Chapter 4
Dispatching Approaches

In this chapter, we discuss dispatching approaches. Dispatching is on the lowest level of the PPC hierarchy described in Chap. 2. We start with a taxonomy of dispatching rules. A dispatching rule assigns a certain index to each job waiting in a queue to be processed on a machine or transported by a vehicle. The job with either the largest or the smallest index is selected to be processed or transported next. Typical attributes are the ready time, the processing time, or the due date of a certain operation or of the job itself. We describe simple dispatching rules that are characterized by an index that is based only on a small number of attributes of a job or the BS and the BP. We continue with the discussion of composite dispatching rules. Composite dispatching rules have an index that is formed by combining indices of simple dispatching rules.

Batching rules are an extension of dispatching rules. A batching rule helps to make the batch formation decision, i.e., which jobs are part of the batch, and also to decide which batch has to be processed next on an available machine. Some dispatching and batching rules use only information related to the jobs that wait in front of a machine or machine group, while other rules may use information regarding machines different from this machine or machine group. The second type of rules are called time-based look-ahead rules.

More sophisticated approaches like rule-based systems and several other approaches to find parameters in batching and dispatching rules are discussed. We present methods to weight indices so that a rule simultaneously works towards multiple objectives. Finally, we also describe an approach to discover appropriate dispatching rules using GAs and discrete-event simulation.

4.1 Motivation and Taxonomy of Dispatching Rules

Dispatching is on the lowest level of the PPC hierarchy described in Chap. 2. Dispatching rules are used to choose the next job that is processed or

L. Mönch et al., *Production Planning and Control for Semiconductor Wafer Fabrication Facilities*, Operations Research/Computer Science Interfaces Series 52, DOI 10.1007/978-1-4614-4472-5_4,

transported by a resource. When the resource is a machine, a dispatching rule selects the next job to be processed among the jobs that are waiting in front of a machine group [29, 116, 240]. When the resource is given by a vehicle, the job is selected among the jobs that wait to be transported by the available vehicle. Note that this is a resource-based point of view, i.e., the resulting dispatching rules are resource-initiated. On the other hand, job-initiated dispatching rules are also possible. In this situation, jobs seek resources with free capacity. This might be important when no job is queueing in front of several available resources and a new job arrives. In this chapter, we focus on resource-initiated dispatching rules because they are more important during heavy material flow.

Dispatching rules are generally myopic in time and space, and it may be difficult to know how and when to adapt them to different situations on the shop floor. However, their decision logic is easy to understand, and they can be implemented with less effort on the shop floor of a wafer fab. Dispatching rules are still the main production control ingredients in many wafer fabs as indicated by Pfund et al. [234] and Sarin et al. [274].

A dispatching rule ranks all the waiting jobs according to an index

$$I_j(a_1,\ldots,a_k) := f(a_1,\ldots,a_k), \tag{4.1}$$

where j denotes the job and $a_i, i = 1,\ldots,k$ are attributes that determine the priority of j. They can be related to j or to properties of the BS or the BP. The right-hand side of expression (4.1) is a function $f : \mathbb{R}^k \to \mathbb{R}$. Often, the indication of all attributes will be suppressed, and only some of the attributes appear on the left-hand side of the priority index. We always assume that the job with either the largest or the smallest value of $I_j(a_1,\ldots,a_k)$ is selected as the job that will be processed or transported next. Based on the number and the nature of the attributes and the form of f, we can develop a taxonomy of dispatching rules.

Simple rules use only one to three attributes to determine the value of the priority index, i.e., $k \leq 3$. Note that the value $k \leq 3$ is somewhat arbitrary as the division between simple and composite dispatching rules. The function f often has the form $f(a_1) := a_1$, $f(a_1) := 1/a_1$, $f(a_1,a_2) := a_1/a_2$, or $f(a_1,a_2) := a_1 - a_2$. Composite dispatching rules are based on more than three attributes, i.e., $k > 3$. At the same time, the form of f is usually more complicated in case of composite dispatching rules; exponentiation, summation, or multiplication are often used. Composite dispatching rules combine several simple dispatching rules together.

A second way of classifying dispatching rules is according to the information on which they are based. A local dispatching rule uses only information that is related to the resource for which the jobs are in queue. On the other hand, a global rule is based on information regarding other resources. Somewhat related to global rules are look-ahead rules because they take future job arrivals from upstream machine groups into account.

A third way to classify dispatching rules is to make a distinction on whether the value of the priority index is time-dependent or not. Dynamic rules are time-dependent, i.e., the current time is an attribute in the priority index, while static rules are not time-dependent.

Besides the types of dispatching rules above, more complex dispatching rules can be constructed using truncation, conditioning, and multilevel approaches. Truncating a dispatching rule refers to the situation where jobs are selected according to a certain dispatching rule, excepting the case where a certain condition is not fulfilled any longer for the jobs. For example, a dispatching rule is used to choose the job to be processed next until at least one job has waited no longer than a specified time. Conditioning implies changing rules according to the BS and the BP state. One might switch back and forth, for example, between two different dispatching rules based on some measure of BS and BP congestion. Multilevel rules may be used to apply tie-breaking or secondary criteria; for example, one might first select the jobs with a small value according to a certain criterion and then use a second dispatching rule to select the job to be processed next among this subgroup of jobs.

In this chapter, we differentiate between simple and composite dispatching rules. We will discuss batching rules because of their practical relevance in wafer fabs. Look-ahead rules are discussed, due to the fact that they are important because of batching- and setup-related decisions.

Usually, discrete-event simulation is used to predict/assess the performance of dispatching approaches in semiconductor manufacturing. Each simulation tool owns a number of built-in dispatching rules. Often, new dispatching rules can be added by means of customizing the simulation tool. It is pointed out by Fowler et al. [87] that comprehensive testing of previously developed flow control approaches in realistic settings is needed. Finally, it is stated in [51, 307] that contradictory results with respect to the performance of dispatching rules are common in complex job shops. It is pointed out by Geiger et al. [96] that the only general conclusion from many years of research on dispatching rules is that there is no dispatching rule that outperforms consistently all the other rules under a variety of BS and BP conditions and performance measures.

We will see in Chap. 5 that dispatching rules can also be used to make scheduling decisions in the list scheduling framework. For some specific situations, even the application of simple dispatching rules within a list scheduling approach leads to the optimal solution. The main idea of this framework consists in applying dispatching in a repeated manner. Hence, a solid knowledge of dispatching rules is also beneficial for scheduling.

There is some relationship between dispatching and order release schemes. We will see in Chap. 6 that many papers support the thesis that when order release approaches become more effective, dispatching decisions will have a diminishing effect on the BS performance.

4.2 Simple Dispatching Rules

In this section, we differentiate between dispatching rules for selecting the next job to be processed on an available machine and to find the next job to be transported by an available vehicle.

4.2.1 JS-Related Dispatching Rules

The FIFO dispatching rule is a popular example of a simple dispatching rule. The corresponding priority index is given by

$$I_j := r_j. \tag{4.2}$$

The job with the smallest index is selected next. The FIFO rule is included as a default rule in virtually all discrete-event simulation packages. Another important example is given by the earliest due date (EDD) dispatching rule. Its priority index is as follows:

$$I_j := d_j. \tag{4.3}$$

EDD selects the job that has the smallest due date. The intention of the EDD rule is to ensure a high on-time delivery performance. Similar to FIFO, EDD is included in most simulation packages. A variant of the EDD rule is the operational due date (ODD) dispatching rule, where the due date of the job is simply replaced by a process step-specific due date d_{jk} for each process step k of job j. These local due dates for the process steps can be obtained from the due dates of the corresponding job by:

$$d_{jk} := d_j - \mathrm{FF} \sum_{h=k+1}^{n_j} p_{jh}, \tag{4.4}$$

where $\mathrm{FF} \geq 1$ is a constant that is called flow factor, p_{jh} is the processing time of operation h of j, and n_j denotes the number of process steps of job j.

Another important rule is the shortest processing time (SPT) dispatching rule. The corresponding index is

$$I_j := 1/p_j, \tag{4.5}$$

where p_j is the processing time of the current process step of job j. This dispatching rule was originally proposed for operating systems where the goal is to keep the number of jobs waiting for a processor as small as possible. It is well known that the SPT rule leads to small CT values in certain types of manufacturing systems because jobs with a small processing time will always be selected first. The application of this rule, a truncation variant of it, and a conditioning variant of SPT for wafer fabs is studied by Rose [267]. It turns out that SPT-type rules do not regularly reduce the ACT value, i.e., for some products this value increases, for some it decreases, and for most of

the products there is no effect. This behavior is caused by the fact that the coefficient of variation of the processing time of the operations is smaller than one for most of the machine groups of a wafer fab, i.e., the processing times of the operations are very similar. Hence, it is hard to predict changes in the CT values in wafer fabs using SPT. Note that the longest processing time (LPT) dispatching rule defined by the index

$$I_j := p_j \tag{4.6}$$

is of course the opposite to the SPT dispatching rule, i.e., the job with the largest processing time is selected first.

Jobs typically are classified in a wafer fab into regular jobs and hot jobs (cf. Sect. 2.2.3). As a result of this classification, different weights can be assigned to the jobs. The highest value first (HVF) dispatching rule selects a job with the highest weight first. The corresponding index of job j is given by

$$I_j := w_j. \tag{4.7}$$

When $w_j = 1$ for all regular jobs, the FIFO dispatching rule is often used as a tie breaker.

A generalization of the SPT rule that incorporates the weights w_j of the jobs is the weighted shortest processing time (WSPT) dispatching rule with index:

$$I_j := w_j / p_j. \tag{4.8}$$

The shortest remaining processing time (SRPT) dispatching rule works towards the selection of jobs that have only a small number of operations to be completed. Its priority index is given by the expression

$$I_j := \sum_{k=l}^{n_j} p_{jk}. \tag{4.9}$$

Totally, $n_j - l + 1$ process steps are necessary to complete job j, i.e., process step l is the current one.

Next, we discuss an example of a dispatching rule that results in good machine utilization and workload balance between the downstream machine groups. It is the fewest lots in the next queue (FLNQ) dispatching rule. This dispatching rule prioritizes jobs with the objective of balancing the workload on different machines. This is accomplished by prioritizing jobs heading to the next operation with the least number of jobs in its queue. The corresponding priority index is given by

$$I_j(t) := n(j), \tag{4.10}$$

where we assume that k is the current process step of job j. Then we denote by $n(j)$ the number of jobs waiting in front of the machine group that corresponds

to the next process step $k+1$ of j. FLNQ-type dispatching rules typically have poor performance in on-time delivery measures like TWT or flow time-related measures like ACT.

The least setup cost (LSC) dispatching rule selects the job to be processed next whose operation requires the smallest setup time among the jobs queueing in front of the machine group. The corresponding priority index is given by

$$I_j := s_{kj}, \tag{4.11}$$

where we denote by s_{kj} the setup time that is needed to process j given that job k was the most recent job processed on the machine. Therefore, the LSC rule is a setup avoidance rule, i.e., when there are jobs that require the current setup state on the machine, then these jobs are selected among the queued jobs. This is a dispatching rule commonly used by many practitioners for dispatching and scheduling problems with sequence-dependent setup times. Whenever there is a job and a machine available, the LSC rule searches for the machine/job combination that causes the least setup time and selects this job to be processed on that machine. When a job is available and the amount of setup between two or more available machines is the same, one will be selected randomly. However, one could also use other tie breakers. We will later see in Sect. 4.3.2 how this rule is used to construct composite dispatching rules.

The flow control (FC) dispatching rule is somewhat similar to the FLNQ rule. The FC dispatching rule calculates the number of remaining production hours per machine for the next machine group in the product's route. More formally, the corresponding index is given by

$$I_j := \frac{m}{\sum_{l \in J(M)} p_l}, \tag{4.12}$$

where we again assume that k is the current process step of j. The quantity m is the number of machines in the machine group M that corresponds to the next process step $k+1$. Finally, we sum up the processing times p_l of all jobs queueing in front of this machine group. This job set is denoted by $J(M)$.

While the dispatching rules discussed so far are general purpose rules, we now discuss an important class of simple dispatching rules for wafer fabs. As defined by Lu et al. [169], fluctuation smoothing policies are a subclass of the least slack (LS) dispatching rule. Known as minimum slack (MS), these dispatching rules give the highest priority to those jobs where the slack is the smallest. The slack of job j that is in buffer b_i is defined by

$$s(j) := \beta(j) - \gamma_i, \tag{4.13}$$

where $\beta(j)$ is the real number attribute of j that is associated with j when it enters the wafer fab. Furthermore, the buffer b_i is associated with the real number γ_i. Of course, we set again

$$I_j := s(j) \tag{4.14}$$

for the resulting index. We will see that different dispatching rules can be obtained from expression (4.13) using particular choices for $\beta(j)$ and γ_i. Among them, there are fluctuation smoothing policies. These policies are used to reduce the mean and standard deviation of CT. This is important because by reducing ACT, the product can get to market faster and can keep up with the changing environments associated with semiconductor manufacturing. Also by reducing the Var(CT) value (see Sect. 3.3.1), companies can predict the completion time of a product much more accurately.

The first fluctuation smoothing policy is a policy for reducing the Var(L) value. It evaluates the slack of a job and lets the one with the least slack process on the available machine first. In this situation, the slack is defined by setting $\beta(j) := d_j$ and $\gamma_i := \xi_i$, where ξ_i is an estimate for the remaining CT of j at the process step that is associated with b_i. Then the quantity $d_j - \xi_i - t$, where t is the current time, is a measure for the urgency of job j. Because all the jobs queueing in buffer b_i have the current time t as a common term, it can be ignored, and consequently it is enough to consider $d_j - \xi_i$ as slack. This rule is also known as the LS dispatching rule. In this situation, we use the crude estimate $\xi_i := \sum_{k=l}^{n_j} p_{jk}$ for the remaining processing time of job j in the wafer fab to obtain the classical global LS rule. Note that the dispatching rule based on index (4.14) attempts to make each job equally late or equally early. The Var(L) value is small when all jobs are either equally too early or too late. Therefore, the resulting dispatching rule is called the fluctuation smoothing for variance of lateness (FSVL) policy (see Lu et al. [169]).

The second fluctuation smoothing policy is used to reduce the Var(CT) value. Therefore, we call the resulting policy FSVCT. It also evaluates the slack of a job and places the job with the least slack on the machine that is associated with buffer b_i next. The slack for this rule is defined by setting $\beta(j) := r_j$ and again $\gamma_i := \xi_i$. This is of course a CT-related measure. Hence, FSVCT attempts to reduce the Var(CT) by a similar argumentation as for the FSVL policy.

The third fluctuation smoothing policy is intended to reduce the ACT value. The resulting dispatching rule is called fluctuation smoothing for mean cycle time (FSMCT) policy. It is known from queueing theory (cf. the description in Sect. 3.2.7 and the Kingman approximation) that the delay of jobs at a server is caused by the burstiness of the job arrivals and the variations in the service time. We cannot influence the service times. Because of this, following Lu et al. [169], we are interested in simultaneously reducing the burstiness of arrivals to all the buffers of the wafer fab. The resulting policy is therefore appropriate for reducing the ACT value.

We start by describing how we reduce the burstiness in the arrivals to buffer b_{k+1}. The basic idea is to set periodic due dates for jobs to reach b_{k+1}. We denote by λ the mean release rate for the wafer fab. Hence, the mean inter-arrival time between jobs is given by $1/\lambda$. Therefore, we set n/λ as the

due date to reach b_{k+1} of the nth job that is released into the wafer fab. When we are able to reduce the variance of lateness Var(L) for reaching b_{k+1}, we obtain an arrival stream at b_{k+1} that is almost deterministic and therefore not bursty.

In order to reduce Var(L) for reaching b_{k+1}, we consider b_{k+1} as the final sink of the wafer fab and use FSVL with respect to this system. We define by ξ_i^k an estimate for the remaining partial CT to go from b_i to b_{k+1}. In this situation, we define $\beta(j) := n/\lambda$, where n is the nth job released into the wafer fab and $\gamma_i := \xi_i^k$. Of course, we have

$$\xi_i^k = \xi_i - \xi_{k+1}, \tag{4.15}$$

and therefore $\gamma_i = \xi_i - \xi_{k+1}$. But because we consider a fixed b_{k+1}, the summand ξ_{k+1} is common for all jobs at the buffers $b_i, i = 1, \ldots, k$ and can be omitted. Therefore, we use

$$s(j) = n/\lambda - \xi_i. \tag{4.16}$$

The expression (4.16) is so far only defined for jobs in buffers $b_i, i \leq k$. Because we are interested in simultaneously reducing the burstiness of job arrivals at all buffers, we extend expression (4.16) to all buffers.

However, it is important to come up with appropriate ξ_i values in expression (4.16). It is discovered in Lu et al. [169] that iterative simulation is quite effective to solve this problem. As described in Sect. 3.2.8, the initial value $\xi_i^{(0)} \equiv 0$ is used within the first simulation run. We obtain estimates $\hat{\xi}_i^{(0)}$ from the first simulation run for the CT of the remaining process steps that are associated with b_i. The new setting $\xi_i^{(1)} := \hat{\xi}_i^{(0)}$ is used in the second simulation. This procedure is repeated in an iterative manner until the difference between two consecutive $\hat{\xi}_i^{(s)}$ and $\hat{\xi}_i^{(s+1)}$ values is small. We refer to Sect. 4.7.2 where more implementation details for a similar problem are presented.

Finally, we refer to Sarin et al. [274] for a more complete description of various simple dispatching rules used in wafer fabs.

4.2.2 MS-Related Dispatching Rules

In this section, we describe how the next job is selected to be transported by a vehicle. As in the case of JS-related dispatching rules, we specify the corresponding dispatching rules by priority indices. In MS-related dispatching rules, we assume that a given set of transportation jobs is waiting to be transported. For simplicity reasons, we assume that each transportation job consists of one job that is to be processed within a wafer fab, usually a FOUP. Performance measures of interest for AMHS are the number of carrier moves, i.e., TP and CDT (see Sect. 3.3.1).

The nearest job first (NJF) dispatching rule is an intuitive greedy heuristic that is defined as follows. An empty vehicle is dispatched to the job whose waiting point is the nearest to the current location of the empty vehicle c. The following index is used to assess job j:

$$I_j := d(j,c), \tag{4.17}$$

where $d(j,c)$ denotes the length of the shortest path between j and c. The job with the smallest index will be selected. It is clear that NJF can lead to high effective utilization of the AMHS, i.e., loaded travel. However, this rule does not take into account the waiting time of jobs for an empty vehicle. Therefore, there is the possibility that some jobs wait a long time for a transport, i.e., the variance of CDT is large. Another limitation of NJF is that this rule becomes inefficient in situations when there are several vehicles available to transport jobs, but they are being blocked by the first vehicle (cf. Liao and Fu [161]).

The first limitation of the NJF dispatching rule is resolved by the longest waiting time (LWT) dispatching rule. It dispatches an empty vehicle c to the job with the LWT among the jobs that are ready for transportation. The corresponding priority index is given by:

$$I_j := wt_j, \tag{4.18}$$

where we denote by wt_j the waiting time of job j for transportation. The job with the largest index is selected next. It is clear that the LWT rule is similar to the FIFO dispatching rule for the BS. Of course, the LWT rule tends to reduce the variation of CDT at the expense of effective vehicle utilization. The second limitation of the NJF rule is considered within the design of the farthest job first (FJF) dispatching rule. Blocking effects can be avoided by dispatching an empty vehicle to the farthest job away from it. The corresponding index is given by expression (4.17). However, we select the job with the largest value of the index in this situation.

The modified nearest job first (MNJF) dispatching is proposed by Liao and Fu [161] to combine the advantage of the NJF and the LWT rule. The rule works as follows. An empty vehicle is dispatched to the job with the longest waiting time, when this time is longer than a threshold value τ. When the set of those jobs is empty, the NJF dispatching rule is applied to determine the job that is selected next. The MNJF rule is a truncation dispatching rule.

The results of simulation experiments provided in [161] show that NJF performs well with respect to TP and average delivery time. But at the same time, the variation in CDT is large when NJF is used. The performance of the MNJF rule, however, is very close to that of NJF, but the CDT variation is much smaller because of the truncation strategy.

Dispatching rules for AMHS that take hot jobs into account are proposed by Liao and Wang [162]. These rules allow for an almost no-wait transport of these high-priority jobs. Additional job-initiated dispatching rules for the MS

are described by Lin et al. [163]. Note that in addition to location and waiting time-related attributes, attributes that take the situation at the different buffers associated with an AMHS into account can be used.

4.3 Composite Dispatching Rules

In this section, we discuss two classes of JS-related composite dispatching rules and one MS-related composite dispatching rule.

4.3.1 Critical Ratio Dispatching Rules

We start with the critical ratio (CR) dispatching rule. It is defined as the ratio of EDD- and SRPT-type rules. Its priority index is given by the following expression:

$$I_j(t) := \frac{d_j - t}{\sum_{k=l}^{n_j} p_{jk}}, \tag{4.19}$$

where l denotes the current process step of j. The job with the smallest CR index is selected first because in this situation, $d_j - t$ is small relative to the remaining processing time, i.e., the job is already late or has only a small amount of slack. Note that the priority index given by expression (4.19) is negative when $t > d_j$. In this situation, the job is already late, as its due date has passed. When $0 \leq I_j(t) \leq 1$, then the job j will most likely be late. Finally, an on-time job will have a critical ratio not smaller than one. A small value for $\sum_{k=l}^{n_j} p_{jk}$ often means that only a small number of process steps to be performed are left. Of course, the numerator $d_j - t$ in expression (4.19) can be replaced by $d_j - \sum_{k=l}^{n_j} p_{jk} - t$. However, this leads to the constant term 1 in the priority index value of all jobs and can therefore be omitted. Sometimes, the following dispatching rule

$$I_j(t) =: \begin{cases} (d_j - t)/\sum_{k=l}^{n_j} p_{jk}, & \text{if } t \leq d_j \\ 1/((t - d_j)\sum_{k=l}^{n_j} p_{jk}), & \text{otherwise} \end{cases} \tag{4.20}$$

is also referred to as the CR rule (see Rose [268]). This rule is an example of a conditioning dispatching rule. When $d_j \geq t$, then a priority index similar to index (4.19) is used, whereas in the case of $d_j < t$, a large value for $t - d_j$ and a large value for $\sum_{k=l}^{n_j} p_{jk}$, i.e., a job that is already late has to perform many process steps, lead to a small value of the priority index.

Results of simulation experiments with CR-type dispatching rules can be found in Rose [268]. This study shows that an appropriate due date setting has a large impact on the performance of CR-type dispatching rules. It is shown that in the case of rather tight due dates, CR-type rules do not perform well with respect to on-time delivery performance and CT.

4.3.2 ATC-Type Dispatching Rules

The apparent tardiness cost (ATC) dispatching rule is proposed by Vepsalainen and Morton [311]. It attempts to reduce TWT values. It is a composite dispatching rule that blends the WSPT and the LS dispatching rules. Its priority index is given by

$$I_j(t) := \frac{w_j}{p_j} \exp\left\{-\frac{(d_j - p_j - t)^+}{\kappa \bar{p}}\right\}, \tag{4.21}$$

where $\bar{p}$ is the average processing time of the jobs that are queueing in front of the machine group, t is the time where the next machine becomes available, and finally κ is a scaling parameter that weighs the slack against the WSPT priority index. We use the notation $x^+ := \max(x, 0)$ for abbreviation. The κ parameter is called the look-ahead (or scaling) parameter. Note that there is an implicit scaling parameter 1 on the WSPT term. It is well known that the performance of ATC-type dispatching rules strongly depends on an appropriate setting of the κ parameter.

As ATC-type rules are found to be sensitive to the κ value setting, ten different κ values from 0.5 to 5 in increments of 0.5 are used by Mehta and Uzsoy [180]. This approach is called grid search in Pfund et al. [236]. For a particular situation, i.e., a certain number of jobs waiting for processing, the ATC rule is used independently for each κ value, the TWT value is calculated for all queueing jobs, and finally the κ value is chosen that provides the smallest TWT value. The κ value chosen is then fixed for a given job set within ATC-type dispatching rules.

Various rules for the selection of this parameter are discussed in Pinedo [240]. The tightness of the due dates, the range of the due dates of the jobs queueing in front of the machine group, and the number of jobs per machine can be used to determine the look-ahead parameter. A heuristic curve-fitting method is used to determine the equations for calculating proper values of the look-ahead parameters. Neural networks are proposed by Kim et al. [140] to find appropriate look-ahead parameters.

There are generalizations of the ATC dispatching rule with respect to sequence-dependent setup times and unequal ready times of the jobs. We consider first the case of sequence-dependent setup times. Because we often have to deal with sequence-dependent setup times, an extension of the ATC dispatching rule to this situation is proposed by Lee and Pinedo [159]. This dispatching rule combines the WSPT dispatching rule, the LS rule, and the LSC rule into a single priority index. The corresponding priority index of job j at time t when job l is processed is given by

$$I_j(t,l) := \frac{w_j}{p_j} \exp\left\{-\frac{(d_j - p_j - t)^+}{\kappa_1 \bar{p}}\right\} \exp\left\{-\frac{s_{lj}}{\kappa_2 \bar{s}}\right\}, \tag{4.22}$$

where s_{lj} denotes the setup time that is required to process job j immediately after job l. The average setup time of the waiting jobs is denoted by $\bar{s}$. This dispatching rule is called ATC with setups (ATCS) for abbreviation. Different expressions to determine appropriate values for κ_1 and κ_2 based on attribute values of the waiting jobs are proposed by Lee and Pinedo [159]. Neural networks are used for the same purpose in Park et al. [229]. Chen et al. [45] develop a four-phase method to determine a set of scaling parameter values that perform well over a wide range of problem instances, i.e. provide robust performance. In the first phase, factor ranges that characterize the problem instances in each machine group are calculated. In the second phase, a face-centered cube design is used to decide the placement of design points in the factor region. The third phase involves adding an explicit scaling factor for the WSPT term and then using designed experiments to find good scaling parameter values at each design point. In the last phase, the central point of the area in which all of the good scaling parameters lie is identified with the robust scaling parameter.

In the case of unequal ready times, it makes sense to wait for a future job arrival in some situations. The resultant dispatching rule is called ATCR, and the corresponding priority index is given by

$$I_j(t) := \frac{w_j}{p_j} \exp\left\{-\frac{(d_j - p_j - t)^+}{\kappa_1 \bar{p}}\right\} \exp\left\{-\frac{(r_j - t)^+}{\kappa_2 \bar{p}}\right\}, \tag{4.23}$$

where κ_1 and κ_2 denote the look-ahead parameters for the slack- and the ready-time-related terms of the priority index, respectively. The slack-related term can be motivated in a similar way as in the ATC priority index (4.21) without ready times, while the ready-time-related term is responsible for reducing the job priority when a job is not ready at t. Results of computational experiments for an approach based on inductive decision trees to select the two look-ahead parameters are presented by Zimmermann et al. [332].

The index (4.22) is extended by Pfund et al. [236] to the situation where ready times of the jobs occur. The resulting priority index is called ATCSR. It is given by

$$I_j(t,l) := \frac{w_j}{p_j} \exp\left\{-\frac{(d_j - p_j - \max(r_j,t))^+}{\kappa_1 \bar{p}}\right\} \exp\left\{-\frac{s_{lj}}{\kappa_2 \bar{s}} - \frac{(r_j - t)^+}{\kappa_3 \bar{p}}\right\}. \tag{4.24}$$

A grid search approach is used to determine appropriate $(\kappa_1, \kappa_2, \kappa_3)$ triples. Furthermore, regression-based approaches are proposed for the same problem. Note that it is also possible to use more sophisticated approaches to calculate the slack of a job within priority indices of ATC-type rules.

Usually, ATC-type dispatching rules lead to small TWT values. However, there is some effort required to find appropriate look-ahead parameters. The results of extensive simulation experiments with different composite dispatching rules in wafer fabs are presented by Bahaji and Kuhl [16].

4.3.3 Composite Dispatching Rules for the MS

Following Jeong and Randhawa [128], we present a dispatching rule for vehicles that combines several attributes to achieve multiple performance measures simultaneously. We assume that each machine has an input buffer and an output buffer. The corresponding priority index is given as follows:

$$I_j := \alpha_1 D_j + \alpha_2 \mathrm{IQ}_j + \alpha_3 \mathrm{OQ}_j, \tag{4.25}$$

where D_j is the unloaded travel distance of an idle vehicle to job j, IQ_j is the remaining space in the input buffer of a machine that is the destination of j, and OQ_j is the remaining space in the outgoing buffer of a machine that is the source of j. The quantity D_j is defined as follows:

$$D_j := \begin{cases} \frac{\max d_k - d_j}{\max d_k - \min d_k}, & \text{if } \max d_k \neq \min d_k \\ 1, & \text{otherwise} \end{cases}, \tag{4.26}$$

where d_k is the distance of the current location of the available vehicle to the machine where job k is in the output buffer. The quantity IQ_j is given by

$$\mathrm{IQ}_j := 1 - n_j^{iq}/c_j^{iq}, \tag{4.27}$$

where n_j^{iq} is the number of jobs in the current input queue of the machine that is the destination of j. The quantity c_j^{iq} represents the capacity of the incoming buffer for the machine to which job j, which is the first job in the output buffer of a machine, is going to move. Finally, the quantity OQ_j is given by

$$\mathrm{OQ}_j := n_j^{oq}/c_j^{oq}, \tag{4.28}$$

where we denote by n_j^{oq} the current number of jobs in the output buffer of the machine that contains j, and c_j^{oq} is the capacity of this output buffer.

The first part of the rule prioritizes the job that is the closest to the newly available empty vehicle, while the second part tends to prefer jobs that are going to machines that have a low queue size in their input buffers. The third part prefers jobs in the output buffers of machines that have a large queue length of the outgoing buffer. Furthermore, the three parts of dispatching rule (4.25) are weighted by using $\alpha_i \geq 0$ and $\alpha_1 + \alpha_2 + \alpha_3 = 1$. A large value of the first part leads to a high utilization of the AMHS, whereas a large value for the second part leads to a decreased value of carrier waiting time because a destination machine whose input buffer is full is not able to accept another job unless the queue is cleared. The third part reduces the probability of machine blocking.

Note that JS- and MS-related dispatching rules are generally considered separately. There are only a few papers that treat them simultaneously (cf. Tyan et al. [300], for example).

4.4 Simulation Results for Assessing Dispatching Rules

We describe the results of a simulation study due to Bullock et al. [37] using the MIMAC 1 model [83] and the FIFO, EDD, SPT, ATCS, FSVL, FSVCT, and the FSMCT dispatching rules. The ATCS dispatching rule is applied only for the steppers. For the remaining machines, the FIFO dispatching rule is used in this situation. The model contains two products, 83 machine groups, and 32 operators. Reentrant flows and rework are contained in the model. Other characteristics of the MIMAC 1 model are shown in Table 4.1.

Table 4.1: Characteristics of the MIMAC 1 model

Product	Weight	Wafers per job	Release size (in jobs)	Constant time until next release (hours)	Raw processing time (days)
1	1	48	1	3.034	13.1
2	5	48	1	6.048	14.9

We consider the performance measures ACT, Var(CT), Var(L), TP, AT, and finally TWT. The simulation time is three years. The first year is truncated to eliminate the initialization bias. Because due dates are not included in the MIMAC 1 model, we set due dates according to

$$d_j := r_j + \sum_{k=1}^{n_j} p_{jk}. \tag{4.29}$$

Ten independent replications of each simulation run are performed to obtain statistically reasonable results. The results of simulation experiments are shown in Table 4.2.

Table 4.2: Simulation results for different dispatching rules

Rule	ACT (days)	Var(CT)	TP (jobs)	AT (days)	Var(L)	TWT (days)
FIFO	38.09	13.67	5,723	24.41	28.63	452,679.10
EDD	25.66	0.15	5,823	11.98	0.26	238,774.59
SPT	25.62	0.03	5,830	11.94	0.30	253,631.29
ATCS	29.34	3.65	5,813	15.66	11.14	189,516.95
FSVL	26.08	0.19	5,827	12.40	0.34	256,435.18
FSVCT	26.53	0.04	5,832	12.85	0.15	275,690.00
FSMCT	25.74	0.04	5,837	12.06	0.07	250,554.04

It turns out that FIFO is outperformed by the remaining dispatching rules with respect to all performance measures because it does not take into account any of the attributes of the jobs except the ready time. The SPT rule performs best with respect to CT and Var(CT). The different fluctuation smoothing policies perform well with respect to Var(CT) and Var(L). As expected, the ATCS dispatching rule outperforms the other rules with respect to TWT, followed by the EDD rule. It is interesting to note that the ATCS rule does not perform well with respect to AT. This can be explained by the fact that this dispatching rule takes the weights of the jobs into account. Only the FIFO rule performs worse.

Related simulation experiments for larger wafer fabs can be found in Mittler and Schömig [186, 187]. The fluctuation smoothing policies especially show a similar behavior in the case of large-scale models. Simulation results for ATC-type dispatching rules in large-scale wafer fabs are presented by Mönch and Zimmermann [200].

4.5 Batching Rules

In this section, we only consider the case of parallel batching, i.e., several jobs can be processed at the same time on the same machine. We assume that at most B jobs can be batched together, i.e., B is the maximum batch size. We note that it is possible to make batch formation decisions based on the number of wafers instead of the number of jobs, but in this monograph, all decisions are based on the number of jobs. The set of jobs that can be used to form a batch is called a family as described for diffusion furnaces in Sect. 2.2.3. We assume that we have f incompatible job families.

Batching rules can be seen as a generalization of dispatching rules, i.e., we have $B = 1$ in the case of pure dispatching rules. When a batch-processing machine becomes available, the next batch has to be formed and then selected to be processed on this machine. Therefore, batch formation is an additional decision when compared to a pure dispatching rule. We assume in the beginning that a large number of jobs are queueing in front of the batch machine group.

One possible way to solve this problem consists of selecting a job among the queueing jobs using one of the dispatching rules described in Sect. 4.2 or 4.3. Then at most $B-1$ additional jobs are selected according to certain criteria among the jobs that are queueing in front of the batch machine group.

The following algorithm is used to determine the batch to be processed next, when only one priority index I_{ij} is used to assess all the jobs j of family i. We assume that jobs with large I_{ij} are selected next to be processed within a batch.

Batching algorithm (BA)

1. Sort all the job within each family $i = 1, \ldots, f$ in nonincreasing order with respect to I_{ij}.

2. Let us denote the length of the list that corresponds to family i by $l(i)$. Consider the first $\min(B, l(i))$ jobs of each family to form batch $B(i)$ of family i. Denote the number of jobs within $B(i)$ by $|B(i)|$.
3. Select the batch with the largest value

$$I_{B(i)} := \frac{|B(i)|}{B} \sum_{j \in B(i)} I_{ij} \tag{4.30}$$

among the families $i = 1, \ldots, f$ to be processed next.

It is clear from expression (4.30) that we assess each batch by taking the sum of the priority indices of the jobs that form the batch. The factor $|B(i)|/B$ makes sure that full batches are preferred compared to batches that contain only a small number of jobs.

We consider two examples for this approach. We may use the EDD dispatching rule to determine the most important jobs within each family. We have to modify priority index (4.3) to $I_{ij} := -d_{ij}$, where we denote by d_{ij} the due date of job j of family i to align with the algorithm BA. Similarly, we can use the ATC dispatching rule with priority index (4.21). The resulting batching rule is called the batched ATC (BATC) rule.

Next, we study the case where not enough jobs are available to form a full batch. In this situation, a decision has to be made whether to start the batch that is not full or to wait until enough jobs are available. We start with the single product case. Let L be the number of jobs in the queue in front of the batch machine group. The resulting decision rule can be formulated as follows.

Algorithm Minimum Batch Size (MBS)

1. Anytime there are at least S jobs in the queue with $S \leq B$, then a batch can be processed. The quantity S is called the minimum batch size.
2. When there are fewer than S jobs in the queue and a machine is available, the machine will remain idle.

Two special cases of the MBS rule are important, namely MBSG where $S = 1$, called the greedy batch policy, and thereby loading the machine every time a new job appears; and MBSF where $S = B$, called the full batch policy, and consequently not loading the machine until it can be loaded at full capacity.

The algorithm MBS has one advantage that makes it appealing to the fab manager. It is extremely easy to implement on the shop floor. This is because MBS requires minimal computation and real-time information to make a decision. The MBS rule is considered as a theoretical standard that is used to assess the performance of other batching policies. This is discussed by Deb and Serfozo [62] who showed that with Poisson arrivals, the MBS rule is the optimal policy among those that only consider the current status of the BS.

However, MBS also has several disadvantages. The first drawback of MBS is how to determine an appropriate minimum batch size prior to, and

during, implementation. Although Gurnani et al. [111] introduce an algorithm for computing the critical number S, the calculation of this value in practice is often quite difficult considering that product mix and production rates typically change according to business forecasts. As a result, the optimal MBS will change as production changes. Another drawback of the MBS algorithm is that it fails to consider the current state of the BS as well as the impact of its dispatching decisions on the entire system. Thus, the MBS batching decision may result in wasted capacity if the MBS chosen is less than the capacity of the machine or if additional idle time is necessary to form a batch. Ultimately, MBS provides local decisions.

Of course, it is possible to combine BA and MBS. This offers possibilities to extend MBS to the multiproduct situation. In this case, we apply MBS to the families where $|B(i)| < B$. Then, we apply the index (4.30) to assess the batches formed for the families that can offer a batch to be processed next.

It is a weakness of the algorithms BA, MBS, and the resulting hybrids that they consider only jobs that are available at the time when the batch has to be formed. But it is intuitively clear that it is reasonable to exploit knowledge of the expected state of the BS. Therefore, it sometimes makes sense to start a non-full batch or to decide to wait for jobs that arrive in the future to increase the fullness of batches. We will study the corresponding look-ahead rules in Sect. 4.6.

4.6 Look-Ahead Rules

Look-ahead rules are dispatching or batching rules that take information related to future job arrivals into account. Such information is important in the case of sequence-dependent setup times and in the case of batch processing. This kind of real-time information is available from the MES in most wafer fabs. In the first case, information with respect to future job arrivals might avoid expensive setup changes by waiting for a job that requires the same setup state, but it is not available now. In the latter case, in some situations, it is reasonable to wait for future job arrivals to increase the fullness of a certain batch. In the remainder of this section, we discuss a rule that dynamically selects the batch size and a rule that makes decisions based on the next arrival of jobs. Additional look-ahead research is also briefly discussed. Finally, we describe batching rules that are generalizations of the algorithm BA using ATC-type heuristics.

4.6.1 Dynamic Batching Heuristic

One possibility for improving the MBS rule is to make more intelligent decisions on batching and possibly wasting less capacity. This can be accomplished by using real-time information to gain knowledge about future

states of the BS. As a result, Glassey and Weng [101] introduce the dynamic batching heuristic (DBH), a heuristic that incorporates real-time data into the decision process.

DBH dynamically determines the batch size of the batch to be formed and processed next based on the BS status at time t_0 and whether idle time to wait for additional jobs should be inserted to minimize total overall delay at the batch machine. It is shown in [101] that the look-ahead procedure of DBH is beneficial with respect to average delay at the batch machine, i.e., the average waiting time of the jobs in queue.

In order to sketch the main idea of DBH, we introduce the following additional notation:

t_0 : time epoch that the batch machine is idle and the number of jobs in the queue is positive
t_j : arriving epoch of the next jth job after t_0
q : number of jobs in queue at t_0
T : processing time
L : look-ahead number, where we assume that the next L arrival epochs are known with certainty at t_0

DBH is proposed for a single batch machine and a single product. The DBH formulation is based on the following two insights:

1. The time of loading the batch-processing machine is either the time that it becomes idle and there are jobs waiting or, if it has been idle, at the time when some job arrives.
2. The batch machine starts service right away, when $q \geq B$. Waiting will only result in more delay under such circumstances.

Since it becomes difficult to predict the later arrivals in a long planning horizon, the DBH is proposed for operating the batch machine in a planning horizon that is equal to the processing time T of the product. This situation is shown in Fig. 4.1. The overall heuristic can be summarized as follows.
Algorithm DBH

1. If the batch machine becomes idle, we have to differentiate two cases. In the first case, i.e., when jobs are in the queue, go to step 2. Otherwise, if the queue is empty, go to step 3.
2. Let t_0 be the time epoch that the batch machine becomes idle. Start the decision heuristic described below.
3. Wait until a job arrives. Let t_0 be its arrival epoch. Start the decision heuristic described below.

We now describe the decision heuristic used in the DBH algorithm. We assume $q < B$ because otherwise we will always start processing a full batch. When the q jobs available at time t_0 are processed immediately at time t_0, the other jobs arriving at t_j, where $t_j < t_0 + T$, will stay in queue at least $T + t_0 - t_j$. On the other hand, when we wait for j job arrivals and then load $q + j$ jobs at time epoch t_j, then each queued job has to wait $t_j - t_0$ time units. Therefore,

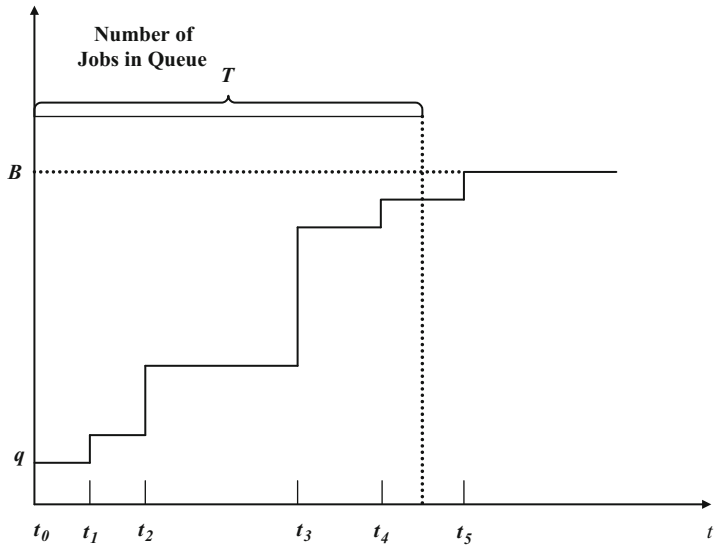

Figure 4.1: Arrival of jobs in the queue of a batch machine over time

the total delay will increase by $(t_j - t_0)q$. The job that arrives at t_j will be processed immediately. Therefore, the delay will be decreased by $(T + t_0 - t_j)j$ because the j jobs that arrived after t_0 will be processed and not have to wait longer. Therefore, the determined net saving is

$$\text{Net}(t_j) = (T + t_0 - t_j)j - (t_j - t_0)q. \tag{4.31}$$

When $\text{Net}(t_j) > 0$, then the delay can be reduced by waiting until t_j to start the batch. In contrast, when $\text{Net}(t_j) < 0$, then the corresponding delay is increased. In case of $\text{Net}(t_j) = 0$, there is no gain or loss. Therefore, it is reasonable to wait until t_i, where i is given by

$$i = arg\max_j \left\{ (T + t_0 - t_j)\, j - (t_j - t_0)\, q \mid 0 \le j \le j_{\max} \right\}. \tag{4.32}$$

We have

$$j^* := arg\max_j \{ t_j \mid t_j \le t_0 + T \}, \tag{4.33}$$

i.e., j^* denotes the maximum number of arrivals within T, and we define $j_{\max} := \min\{j^*, B - q, L\}$. Therefore, $t_{j_{\max}}$ is the last possible loading epoch. It is determined by the look-ahead number L and T. Of course, the setting $L \le B - 1$ makes sense because of the maximum batch size of B and $1 \le q$.

Although DBH performs better than MBS in all situations, it requires much more computation than MBS and also requires real-time information

to make dispatching decisions. Likewise, the number of jobs to look ahead is necessary before DBH can be implemented. It is also inflexible with respect to the planning horizon length.

4.6.2 Next Arrival Control Heuristic

Half of the potential benefit in using DBH is gained by looking ahead only to the next arrival (see Glassey and Weng [101]). This is the starting point for the next arrival control heuristic (NACH) proposed in [84]. Fowler et al. [84] note also that the further ahead one looks, the greater the potential impact of the decision on arrivals that occur outside of the time horizon of T time units as suggested by Glassey and Weng [101]. An extension of the original NACH approach for parallel batch machines and multiple products is provided by Fowler et al. [85].

In this monograph, we describe NACH in a slightly generalized context, where the status of critical machines in subsequent downstream processing is taken into account during batch processing decision-making [287]. We describe a methodology that is intended to balance the time a job spends waiting for batching with the time spent in setup at downstream machines. The resulting heuristic is called NACH-setup. We will use the following notation to describe NACH-setup:

q_j : number of jobs in queue for product j
N : total number of products
T_j : processing time of a batch of product j
S_j : downstream setup required by product j given the current status of the BS
W_j : weighted processing time for a batch of product j
TN_j : time until the next arrival of a job of product j
t_{1j} : time of the next arrival of a job of product j

The NACH-setup logic consists of two cases. The first case, a push decision, occurs when a batch machine is idle and a job arrives. At this point, a trade-off similar to the one employed by DBH is made to determine if the machine should begin processing this product now or wait for the next arrival. The second case, a pull decision, occurs when a machine has just finished processing and must choose whether or not to pull jobs and immediately begin processing again. If the decision is to pull jobs, the type of product to process must be determined.

We start by describing the push decision logic. It is similar to DBH with $L = 1$. A batch loading decision is said to occur at epoch t_0. If there are jobs in queue, t_0 corresponds to the time epoch that the machine becomes idle. Otherwise, t_0 corresponds to the arrival epoch of the next job. The potential times the next load begins is specified by the number of future arrivals that will occur before the next load begins.

If $B \leq q$ at time t_0, then a full load is available, i.e., waiting will only result in an increased delay. Therefore, a full load will be dispatched to the machine.

However, if $0 < q < B$, a decision must be made whether to load the machine immediately or to wait for the next arrival. The decision of whether or not to wait is determined by calculating the net decrease in delay, if any, caused by waiting similar to the case of DBH. We obtain for the corresponding net change in the delay for product j:

$$\Delta_{1j} := (T_j + S_j + t_0 - t_{1j}) - (t_{1j} - t_0)q_j. \tag{4.34}$$

Note that the main difference between expression (4.31) and (4.34) is the additional setup time in the first term of expression (4.34). When $\Delta_{1j} > 0$, then the delay is reduced by waiting until t_{1j} to start the batch. We note that in the case of the push logic, we have to consider only the arriving product type in determining whether to make a batch or not. Let this product type be denoted by j. The push decision can be formalized as follows.
Algorithm Push Decision (Push)

1. Increase the inventory of this product by one. Determine the number of idle batch machines. Denote this value by m_{idle}. If $m_{\text{idle}} = 0$, then stop, i.e., no push decision is made at this point of time. If $m_{\text{idle}} > 1$, then go to step 4. If there is a full load of this product, then also go to step 4.
2. Determine the time of the next arrival of this product, i.e., find t_{1j}. Let t_0 be the current time. If $t_{1j} > t_0 + T_j$, then set $t_{1j} := t_0 + T_j$.
3. Calculate Δ_{1j} for the product that enters the queue to determine whether it is worth waiting for the next arrival of this product before making a batch. If $\Delta_{1j} > 0$, then stop, i.e., no push decision is made at this point of time.
4. Start a batch of this product now on an idle batch machine. Decrease the inventory of this product by the size of this batch.

The more complex issue, the influence of one product on another, is embedded into the pull decision logic. Therefore, we continue by describing the pull decision logic. A pull decision is necessary when a specific batch machine becomes idle immediately after completing its previous batch. Again, the benefit to wait can be expressed as Δ_{1j} using expression (4.34). If we have $\Delta_{1j} > 0$ for all j, then the batch machine has to wait. The pull decision can be summarized as follows.
Algorithm Pull Decision (Pull)

1. If no jobs are waiting, then stop, i.e., no pull decision is required.
   Determine the number of idle batch machines, including this batch machine. Denote this number by m_{idle}. Determine the time of the next batch machine completion. This time is denoted by t_C. Set $t_C := \infty$ if no batch machines are busy. Set the selected product indicator j_{prod} to 0.
2. If there is no full load for any product, then go to step 3.
   Determine the product among those that have a full load that will cause the weighted shortest processing time. Set j_{prod} to that product number and go to step 6.

3. Determine the next arrival time t_{1j} for all j. If $m_{\text{idle}} > 1$ or $t_c < t_{1j}$, then calculate the delay due to processing j (see Eq. (4.37)). Set j_{prod} to that product number of the product with the corresponding minimum value and go to step 6.
4. Calculate Δ_{1j} for each j to determine whether it is worth waiting for the next arrival of that product to appear before making a batch.
   If it is worth waiting until the next arrival appears for all products, i.e., if $\Delta_{1j} > 0$ for all products, then stop. In this case, no pull decision has to be made.
   On the other hand, if it is not worth waiting until the next arrival for any of the products, i.e., $\Delta_{1j} \leq 0$, set j_{prod} to the product number of that product that causes no setup time. If no products exist without setup time, set j_{prod} to the product number of the product with the weighted shortest processing time and go to step 6.
5. For each product for which it is worth waiting for the next arrival, i.e., $\Delta_{1j} > 0$, determine the total waiting time incurred by all products by waiting for the next arrival of this product.
   For each product for which it is not worth waiting for the next arrival, determine the total waiting time incurred by all products when a batch of this product is started now.
   Determine the minimum of the above and set j_{prod} to the product number of the corresponding product.
   If the minimum is for a product for which it is worth waiting, then stop. In this case, no pull decision is required.
6. Start a batch of product j_{prod} on the batch machine that just completed. Decrease the inventory of that product by the size of the batch.

If it is determined that all products should start processing now, the algorithm Pull chooses the batch that requires no setup downstream. If no such batch can be formed, the algorithm selects a batch with the weighted shortest processing time. It has been shown that using a WSPT scheme leads to schedules with a minimum mean flow time (cf. Pinedo [240]). The weighted value for each product, W_j, is defined as follows:

$$W_j := (T_j + S_j) \sum_{i=1, i \neq j}^{N} q_i, \quad j = 1, 2, \ldots, N. \tag{4.35}$$

The quantity W_j represents the total delay incurred by all of the other products at the batch machine by starting j immediately. The product with the minimum value is selected in step 2 and step 4.

If it is found in step 5 that some products should wait and some should begin processing, the total delay over all products as a result of waiting, DW_j, or processing, DP_j, is computed as follows:

$$\mathrm{DW}_j := \mathrm{TN}_j \sum_{i=1}^{N} q_i + \sum_{i=1, i\neq j}^{N} \left((T_j + S_j)q_i + (T_j + S_j + \mathrm{TN}_j - \mathrm{TN}_i)^+\right),\ j \in S_1, \quad (4.36)$$

$$\mathrm{DP}_j := (T_j + S_j) \sum_{i=1, i\neq j}^{N} q_i + \sum_{i=1}^{N} (T_j + S_j - \mathrm{TN}_i)^+,\ j \in S_2. \quad (4.37)$$

Recall that we set $x^+ := \max(x, 0)$ for abbreviation. S_1 is the set of products for which it is determined to wait, and S_2 is the set of products for which it is determined to begin processing. The minimum of these values is selected, and the appropriate action is taken. The first term of the right side of Eq. (4.36) determines the additional waiting time incurred by those jobs already in the queue until the next arrival of j. The second term calculates the additional waiting time for all other products if j begins processing at the time of its next arrival. The $(T_j + S_j)q_i$ portion is the delay for those products already in queue, and the $(T_j + S_j + \mathrm{TN}_j - \mathrm{TN}_i)^+$ portion is the delay (if any) for the next arrival of the other products. The first term of the right side of Eq. (4.37) represents the additional waiting time gained by those jobs already in the queue, while the second term corresponds to the additional waiting time gained by the next arrival of each product. The resultant heuristic is called NACH-setup for abbreviation. This heuristic is very similar to the multi-product NACH procedure proposed by Fowler et al. [85]; however, NACH does not take setups into account.

To assess the performance of the NACH-setup heuristic, simulation experiments are performed using the simulation engine Factory Explorer. The first model is a three-machine system with multiple products. This system is used to compare MBSG, MBSF, NACH, and NACH-setup in a simple controllable environment. The three-machine system is comprised of a serial machine, a batch machine, and another serial machine with setups. The first machine (machine 1) is a dummy machine with infinite capacity used only to get products into the system. The batch machine (machine 2) and the next serial machine (machine 3) have limited capacity, only one machine each, and parameters typical of machines used in semiconductor manufacturing. The different products have identical process flows, which are shown in Fig. 4.2.

We continue by describing the design of experiments used. Five factors are included in the design of experiments for the simple system. The factors are listed below:

- Number of products
- Product mix
- Traffic intensity at the batch step (machine 2)
- Traffic intensity at the setup step (machine 3)
- Dispatching policy at the batch machine

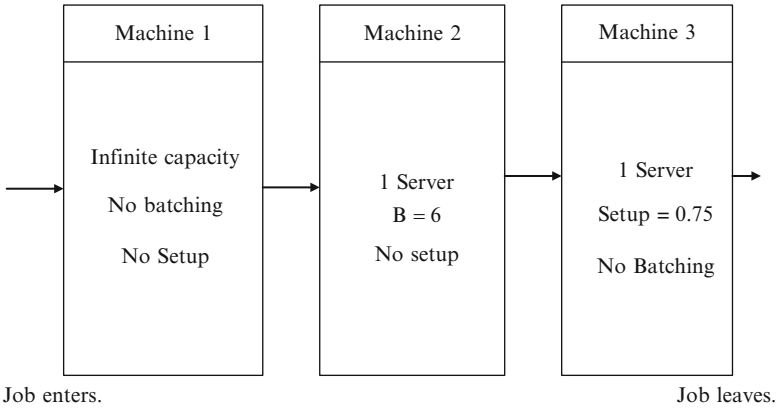

Figure 4.2: Three-machine system with two or four products

The input rate to the system is varied depending upon product mix and batch step traffic intensity, and exponential inter-arrival times are assumed. The processing time at machine 3, the machine with setup, is also varied depending upon product mix, input rate, and traffic intensity at the setup machine. The remainder of the system parameters for this experiment were as follows:

- Number of wafers per job: 48 wafers
- Processing time at machine 1: 2.5 h (deterministic)
- Processing time at machine 2: 2.5 h (deterministic)
- Capacity of machine 2: $B = 6$ jobs
- Setup time between products: $s = 0.75$ h

Altering the input rate to the system controls the traffic intensity at the batch step. Changing the processing time required for each product at the setup step controls the traffic intensity at the setup step. Notice that the experimental range of these values is small, i.e., 0.720–0.734. This is done to examine a range of high utilization at the setup step while avoiding an unstable system. The full experimental design is shown in Table 4.3.

Each data point is replicated three times, each with a run length of 8,640 h and statistics cleared after 1,720 h to take into account for initialization bias. The traffic intensity at the setup step was calculated disregarding any potential setup. Figure 4.3 shows the variation in CT over the six traffic intensity levels at the setup step for the two-product, equal-mix case.

We observe that NACH-setup outperforms all the other policies, except at setup step traffic intensity level 1. At this point, the setup step is not limiting the flow, thus dispatching based on downstream setup will likely not be beneficial. This is supported by the fact that, as the traffic intensity level at the setup step increases, the performance of NACH-setup compared to the other policies is superior.

Table 4.3: Design of experiments

Factor	Level	Count
Number of products	2, 4	2
Product mix	Equal (50%,50%), (25%,25%,25%,25%) Dominant (70%,30%), (40%,40%,10%,10%)	2
Traffic intensity at batch step	0.5, 0.8	2
Traffic intensity at setup step	0.720, 0.723, 0.726 0.729, 0.731, 0.734	6
Dispatching policy	MBSG, MBSF, NACH, NACH-setup	4
	Total factor combinations	192
	Number of replications	3
	Number of simulation runs	576

In Table 4.4, the CT values for the two-product, dominant mix case are shown. With one product dominant over the other, the behavior of the manufacturing system is similar to a system with one product. As expected, because of the dominant-product mix, NACH performs best until traffic intensity level 4. Once this situation is reached, the gain in CT because of the reduced setup downstream becomes critical as the setup step begins to limit the flow of jobs through the system. In the four-product, equal-mix case,

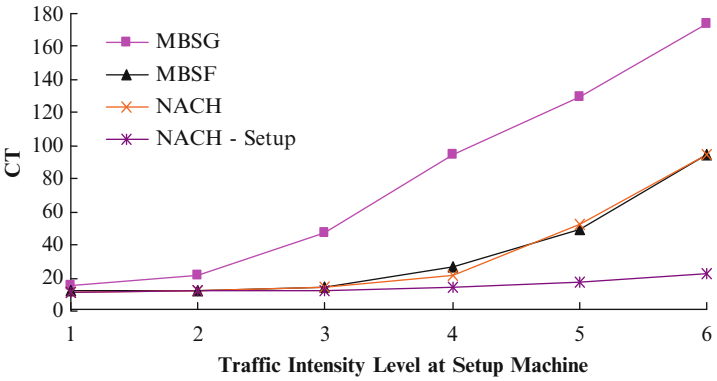

Figure 4.3: CT at the setup step, two products, equal mix

NACH-setup outperforms all the other policies, and, as expected because of the additional setup due to the larger number of products, the difference between them is more substantial. MSBF outperforms NACH because the additional setup time favors larger batches. As in the two-product situation, the differences in CT between the policies are reduced in the four-product, dominant-product mix scenario, and it does not become apparent until traffic

intensity level 4. The detailed results for the four-product case can be found in [287]. Additional experiments using the MIMAC 1 model are also described in [287]. In case of a high input rate, no statistically significant difference between the four policies can be found. There is no significant difference between MBSG, NACH, and NACH-setup in the low- and medium-input rate cases. However, all three policies are superior to MBSF. In addition, NACH-setup performs well for all three input rates, while the relative performance of the others changes with respect to the different input levels.

Table 4.4: Simulation results for two products, dominant mix

Traffic intensity level	MBSG	MBSF	NACH	NACH-setup
1 - 0.720	14.425	12.025	11.050	13.125
2 - 0.723	18.100	12.400	11.500	13.350
3 - 0.726	28.875	12.975	12.375	13.550
4 - 0.729	63.900	34.675	13.700	14.475
5 - 0.731	87.700	53.025	16.375	15.750
6 - 0.734	133.300	75.250	28.225	17.700
Average	57.717	33.558	15.538	14.658

4.6.3 Additional Look-Ahead Research

Weng and Leachman [320] address the same problem as Glassey and Weng [101] and Fowler et al. [84]. However, their minimum cost rate (MCR) heuristic has some noticeable differences from DBH and NACH. MCR seeks to minimize the holding cost per unit time, which is like minimizing the weighted (by cost) number of jobs in queue. Robinson et al. [260] present a heuristic that is essentially a combination of NACH and MCR. The cost rate function used in MCR is incorporated into the rolling horizon scheme used in NACH. We refer to Robinson et al. [261] for a review and a comparison of various real-time control strategies for batch machines in wafer fabs until 2000.

Instead of calculating a threshold number of jobs in queue, Cigolini et al. [52] determine dynamically the length of the time window in a more recent paper. The resulting look-ahead heuristic is called wait no longer than time (WNLTT).

Ham and Fowler [114] propose an extension of NACH. The heuristic, called NACH+, is based on the idea that the incoming inventory into the batch operations is controlled such that unnecessary waiting time does not happen.

4.6.4 BATC-Type Rules

We continue with an extension of the algorithm BA described in Sect. 4.5. Again, we assume that a batch-processing machine is available for processing, and we have to determine the next batch to be processed. In contrast to the algorithm BA, the present extension takes ready times of the jobs into account. This kind of ready time information is typically provided by the MES.

For this purpose, we consider a time window $(t, t+\Delta t)$, where t is the current time. We define the set

$$J(i,t,\Delta t) := \left\{ij | r_{ij} \leq t+\Delta t\right\}, \tag{4.38}$$

where we represent job j of family i by ij. We sort the elements of $J(i,t,\Delta t)$ with respect to the index

$$I_{ij}(t) := \frac{w_{ij}}{p_i} \exp\left\{-\frac{(d_{ij}-p_i-t+(r_{ij}-t)^+)^+}{\kappa \bar{p}}\right\} \tag{4.39}$$

in nonincreasing order, where p_i is the processing time of the jobs of family i and r_{ij} is the ready time of job j of family i. Note that index (4.39) is similar to index (4.24) when no setup times occur. In the next step, we select the first thresh jobs from this list and form all the possible batches. We assess each of these potential batches by using the batch index

$$I_{bi}(t) := \frac{w_{bi}}{p_i} \exp\left\{-\frac{(d_{bi}-p_i-t+(r_{bi}-t)^+)^+}{\kappa \bar{p}}\right\} \frac{n_{bi}}{B}, \tag{4.40}$$

where w_{bi} is the average weight of the n_{bi} jobs that form the batch bi of family i, r_{bi} is the maximum ready time among the jobs that form the batch, and finally d_{bi} is the minimum due date. We summarize the algorithm as follows.
Algorithm Dynamic Batching Dispatching Heuristic (DBDH)

1. Determine the sets $J(i,t,\Delta t)$ for family $i = 1,\ldots,f$. The quantity t is the time where a batching machine is available for processing.
2. Sort all the jobs within each family $i = 1,\ldots,f$ in nonincreasing order with respect to $I_{ij}(t)$ given by expression (4.39).
3. The length of the list that corresponds to family i is denoted by $l(i)$. Consider the first $\min\{\text{thresh}, l(i)\}$ jobs to form potential batches. Assess each of these batches using index $I_{bi}(t)$.
4. Select the batch with the largest $I_{bi}(t)$ value among the families $i = 1,\ldots,f$ to be processed next.

Note that Δt and thresh are parameters of DBDH that have to be selected carefully. Large values of thresh might lead to a huge computational burden, whereas large values for Δt might decrease the quality of the future job arrival information represented by r_{ij}.

We continue with the presentation of the results of some computational experiments. The MIMAC 1 model [83] is used in a slightly modified version. This simulation model consists of two different process flows with more than 200 process steps and over 80 different machine groups.

There are 16 batch machine groups among the machine groups of the MIMAC 1 model. Machine group OXIDE_1 is the bottleneck of the wafer fab. Table 4.5 provides information on this particular batch machine group. In Table 4.5, we denote by $B_{\min}$ the minimum batch size in jobs and by $B_{\max}$ the maximum batch size in jobs. The processing time of the different job families is between 135 min and 1,410 min. The utilization is determined by simulation experiments with the FIFO dispatching rule.

Table 4.5: Bottleneck batching machine group information

Machine group	Number of machines	$B_{\min}$	$B_{\max}$	Utilization (%)
OXIDE_1	3	2	6	84.19

We use a slack-based dispatching rule for the non-batching machines (cf. Sect. 4.2.1). The rule selects the job with the smallest slack for the process step. For the calculation of the slack of the jobs waiting in front of a certain machine group, we simply multiply the processing time by a flow factor. For that purpose, we calculate the difference between the due date of the job and the current time as used in the LS index (4.13). Based on this information, we assign a flow factor to each job. This scheme allows us to determine local due dates for each single process step, i.e., future job arrival information is available at the batch machines. We repeat the calculation of the flow factors every 15 min.

In our experiments, we consider a moderate workload in the system. Machine TTF and TTR (see Sect. 3.2.8) are exponentially distributed. The model is initialized using a WIP distribution of the wafer fab. The length of a single simulation run is 100 days in our experiments. We take five independent replications of each simulation run in order to obtain statistically meaningful results.

We continue by presenting the design of experiments used. The main performance measures are TWT, CT, and TP. Therefore, we set due dates according to the following expression:

$$d_j := r_j + \mathrm{FF} \sum_{k=1}^{n_j} p_{jk}, \tag{4.41}$$

where FF is the flow factor. Furthermore, we define weights of the jobs according to the following two discrete distributions:

$$D_1 := \begin{cases} w_j = 1, \ p_1 = 0.5 \\ w_j = 5, \ p_2 = 0.35 \\ w_j = 10, \ p_3 = 0.15 \end{cases} \tag{4.42}$$

and

$$D_2 := \begin{cases} w_j = 1, \ p_1 = 0.5 \\ w_j = 2, \ p_2 = 0.45. \\ w_j = 10, \ p_3 = 0.05 \end{cases} \tag{4.43}$$

D_1 mimics the situation where a large number of jobs have a large weight, whereas a large number of jobs have a medium weight in D_2 and only a very small portion of the jobs have a large weight in this situation. We summarize the design of experiments in Table 4.6. We denote by $\bar{p}$ the average processing time of the jobs on the batch machines.

Table 4.6: Design of experiments

Factor	Level		Count
FF	1	2 for all the jobs,	2
	2	2 for 50 % of the jobs 1.5 for 50 % of the jobs	
w_j	1	$\sim D_1$	2
	2	$\sim D_2$	
Δt	1	$0.25\bar{p}$	2
	2	$0.5\bar{p}$	
Overall number of experiments			8

The look-ahead parameter κ in DBDH is selected from the grid $\{0.1, 0.2, \ldots, 6.5\}$. The κ value that leads to the smallest TWT value is finally used whenever DBDH makes a decision. Within the experiments, $\mathsf{thresh} = 15$ is chosen.

The corresponding results of the DBDH-based batching strategy are shown in Table 4.7. We use a batching scheme based on the FIFO dispatching rule as a reference. Note that when the FIFO dispatching rule is used for batching, Δt does not have any impact on the decision-making. Therefore, we have to conduct only four simulation experiments in this situation. Consequently, we have to perform a total of 12 different simulation experiments.

We use the notation (level of factor 1—level of factor 2—level of factor 3) in order to indicate the considered factor combinations for DBDH. All the results in Table 4.7 are the ratios of the performance measure values of DBDH and FIFO for the same values of the levels of the first, the second, and finally the third factor. Low values are good for TWT and CT, while we are interested in high values for TP.

Table 4.7: Computational results for DBDH

Factor combination	TWT	CT	TP
1-1-1	0.1031	0.9685	1.0099
1-1-2	0.1191	0.9725	1.0089
1-2-1	0.0918	0.9677	1.0135
1-2-2	0.1475	0.9732	1.0056
2-1-1	0.2913	0.9631	1.0081
2-1-2	0.3035	0.9703	1.0082
2-2-1	0.4230	0.9588	1.0089
2-2-2	0.4363	0.9643	1.0085

From the results shown in Table 4.7, we can see that the algorithm DBDH outperforms the FIFO rule at all factor settings for all performance measures. The TWT values are sensitive to the choice of Δt. In our experimental design, smaller values for Δt lead to slightly better results. Choosing a larger Δt value causes fuller utilized batches and a larger queue size. The machines have to wait longer for jobs that arrive during the given time window. Therefore, this leads to fuller batches. Hence, a careful selection of the Δt values is important for the performance of DBDH.

More computational results can be found in Mönch and Habenicht [194]. It is also shown, by comparison with the algorithm BA, that taking future job arrival information into account can lead to TWT reductions.

4.7 More Sophisticated Approaches

In this section, we discuss rule-based systems, the selection of parameters of dispatching rules using iterative simulation, the construction of appropriate blended dispatching rules, and finally the automated discovery of dispatching rules.

4.7.1 Rule-Based Systems

A rule-based system determines a priority value for each job or batch based on hierarchically structured rule-based criteria systems. In a certain sense, a rule-based dispatch system is a combination of composite, truncation, conditioning, and finally multilevel dispatching rules. Powerful rule-based systems are in use in wafer fabs (see Appleton-Day and Shao [9]).

The main ingredient of a rule-based system is a composite dispatching rule, given by the following priority index for each job j:

$$I_j := \sum_{k=1}^{C} w_k c_k, \tag{4.44}$$

where C is the number of first-order criteria, c_k is the value of the kth first-order criterion, and $w_k \geq 0$ with $\sum_{k=1}^{C} w_k = 1$ are weights to balance the importance of the different first-order criteria. Each c_k can be further refined by appropriate subcriteria, i.e., we consider second-order criteria $c_{kl}, l = 1, \ldots, l_k$, where we denote by l_k the number of second-order criteria for the first-order criterion c_k. Higher-order criteria can be taken into account by following this approach in a recursive manner. The resulting criteria hierarchy is a tree. A set of IF-THEN rules is used to evaluate each leaf of this tree based on certain BS- and BP-related data with a certain attribute value. Typical attribute values are LOW, MEDIUM, LARGE, and HUGE. An attribute value is assigned to each $(l+1)$-order criterion based on the attribute value combination of its l-order subcriteria. By proceeding in a recursive manner, a positive real number can be assigned to each first-order criterion that is necessary to compute the left-hand side I_j in expression (4.44).

Following Thiel et al. [296, 297], we introduce the following example for a rule-based system. The rule-based system consists of the following first-order criteria:

1. On-time delivery performance-related criterion, called on-time urgency criterion
2. Setup-related criterion
3. Load-related criterion

Note that the second criterion deals with setup avoidance issues, while the third one is related to batch formation issues. It is obvious that the second and third criterion are TP-related and therefore in potential conflict with the first criterion.

We show the second- and third-order subcriteria for the on-time urgency in Fig. 4.4. The first second-order criterion measures the slack related to the current process step of the job while the second second-order criterion is concerned with the importance of the due date and is comprised of three third-order criteria. The first third-order criterion considers the progress of processing a job measured in the number of completed process steps. This subcriterion is motivated by the fact that a potential due date violation is less important when the number of already completed process steps is small. The second third-order criterion takes into account whether the job is a regular or a hot job, whereas the third second-order criterion measures the importance of meeting the due date of the job with respect to the type of the customer associated with this job. Specifically, the progress of a job is measured by

$$\text{prog}(j) := l_j / n_j, \tag{4.45}$$

where l_j is the current process step of j, and n_j denotes the number of all process steps of job j. We show the IF-THEN rules with respect to the progress of the job-related subcriterion:

IF $\text{prog}(j) < 1/3$ THEN c_{121} = “LOW”

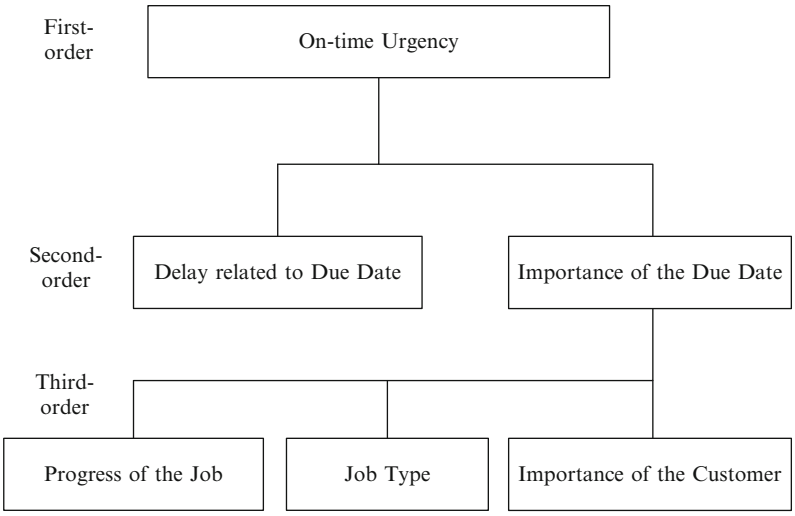

Figure 4.4: Criteria hierarchy related to On-time urgency

IF $1/3 \leq \text{prog}(j) \leq 2/3$ THEN c_{121} = "MEDIUM"
IF $2/3 < \text{prog}(j)$ THEN c_{121} = "LARGE"

It is obvious that the values 1/3 and 2/3 are prescribed values that can be modified to model user preferences. Because these values are arbitrary, extensive simulation-based assessment of rule-based dispatching systems is necessary.

4.7.2 Determining Parameters of Dispatching Rules Based on Iterative Simulation

We continue by studying global dispatching rules that take waiting times into account. The waiting times for process steps that have to be performed in the future are unknown. The waiting times depend, for example, on the product mix, on the load of the wafer fab, and on the control strategy used.

Global variants of the ATC dispatching rule are discussed by Vepsalainen and Morton [312]. A solution is proposed that is based on iterative simulation (cf. the discussion in Sect. 3.2.8). The method is called lead time iteration. Based on a crude initial waiting time estimate, successive adjustments of the waiting time are performed by using the measured waiting time from the current simulation run. This method is also used by Ovacik and Uzsoy [223] in order to determine appropriate internal due dates for an ODD-type dispatching rule in the test area of a back-end facility. A lead time iteration scheme is used to estimate waiting time used in the FSMCT dispatching rule in [169] (cf. the discussion in Sect. 4.2.1). We consider a global ATCS rule as

proposed by Vepsalainen and Morton [312]. The index has to be calculated as follows:

$$I_{ji}(t,lk) := \frac{w_j}{p_{ji}} \exp\left\{ -\frac{\left(d_j - p_{ji} - t - \sum_{g=i+1}^{n_j}(wt_{jg} + p_{jg})\right)^+}{\kappa_1 \bar{p}} - \frac{s_{kl,ji}}{\kappa_2 \bar{s}} \right\}, \quad (4.46)$$

where we assume that i is the current process step of job j. The average of the sum of the processing times of the remaining process steps for each job is denoted by $\bar{p}$. The waiting time associated with process step k of job j is denoted by wt_{jk}. The quantity $s_{kl,ji}$ is the setup time that is necessary when process step l of job k is processed before ji. Again, κ_1 and κ_2 are scaling parameters. The resulting dispatching rule is called global ATCS (GATCS). Appropriate values for wt_{jk} are unknown in the beginning because they are a result of the dispatching strategy used. Therefore, we use iterative simulation to determine them. The resulting procedure can be formulated as follows.
Algorithm Lead Time Iteration Procedure (LTIP)

1. Set $l = 1$. Start by an initial waiting time setting using

$$wt_{jk}^{(l)} := (\mathrm{FF} - 1)p_{jk},\ k = i+1,\ldots,n_j, \quad (4.47)$$

   where we denote by $\mathrm{FF} \geq 1$ the flow factor. Initial values for FF can be obtained from a simulation run using the FIFO dispatching rule.
2. Dispatch the wafer fab using the GATCS dispatching rule and the waiting time estimates for the current iteration.
3. Calculate the actual waiting time $q_{jk}^{(l)}$ of each process step jk from simulation run l. In this situation, the waiting time is defined as the time between the completion of process step $j,k-1$ and the start time of process step jk, i.e., the transportation time is included.
4. Update the waiting time estimates as follows:

$$wt_{jk}^{(l+1)} := (1-\alpha)wt_{jk}^{(l)} + \alpha q_{jk}^{(l)},\ k = i+1,\ldots,n_j, \quad (4.48)$$

   where $0 \leq \alpha \leq 1$ denotes a fixed smoothing factor.
5. Terminate the procedure if the stopping condition $\max_{jk} |wt_{jk}^{(l+1)} - w_{jk}^{(l)}| < \varepsilon$ is valid; otherwise, update $l := l+1$ and go to step 2. The quantity ε is a small prescribed value.

Usually, four to eight iterations are enough to fulfill the LTIP stopping condition for reasonable values of ε. The update scheme for the waiting times in step 4 is an exponential smoothing-type approach. It takes the current measured waiting time from the simulation run and the estimated waiting time from the previous iteration into account. Typical α values are 0.7 and 0.9 (see Mönch and Zimmermann [197]). Note that it is also possible to use a more sophisticated update scheme based on double exponential smoothing.

The architecture described in Sect. 3.3.2 is used to implement LTIP. The object-oriented database is used to store the waiting time for each single process step in each iteration. It makes the waiting times persistent because they are required in future iterations.

LTIP allows for large TWT reductions compared to a FIFO dispatching strategy. At the same time, CT decreases and TP increases. Detailed computational results for the MIMAC 1 model can be found in [197]. LTIP is a simple but powerful technique to improve the performance of dispatching rules.

4.7.3 Construction of Blended Dispatching Rules

We consider a blended dispatching rule for the case of l different performance measures. Each performance measure a of interest is represented by a priority index $I^a_{jis}(t) \in [0,1]$ for processing job j of product i at stage s at time t. We obtain for the weighted priority for job j of product i on stage s at time t:

$$P^t_{jis} := \sum_{a=1}^{l} w_a I^a_{jis}(t), \tag{4.49}$$

where $w_a \geq 0$ is the weight of performance measure a and $\sum_{a=1}^{l} w_a = 1$ is valid. Furthermore, it can be achieved by an appropriate transformation of the priority indices of the jobs that the sum of all priority index values for all the jobs of all products queued at machine group m sum up to one as shown in the following expression:

$$\sum_{i=1}^{p} \sum_{s \in J_i(m)} \sum_{j=1}^{n_{si}} I^a_{jis}(t) = 1, \tag{4.50}$$

where p is the number of different products, n_{si} is the number of jobs of product i at stage s, and finally $J_i(m)$ is the set of all stages in the process flow of product i that require machine group m.

We have to determine appropriate values for the weights. Therefore, we express the values of each performance measure as a mapping of the weights. To do this, we look for a mapping:

$$P_a : (w_1, \ldots, w_l) \rightarrow \mathbb{R} \tag{4.51}$$

for each performance measure a. $P_a(w_1, \ldots, w_l)$ is called the response for the weight setting $w_1, \ldots, w_l$. The same combined dispatching weights and rules are used at all machine groups. However, due to the different processing natures of non-batching and batching machines, two varying yet similar approaches are used. For a non-batching machine, the combined criterion is calculated for each job queued in front of machine group m. In the case of a batching machine, the only difference is that jobs are grouped together into

batches based on similar batching requirements, i.e., the same incompatible job family. The combined criterion for the different jobs that form the batch is aggregated into a single index value by adding the weighted criterion for each job in the batch similar to algorithm BA in Sect. 4.5.

Next, we describe how the response for each objective is determined by designed experiments using discrete-event simulation. Model parameters are estimated most effectively when proper experimental designs are used to collect the data (see Montgomery [208]). There are many experimental designs to choose from including factorial, central composite, and D-optimal. However, there are several assumptions that need to be made about the design and the data for the experimental methods to be effective. One critical assumption is the independence of the experimental variables. Least square methods cause erroneous results in the model parameter estimates if this assumption is violated.

In the situation discussed in this section, the experimental variables, i.e., the different w_a, are not independent. The weights identify the proportional contribution of each dispatching rule index in the overall blended priority index. The response variables are a function of the proportions of the different weights. The actual value of a weight is not important, but, rather, it is the relative size when compared with another weight that is important. For example, in case of four criteria, the weights 0.05, 0.05, 0.15, and 0.25 are possible, or they can be 200, 200, 600, and 1,000. The same results are obtained for each of the two weight sets because the ratios of the weights when compared to each other are the same for each set. Both sets of weights can be normalized to 0.1, 0.1, 0.3, and 0.5, i.e., their weight sum is one.

Due to the lack of independence of the weights, standard experimental design strategies have to be abandoned and a mixture design chosen that will accommodate this lack of independence. Mixture experiments address the issue where the components of the experiment have to add to 100 %. There are several mixture designs to choose from depending on the degree of the polynomial that the experimenter is anticipating to best fit the process. For mixture experiments, the experimental design region is a simplex that is a regularly sided figure with q vertices in $q-1$ dimensions (see Montgomery [208]). Let us consider the simplex centroid design with q factors in more detail. This design consists of 2^q-1 design points. This is the number of vectors with q components where k components have the value $1/k$ for $1 \leq k \leq q$ and the remaining components are 0. An example with three different criteria is given in Table 4.8. Note that in Table 4.8 in addition to the 2^3-1 regular design points, three augmented points are added to give more degrees of freedom for error lack of fit and model significance analysis.

Based on the mixture design, a response surface is constructed for each single criterion as described in Sect. 3.2.9. Note that each design point corresponds to a blended dispatching rule with a certain weight setting. Simulation experiments with simulation models of wafer fabs have to be carried out to find the response values. Different analyses of variance have to be

Table 4.8: Simplex centroid design for a mixture experiment

Run number	w_1	w_2	w_3
1	1	0	0
2	0	1	0
3	0	0	1
4	1/2	1/2	0
5	0	1/2	1/2
6	1/2	0	1/2
7	1/3	1/3	1/3
8	2/3	1/6	1/6
9	1/6	2/3	1/6
10	1/6	1/6	2/3

performed for the different responses to determine statistically significant factors. An individual optimum can be determined for each single P_a. However, a global optimum is desired. Therefore, a multiple response optimization using the desirability function approach (cf. Sect. 3.3.1) is performed. The individual meta-models are transferred to a desirability function with values between zero and one, where one is the most desirable. The corresponding desirability function for criterion a is denoted by d_a. Finally, the desirability functions are transformed into a combined objective function using the geometric mean of the individual desirabilities as described in Sect. 3.3.1. Optimization of the combined objective function can be accomplished using different pattern search algorithms such as the algorithm of Hooke and Jeeves [118].

In Dabbas et al. [56, 57, 58], the above approach is used to determine a combined criterion for four objectives that are similar to AT, Var(L), ACT, and finally Var(CT), defined in Sect. 3.3.1. The dispatching rules used are the CR dispatching rule, the throughput (TP) dispatching rule, the line balance (LB) rule, and finally the FC rule.

The CR dispatching rule works, as discussed in Sect. 4.3.1. The TP rule works as follows. The SPT rule is applied to non-batching machines, while for batching machines, the loads with the highest number of wafers per hour get the highest priority because in this situation, the batch fullness can be increased. A more global dispatching rule is the LB rule. Using LB, products at stages with higher deviations from their WIP goal get higher priority. WIP goals determine the average WIP required at each stage of a process flow such that output requirements are met. WIP goals tie the required TP rate of product i to the CT at stage j using Little's law (cf. Sect. 3.2.7):

$$\mathbf{WIP}(L_{ij}) = \lambda_{ij}\mathbf{CT}_{ij}, \tag{4.52}$$

where $\mathbf{WIP}(L_{ij})$ is the WIP goal for product i at stage j, λ_{ij} is the corresponding TP rate, and $\mathbf{CT}_{ij}$ is the CT value, i.e., the sum of waiting time and processing time. The quantity λ_{ij} can be calculated by dividing the required

daily output for i by the number of process steps to determine the daily output of i at stage j because then the line is balanced. The goal CT values $\mathbf{CT}_{ij}$ can be derived from simulation studies.

Finally, the FC dispatching rule prioritizes jobs with the objective to balance the workload on the different machines. This rule is described in more detail in Sect. 4.2.1.

The reason for selecting the above dispatching rules is that they each target a different performance measure of interest. CR identifies the quality of dispatching from a point of view of on-time delivery, i.e., AT. The TP rule improves the quality of dispatching from a CT point of view. Its objective is to maximize TP at machines regardless of due date priorities. The LB rule has the objective of minimizing the WIP variation vs. a pre-set goal in an effort to linearize output, while FC takes the workload balancing perspective.

The blended dispatching approach was compared to single dispatching criteria like CR, SPT, or FLNQ in [56, 57] using full wafer fab simulation models. The simulation results indicate that CT, Var(L), and on-time delivery-related performance measures are improved to a large extent. The blended approach shows a behavior similar to the CR dispatching rule with respect to Var(CT), but it outperforms the two remaining dispatching rules. It is also demonstrated by the simulation experiments that the resultant WIP profile is more stable and has a lower average value when using the blended dispatching approach. Lower WIP translates to lower ACT values, whereas less variability in the profile translates to lower Var(CT) values and consequently better overall performance measure values.

4.7.4 Automated Discovery of Dispatching Rules

So far, we assume that we manually select dispatching rules out of a given set of rules. Then simulation experiments have to be carried out to assess the performance of the rules, and finally, an appropriate rule is selected. This approach is in a certain sense rigid. That is why we allow for the selection of appropriate weights to construct blended dispatching rules in Sect. 4.7.3. This approach is less rigid; however, only the weights can be changed and not the structure of the rules.

In this section, we describe work that is related to an automated discovery of dispatching rules. A dispatching rule is represented by an index that assesses all jobs awaiting processing at a given machine at time t when the resource is available (cf. Sect. 4.1 for details). The index takes several attribute values into account. The priority index might be considered as a logical expression. Selecting a dispatching rule means specifying the priority index.

We discuss an approach to construct new dispatching rules based on a given set of primitives. These primitives belong to two subsets (see Geiger et al. [96]):

- A set of relational and conditional functions denoted by F.
- A set of terminals T that is problem-specific and consists of a set of variables and numerical constants. Terminals cannot be broken down into smaller units.

The set of relational and conditional functions is given by unary and binary operators and functions. Examples for unary operators are the four basic arithmetic operators $+, -, /, *$, whereas EXP, ABS, MAX, MIN are examples of functions. Furthermore, the conditional function IF3 is often important. It is the ternary version of the IF-THEN-ELSE expression used in programming languages, i.e., if $a \geq 0$ then b else c, where a, b, and c are expressions. The precedence relationship of the functions is given. It is preserved and cannot be redefined. More examples of functions F to construct dispatching rules can be found in [96, 295].

We continue with examples for terminals. The following terminals are used, and most of them refer to a given job j:

- PT: The processing time p_j is modeled using this terminal.
- DD: This terminal is used to refer to d_j.
- W: The weight w_j is expressed by this terminal.
- CurT: This is the current time t where the dispatching decision is made.
- Con: A constant value, i.e., a number $z \in \mathbb{R}$, is denoted by this terminal.
- AvgPT: This terminal denotes the average processing time of all jobs waiting for processing. It represents the quantity $\bar{p}$.

It becomes clear that the terminals model the attribute values within the priority indices. More examples for terminals used to construct dispatching rules can be found in [95, 96, 237].

The logical expression of the priority index is first transformed into an intermediate representation using prefix notation. In prefix notation, the function is written before the arguments it operates on (see Preiss [248]). Compared to the logical expression, the prefix notation has the advantage that an expression tree can be derived from it automatically by parsing the expression in prefix notation from left to right. Terminals are the leaves of an expression tree, i.e., variables or constants in the logical expression for a certain priority index. Each single function in a logical expression is represented by a node of the expression tree. When a symbol $s \in F$ is detected during the parsing process, then a node is created in the expression tree. When a terminal $\tau \in T$ is found, then a leaf is added to the tree. When a subtree cannot have more leaves, then a new subtree is started when the next symbol τ is detected.

On the other hand, each expression tree can be transformed automatically into an expression in prefix notation by traversing the tree recursively using the following procedure.

Algorithm Traverse Expression Tree

1. When a visited node is a terminal, then it is written at the end of the partial expression in prefix notation.

2. When a nonterminal symbol is found, then a left parenthesis is written. The symbol is written into the expression after the left parenthesis.
3. The left subtree is traversed using step 1 and step 2.
4. The right subtree is traversed, if any, using step 1 and step 2, and finally, a right parenthesis is written.

Let us consider the ATC dispatching rule given by index (4.21) to illustrate these rather abstract concepts. In this example, the index can be represented as:

$$(*(/W\ \mathrm{PT})(\mathrm{EXP}(-(/(\mathrm{MAX}(-(\mathrm{DD}\ (-\mathrm{PT}\ \mathrm{CurT}))0)(*\mathrm{Const}\ \mathrm{AvgPT}))) \quad (4.53)$$

using the notation for functions and terminals introduced. The resulting tree is shown in Fig. 4.5.

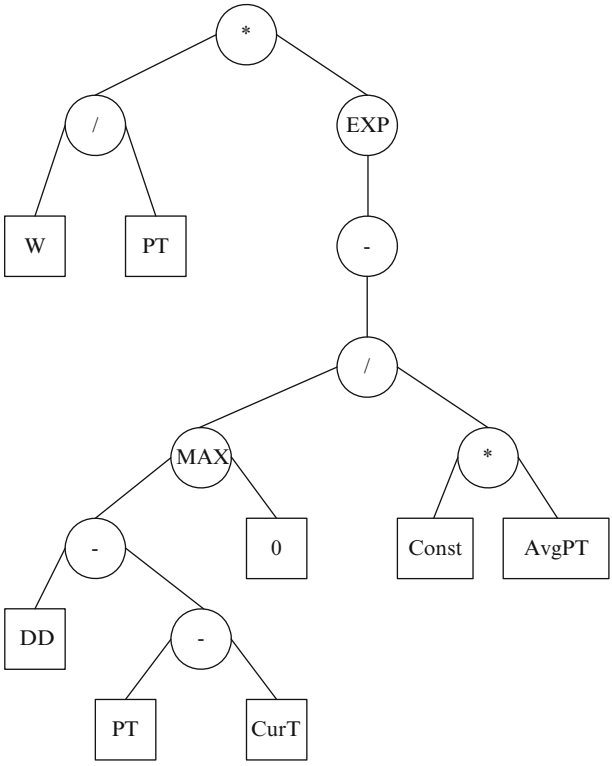

Figure 4.5: Tree representation of the ATC dispatching rule

The basic learning system model described in Sect. 3.2.10 is used to discover new dispatching rules. The corresponding learning element is realized using genetic programming (GP). The performance element is the constructed dispatching rule. Feedback is available from discrete-event simulation. GP is

a special kind of a GA. It uses tree structures of variable lengths to represent solution candidates and can be used to automatically discover logical expressions or even computer programs.

GP starts from a candidate set of dispatching rules called a population (cf. Sect. 3.2.6). These rules are either generated randomly or based on heuristics. The quality of each dispatching rule is assessed using discrete-event simulation and a set of performance measures of interest. The BS and the BP are represented by an appropriate simulation model. Each dispatching rule from the candidate set is equipped after the performance assessment with a set of values for the performance measures.

The reasoning mechanism consists of a selection component and a component that generates new dispatching rules. The selection component uses the performance measure values for each rule of the candidate set to choose a set of high-performing dispatching rules that form the basis for generating a new candidate set. The entire cycle is repeated until a certain stopping criterion is fulfilled.

New dispatching rules will be generated using crossover and mutation operators as in common GAs. The crossover operator works as follows. Two trees are randomly selected from the current set of candidate rules. A subtree is identified in each parent rule randomly. These subtrees are swapped in the next step between the two parent rules. It is clear that because of the entire subtrees used, only feasible offspring are produced. The mutation operator starts by randomly selecting a subtree from a parent rule, then this subtree is replaced by a randomly generated subtree using the sets F and T.

Experiments with the described discovery approach using simulation models of large-scale wafer fabs are described by Pickardt et al. [237]. The discovered rules clearly outperform ATCS-type dispatching rules. The basic discovery approach is extended to automatically learn batching rules for single machine scheduling by Geiger and Uzsoy [95]. Overall, it seems that discovering dispatching rules automatically is a promising direction of future research.