Evolutionary Methods for the Design of Dispatching Rules for Complex and Dynamic Scheduling Problems

by

Christoph W. Pickardt

Thesis

Submitted to the University of Warwick

for the degree of

Doctor of Philosophy

Warwick Business School

July 2013

THE UNIVERSITY OF WARWICK

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank both my supervisors, Professor Juergen Branke and Professor Bo Chen, for their support, especially Juergen Branke for having been so helpful and accessible throughout the period of my PhD, always showing great interest in my research, which resulted in numerous constructive and fruitful discussions.

I would also like to thank Torsten Hildebrandt and Jens Heger at the BIBA institute at the University of Bremen, Germany for their contribution to our joint project on Learning and Self-Organisation in Production Planning and Control, and the regular exchange of ideas, which I have found inspiring and very motivating. In particular, I thank Torsten for his work on the Jasima simulation environment and the technical support he provided with regard to its extension and its integration with the algorithms developed as part of this thesis.

I am further very grateful to the German Research Council (DFG) for funding the first two years of my PhD under Grant BR 1592/7-1, and to the Warwick Business School for funding the third year, which allowed me to conduct this research.

Declaration

This thesis is submitted to the University of Warwick in support of my application for the degree of Doctor of Philosophy. It has been composed by myself and has not been submitted in any previous application for any degree apart from the algorithmic components described in Section 5.2, which were previously developed as part of a thesis submitted for a Master of Science degree to the Karlsruhe Institute of Technology, Germany. The work presented (including data generated and data analysis) was carried out by the author.

Parts of this thesis have been published by the author:

- Branke, J. and C. W. Pickardt (2011). Evolutionary search for difficult problem instances to support the design of job shop dispatching rules. *European Journal of Operational Research* 212(1), 22–32.
- Pickardt, C. W. and J. Branke (2012). Setup-oriented dispatching rules — a survey. *International Journal of Production Research* 50(20), 5823–5842.
- Pickardt, C., J. Branke, T. Hildebrandt, J. Heger, and B. Scholz-Reiter (2010). Generating dispatching rules for semiconductor manufacturing to minimize weighted tardiness. In B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan (Eds.), *Proceedings of the 2010 Winter Simulation Conference*, pp. 2504–2515. Piscataway, NJ: IEEE Press.
- Pickardt, C. W., T. Hildebrandt, J. Branke, J. Heger, and B. Scholz-Reiter (2013). Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems. *International Journal of Production Economics* 145(1), 67–77.

Abstract

Three methods, based on Evolutionary Algorithms (EAs), to support and automate the design of dispatching rules for complex and dynamic scheduling problems are proposed in this thesis. The first method employs an EA to search for problem instances on which a given dispatching rule performs badly. These instances can then be analysed to reveal weaknesses of the tested rule, thereby providing guidelines for the design of a better rule. The other two methods are hyper-heuristics, which employ an EA directly to generate effective dispatching rules. In particular, one hyper-heuristic is based on a specific type of EA, called Genetic Programming (GP), and generates a single rule from basic job and machine attributes, while the other generates a set of work centre-specific rules by selecting a (potentially) different rule for each work centre from a number of existing rules. Each of the three methods is applied to some complex and dynamic scheduling problem(s), and the resulting dispatching rules are tested against benchmark rules from the literature. In each case, the benchmark rules are shown to be outperformed by a rule (set) that results from the application of the respective method, which demonstrates the effectiveness of the proposed methods.

Chapter 1

Introduction

Scheduling is concerned with the allocation of limited resources to tasks over time, with the basic aim to ensure an effective and efficient use of the available resources. A classic problem area is the scheduling of manufacturing systems, where machines (the resources) have to be allocated to jobs (the tasks) in a way that the cost of production is minimised. Some of the costs that are typically affected by a production schedule are the holding costs of in-process inventory, contractual penalties for late deliveries, setup costs, and the costs of scrap and rework, which illustrates the importance of production scheduling to manufacturers in their endeavour to become and remain competitive.

In general, scheduling problems can be addressed with a variety of solution methods that have different strengths and weaknesses (see Pinedo, 2008). However, real-world production environments tend to be highly complex and dynamic, making the use of some of these methods impractical. A widely popular approach to solving production scheduling problems in practice are dispatching rules (Pfund et al., 2006), which are simple heuristics that progressively construct solutions by scheduling one operation at a time. More precisely, whenever a machine is available and there are jobs waiting to be processed on that machine, dispatching rules compute a priority index for each job as a function of some job attributes, e.g. its due date or the processing time of its imminent operation, and machine attributes, e.g. its current setup, and schedule only the imminent operation of the job with the highest priority. Dispatching rules thus solve a global problem by making local scheduling decisions based on local information, which allows them to be executed very quickly, irrespective of the complexity of the overall problem. Their ability to take decisions in real-time is particularly advantageous in dealing with dynamic problems as it allows for prompt reactions to sudden changes due to unexpected events, which can be taken into account as they emerge. Apart from their low computational and information requirements, dispatching rules are further favoured for their simple and intuitive nature, their ease of implementation and their flexibility to incorporate domain knowledge and expertise (Aytug et al., 2005).

The main disadvantage of dispatching rules is that, due to their locally restricted horizon, they cannot assess how a local scheduling decision affects the quality of the global schedule. In consequence, no rule can outperform all other rules across different shop configurations, objective functions or operating conditions (Blackstone et al., 1982; Kiran and Smith, 1984; Haupt, 1989; Holthaus and Rajendran, 2000), but they have to be carefully designed for the problem at hand so that, despite their local horizon, their decisions result in a good global performance. However, the manual design of effective dispatching rules for complex and dynamic problems is usually a tedious trial-and-error process, with candidate rules tested in a simulation model of the considered shop, modified and retested until they meet the demands for actual implementation, which requires a significant amount of expertise, time and coding-effort (Geiger et al., 2006). This motivates the development of methods to support and automate the design of dispatching rules, which is the main concern of this thesis.

Three different methods are proposed in this work, which are each based on an Evolutionary Algorithm (EA). The first method employs an EA to search for job shop problem instances on which a given dispatching rule performs poorly. The idea of this method is to support the development of dispatching rules by revealing the weaknesses of existing rules. Although similar methods have been developed by Kosoresow and Johnson (2002) and Julstrom (2009), this work appears to be the first to apply the idea to dispatching rules, and the area of scheduling in general. Moreover, the proposed EA is designed to search for problem instances where the poor performance of a tested dispatching rule can be attributed to a single suboptimal decision it takes, which is a key difference to previous work. The advantage of focussing on single decision points is that it greatly facilitates the analysis of problem instances and the identification of conditions that cause the rule to fail. The effectiveness of the method is assessed by applying it to a benchmark rule for a well-studied dynamic job shop problem with the objective of minimising mean flow time, where the insights gained from the analysis of the evolved problem instances are incorporated in the design of two new dispatching rules. The new rules are then evaluated against the benchmark rule and some other effective rules.

In contrast to the first method, the second method employs an EA directly to search for effective rules, thereby automating the development process. Such methods that operate on a search space of heuristics are commonly referred to as hyper-heuristics (Burke et al., 2010). The proposed hyper-heuristic is based on a specific type of EA, called Genetic Programming (GP), and generates a dispatching rule from basic job and machine attributes. Several researchers have developed similar GP-based hyper-heuristics for various scheduling problems, with promising results (e.g. Atlan et al., 1994; Geiger and Uzsoy, 2008; Jakobović and Marasović, 2012). However, a shortcoming of these studies is that they predominantly focus on static problems of small to moderate size, which are likely to be better solved by some global solution method than some dispatching rule. This work investigates the potential of GP hyper-heuristics to generate effective rules for complex, dynamic problems. In particular, the developed hyper-heuristic is applied to some dynamic, flexible job shop problems from semiconductor manufacturing with the objectives of minimising mean flow time and mean weighted tardiness, respectively. These problems feature complex processing characteristics such as sequence-dependent setup times and batch processing, which make the manual design of effective dispatching rules very challenging. To assess the effectiveness of the hyper-heuristic, the evolved dispatching rules are evaluated against some benchmark rules from the literature, and are further tested for their robustness to changes in the job arrival pattern and due date setting.

The third method is another hyper-heuristic that (automatically) generates a set of work centre-specific rules by selecting a (potentially) different rule for each work centre from a number of existing rules. The motivation for this method is that using different dispatching rules at different work centres has been found to be beneficial for some problems where work centres vary with respect to their position in the shop (Barrett and Barman, 1986; LaForge and Barman, 1989; Mahmoodi et al., 1996; Sarper and Henry, 1996; Barman, 1997, 1998) and/or workload (Raman, Talbot, and Rachamadugu, 1989; Ruben and Mahmoodi, 1998; Bokhorst, Nomden, and Slomp, 2008). Moreover, similar hyper-heuristics have been developed and shown to be effective by Baek and Yoon (2002) and Yang et al. (2007) for small problems with up to ten work centres. This work investigates the potential of EA-based hyper-heuristics to generate sets of work centre-specific rules for complex, dynamic problems that feature a large set of heterogeneous work centres. The proposed hyper-heuristic is specifically designed for shops that contain both serial and batch processing work centres, and further extends previously developed hyper-heuristics by optimising the information horizon of each work centre, i.e. the degree to which jobs that are expected to arrive in the near future are considered in the decision making. The latter extension is motivated by the observation that it may be beneficial to wait for future jobs at some work centres, e.g. to save a long setup time or to support the operation of a downstream bottleneck, while it may be counterproductive at others, e.g. at highly utilised work centres. The hyper-heuristic is applied to the same problems as the GP hyper-heuristic, which feature different types of work centres that vary with regard to their setup requirements, batch processing capacities, number of parallel machines, workload and their position in the shop. To assess the effectiveness of the rule set hyper-heuristic, the evolved rule sets are evaluated against some benchmark rules, including those evolved by the GP hyper-heuristic, and are further tested for their robustness to changes in the job arrival pattern and due date setting.

This thesis is organised as follows. Chapter 2 gives a brief introduction to scheduling problems in production and methods to solve them, and introduces some basic terminology and notation that is used within the rest of the thesis. Chapter 3 contains a review of dispatching rules, with a focus on promising design concepts for selected problems of interest. Chapter 4 describes the methodology of this work, specifically the tools and techniques used for the development and evaluation of the three methods, which are subsequently discussed in three separate chapters. In particular, the design and application of the method that employs an EA to search for problem instances is described in Chapter 5, followed by a presentation of the GP hyper-heuristic and the rule set hyper-heuristic in Chapter 6 and Chapter 7, respectively. The thesis ends with some general conclusions drawn in Chapter 8.

Chapter 2

Production Scheduling

Scheduling problems in production are generally defined by a number of jobs, n, that have to be processed by a set of m machines. The task is to determine a processing order for the n jobs on each of the m machines, i.e. a schedule, that optimises a given objective function. A scheduling problem can be either static or dynamic. *Static* scheduling problems are deterministic and fully defined problems with a limited number of jobs, whose attributes are completely known. In contrast, *dynamic* problems are subject to constant change over time due to random, stochastic events such as new job arrivals or machine breakdowns. This work is primarily concerned with dynamic scheduling problems, which is motivated by the fact that real-world production environments are typically dynamic in nature.

This chapter provides a general introduction to scheduling problems and methods to solve them, with a focus on those type of problems that are most relevant for this work (for a more extensive discussion on problems and solution techniques in scheduling see Pinedo, 2008). The first section introduces the terminology and the notation used to describe scheduling problems in this thesis. It also discusses some issues related to the complexity of scheduling problems. This is followed by another section covering solution methods for static and dynamic scheduling problems.

2.1 Problem Definition and Notation

A convenient way to describe a specific scheduling problem is by means of the $\alpha|\beta|\gamma$ notation of Graham et al. (1979), which is adopted herein. The three fields represent the shop configuration, processing characteristics and objective function of the scheduling problem, which are separately discussed in the following.

2.1.1 The α Field — Shop Configuration

A number of shop configurations are commonly described in the scheduling literature. The simplest configuration is a single machine that is responsible for processing all jobs, which is denoted by a 1 in the α field. *Pm* denotes a *parallel machine* environment, where *m* identical machines of the same functionality are available to process a job, while *Qm* is used to indicate that the parallel machines differ with respect to their processing speed. *Multi-stage* problems, which are generally NP-hard (Garey et al., 1976), are characterised by jobs that consist of a number of processing steps, or operations, that need to be performed on distinct machines. If all jobs follow the same route that involves visiting each of the *m* distinct machines exactly once, the shop is called a *flow shop*, which is denoted by Fm. Hence, every job consists of exactly *m* operations in a flow shop. In the other extreme of an ideal *job shop*, each job has its own specified route and may revisit some of the *m* machines several times while leaving out others completely. This implies that the number of operations can vary between jobs in a job shop. In practice, the number of different job routes is often limited by the product portfolio, which leads to a flow pattern that lies in between the two extremes of the linear flow of a flow shop and the random pattern of an ideal job shop. Such a shop is referred to as a *flow-dominant* job shop herein, and like the ideal job shop, is denoted by Jm. Flow shops and job shops are further called *flexible*, if they contain at least one work centre that consists of more than one machine, where a *work centre* is defined as a set of parallel machines sharing the same input and output buffer. FFc and FJc denote a flexible flow or job shop, respectively, with *c* work centres.

An important characteristic of multi-stage problem is the workload balance among work centres. In a perfectly balanced shop all work centres are equally utilised, where *utilisation* of a work centre is defined as the fraction of busy time averaged across its machines. On the other hand, an unbalanced shop implies that some work centres have a higher workload than others. Then, the utilisation of the shop is given by the utilisation of the *bottleneck*, i.e. the mostly utilised work centre.

2.1.2 The β Field — Processing Characteristics

The standard assumptions with respect to the processing characteristics of production scheduling problems are that

- a job can be processed by only one machine at at time (no splitting or overlapping),
- a machine can process only one job at a time (no batch processing),
- a started operation must be completed before the respective machine can process another job (no preemption),
- an operation can only be performed on the specified work centre (no alternate routeings),

- the processing of jobs is independent of the progress of other jobs (no assembly operations or precedence constraints),
- processing times are independent of the schedule and cover all activities related to the execution of an operation (no sequence-dependent setup times),
- machines are the only limiting resource (no lack of operators, tools or material),
- there are no space or time constraints on input and output buffers (no blocking, scrap or rework),
- all machines are continuously available (no breakdowns or maintenance activities),
- all jobs are available to be scheduled instantly (no dynamic job arrivals).

A problem that fulfils all of the above conditions is typically identified by an empty β field. Otherwise, specific entries are used to indicate certain deviations from these assumptions as illustrated in the following.

Dynamic Job Arrivals

If jobs arrive at the shop over time, i.e. each job j has its own specific release date r_j, this is denoted by an r_j entry in the β field. Dynamic job arrivals generally increase the complexity of scheduling problems, and make some otherwise simple single machine problems become NP-hard (Lenstra et al., 1977). In reality, perfect information on future jobs and their release dates is often not available, as the job arrival process may depend on external customers placing their orders. It is thus assumed in this work that future jobs and their characteristics are unknown until the time of their release to the shop.

Sequence-Dependent Setup Times

An s_{ef} entry in the β field indicates that machines require to be set up properly to process a particular job, and that the durations of these setups are *sequence-dependent*, i.e. they depend on the current state of the machine, resulting from the previously executed operation. Contrary to sequence-independent setup times, which can be simply treated as part of the processing time of an operation, sequence-dependent setup times have to be explicitly modelled and largely increase the complexity of a scheduling problem. This is illustrated by the fact that even the most trivial single machine problems become NP-hard in the presence of sequence-dependent setup times (Monma and Potts, 1989; Potts and Kovalyov, 2000).

One way to represent the setup characteristics of a machine is by a so-called setup time matrix. In practice, the number of different operations a machine can perform will be limited, implying a limited number of machine setups. Jobs requiring the same setup can then be grouped into *families* with the implication that changeovers only become necessary when switching from one family of jobs to another. The setup time matrix is a square matrix of dimension equal to the number of families, where its entries s_{ef} specify the time for changing over from a job belonging to family e to one belonging to family f. Since no setup is required between the processing of jobs of the same family, all elements on the main diagonal are equal to 0 in a setup time matrix. Moreover, setup time matrices normally satisfy the triangle inequality, $s_{ef} + s_{fg} \geq s_{eg}\ \forall e, f, g$ (Pinedo, 2008, Chapter 4). The inequality simply states that the direct transition from one to the other family setup is always the fastest, which is usually the case in reality. Otherwise, the matrix can be made compliant with the triangle inequality by substituting the respective entry s_{eg} by the lower value of $s_{ef} + s_{fg}$. Figure 2.1 shows an example of a setup time matrix, which satisfies the triangle inequality, for a machine that can be set up in four different ways. To illustrate, the time for changing over from the setup required for processing jobs of family 1 to the setup required for jobs of family 3 is $s_{13} = 2$ in this case.

0	3	2	3
5	0	2	3
5	3	0	3
5	3	2	0

Figure 2.1: Example of a setup time matrix for a machine with four possible setups.

A special case of sequence-dependent setup times is the one where all changeovers require the same amount of time, i.e. all entries of the setup time matrix apart from the zero-diagonal have the same value. Although setup times are then independent of the family sequence (and have been referred to as being sequence-independent in the literature), they are clearly not independent of the job sequence as no changeover is required between two jobs of the same family, and thus have to be modelled as sequence-dependent.

In multi-stage problems, there may be more than one work centre featuring sequence-dependent setup times. Then, one matrix, which applies to every machine of the respective work centre, is needed for each of these work centres. Moreover, since the concept of family membership is generally related to operations and not to jobs, the association of jobs to families can vary for their different operations in multi-stage problems.

Batch Processing

In contrast to *serial processing* machines, *batch processing* machines can process more than one job at the same time as a batch. In the literature, a distinction is made between batch processing problems with compatible and incompatible families. In the case of compatible families, a batch can be comprised of jobs with different processing times and of different sizes, and the only constraint regarding the formation of batches is that the total size of a batch may not exceed the capacity of the machine. The processing time of a batch is then determined by the maximum of the processing times of the included jobs. The various possibilites to form batches make these problems difficult to solve, and single machines problems of this type are already NP-hard, in general (Uzsoy, 1994; Potts and Kovalyov, 2000). On the other hand, batch processing problems with incompatible families only allow for jobs of the same family to be batched together, which strongly reduces the number of possible batches that can be formed. In consequence, some common single batch processing machine problems are still easy to solve if families are incompatible (Uzsoy, 1995).

In the context of batch processing, *families* are defined by jobs that share the same processing requirements with regard to machine setup, time and space, and, as in the case of sequence-dependent setup times, are related to operations in multi-stage problems. Moreover, the capacity of the machine in terms of the maximum batch size B_f can vary for different families f. In this work, only batch processing problems with incompatible families are considered, which are identified by a B_f entry in the β field.

2.1.3 The γ Field — Objective Function

In practice, the ultimate goal of production scheduling is to minimise the cost of production. However, since cost structures tend to vary widely across companies, general, time-based objective functions that represent a certain cost component are more commonly used in the literature. Most of these functions can be classified as being either related to the efficiency of the shop or its ability to adhere to due dates.

A classic efficiency-related objective of scheduling problems is to minimise the *makespan* of jobs, which is the total time required to finish a given set of jobs. In mathematical terms, the makespan is defined as $\max_{j \in N}(C_j)$, where N denotes the set of jobs and C_j the completion time of job j. Problems with the makespan objective are indicated by a $C_{\max}$ symbol in the γ field. The makespan is an intuitive measure of efficiency for static scheduling problems that feature a limited and fully defined set of jobs. However, it is generally not a very sensible criterion for dynamic scheduling problems, in which new jobs arrive continuously.

A more suitable objective for dynamic problems is the minimisation of the mean flow time, where the *flow time* of job j is defined as $C_j - r_j$, i.e. the length of time the job spends in the shop, from the time of its arrival to the time of its completion. The mean flow time is therefore calculated as

$$\frac{1}{n}\sum_{j=1}^{n}(C_j - r_j) = \frac{1}{n}\sum_{j=1}^{n} C_j - \frac{1}{n}\sum_{j=1}^{n} r_j. \tag{2.1}$$

By Little's Law (Little, 1961), the mean flow time is proportional to the average number of jobs in the shop, and mean flow time is thus a widely accepted measure for the holding costs of in-process inventory. In Equation (2.1), the last term on the right corresponds to the mean of the job release dates, which are normally not decision variables of a scheduling problem, but assumed as fixed. Then, they can be taken out of the objective function, and the objective reduces to the minimisation of the mean completion time. Moreover, the flow time of job j can also be written as $p_j + q_j$, where p_j indicates its processing time and q_j its queueing or waiting time, which is the reducible component of the flow time. In other words, there is no need to distinguish between the objectives of minimising average number of jobs in shop, mean (or total) flow time, mean (or total) waiting time and mean (or total) completion time, in this work, as they all refer to the same objective, which is denoted by $\overline{C}$.

Most due date-related objective functions, used to represent costs related to the violation of due dates such as contractual penalties for late deliveries or costs of customer badwill, are some function of the tardiness of jobs. The *tardiness* of job j, denoted by T_j, is defined as the non-negative difference between its completion time and its due date d_j, i.e. $T_j = \max(C_j - d_j, 0)$. Common objectives are the minimisation of the maximum or mean (or total) tardiness of all jobs, indicated by $T_{\max}$ and $\overline{T}$, respectively, and the proportion (or number) of tardy jobs, denoted by $\overline{U}$. The proportion of tardy jobs can further be seen as one of two components of the mean tardiness, with the second one being the *conditional mean tardiness*, which is the mean tardiness of the tardy jobs and denoted by $\overline{T}^{\mathrm{c}}$ herein. The relationship between the three measures is given by $\overline{T} = \overline{U} \times \overline{T}^{\mathrm{c}}$.

Pinedo (2008) establishes a complexity hierarchy among the different objectives, according to which a scheduling problem with the objective of minimising proportion tardy or mean tardiness is generally more complex than the corresponding problem with the maximum tardiness objective, which in turn is harder to solve than the respective makespan problem. In addition, solving a problem for the minimum mean tardiness is at least as hard as to solve it for the minimum mean flow time (Pinedo, 2008, Chapter 2).

In some situations, the costs of holding a job or violating its due date may depend on the specific job. This is expressed by weighted objective functions, where the contribution of each job j, e.g. in terms of tardiness or flow time, is multiplied by a weight w_j that reflects its relative importance. The symbols $\overline{wC}$ and $\overline{wT}$ indicate the objectives of minimising mean weighted flow time and mean weighted tardiness, respectively. Clearly, (unequal) job weights increase the complexity of scheduling problems.

The objectives discussed so far are all *regular*, i.e. the objective function values do not improve if the completion of a job is delayed. An example of a non-regular objective is the minimisation of mean earliness, where the *earliness* of job j, denoted by E_j, is defined as $E_j = \max(d_j - C_j, 0)$. Such objectives that penalise the earliness of jobs are characteristic of just-in-time environments, where the aim is to finish jobs dead on time.

To illustrate the notation described in this section, consider a scheduling problem from semiconductor manufacturing. Production systems in this industry are characterised by a heterogeneous set of work centres, of which some contain multiple machines operating in parallel, and by product-specific routes that involve revisitations. Moreover, some of the machines in the shop feature sequence-dependent setup times while others are capable of processing several jobs simultaneously as a batch (Uzsoy et al., 1992; Pfund et al., 2006). Provided that job arrivals are dynamic and that the objective is to minimise mean weighted tardiness, this problem is then denoted by $FJc|r_j, s_{ef}, B_f|\overline{wT}$.

2.2 Solution Methods

In discussing solution methods, a distinction has to be made between methods for static and dynamic scheduling problems.

2.2.1 Static Scheduling Problems

Static problems are fully defined and, in theory, can be solved to optimality. However, scheduling problems in practice are often too complex for exact methods to find the optimum in a reasonable amount of time, which motivates the use of heuristics and other methods that focus on delivering a good, but not necessarily optimal solution in a short time. Figure 2.2 presents an overview of methods used to solve static scheduling problems.

Exact solution methods generally involve the formulation of the problem as a (dynamic, mathematical or constraint) programme that is then solved by a complete and systematic search of the solution space. Many of the more complex scheduling problems can be formulated as mixed integer programmes (see, e.g. Mason et al., 2005; Pan and Chen, 2005), which are typically tackled by the well-known branch-and-bound procedure. Exact methods can further be used as heuristics by restricting their search to certain parts of the solution space, or by setting a limit on the time the algorithm is allowed to run for, after which the best solution found so far is returned.

A different kind of search-based heuristic are the so-called metaheuristics, which start from initial (randomly generated) solutions that they seek to improve upon by means of local search techniques in combination with mechanisms that allow them to escape local optima and explore other parts of the solution space. A popular subcategory of metaheuristics are evolutionary algorithms, which have been widely applied to scheduling (Hart et al., 2005).

An alternative heuristic approach to search-based heuristics is to decompose a given scheduling problem into smaller subproblems that can be solved more easily. These partial solutions are then combined to yield an overall solution to the original problem. Decomposition methods for static scheduling problems are typically job- or machine-based. An example of a machine-based decomposition method is the well-known shifting bottleneck heuristic, which schedules machines in non-increasing order of their criticality and is commonly used to address (flexible) job shop problems (see Mason et al., 2002; Pinedo, 2008, Chapter 7).

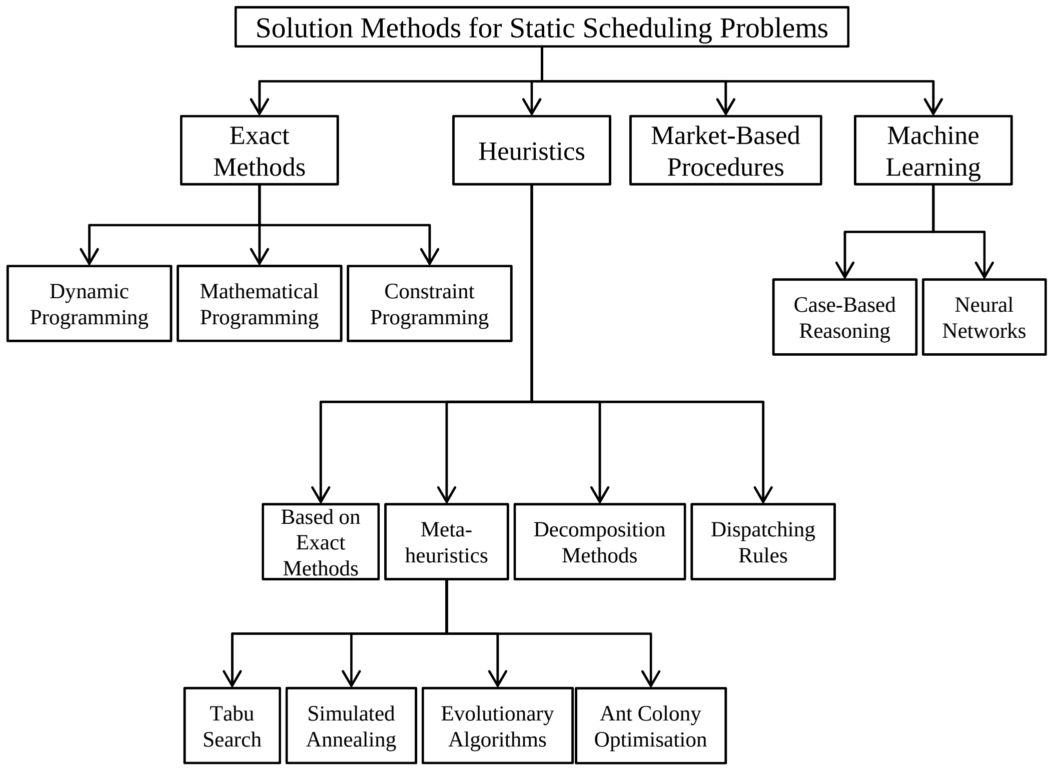


Figure 2.2: An overview of solution methods for static scheduling problems.

A particularly simple type of scheduling heuristic are dispatching rules, which progressively construct solutions by scheduling one operation at a time. Whenever a machine is available and there are jobs waiting to be processed on that machine, dispatching rules compute a priority index for each job (or batch) as a function of some job (or batch) attributes, e.g. its weight or due date, and machine attributes, e.g. its current setup, and schedule only the imminent operation of the job (or the batch) with the highest priority. Since dispatching rules lack a global perspective on the problem they tend to produce worse solutions than more sophisticated heuristics. However, their local horizon allows them to be executed very quickly, irrespective of the complexity of the overall problem.

Another approach that is based on local decision making are market-based procedures, which model scheduling decisions as a negotiation process between agents associated with jobs and agents representing the machines. Each job agent holds a certain budget, which it can use to pay for a processing time on the required machine(s). To reserve a time slot for the imminent operation of a job, the job agent makes a call for bids to which the agents of the machine(s) that can process the operation reply with bids. Each bid consist of a price and time slot, where the prices are set by the machine agents with the aim to maximise their own profit. The job agent then accepts the bid that it considers to be the best value for money, thereby scheduling the operation. A challenge with regard to the application of market-based procedures is the determination of suitable budgets, as well as effective pricing and acceptance rules, which are likely to be problem-specific. Toptal and Sabuncuoglu (2010) provide a more detailed discussion of market-based scheduling procedures.

Finally, static scheduling problems can also be addressed by means of machine learning algorithms. These algorithms solve problems by inference from good solutions to similar instances. Hence, they can only be employed in combination with another method or a human expert, which or who generates solutions to example problems that can be used to train the machine learning algorithm. Some scheduling algorithms based on machine learning can be found in Pinedo (2008, Chapter 18).

2.2.2 Dynamic Scheduling Problems

Contrary to static problems, dynamic scheduling problems cannot be solved optimally as the optimal solution depends on future uncertain events that occur only after a solution has been implemented. A key distinctive feature of solution methods for dynamic problems is how they account for these uncertainties, which can be in a proactive or reactive manner. Figure 2.3 provides an overview of the different methods for dynamic scheduling problems, based on the surveys by Aytug et al. (2005) and Ouelhadj and Petrovic (2009).

Robust scheduling methods take a proactive approach to uncertainty by generating flexible (or robust) schedules that, up to a certain degree, can accommodate future changes without a severe deterioration of the objective function value. Measures to increase the robustness of a schedule can include the insertion of idle times at machines that are likely to break down, or the upkeep of a minimum queue length at the bottleneck work centre to reduce the risk of a temporarily starving bottleneck due to a breakdown at an upstream machine (Pinedo, 2008, Chapter 18). The drawback of such measures is that they compromise performance to account for disruptions that do not necessarily occur, e.g. processing capacity may be wasted as a result of reserving machine time for repair work that turns out to not be required in the end. It is therefore important to have a good understanding of the uncertainties involved in a problem, their probability of occurrence and their potential impact on the quality of a schedule when using robust scheduling methods. This knowledge can then be incorporated into the solution method, e.g. into the constraints or objective function of a stochastic mathematical programme (Kouvelis et al., 2000). However, since predictions regarding future developments become increasingly inaccurate for longer time horizons, associated with the exponential increase of possible combinations of future events over time, robust scheduling methods are only practical for problems with relatively short time horizons.

Predictive-reactive methods are two-stage procedures that first generate a so-called predictive schedule on the basis of the initial problem data, and then revise and modify this schedule over time as new information becomes available. They can be classified as event-driven or

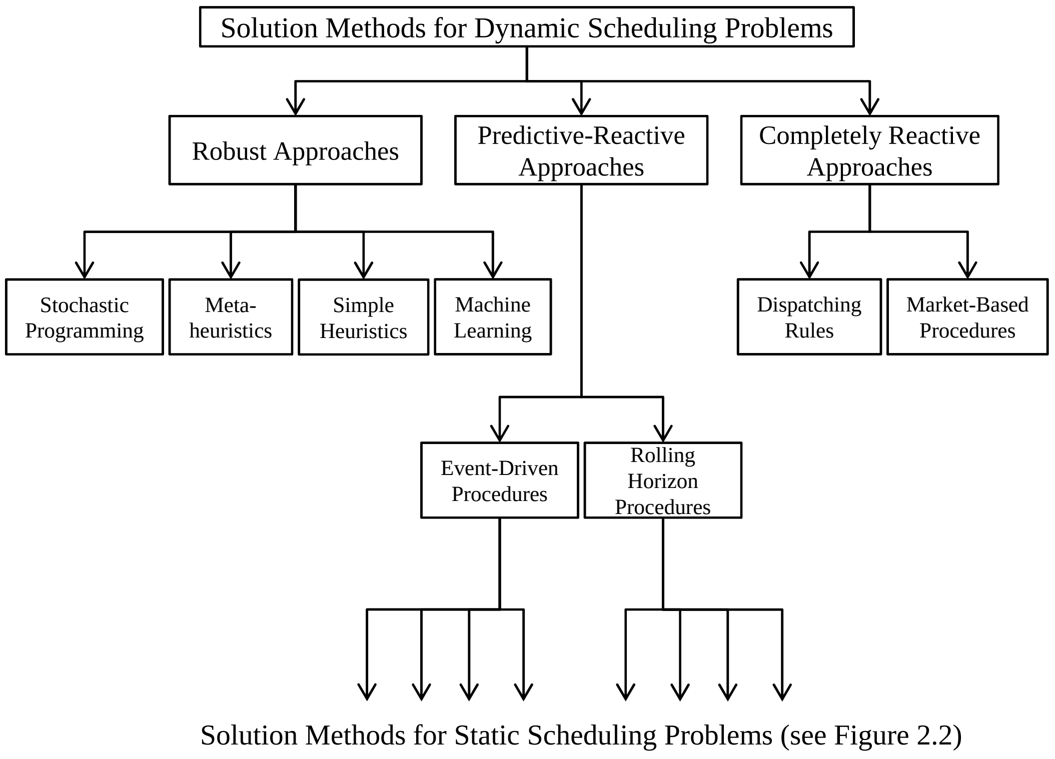

Solution Methods for Dynamic Scheduling Problems
Robust Approaches
Predictive-Reactive Approaches
Completely Reactive Approaches
Stochastic Programming
Meta-heuristics
Simple Heuristics
Machine Learning
Dispatching Rules
Market-Based Procedures
Event-Driven Procedures
Rolling Horizon Procedures
Solution Methods for Static Scheduling Problems (see Figure 2.2)

Figure 2.3: An overview of solution methods for dynamic scheduling problems.

time-driven, depending on the mechanism that initiates revisions of the schedule. Event-driven procedures revise the schedule in response to unexpected events, while time-driven procedures, commonly referred to as rolling horizon procedures, reoptimise the schedule at regular intervals. Concerning the generation of the predictive and revised schedules, any of the solution methods for static scheduling problems, shown in Figure 2.2, can be used in principle, where the suitability of a method for a given problem generally depends on the nature of the present uncertainties in terms of their disruptive power and the available time to react. In the case that the only random events concern the arrival of new jobs that hardly affect the quality of the current schedule, a rolling horizon procedure, which periodically applies a computationally expensive branch-and-bound algorithm, may be the method of choice (see, e.g. Ovacik and Uzsoy, 1994). On the other hand, a problem in which unexpected machine breakdowns can cause sudden and serious disruptions may require for an event-driven method that, in case of a breakdown, quickly restores feasibility of the schedule by means of a simple heuristic (see, e.g. Yamamoto and Nof, 1985). In practice, hybrid methods that generally follow a periodic reoptimisation but are able to react flexibly, if the disruption caused by a particular event is severe, are often employed.

Completely reactive methods are characterised by making scheduling decisions in real-time, immediately before they are implemented, thereby renouncing to plan into the future. The advantage of this approach is that changes due to random events are taken into account as they emerge, which allows for prompt reactions. However, in order to be able to schedule in real-time, reactive methods have to rely on procedures with low computational and information requirements such as dispatching rules or simple agent negotiation protocols, which take scheduling decisions on the basis of a locally restricted information horizon with little consideration of the overall problem structure.

In general, there is broad agreement in the literature that the higher the dynamics of a scheduling problem, the better completely reactive methods fare in comparison to predictive-reactive methods (Aytug et al., 2005). Clearly, a globally optimised schedule becomes obsolete more quickly if the frequency and impact of random events is high, up to a point where the assumptions underlying the schedule become invalid only moments after the very first part of the schedule has been implemented. Then, the global solution method effectively solves the wrong problem(s) and thus may well lead to poorer scheduling decisions than a locally restricted method that does not plan into the future at all.

The use of a predictive-reactive approach based on global solution methods may also become impractical if the given scheduling problem is too complex to permit the generation of good global schedules in a reasonable amount of time (Ouelhadj and Petrovic, 2009). The limits of global optimisation methods is impressively demonstrated by a study by Mason et al. (2005), who apply the well-known CPLEX solver to a mixed integer programming formulation of the $FJc|r_j, s_{ef}, B_f|\overline{wT}$ problem. CPLEX is shown to be unable to find, within six hours of CPU time, a feasible, let alone the optimal solution to problem instances with only five machines and nine jobs, consisting of six operations each. In contrast, real-world problems from semiconductor manufacturing, which feature similar processing characteristics (see Section 2.1), typically require thousands of operations to be scheduled on hundreds of machines at a time, making the generation of good global schedules extremely difficult.

In summary, a variety of methods exist for solving scheduling problems, which further can be employed as hybrid methods in various combinations. Solution methods for dynamic problems can be classified as robust, predictive-reactive and completely reactive. Robust methods try to accommodate uncertainty by anticipating future events, and by themselves are thus only practical for problems with relatively short time horizons. On the other hand, scheduling problems can be too dynamic or complex for predictive-reactive methods to be effective, which motivates the use of completely reactive methods with their local decision making.

Completely reactive methods can be based either on dispatching rules or market-based procedures. While the latter have to be properly configured for the problem at hand, which is generally not easy, dispatching rules are known for their simple and intuitive nature, their ease of implementation and their flexibility to incorporate domain knowledge and expertise

(Aytug et al., 2005). These factors, together with the low computational and information requirements of dispatching rules, explain their wide usage in practice (Pfund et al., 2006) and the ongoing research on the development and evaluation of dispatching rules. The following chapter contains a review of dispatching rules and their underlying design concepts.

Chapter 3

The Design of Dispatching Rules

Since dispatching rules are simple heuristics with a locally restricted information horizon, it is obvious that no rule can outperform all other rules across different shop configurations, objective functions or operating conditions (Blackstone et al., 1982; Kiran and Smith, 1984; Haupt, 1989; Holthaus and Rajendran, 2000). Hence, dispatching rules should be generally designed for the problem at hand, and their priority indices should incorporate attributes that are specifically relevant with respect to the given problem. In consequence, many different dispatching rules have been developed over the last few decades (for extensive surveys see, e.g. Panwalkar and Iskander, 1977; Blackstone et al., 1982; Haupt, 1989).

The aim of this chapter is to highlight the issues in developing effective dispatching rules as well as to identify the most promising design concepts from the literature for selected problems of interest. In particular, the scope of the review is on rules designed for scheduling problems with the objective to minimise mean flow time or mean (weighted) tardiness, where problems are covered in increasing degree of their difficulty. Thus, the first section presents rules for simple single stage problems, which provide the basis for the development of more sophisticated rules for problems with sequence-dependent setup times, batch processing problems or multi-stage problems, discussed in the three subsequent sections. Throughout this thesis, priority indices are defined so that a higher index I_j corresponds to a higher priority of the job.

3.1 Simple Single Stage Problems

The design of effective dispatching rules for scheduling problems with a single processing stage or work centre and without complex characteristics such as sequence-dependent setup times is comparatively easy, and similar principals apply with regard to single machine and parallel machine problems.

3.1.1 Minimising Mean Flow Time

The $1||\overline{C}$ and $Pm||\overline{C}$ problems are both optimally solved by the Shortest Processing Time (SPT) rule, according to which jobs are scheduled in non-decreasing order of their processing times (Smith, 1956; Pinedo, 2008, Chapters 3 and 5). Thus, the priority index of SPT for job j can be simply defined as

$$I_j^{\text{SPT}} = -p_j. \tag{3.1}$$

The application of the SPT rule is further the best one can do for the analogous problems with dynamic job arrivals without information on future jobs, which are assumed to be unknown in this work.

3.1.2 Minimising Mean Weighted Tardiness

Du and Leung (1990) show that the easiest type of this problem category, namely the $1||\overline{T}$ problem, is already NP-hard. Hence, a dispatching rule (polynomial time algorithm) that optimally solves the $1||\overline{wT}$ or $Pm||\overline{wT}$ problem is not known, and its existence is very unlikely. However, there are two extreme cases that can be solved by a simple dispatching rule.

First, consider the case in which it is impossible to finish any job on time. The mean weighted tardiness can then be rewritten as

$$\frac{1}{n}\sum_{j=1}^{n} w_j T_j = \frac{1}{n}\sum_{j=1}^{n} w_j (C_j - d_j) = \frac{1}{n}\sum_{j=1}^{n} w_j C_j - \frac{1}{n}\sum_{j=1}^{n} w_j d_j, \tag{3.2}$$

where the last term of the expression on the right is a constant and independent of the schedule. Hence, the objective of minimising mean weighted tardiness becomes the same as minimising mean weighted completion time in this case. The $1||\overline{wC}$ problem, in turn, is optimally solved by the Weighted Shortest Processing Time (WSPT) rule by Smith (1956), which also yields close to optimal solutions to the $Pm||\overline{wC}$ problem (Pinedo, 2008, Chapters 3 and 5). The priority index of job j with WSPT is the ratio of its weight to its processing time, i.e.

$$I_j^{\text{WSPT}} = \frac{w_j}{p_j}. \tag{3.3}$$

The other extreme case is the one where it is possible to finish every job on time. Then, an algorithm that minimises the maximum tardiness will produce a solution with a maximum tardiness of 0, which is obviously also optimal with respect to the mean weighted tardiness criterion. The $1||T_{\max}$ problem is optimally solved by the Earliest Due Date (EDD) rule, which simply schedules jobs in non-decreasing order of their due dates (Jackson, 1955; Pinedo, 2008, Chapter 3). The priority index of EDD is thus given by

$$I_j^{\text{EDD}} = -d_j \tag{3.4}$$

or

$$I_j^{\text{EDD}} = -(d_j - t), \tag{3.5}$$

where t refers to the current time.

Both priority indices represent the same rule as the current time is the same for all jobs and hence does not affect the ranking of priorities. However, as soon as EDD is integrated with other rules into so-called composite dispatching rules, the formulation of its priority index does become important. If the absolute due date from Equation (3.4) is integrated with other job attributes, their relative proportions, and hence the behaviour of the rule, may change over time. To illustrate, a dispatching rule with a priority index $I_j = -\max(10p_j, d_j)$ might operate like SPT in the beginning and then switch to EDD dispatching as time passes and due dates take on substantially larger values than the processing times, which do not change with time. Clearly, such a time-dependent switch is generally not desired and can be avoided by relating due dates to the current time as in Equation (3.5), before integrating them with other attributes. In fact, the term $d_j - t$, which is called the *remaining allowance* of job j, appears in many effective due date-oriented rules, as shown in the following.

The analytical result that EDD and WSPT each lead to an optimal solution to the $1||\overline{wT}$ problem under opposed extreme conditions seems to carry over to the problem with dynamic job arrivals to some extent. Baker and Bertrand (1982) apply (W)SPT and EDD to the $1|r_j|\overline{T}$ problem, and find that SPT clearly outperforms EDD under tight due date levels, while the latter is the better rule if due dates are loose. As a consequence, they propose the Modified Due Date (MDD) rule, which behaves like SPT or EDD in the respective extreme conditions and presents a compromise for intermediate tightness settings. Its priority index is defined as

$$I_j^{\text{MDD}} = -\max(t + p_j, d_j), \tag{3.6}$$

i.e. the priority of a job is determined by the maximum of its earliest possible completion time and its due date. Kanet and Li (2004) reformulate the priority index of MDD, by substracting t, to

$$I_j^{\text{MDD}} = -\max(p_j, d_j - t). \tag{3.7}$$

The terms of earliest possible completion time and due date in the original formulation are thus replaced by the processing time and remaining allowance of a job, the priority indices of SPT and EDD. Baker and Bertrand (1982) test MDD against SPT and EDD on the $1|r_j|\overline{T}$ problem with various levels of shop utilisation and due date tightness, as well as different due date assignment rules, and report that MDD dominates its component rules across all experimental conditions. The dominance of MDD over EDD and SPT for the $1|r_j|\overline{T}$ problem is confirmed by Moser and Engell (1992a) for different levels of utilisation and due date tightness.

Kanet and Li (2004) extend the MDD rule to the Weighted Modified Due Date (WMDD) rule to account for unequal job weights. They formulate this extension on the basis of the MDD index from Equation (3.7) and define its priority index as

$$I_j^{\text{WMDD}} = -\frac{1}{w_j}(\max(p_j, d_j - t)). \tag{3.8}$$

Their decision not to base the WMDD rule on the original MDD index from Equation (3.6) is motivated by the aim to create a rule that reverts to WSPT behaviour when due dates are very tight. This is achieved by their WMDD formulation, but not by an extension that is based on the priority index from Equation (3.6), which would prioritise jobs in non-increasing order of the term $(t + p_j)/w_j$ in tight due date settings. They also mention some preliminary simulation experiments showing this alternative rule to be inferior, which supports the earlier notion that it is more sensible to use the remaining allowance than absolute due dates within composite rules. Kanet and Li compare the performance of WMDD for the $1||\overline{wT}$ and the $1|r_j|\overline{wT}$ problem to that of various other rules, and conclude that WMDD is one of the most effective rules tested.

In trying to minimise mean (weighted) tardiness, it is important to discriminate jobs on the basis of how urgent they are with respect to their individual due dates. One such measure for the urgency of a job is the *critical ratio*, which is generally defined as the fraction of the remaining allowance of a job to the remaining time required for its completion (Blackstone et al., 1982; Baker, 1984). Thus, the more urgent a job, the smaller its critical ratio, with a ratio lower than 1 indicating that the job is too far behind schedule to finish on time. By prioritising jobs in non-decreasing order of their critical ratio, the Critical Ratio (CR) rule aims to minimise mean tardiness by processing the more urgent jobs first. In the context of simple single stage problems, the remaining time required for the completion of a job is just its processing time and the priority index of CR is thus given by

$$I_j^{\text{CR}} = -\frac{d_j - t}{p_j}. \tag{3.9}$$

A general issue with rules that are based on a ratio as priority index is that their behaviour can change as a result of a change of sign in either the numerator or denominator. To illustrate, the CR rule generally gives priority to jobs that require more time before they are completed. However, when some jobs have missed their due dates (at which point their priority indices become positive) the rule tends to prioritise the shorter among those late jobs. This anomaly has been criticised as one of the main drawbacks of ratio type rules, and efforts have been made to rectify it (Adam and Surkis, 1980; Kanet, 1982). On the other hand, the shift towards the prioritisation of short jobs when some jobs are late can be considered a beneficial adaptation to the changing conditions (Vepsalainen and Morton, 1987), as scheduling the tardy jobs first, in non-decreasing order of their processing times, is actually a sensible strategy to reduce mean tardiness. The anomaly, however, does mean that the incorporation of weights into the priority index of the CR rule is not trivial. The aim of a weighted variant of CR should be to always assign a higher priority to jobs with larger weights, no matter whether jobs are late or not. In practice, a hierarchical rule is often used that prioritises jobs in non-increasing order of their weights, and uses the CR priority index to break ties (Pfund et al., 2006).

Another popular measure for the urgency of a job is the difference (rather than the fraction) of the remaining allowance and the remaining time required for its completion, commonly known as the *slack* (Baker, 1984). For simple single stage problems, the slack S_j of job j is calculated as

$$S_j = d_j - t - p_j. \tag{3.10}$$

The direct use of the slack as a priority index in form of the Minimum Slack (MS) rule, which schedules jobs in non-decreasing order of their slack, is generally not recommended. This is due to the inherent 'anti-SPT' logic (Baker, 1984; Vepsalainen and Morton, 1987), i.e. the tendency to prioritise longer jobs and thereby counteracting SPT, which can lead to a very poor performance in conditions of tight due dates. However, the slack is a recurring component of many of the more effective due date-oriented rules, e.g. the priority index of the CR rule can be reformulated as

$$I_j^{\text{CR}} = -\frac{d_j - t}{p_j} - \frac{p_j}{p_j} = -\frac{S_j}{p_j}, \tag{3.11}$$

since p_j/p_j is a constant and constants do not affect the relative priority of jobs. The expression on the right in Equation (3.11) is the priority index of the Slack per Remaining Processing Time (S/RPT) rule, which is thus equivalent to CR (Kutanoglu and Sabuncuoglu, 1999). Moreover, there is an alternative slack-based formulation of the WMDD rule, i.e.

$$I_j^{\text{WMDD}} = \begin{cases} -\dfrac{d_j - t}{w_j} & \text{if } S_j \geq 0, \\ -\dfrac{p_j}{w_j} & \text{otherwise,} \end{cases} \tag{3.12}$$

in which the slack can be seen to trigger a switch from a weighted version of EDD to WSPT when jobs become time-critical (Baker and Bertrand, 1982).

Another promising approach, in which the slack plays an important role, are dispatching rules that determine the priority of a job by estimating the cost of delaying it by one unit of time. Clearly, for a job that is behind schedule, the penalty for delaying it for a further time unit in terms of increased weighted tardiness is equal to the weight of the job. However, if a job has some positive slack it is not as clear to what increase of weighted tardiness the decision of delaying it will ultimately lead, particularly not for more complex scheduling problems. Then, the cost of taking that decision can only be estimated, e.g. by taking into account the slack of the job. Two popular examples of this type of rule are the Cost Over Time (COVERT) and the Apparent Tardiness Cost (ATC) rule. While both rules are similar in that they prioritise jobs in non-increasing order of the ratio of its estimated delay cost to its (imminent) processing time, they differ with respect to how they estimate the delay cost.

The priority index of the COVERT rule, developed by Carroll (1965), is given by

$$I_j^{\mathrm{COVERT}} = \frac{w_j}{p_j}\left(\max\left(1 - \frac{\max(S_j, 0)}{k\hat{q}_j}, 0\right)\right), \tag{3.13}$$

where $\hat{q}_j$ indicates the estimated queueing time of job j and k is a scaling parameter. As can be seen from Equation (3.13), the estimated delay cost (and the priority index) of a job decreases linearly in the amount of its slack, up to a point where the slack exceeds $k\hat{q}_j$. Such jobs are considered to have excessive slack so that a delay does not cause any increase of weighted tardiness, and their estimated delay cost is thus 0. Obviously, the performance of COVERT is affected to some extent by the method used to determine the queueing times $\hat{q}_j$. Carroll (1965) runs a preliminary simulation experiment with the First Come First Served (FCFS) rule to obtain an estimate for the queueing times. Other possible options are to use recently observed queueing times or initial allowances (Russell et al., 1987).

The ATC rule originates from a paper by Morton and Rachamadugu (1982), and its priority index is defined as

$$I_j^{\mathrm{ATC}} = \frac{w_j}{p_j}\exp\left(-\frac{\max(S_j, 0)}{k\overline{p}}\right). \tag{3.14}$$

The main difference to COVERT is that ATC relies on an exponential instead of a linear function to discount the estimated delay cost for the slack of a job. Moreover, the queueing time estimate in the denominator of the discount function is replaced by the average processing time in the queue $\overline{p}$. Morton and Rachamadugu report that the exponential discount function is more effective than a linear one in reducing mean weighted tardiness of single machine problems. They also test ATC against several dispatching rules on the $1||\overline{wT}$ and the $Pm||\overline{wT}$ problems, including WSPT and EDD, and conclude that ATC is superior across different levels of due date tightness and due date range (Morton and Rachamadugu, 1982; Rachamadugu and Morton, 1983). However, in a few cases the scaling parameter k has to be adjusted to the problem parameters for ATC to outperform the other rules.

The effect of the scaling parameter on the mode of operation of ATC and COVERT, respectively, is shown in Figure 3.1. For illustrative purposes, it is assumed that $\hat{q}_j = \overline{p}$. On the one hand, if k is chosen very large, the priority indices become insensitive to the slack of a job and both rules converge to the WSPT rule, which is good for tight due dates. On the other hand, a very small k value leads to a sharp increase in the priority indices as jobs approach the depletion of their slack, which is a more suitable strategy for loose due date settings. In the original development of the COVERT rule, Carroll (1965) sets $k = 1$, according to Russell et al. (1987). For ATC, Morton and Rachamadugu (1982) and Rachamadugu and Morton (1983) find that $k = 2$ works well for single stage problems, but also observe that a lower parameter value can be beneficial in loose due date conditions.

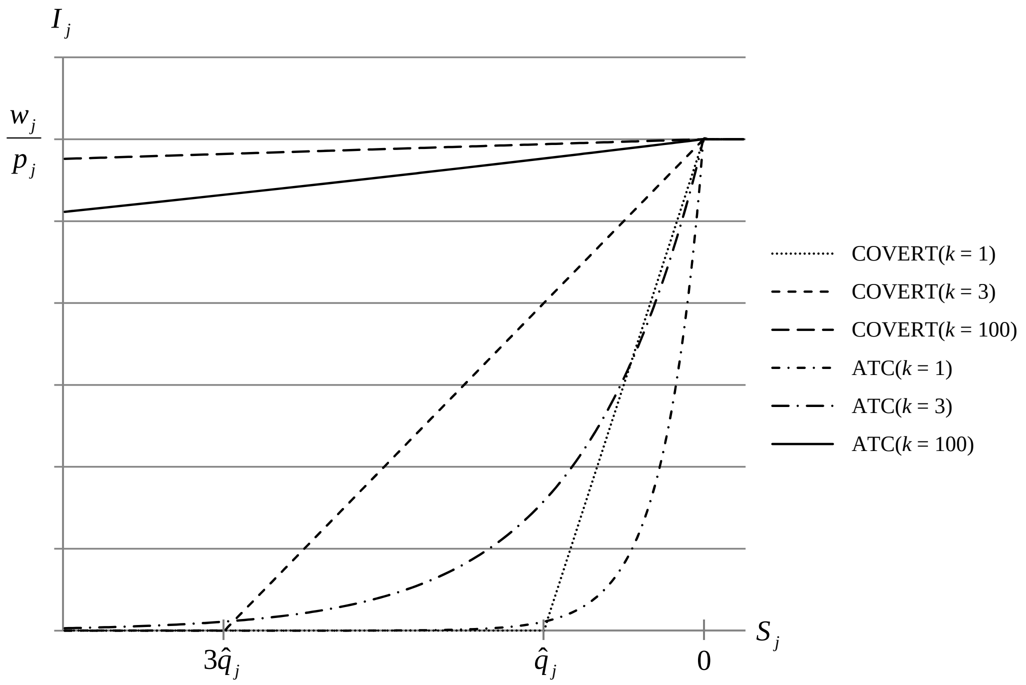


Figure 3.1: Influence of the scaling parameter k on the priority index I_j of the COVERT and ATC rules as a function of the slack S_j of job j (see also Morton and Rachamadugu, 1982).

Kanet and Li (2004) compare the performance of ATC and COVERT to that of their WMDD rule. Regarding the $1|r_j|\overline{wT}$ problem, all three rules are shown to perform similar, in general, while COVERT is found to be less effective than ATC and WMDD for the $1||\overline{wT}$ problem. Moreover, WMDD is reported to outperform ATC on the $1||\overline{wT}$ problem if due dates are tight.

In summary, SPT is clearly the best dispatching rule to minimise mean flow time in simple single stage problems, while WMDD, ATC and COVERT seem to be the most promising rules for the minimisation of mean weighted tardiness. In addition, CR represents a popular rule for the latter objective in the case that jobs have unit weights, and is also commonly used as a tie breaking rule.

3.2 Problems with Sequence-Dependent Setup Times [1]

When setup times are sequence-dependent, the use of a dispatching rule that does not account for setup times is likely to cause unnecessarily frequent and lengthy setups, thereby wasting

[1]This section is published in similar form in C. W. Pickardt and J. Branke (2012), Setup-oriented dispatching rules — a survey, *International Journal of Production Research* 50(20), 5823–5842.

a large amount of the processing capacity of the work centre on changeovers. In dynamic problems, this can lead to the buildup of large queues, implying long waiting times, and even to an unstable system, i.e. the number of jobs in the shop grows indefinitely, if the long-term processing rate of the work centre is allowed to fall below the job arrival rate. Hence, dispatching rules for problems with sequence-dependent setup times have to be setup-oriented and take setup times explicitly into account in order to be effective.

Setup-oriented dispatching rules from the literature can be classified into three categories: Purely setup-oriented, composite and family-based rules, where family-based rules can be further subdivided into exhaustive and truncated rules, as illustrated in Figure 3.2. Purely setup-oriented rules are primarily concerned with the minimisation of setup times. A popular example is the Shortest Setup Time (SST) rule, originally proposed by Gavett (1965), which assigns highest priority to the job requiring the least setup time given the current machine setup. However, it will be generally insufficient to focus solely on setup time reductions if the objective is to minimise mean flow time or mean weighted tardiness. Then, the importance of minimising setup times has to be traded off with potentially conflicting objectives such as giving priority to urgent jobs, which motivates the design of composite or family-based rules. Composite setup-oriented rules employ a single priority index that incorporates a term for setup avoidance, which allows them to trade off different criteria simultaneously, whereas family-based rules follow a hierarchical approach and avoid setup times by combining the processing of jobs that belong to the same family.

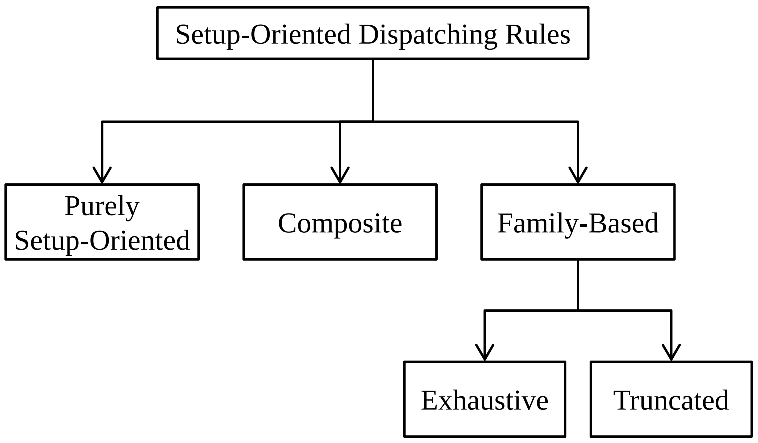


Figure 3.2: Classification of setup-oriented dispatching rules.

Mosier et al. (1984) model the decision making of family-based rules as a multi-stage process involving the consideration of three interrelated decisions, namely:

1. When to switch to another family, i.e. when to conduct a setup.
2. Which family to select, if a switch is made.
3. Which job to choose from the selected family.

The third decision does not affect setup times and thus can be properly resolved by any conventional dispatching rule. Regarding the first decision, family-based rules are distinguished as exhaustive or truncated. Exhaustive rules never switch to another family as long as jobs of the current family are waiting to be processed. While this strategy effectively reduces the frequency of setups, it also introduces a certain amount of inflexibility that can lead to poor results in some situations. Truncated rules address this inflexibility by permitting changeovers prior to the exhaustion of the current family, provided that a certain (problem-dependent) criterion is met.

3.2.1 Minimising Mean Flow Time

Unlike the corresponding problem without sequence-dependent setup times, the $1|s_{ef}|\overline{C}$ problem is NP-hard (Potts and Kovalyov, 2000), which precludes the availability of an optimal dispatching rule. However, an optimal solution to this problem is characterised by scheduling jobs within each setup family in non-decreasing order of their processing times (Pinedo, 2008, Chapter 4), which indicates that an effective rule needs to integrate the principle of setup avoidance with SPT in some way.

Several composite rules that combine SST and SPT have been developed in the literature. Wilbrecht and Prescott (1969) propose a simple additive combination of the two rules, which is commonly referred to as the Shortest Setup and Processing Time (SSPT) rule. Its priority index is given by

$$I_j^{\text{SSPT}} = -(s_{ef_j} + p_j), \tag{3.15}$$

where s_{ef_j} denotes the time required to change over from the previously processed family e to the family f_j of job j. Kochhar and Morris (1987) present a similar rule, called Shortest Normalised Setup and Processing Time (SNSPT) in this work, with a priority index of

$$I_j^{\text{SNSPT}} = -\left(X\frac{s_{ef_j}}{\overline{s}} + Y\frac{p_j}{\overline{p}}\right). \tag{3.16}$$

In contrast to SSPT, the SNSPT rule normalises the setup and processing time by dividing them by their respective queue averages $\overline{s}$ and $\overline{p}$, and weights the two terms before summing them up. Given that no guidelines are provided for setting the weighting factors, it appears reasonable to set $X = Y$. Another composite rule is proposed by Raman, Rachamadugu, and Talbot (1989), which is referred to as the Shortest Average Setup and Processing Time (SASPT) rule herein and computes the priority index of a job as

$$I_j^{\text{SASPT}} = -\left(\frac{s_{ef_j}}{n_{f_j}^{\text{Q}}} + p_j\right), \tag{3.17}$$

where $n_{f_j}^{\text{Q}}$ indicates the number of jobs in the queue that belong to the same family as job j. The underlying assumption of this rule is that the machine processes all waiting jobs of a family consecutively so that the setup time s_{ef_j} is spread over $n_{f_j}^{\text{Q}}$ jobs. The ratio of the two attributes then gives the average setup time per job of that family. Unfortunately, there appears to be a lack of studies on the relative performance of these composite rules or how they compare with promising family-based rules.

Since it is optimal to schedule jobs within each family in non-decreasing order of their processing times, SPT should be generally used for the job selection within family-based rules for single stage problems with the objective to minimise mean flow time. Hence, the discussion of family-based rules for the minimisation of mean flow time can focus on the design of family selection rules.

The employment of an effective family selection rule is crucial as they largely determine the overall performance of family-based rules. This is especially the case for exhaustive rules, which are committed to processing all jobs of a family once that family has been chosen, and therefore cannot easily rectify a poor decision taken at the family selection level. Consequently, the more promising family selection rules from the literature make use of information on the entire set of jobs in the queue. Arzi and Raviv (1998) propose a simple rule that determines the priority of a family by dividing its required setup time by the number of jobs in the queue that belong to it. This rule is called Family Shortest Average Setup Time (FSAST) herein, and its priority index I_f for family f is defined as

$$I_f^{\text{FSAST}} = -\left(\frac{s_{ef}}{n_f^{\text{Q}}}\right), \tag{3.18}$$

where n_f^{Q} indicates the number of jobs of family f in the queue. Russell and Philipoom (1991) develop a family selection rule, referred to as Family Shortest Average Setup and Processing Time (FSASPT), that additionally takes into account the processing times of the waiting jobs of a family. Its priority index is given by

$$I_f^{\text{FSASPT}} = -\frac{1}{n_f^{\text{Q}}}\left(s_{ef} + \sum_{j \in N_f^{\text{Q}}} p_j\right), \tag{3.19}$$

where N_f^{Q} denotes the set of queueing jobs that belong to family f. FSASPT thus prioritises the family with the highest job processing rate, i.e. the largest number of jobs that can be completed per unit of time, given the current queue content.

By definition, exhaustive family-based rules only change the setup when there is no job left in the queue that can be processed with the current setup, and are therefore fully defined by the two rules they apply to determine the next family, and the next job within a family. Nomden et al. (2008) test several exhaustive rules, based on different family selection rules, including FSAST and FSASPT, and the SPT job selection rule, on the $1|r_j, s_{ef}|\overline{C}$ problem. They conclude that FSASPT is very effective across various settings for the number of families, utilisation and the *s/p ratio*, the ratio of average setup time to average processing time, though it is outperformed by FSAST when utilisation is high and the s/p ratio large. These findings are confirmed by Van der Zee (2010), who also applies the exhaustive FSAST and FSASPT rules with SPT job selection to the $1|r_j, s_{ef}|\overline{C}$ problem. In addition, Van der Zee finds FSAST to perform particularly well if the variance of setup times is large (relative to the variance of processing times). These results can be explained by the fact that FSAST concentrates on the minimisation of setup times, which becomes most important when setups take a long time and work centres are highly utilised. Moreover, a small variance of processing times limits the potential of FSASPT to select a family with a small average processing time as they will all be similar.

On the other hand, a large variance of job processing times can lead to situations where there are many jobs of another family in the queue with substantially shorter processing times than the remaining jobs of the current family. In such a case, a truncated rule that switches to the other family although there are still jobs of the current family left could minimise mean flow time more effectively. Mosier et al. (1984) design a truncated rule that permits a changeover after the completion of a certain number of jobs, subsequently referred to as a sequential batch. The size of this sequential batch (in terms of number of jobs) is determined at the moment a new family is selected and is simply equal to the number of jobs of that family waiting in the queue at that point. However, with new jobs arriving over time, the completed jobs do not necessarily have to correspond to those that were originally in the queue, but can include some that arrived more recently.

Van der Zee (2010) combines this rule with FSASPT and calls it the Gated FSASPT (FSASPT_GA) rule, and proposes two alternative truncated rules, the Adaptive FSASPT (FSASPT_AD) and the Hybrid FSASPT (FSASPT_HY) rule. The FSASPT_AD rule sorts the queueing jobs of every family in non-decreasing order of their processing times and applies the FSASPT rule to every partial sequential batch of every family, including those that consist only of the job with the shortest processing time. The family related to the batch with the highest priority is then selected for processing, and the number of jobs to be processed before the rule is allowed to truncate is determined by the size of this batch. However, to prevent the system from becoming unstable, only sequential batches that are larger than a lower bound, which is based on the required setup time and utilisation level, are eligible to be chosen. The FSASPT_HY rule extends FSASPT_AD by successively increasing the batch size of the chosen family until its priority becomes lower than that of a partial batch of another family. In case that even the largest batch, containing all queueing jobs from the family, has a higher priority than any batch of any other family, FSASPT_HY temporarily switches to an exhaustive mode. Van der Zee tests the three truncated rules on the $1|r_j, s_{ef}|\overline{C}$ problem against the exhaustive FSASPT and FSAST rules (all with SPT job selection) and concludes that FSASPT_AD yields the lowest mean flow time for moderate to low utilisation levels and s/p ratios, especially if the variances of processing and setup times are both large. In contrast, FSASPT_AD and FSASPT_GA are shown to be significantly worse than the exhaustive rules for high utilisation levels or s/p ratios, and a small variance of processing times, while FSASPT_HY is reported to hold an intermediate position.

A few researchers have developed family selection rules that coordinate the setups of parallel machines. Jensen et al. (1998) present two rules that, in determining the priority of a family f, first count the number of machines m_f that are currently set up for it. The priority of a family is then based on a subset of the n_f^{Q} queueing jobs of that family, namely the n_f^{Q}/m_f jobs with the lowest priorities. Hence, the more machines are set up for a given family the lower its priority will be, thereby discouraging machines to work on the same family simultaneously. Similarly, Arzi and Raviv (1998) drive parallel machines to work on different job families with a decision logic that, upon its arrival, assigns every job to a machine that is currently processing its family, provided that such a machine exists and there is no machine running idle at that time. They further improve their rule by grouping together families for which switching between setups takes a relatively short time. The rule then restrains machines from loading jobs that belong to a family group already in process on another machine. These more advanced strategies generally reduce the time spent on changeovers, and thus can achieve a lower mean flow time.

3.2.2 Minimising Mean Weighted Tardiness

Since the WMDD, COVERT, ATC and CR rules effectively minimise mean (weighted) tardiness in single stage problems without sequence-dependent setups, their integration with a mechanism that minimises setup times appears to be a promising way to the design of effective setup-oriented rules.

A simple approach to the development of setup-oriented variants of these rules is taken by Raman, Rachamadugu, and Talbot (1989), who modify MDD and ATC by substituting every processing time term in their priority indices with the sum of (current) setup and processing time. The slack within these composite rules is therefore altered to

$$S_j = d_j - t - p_j - s_{ef_j}. \tag{3.20}$$

As Lee, Bhaskaran, and Pinedo (1997) point out, the problem with this slack definition is that it leads due date-oriented rules to give a higher priority to jobs that require a longer setup time since they tend to have less slack according to Equation (3.20). Such a rule then acts in a way that is counterproductive to the minimisation of setup times and is likely to perform very poorly in consequence.

This observation motivates Lee, Bhaskaran, and Pinedo to propose an alternative extension of ATC that considers slack and setup time separately. The priority index of this rule, called ATC with Setups (ATCS), is given by

$$I_j^{\text{ATCS}} = \frac{w_j}{p_j} \exp\left(-\frac{\max(d_j - t - p_j, 0)}{k_1 \overline{p}}\right) \exp\left(-\frac{s_{ef_j}}{k_2 \overline{s}}\right). \tag{3.21}$$

A disadvantage of this separation is that it comes at the cost of a second scaling parameter k_2, in addition to the one from the original ATC rule, which is denoted by k_1 in Equation (3.21). Lee, Bhaskaran, and Pinedo establish laws for setting the two parameters experimentally and test ATCS with thusly tuned parameter values on the $1|s_{ef}|\overline{wT}$ problem against the setup-oriented ATC rule from Raman, Rachamadugu, and Talbot (1989). The ATCS rule is shown to be clearly better, irrespective of the due date setting or the s/p ratio. In a follow-up paper, Lee and Pinedo (1997) apply the ATCS rule to the $Pm|s_{ef}|\overline{wT}$ problem and find that it achieves close to optimal solutions when due dates are tight and the s/p ratio is not too high. However, they derive and make use of different laws to tune the parameters, which indicates that a good parameter setting is highly problem-dependent.

Chiang and Fu (2009) design a composite setup-oriented rule that is based on the CR rule. Their Enhanced CR (ECR) rule evaluates the effect of prioritising one job on the urgency of all other jobs in the queue before making a decision on which job to select. For single stage problems the priority index of ECR is computed as

$$I_j^{\text{ECR}} = -\sum_{y \in N^{\text{Q}}, y \neq j} urg(p_y, d_y - t - s_{ef_j} - p_j - s_{f_j g_y}), \tag{3.22}$$

where N^{Q} denotes the set of jobs in the queue. The two arguments of the urgency function urg correspond to the numerator and denominator of the critical ratio of a job, namely its remaining processing time and remaining allowance (see Section 3.1.2). The latter is determined under the assumption that job j is processed first and the current allowance of job y therefore decreases by the setup and processing time required for the completion of job j, and the time

$s_{f_j g_y}$ it takes to subsequently switch from the setup for family f_j to that needed for family g_y of job y. Hence, a job that involves extensive processing and setup times tends to receive a low priority with ECR as it largely increases the urgency, a non-increasing function of remaining allowance, of all other waiting jobs. Moreover, the ECR rule has an extra capability of minimising setup times as it does not only consider the time required to get to a certain setup, but also the time to leave the setup afterwards to process another job from the queue. On the other hand, the performance of ECR is strongly influenced by the used urgency function, where the one proposed by Chiang and Fu ignores job weights and is primarily designed to minimise the proportion of tardy jobs. Thus, the applicability of the ECR rule is restricted by the absence of a suitable urgency function for the minimisation of mean weighted tardiness.

Regarding family-based rules, there seems to be a general lack of sophisticated family selection rules for due date-related objectives. Exhaustive rules suggested in the literature typically use a common dispatching rule to choose the next family, e.g. the family that contains the job with the earliest (modified) due date or the smallest critical ratio (see, e.g. Mahmoodi et al., 1990; Russell and Philipoom, 1991; Mahmoodi et al., 1992; Ruben et al., 1993; Kim and Bobrowski, 1994; Frazier, 1996; Vinod and Sridharan, 2008). This is likely to result in a poor performance as the urgency (and weight) of all other queueing jobs of a family are completely ignored in the decision making. On the other hand, the suitability of exhaustive rules for the minimisation of mean (weighted) tardiness is generally questionable as they make jobs, which approach their respective due dates, wait for the completion of less urgent jobs, simply because the machine is set up for the latter. Instead, it appears more sensible to resort to truncated family-based rules that allow for a changeover to another family whenever a job becomes time-critical and there is no urgency to process the jobs of the current family. However, none of the due date-oriented truncated rules developed so far seems to be particularly effective in minimising mean (weighted) tardiness (see, e.g. Mahmoodi and Dooley, 1991; Russell and Philipoom, 1991; Wemmerlöv and Vakharia, 1991).

In summary, rules that combine setup avoidance with SPT are most effective in reducing mean flow time in single stage problems with sequence-dependent setup times, and the group of promising rules comprises both composite and family-based rules. In contrast, there appears to be a lack of effective family-based rules for the objective of minimising mean (weighted) tardiness, leaving the composite ATCS rule as the benchmark.

3.3 Batch Processing Problems with Incompatible Families

Dispatching rules for work centres with batch processing capabilities have to select a batch of jobs to be processed next, which implies the formation of batches from the set of waiting jobs. In the case of incompatible families, when only jobs of the same type, i.e. of equivalent size and with equivalent processing requirements, can be batched together, batch formation can be handled as a separate subproblem. More specifically, it is sufficient to form the best batch of each family, containing the jobs with the highest priorities, and then choose one of these batches for processing. While a common dispatching rule can be used to prioritise the jobs within each family, the comparison of batches requires for a more sophisticated dispatching rule that takes into account the content of each batch. In particular, dispatching rules need to consider the size of a batch, in terms of the number of jobs it contains, relative to the maximum number of jobs the machine could accommodate for that family. Otherwise, there is the risk of an excessive waste of the capacity of the work centre by the repeated processing of near-empty batches, which, as in the case of sequence-dependent setup times, can lead to an unstable system in dynamic problems.

3.3.1 Minimising Mean Flow Time

Uzsoy (1995) shows that, for single stage batch processing problems with incompatible families and a regular objective such as the minimisation of mean flow time, it is optimal to make batches always as full as possible (see also Webster and Baker, 1995). In other words, unless there is a penalty associated with the early completion of jobs, nothing can be gained by leaving some of the capacity of a machine unused. This finding and the analogy between the $Pm|B_f|\overline{C}$ and the $Pm||\overline{wC}$ problem (see also Dobson and Nambimadom, 2001), allows Uzsoy to derive a dispatching rule that solves the $1|B_f|\overline{C}$ problem optimally and yields good solutions to the $Pm|B_f|\overline{C}$ problem. This rule, called Batch SPT (BSPT) herein, forms the largest possible batch of each family using any of the available jobs and then, analogue to the WSPT rule, selects the batch b with the largest ratio of its size (its weight) to its processing time. Thus, the priority index of BSPT is defined as

$$I_b^{\text{BSPT}} = \frac{n_b}{p_b}, \tag{3.23}$$

where p_b indicates the processing time of batch b and n_b the number of jobs it contains.

Although the BSPT rule is very effective for the $1|B_f|\overline{C}$ and the $Pm|B_f|\overline{C}$ problem, it does not guarantee stability for the corresponding dynamic problems. To illustrate, consider a single machine problem with dynamically arriving jobs that belong to one of the two families e and f. The machine can process up to four jobs of family e in a batch, requiring four units of time, i.e. $B_e = 4$ and $p_e = 4$, or up to four jobs of family f in 20 time units, i.e. $B_f = 4$ and $p_f = 20$. It follows that a batch containing only one job of family e has a priority index of 1/4 with BSPT, which is larger than 4/20, the priority index of a full batch of family f. If the arrival rate of family e exceeds its processing rate based on one-job batches, i.e. one job per four units of time, there will always be jobs of family e in the queue in the long term. Then, BSPT will never choose a batch of family f for processing, resulting in the system to become unstable.

One way to ensure system stability is by the application of a threshold policy that only allows the start of a batch b if its size n_b is equal to or larger than a given threshold value L. This policy, which originates from a paper by Neuts (1967), is commonly referred to as the Minimum Batch Size (MBS) rule in the literature. Deb and Serfozo (1973) show that MBS with the optimal L-value yields the long-term minimum average number of jobs in the queue, and hence the minimum mean flow time, for the special case of the $1|r_j, B_f|\overline{C}$ problem with one family and a Poisson arrival time distribution. For problems with multiple families with different arrival rates and processing times, Akçalı et al. (2000) suggest to use family-specific threshold values that are increasing in the difference between the processing time and the mean interarrival time of a family. The dispatching rule is then restrained to select only among those batches that are large enough to comply with the family-specific threshold. Duenyas and Neale (1997) design a dispatching rule that combines BSPT and MBS in a similar manner. This rule, called MBSBSPT in this work, sorts families in non-increasing order of their BSPT index B_f/p_f and determines the family with the highest priority that can form a full batch. This family and all families that have a higher BSPT index and can form a batch that is at least as large as their MBS threshold are then eligible for processing, where the family-specific thresholds depend on the family that can form the full batch and are based on a sufficient condition for a stable system. If none of the families can form a full batch, then all families are eligible for processing. The largest possible batch is formed for each family and the one with the highest BSPT index is selected. The batch is started immediately unless there is a benefit in leaving the machine idle, which is estimated using the arrival rates for the individual families. Duenyas and Neale apply their MBSBSPT rule to the $1|r_j, B_f|\overline{C}$ problem with different utilisation levels, number of families, family mixes and machine capacities. They find that their rule consistently yields a mean flow time that is very close to a lower bound on the optimum solution obtained by stochastic dynamic programming.

The main disadvantage of MBS-based rules in general, and MBSBSPT specifically, is that their effectiveness depends largely on adequately set (family-specific) threshold values (Glassey and Weng, 1991; Van der Zee et al., 1997). In the literature, these are typically determined using the arrival rates of the individual families (see, e.g. Duenyas and Neale, 1997; Akçalı et al., 2000). However, arrival rates are generally unknown, and can only be estimated from historic job arrivals. Furthermore, subtle changes to arrival rates over time are very difficult to detect, making it difficult to adjust the thresholds in time, which could impair the performance of an MBS-based rule to some extent.

Alternatively, system instability can also be avoided by giving priority to batches that use more of the available machine capacity. A simple implementation of this strategy is a rule, referred to as the Full Batch (FB) rule herein, that distinguishes only between full and partial batches and only allows for the selection of a partial batch if no full batch can be formed.

A slightly more sophisticated variant, called Most Complete Batch (MCB) is this work, assesses the degree of fullness in the prioritisation of batches, i.e. its priority index is given by

$$I_b^{\mathrm{MCB}} = \frac{n_b}{B_{f_b}}, \tag{3.24}$$

where B_{f_b} denotes the maximum size of a batch of the family f_b that batch b belongs to. Both MCB and FB guarantee a stable system and could be combined with BSPT, e.g. by using the latter as a tie-breaking rule, to create promising rules for the minimisation of mean flow time of single stage batch processing problems.

3.3.2 Minimising Mean Weighted Tardiness

The minimisation of mean weighted tardiness represents another regular objective, for which there is no incentive to form partial batches that could be filled with some of the waiting jobs instead (Uzsoy, 1995; Webster and Baker, 1995). Hence, all batches apart from possibly the last batch of each family are full batches in an optimal solution to the $1|B_f|\overline{wT}$ or the $Pm|B_f|\overline{wT}$ problem. Moreover, due to the analogy between the $1|B_f|\overline{wC}$ and the $1||\overline{wC}$ problem (Uzsoy, 1995; Dobson and Nambimadom, 2001), the extreme case of the $1|B_f|\overline{wT}$ problem, in which it is impossible to complete any job on time, can be solved optimally by a batch processing version of the WSPT rule (see also Section 3.1.2). This rule, developed by Uzsoy (1995) and referred to as the Batch WSPT (BWSPT) rule in this work, forms the largest possible batch of each family by filling them with jobs in non-increasing order of their weights. The priority index of a formed batch is then given by

$$I_b^{\mathrm{BWSPT}} = \frac{w_b}{p_b}, \tag{3.25}$$

where the weight w_b of batch b equals the sum of the weights of the jobs it contains, and the batch with the highest BWSPT index is selected for processing.

In addition, Uzsoy (1995) derives a batch processing version of the EDD rule, called Batch EDD (BEDD) herein, that solves the $1|B_f|T_{\max}$ problem optimally, and thus also the $1|B_f|\overline{wT}$ problem in the other extreme case when it is possible to finish all jobs on time. BEDD fills batches with jobs in non-decreasing order of their due dates and, analogue to EDD, sequences batches in non-decreasing order of their due dates. Hence, the priority index of BEDD is

$$I_b^{\mathrm{BEDD}} = -d_b, \tag{3.26}$$

where the due date d_b of batch b is determined by the earliest of the due dates of the jobs it contains.

The fact that BWSPT and BEDD yield optimal solutions to the $1|B_f|\overline{wT}$ problem, and close to optimal solutions to the $Pm|B_f|\overline{wT}$ problem (Uzsoy, 1995), in opposed conditions of extremely tight and loose due dates indicates that a generally effective rule should somehow integrate the two rules. In particular, with regard to the formation of batches, such a rule has to trade off between the prioritisation of a job with a larger weight and more relaxed due date, and giving priority to a more urgent job with a smaller weight. In many ways, the considerations to be made resemble those involved in solving serial processing problems for the minimum mean weighted tardiness, and the extension of effective rules such as WMDD, COVERT or ATC to batch processing problems thus appears promising.

Several ATC-based rules have been developed for batch processing problems. Mehta and Uzsoy (1998) design the Batch ATC (BATC) rule to address the $1|B_f|\overline{T}$ problem, for which the formation of batches is easy as it is optimal to include jobs in non-decreasing order of their due dates (Mehta and Uzsoy, 1998). The priority index of BATC is defined as

$$I_b^{\text{BATC}} = \frac{1}{p_b} \exp\left(-\frac{\sum_{j \in N_b} \max(d_j - t - p_j, 0)}{k\overline{p}} \right), \tag{3.27}$$

where N_b denotes the set of jobs contained in batch b and $\overline{p}$ now refers to the average processing time of the formed batches. Similarly, Mason et al. (2002) formulate a batch processing extension of the ATCS rule that accounts for the weight as well as the relative fullness of a batch. For problems without sequence-dependent setup times the priority index of this rule can be simplified to

$$I_b^{\text{BATCS}} = \frac{w_b}{p_b} \exp\left(-\frac{\max(d_b - t - p_b, 0)}{k\overline{p}} \right) \frac{1}{B_{f_b}}. \tag{3.28}$$

Regarding the formation of batches, Mason et al. propose to make a batch for every feasible combination of available jobs, including partial batches, which is rather inefficient. Perez et al. (2005) present another batch processing adaptation of the ATC rule, which forms only one batch per family by applying ATC to select the jobs to be included. The priority index of a batch is then simply given by the sum of priority indices of jobs it contains, i.e.

$$I_b^{\text{BATC}^{\text{x}}} = \sum_{j \in N_b} \frac{w_j}{p_j} \exp\left(-\frac{\max(d_j - t - p_j, 0)}{k\overline{p}} \right), \tag{3.29}$$

where $\overline{p}$ refers to the average processing time of all jobs in the queue, as in the original ATC rule. Perez et al. test BATC$^{\text{x}}$ against BEDD on the $1|B_f|\overline{wT}$ problem with different values of number of families, number of jobs per family and machine capacity, as well as different due date settings. Their results show BATC$^{\text{x}}$ to outperform BEDD across all experimental conditions, which demonstrates the potential of ATC-based rules for batch processing problems.

Despite the lack of studies on the relative performance of the different BATC formulations, their effectiveness can be gauged by a closer examination of their respective priority indices. First of all, a drawback of the BATC rule is that it ignores the size (or weight) of a batch.

Clearly, the (weighted) tardiness penalty associated with a unit of time delay of a batch that is entirely composed of tardy jobs is equal to the number (or sum of weights) of jobs the batch contains. Hence, replacing the 1 in the numerator of the first ratio in Equation (3.27) by w_b (which is equal to n_b in the case of unit job weights) is likely to make the rule more effective. The main difference between BATC, BATCS and BATC$^{\text{x}}$, however, is the way in which they estimate the cost of delaying a batch for another unit of time that is not entirely composed of tardy jobs. The BATCS rule is a little simplistic in this respect as it considers only the slack of the most urgent job in the batch, ignoring the slack of all other jobs. The determination of the delay cost of a batch within the BATC rule is also flawed, as it mixes the estimated delay costs of the individual jobs and discounts the total cost for each job with positive slack. In consequence, a job with excessive slack may offset the delay cost of other jobs, even if these are already late and would increase weighted tardiness for certain if the batch is further delayed. Clearly, this can result in very poor scheduling decisions. BATC$^{\text{x}}$, on the contrary, discounts the weight of each job for its slack and then adds up the discounted weights, which seems a more appropriate way to estimate the delay cost of a batch. Hence, the BATC$^{\text{x}}$ rule is likely to be the most effective of the three rules.

Another question is whether or not to incorporate the extra factor of the BATCS rule, i.e. $1/B_{f_b}$, which only has an effect if the maximum batch size differs across families. An argument against its inclusion is the fact that BATC$^{\text{x}}$ in its original form reverts to the BWSPT rule for extremely tight due dates, which can be regarded as desirable, given the optimality of BWSPT for the $1|B_f|\overline{wT}$ problem under these conditions. On the other hand, as discussed in the previous section, a rule that is optimal for a static batch processing machine problem can still lead to an unstable system when applied to the corresponding dynamic problem. The extra factor of BATCS decreases the probability of instability by relating the estimated delay cost of a batch, which is a non-decreasing function of its size, to the maximum batch size, thereby increasing the priority of fuller batches. However, the factor does not guarantee stability, and an adequate integration of the original BATC$^{\text{x}}$ rule with the FB or MCB rule may actually be more effective.

Kim et al. (2001) develop a rule, named PUCH (Processing Urgency Classification Heuristic), that integrates the CR rule with MBS. PUCH first applies the CR rule to determine the most critical job in the queue. If the critical ratio of this job is less than or equal to a predefined threshold value u_1, it is regarded as very urgent and processed immediately as part of a batch of maximum possible size. If its critical ratio is greater than u_1, but less than another predefined threshold value u_2, the job is considered to be moderately urgent. Then, PUCH assesses for every moderately urgent job, in non-decreasing order of their critical ratios, whether a batch can be formed of its family that is large enough to satisfy the family-specific MBS threshold, and starts the largest possible batch of that family, if that is the case. Otherwise, if none of the jobs in the queue is even moderately urgent or none of those families with moderately urgent jobs can form a large enough batch, PUCH simply starts a batch of the family with the most number of jobs in the queue that can satisfy the MBS threshold, or leaves the machine idle if no such family exists. Regarding the formation of batches, PUCH includes jobs in non-decreasing order of their slack, which for single stage batch processing problems is the same as including jobs in non-decreasing order of their due dates, as the (remaining) processing times of jobs of the same family are all equal.

The key idea of the PUCH rule is to allow urgent jobs to violate minimum batch size requirements in order to finish before their due date. An important issue with respect to the application of PUCH thus concerns the definition of 'urgent', i.e. the selection of an adequate u_1-value. Clearly, if u_1 is set smaller than 1, PUCH starts to expedite jobs only after it is certain that they cannot be completed on time, which defeats the intention of the rule. On the other hand, a very large u_1-value effectively corresponds to removing the MBS rule, which is likely to cause an inefficient operation of the batch processing work centre, thereby making more jobs time-critical in turn. Using simulation, Kim et al. find that setting $u_1 = 2.5$ and $u_2 = 4$ works well for their problem. However, in general, the best values for u_1 and u_2 can be expected to depend not only on the problem, but also on the values chosen for the MBS thresholds and vice versa. The various parameters and the previously discussed challenges associated with the use of MBS-based rules makes the application of PUCH rather difficult. Another drawback is that the rule does not account for unequal job weights in its current form.

In summary, BSPT is the best rule to minimise mean flow time in static single stage batch processing problems, while BATC^x seems to be the most promising rule for the minimisation of mean weighted tardiness. However, both rules can cause instability in dynamic problems, for which they should be combined with rules such as FB or MCB.

3.4 Multi-Stage Problems

In multi-stage problems, jobs have to visit several work centres in a certain order, which implies that a scheduling decision taken at one work centre generally affects the set of jobs, and therefore the decision options, available at another work centre at a later point in time. This interdependency of scheduling decisions makes the development of effective dispatching rules for multi-stage problems very difficult. The challenge is to design rules that operate on a locally restricted horizon and, at the same time, result in a good global performance.

3.4.1 Minimising Mean Flow Time

Since SPT is the best rule for simple single stage problems with the objective to minimise mean flow time, its adaptation provides a natural starting point in the design of effective rules for the corresponding multi-stage problems.

There are several obvious ways to adapt SPT to multi-stage problems. One possibility is to base the priority of a job only on the processing time of its imminent operation on the considered work centre, thereby ignoring all subsequent operations of the respective job. The priority index of this rule, commonly identified by the name of its single stage relative, is thus

$$I_j^{\text{SPT}} = -p_{j,i}, \tag{3.30}$$

where $p_{j,i}$ indicates the processing time of the imminent operation i of job j. Another sensible adaptation is the Least Work Remaining (LWKR) rule, which prioritises jobs according to the processing times of all remaining operations. Its priority index is given by

$$I_j^{\text{LWKR}} = -\sum_{o=i}^{l_j} p_{j,o}, \tag{3.31}$$

where $p_{j,o}$ denotes the processing time of operation o of job j, and l_j the number of operations of job j. Hence, while SPT focusses on short operations and on minimising the local mean flow time at each individual work centre, LWKR assumes a more global perspective and aims to quickly push through short jobs.

Although both rules have their intuitive appeal, the general conclusion in the literature is that SPT is the more effective rule for the minimisation of mean flow time, both in job shops (Blackstone et al., 1982; Baker and Kanet, 1983; Haupt, 1989) and flow shops (Hunsucker and Shah, 1994; Brah and Wheeler, 1998; El-Bouri et al., 2008). An exception is the study by Rajendran and Alicke (2007), who examine the $Fm||\overline{C}$ problem with unbalanced workloads and find that LWKR gains considerably on SPT with an increasing number of processing stages until it starts to outperform the latter rule. However, this result does not seem to carry over to other problems, not even to flow shop problems with dynamic job arrivals and/or balanced workloads.

The drawback of conventional dispatching rules such as SPT and LWKR is that their information horizon is restricted to jobs currently queueing at the considered work centre. Hence, their decision making is myopic in that interactions with other work centres are completely ignored. Clearly, such a local horizon can easily lead to decisions that are bad with respect to the global performance of the shop, which motivates the design of dispatching rules that use some more global information.

Consideration of Upstream Jobs

Gere (1966) proposes to provide dispatching rules with information on jobs that are expected to arrive from upstream work centres in the near future. This information allows a dispatching rule to *look ahead*, i.e. to assess the effect a particular decision has on future conditions at the local work centre. Moreover, these look-ahead dispatching rules can make a decision to leave a machine idle in anticipation of a job with high priority. Waiting for a future job, however, always comes at the expense of delaying jobs that are already in the queue, which has to be traded off against the benefit of being able to process the future job without delay.

To illustrate this trade-off, consider a simple single machine problem with the objective of minimising mean flow time. Assume that there are n^{Q} jobs waiting at the machine, and that another job with higher priority, i.e. shorter processing time, is due to arrive before the highest prioritised job in the queue can be completed (otherwise there is no incentive to wait for the future job). Without loss of generality, let the jobs be indexed in non-decreasing order of their processing times. Then, the mean queueing time of the $n^{\mathrm{Q}} + 1$ jobs that results from waiting for the future job (Job 1) and processing all jobs in order of the (optimal) SPT rule equals

$$\frac{1}{n^{\mathrm{Q}}+1}\left(n^{\mathrm{Q}}(r_1 - t) + \sum_{j=1}^{n^{\mathrm{Q}}}((n^{\mathrm{Q}} + 1 - j)p_j)\right), \tag{3.32}$$

where r_1 denotes the time of arrival of Job 1 to the machine. In contrast, processing the shortest job in the queue, i.e. Job 2, immediately and the remaining jobs in SPT order yields a mean queueing time of

$$\frac{1}{n^{\mathrm{Q}}+1}\left((p_2 - (r_1 - t)) + \sum_{j=1}^{n^{\mathrm{Q}}}((n^{\mathrm{Q}} + 1 - j)p_j) - p_1\right). \tag{3.33}$$

It is therefore better to wait for Job 1 if, and only if, the expression in Equation (3.32) is smaller than that in Equation (3.33) or, rearranging terms, if

$$(n^{\mathrm{Q}} + 1)(r_1 - t) + p_1 < p_2. \tag{3.34}$$

The first term in this inequality represents the penalty of waiting in terms of increase of total flow time, which is shown to be proportional to the number of jobs in the queue. Hence, it is generally not beneficial to wait for a future job at highly utilised work centres, which are characterised by large queues. On the other hand, the inequality indicates that look-ahead rules can reduce the mean flow time over conventional dispatching rules at moderately utilised work centres, especially in situations where a job with a much shorter processing time than any of the waiting jobs is expected to arrive in the very near future.

The potential of look-ahead dispatching rules tends to increase for problems with sequence-dependent setup times. More specifically, their ability to anticipate future jobs that require little or no setup time, given the current machine setup, may save some long changeover operations. Nomden et al. (2008) extend several exhaustive family-based rules so they take future jobs into account in the determination of family priorities and can select a family without jobs in the queue, in which case the machine is left idle until the next job arrival. They test these look-ahead extensions on the $1|r_j, s_{ef}|\overline{C}$ problem and find that the performance of the already effective FSASPT and FSAST rules is further improved, particularly in conditions of a high s/p ratio and low utilisation. Despite the apparent potential of look-ahead rules to reduce mean flow time in problems with sequence-dependent setup times, little work seems to have been done in this area.

In contrast, a number of look-ahead rules have been developed for batch processing problems (with incompatible families), which are the topic of the remaining discussion (see also Van der Zee, 2003). The design of look-ahead rules for batch processing problems is particularly promising as batch operations are typically characterised by long processing times (relative to the interarrival times) and the information on future job arrivals can help to start a batch operation at the right moment. To illustrate, if several jobs are expected to arrive in the near future, allowing for the formation of a substantially larger batch, then to wait for their arrivals is likely to result in a lower mean flow time than to process a smaller batch immediately.

One of the simplest look-ahead rules, which is designed for problems with only one family, is the Dynamic Batching Heuristic (DBH) proposed by Glassey and Weng (1991). DBH considers all jobs that arrive by the time a batch operation would finish if it were started immediately, and up to the point where a full batch could be formed. Its priority index can be expressed as

$$I_j^{\mathrm{DBH}} = \sum_{y \in N^{\mathrm{L}}, r_{y,i} \leq r_{j,i}} (t + p_{j,i} - r_{j,i}) - n^{\mathrm{Q}} \times \max(r_{j,i} - t, 0), \tag{3.35}$$

where N^{L} denotes the set of look-ahead jobs, i.e. the jobs that arrive at the considered work centre in the near future, and $r_{j,i}$ indicates the arrival time of job j to the work centre for its imminent operation. Thus, if j is a look-ahead job then $r_{j,i} > t$ holds, and $r_{j,i} \leq t$ otherwise. The first term of the priority index of DBH measures the total queueing time saved by waiting for the arrival of job j before starting a batch operation, which relates to future jobs that arrive before j and would have to wait until $t + p_{j,i}$ if a batch is processed immediately (in batch processing problems with only one family all batches require the same processing time, i.e. $p_{j,i}$). The second term, on the other hand, represents the additional queueing time that would incur for jobs already in the queue at the time of decision making. The priority index of DBH is therefore an estimate of the net reduction of total waiting time over the horizon of one batch operation processing time, associated with the prioritisation of a look-ahead job j. Moreover, since $I_j^{\mathrm{DBH}} = 0$ for any non-future job j, the machine is made to wait whenever there is a look-ahead job with a positive estimated net reduction of total waiting time.

In several papers (Glassey and Weng, 1991; Fowler et al., 1992; Weng and Leachman, 1993; Van der Zee et al., 1997), DBH is applied to the $1|r_j, B_f|\overline{C}$ problem and shown to outperform MBS with an optimally chosen threshold value across various conditions, where the relative advantage of DBH is found to be larger for a more irregular job arrival pattern, a lower utilisation and a smaller machine capacity. In particular, the fact that DBH shows to outperform MBS in the case that job arrivals follow a Poisson process, for which MBS is optimal under the assumption that information on future jobs is unavailable (see Section 3.3.1), demonstrates the potential of look-ahead rules for batch processing problems.

Fowler et al. (1992) develop a look-ahead rule, named Next Arrival Control Heuristic (NACH), for batch processing problems with multiple families, and extend its applicability to problems with parallel machines in a follow-up paper (Fowler et al., 2000). NACH features a sequential decision logic that involves the computation of up to three priority indices. Whenever a machine completes an operation and there are jobs waiting to be processed, NACH determines first whether it is possible to form a full batch with the jobs in the queue. If this is the case, a full batch of the family f with the highest NACH-Start (NACH-S) index, defined as

$$I_f^{\text{NACH-S}} = -\left(\sum_{g \neq f} n_g^{\text{Q}} p_f \right), \tag{3.36}$$

is formed and processed immediately. Else, if the work centre consists of several machines, of which more than one is running idle at the point of decision making, a partial batch, as large as possible, of the family with the highest NACH-S index is processed immediately. If neither of the first two conditions apply, NACH employs the DBH rule to determine for each family, separately and in isolation of the other families, whether it is beneficial to wait for the next respective job arrival or not. To account for parallel machines, Fowler et al. (2000) extend the priority index of DBH to

$$I_j^{\text{DBH}} = \sum_{y \in N^{\text{L}}, r_{y,i} \leq r_{j,i}} (\min(\tau, t + p_{j,i}) - r_{j,i}) - n^{\text{Q}} \times \max(r_{j,i} - t, 0), \tag{3.37}$$

where τ refers to the time at which the next of the parallel machines completes its current operation. If, for every family, the DBH index of the next arriving job is negative, i.e. the total waiting time is estimated to increase as a result of waiting, then the largest possible batch of the family with the highest NACH-S index is formed and processed immediately. Otherwise, if the DBH rule indicates that, for every family, it is beneficial to wait for the next arrival, the machine is left idle. However, if the result of the evaluation is mixed, i.e. DBH determines that waiting is the preferred decision for some families but not all, then another index, referred to as NACH-Start or Wait (NACH-S/W) herein, is employed to make a decision. This priority index is given by

$$\begin{aligned} I_j^{\text{NACH-S/W}} = -\Bigg(& \sum_{g} n_g^{\text{Q}} \times \max(r_{j,i} - t, 0) + \sum_{g \neq f_{j,i}} n_g^{\text{Q}} p_{j,i} \\ & + \sum_{y \in N^{\text{L}}, y \neq j} \max(0, \max(t, r_{j,i}) + p_{j,i} - r_{y,i}) \Bigg), \end{aligned} \tag{3.38}$$

where $f_{j,i}$ denotes the family of the imminent operation of job j. NACH-S/W, which just like NACH-S measures the increase of total waiting time, is applied to all queueing jobs and all jobs of the set of look-ahead jobs N^{L}, which only includes one job per family in this case, namely the one with the earliest arrival time. If the job with the highest priority, i.e. the one associated with the smallest increase of total waiting time, is a look-ahead job, the machine is left idle. Otherwise, the machine starts processing a batch that contains the job with the highest priority and all other jobs in the queue that belong to the same family. Once the decision is made to leave a machine idle, the NACH rule determines for every new job arrival whether it is now possible to form a full batch, and if not, whether starting a partial batch is, according to DBH, better than to wait for the subsequent arrival of that family. If one of these two conditions is satisfied, NACH starts processing a (full or partial) batch with the newly arrived job, and else leaves the machine idle.

For the $1|r_j, B_f|\overline{C}$ problem with a single family, NACH performs very similar to the DBH rule, in general (Fowler et al., 1992; Weng and Leachman, 1993; Van der Zee et al., 1997), which can be expected as NACH essentially reduces to the application of DBH to decide whether or not to wait for the next arrival of the only family, in this case. On the other hand, Van der Zee et al. (1997) point out some flaws in the decision logic of NACH that are likely to impair its performance for problems with multiple families. First of all, the separate consideration of families by the DBH rule to determine whether or not it is beneficial to wait does not account for interdependencies between jobs of different families. Secondly, NACH only accounts for the next arriving job of each family, thereby ignoring all other jobs that are expected to arrive within the considered time horizon and thus could be negatively affected by a taken decision. Clearly, such a selective use of arrival time information can easily result in poor decisions and reduce the effectiveness of a look-ahead rule.

An alternative to the NACH rule is the Minimum Cost Rate (MCR) rule by Weng and Leachman (1993). Similar to NACH, MCR checks first whether a full batch can be formed with the set of queueing jobs and, if that is the case, starts a full batch of the family f with the highest MCR-Start (MCR-S) index, which is computed as

$$I_f^{\text{MCR-S}} = -\frac{1}{p_f}\left(\left(\max(n_f^{\mathrm{Q}} - B_f, 0) + \sum_{g \neq f} n_g^{\mathrm{Q}}\right)p_f + \sum_{\substack{y \in N^{\mathrm{L}} \\ r_{y,i} \leq t + p_f}} (t + p_f - r_{y,i})\right). \tag{3.39}$$

Otherwise, all jobs (and families) are eligible to be selected, in particular all look-ahead jobs up to and including the one that allows for the formation of a full batch of the respective family. The priority of job j is then determined by the MCR-Start or Wait (MCR-S/W) index, given by

$$I_j^{\text{MCR-S/W}} = \begin{cases} -\dfrac{1}{p_{j,i}}\left(\displaystyle\sum_{g\neq f_{j,i}} n_g^{\text{Q}} p_{j,i} + \sum_{\substack{y\in N^{\text{L}} \\ r_{y,i}\leq t+p_{j,i}}} (t+p_{j,i}-r_{y,i})\right) & \text{if } j \in N^{\text{Q}}, \\ -\dfrac{1}{(r_{j,i}+p_{j,i}-t)}\left(n_{f_{j,i}}^{\text{Q}}(r_{j,i}-t) + \displaystyle\sum_{y\in N_{f_{j,i}}^{\text{L}}, r_{y,i}\leq r_{j,i}} (r_{j,i}-r_{y,i})\right. & \\ \quad + \displaystyle\sum_{\substack{y\in N_{f_{j,i}}^{\text{L}} \\ r_{j,i}<r_{y,i}\leq r_{j,i}+p_{j,i}}} (r_{j,i}+p_{j,i}-r_{y,i}) + \sum_{g\neq f_{j,i}} \left(n_g^{\text{Q}}(r_{j,i}+p_{j,i}-t)\right. & \\ \quad\quad \left.\left. + \displaystyle\sum_{\substack{y\in N_g^{\text{L}} \\ r_{y,i}\leq r_{j,i}+p_{j,i}}} (r_{j,i}+p_{j,i}-r_{y,i})\right)\right) & \text{otherwise,} \end{cases} \tag{3.40}$$

where N_g^{L} refers to the set of look-ahead jobs of family g. Both indices measure the average queue length that would result from prioritising job j (or family f) over a time horizon that extends to the completion of its operation, i.e. $t + p_{j,i}$ for jobs that are started immediately and $r_{j,i} + p_{j,i}$ for jobs that arrive at $r_{j,i}$, which the MCR rule seeks to minimise. More specifically, in each case the expression within the outer brackets measures the total waiting time within the respective time horizon, which is divided by the length of the horizon to give the average queue length. The total waiting time that would result from prioritising a queueing job, given by Equation (3.39) and the first case in Equation (3.40), is composed of two terms, which measure the waiting time for other jobs in the queue that are not part of the batch, and the waiting time for future jobs that arrive before the operation is completed, respectively. On the other hand, the total waiting time associated with the prioritisation of a look-ahead job, i.e. the second case in Equation (3.40), is composed of five terms, which represent queueing jobs of the same family as job j, look-ahead jobs of the same family that arrive before and after j, and queueing and look-ahead jobs of other families, respectively. Depending on whether the job with the highest priority is a queueing or look-ahead job, the machine either starts processing a batch containing that job or waits for the job to arrive.

In contrast to NACH, MCR takes into account all jobs that are expected to arrive within the considered time horizon, not only the next arrival of each family. This could be the reason for that MCR is commonly found to be slightly more effective than NACH for the $1|r_j, B_f|\overline{C}$ problem with multiple families (Weng and Leachman, 1993; Robinson et al., 1995; Van der Zee et al., 1997). On the other hand, results from Weng and Leachman (1993) and Van der Zee et al. (1997) for the $1|r_j, B_f|\overline{C}$ problem with a single family indicate that NACH starts to outperform MCR under higher utilisation levels. More generally, the superiority of one rule over the other seems to depend on a combination of factors such as the number of families, the utilisation and the distributions of interarrival and processing times (Van der Zee et al., 1997). However, a drawback of MCR in comparison with NACH is that it is not designed for work centres with parallel machines.

Van der Zee et al. (1997) develop the Dynamic Job Assignment Heuristic (DJAH), a modification of the MCR rule that is also suitable for parallel machines. The main difference to MCR is that the total waiting time in the indices is divided by the number of jobs in the batch instead of by the length of the time horizon. The DJAH-S index, which is used to determine the family with the highest priority among those that can form a full batch with jobs in the queue (if applicable), is therefore given by

$$\begin{aligned} I_f^{\text{DJAH-S}} = -\frac{1}{B_f}\Bigg(&\bigg(\max(n_f^{\text{Q}} - B_f, 0) + \sum_{g \neq f} n_g^{\text{Q}}\bigg) \times \min(\tau - t, p_f) \\ &+ \sum_{\substack{y \in N^{\text{L}} \\ r_{y,i} \leq \min(\tau, t+p_f)}} (\min(\tau, t + p_f) - r_{y,i})\Bigg), \end{aligned} \tag{3.41}$$

which, apart from denominator (B_f instead of p_f), corresponds to the MCR-S index for work centres that consist of a single machine. Similarly, the DJAH-S/W priority index, which is employed in the case that no full batch can be formed, is an analogue modification of the MCR-S/W index and defined as

$$I_j^{\text{DJAH-S/W}} = \begin{cases} -\dfrac{1}{n_{f_{j,i}}^{\text{Q}}}\Bigg(\displaystyle\sum_{g \neq f_{j,i}} n_g^{\text{Q}} \times \min(\tau - t, p_{j,i}) \\ \qquad + \displaystyle\sum_{\substack{y \in N^{\text{L}} \\ r_{y,i} \leq \min(\tau, t+p_{j,i})}} (\min(\tau, t + p_{j,i}) - r_{y,i})\Bigg) & \text{if } j \in N^{\text{Q}}, \\ -\dfrac{1}{(n_{f_{j,i}}^{\text{Q}} + 1)}\Bigg(n_{f_{j,i}}^{\text{Q}}(r_{j,i} - t) + \displaystyle\sum_{g \neq f_{j,i}} n_g^{\text{Q}}(\min(\tau, r_{j,i} + p_{j,i}) - t) \\ \qquad + \displaystyle\sum_{\substack{y \in N^{\text{L}} \\ r_{y,i} \leq \min(\tau, r_{j,i}+p_{j,i})}} (\min(\tau, r_{j,i} + p_{j,i}) - r_{y,i}) - p_{j,i}\Bigg) & \text{otherwise,} \end{cases} \tag{3.42}$$

In accordance with the MCR rule, N^{L} denotes the set of all look-ahead jobs. However, the DJAH-SW index is only computed for jobs in the queue and the next arrival of each family, which is more in the spirit of the NACH rule and the reason for why some of the terms in the second case in Equation (3.40) can be combined. Moreover, the processing time of a batch in the denominator of MCR is again replaced by its size, which is equal to the number of jobs in the queue belonging to the respective family $n_{f_{j,i}}^{\text{Q}}$, plus the extra look-ahead job if the decision to wait for its arrival is made.

Van der Zee et al. test DJAH against NACH and MCR on the $1|r_j, B_f|\overline{C}$ problem with multiple families and report that it dominates both rules across different utilisation levels, family mixes, machine capacities and processing time distributions. An exception is the case of uniformly distributed interarrival times in combination with a moderate to low utilisation, for which NACH shows to perform best. In addition, their results for the corresponding single family problem indicate that MCR is slightly more effective than DJAH in conditions of a lower utilisation and/or larger machine capacity, while NACH seems to fare marginally better for a high utilisation level. However, the differences found between the mean flow time performance of DJAH, NACH and MCR are generally small in comparison to the advantage they have over rules that do not take future job arrivals into account (Fowler et al., 1992; Weng and Leachman, 1993; Robinson et al., 1995; Van der Zee et al., 1997).

Duenyas and Neale (1997) present a variant of their MBSBSPT rule that bases the decision to leave the machine idle on the time of the next job arrival of every family instead of on arrival rates. This modification is shown to improve the effectiveness of the rule to an extent that it outperforms NACH on the $1|r_j, B_f|\overline{C}$ problem for various settings of number of families, family mix, utilisation, machine capacity, and different interarrival or processing time distributions, except for conditions of a very low utilisation.

An important issue in connection with look-ahead rules is the accuracy and completeness of the arrival time information, as perfect information is generally not available. To illustrate, in an ideal job shop, any work centre can receive jobs from outside the shop, which cannot be predicted according to the assumptions made in this work, or from other work centres within the shop, for which arrival times may be estimated. These estimations can be further precise, e.g. if the job has already started the immediately preceding operation and will join the queue directly afterwards, or imprecise, e.g. if the job still has to undergo several operations on other work centres before it is sent to the considered work centre, in which case its arrival time is subject to future scheduling decisions and random events. Some researchers have studied the effect of inaccurate and incomplete arrival time data on the performance of look-ahead rules in the context of batch processing problems. Their findings can be summarised in that less information-intensive rules turn out to be more robust, and that negative effects of imperfect information can be alleviated by reevaluating decisions to leave a machine idle as soon as new information on future arrivals becomes available (Fowler et al., 1992; Robinson et al., 1995; Van der Zee et al., 1997).

Overall, the above studies demonstrate the benefit of using look-ahead rules for the minimisation of mean flow time, especially at work centres involving sequence-dependent setup times or batch processing. Moreover, look-ahead rules are found to be particularly effective in conditions where a lot can be gained from looking ahead, e.g. setup times are long or job arrivals are irregular, and the penalty for leaving a machine idle is not too high, i.e. utilisation is moderate and queues are small. However, a local reduction of mean flow time does not necessarily translate into a lower global mean flow time, e.g. if the considered work centre is upstream of a bottleneck that largely determines the performance of the shop. Then, supporting an efficient operation of the downstream bottleneck may be more important than local improvements.

Consideration of Downstream Work Centres

While look-ahead rules assess the effect of each possible scheduling decision on the future conditions at the local work centre, it can also be beneficial to assess how a decision affects the future conditions at other, specifically downstream work centres. In particular, unnecessary idle times at highly utilised work centres that are due to a temporarily shortage of jobs may be avoided by ensuring that these work centres are provided with a constant job supply.

A rule that operates in this way is the Processing Time plus Work In Next Queue (PT+WINQ) rule, presented by Holthaus and Rajendran (1997a,b). The priority index of this rule is given by

$$I_j^{\mathrm{PT+WINQ}} = -(p_{j,i} + W_{z_{j,i+1}}), \tag{3.43}$$

where $W_{z_{j,i+1}}$ denotes the current work content, i.e. the sum of operation processing times of jobs in the queue and the remaining processing time of the operation in process, of the work centre $z_{j,i+1}$, required for the subsequent operation of job j. If there is no subsequent operation, i.e. the imminent operation of the job is its last, then $W_{z_{j,i+1}} = 0$. Hence, this component, called WINQ in the following, assigns maximum priority to a job that will leave the system after the completion of its imminent operation. By considering the queue lengths of work centres that jobs visit for their next operation in addition to the processing time of their imminent operation, the PT+WINQ rule is able to trade off the minimisation of the local mean flow time with the importance of balancing the work content of other work centres to avoid idle times due to starvation. As a consequence, PT+WINQ shows to perform significantly better for the $Jm|r_j|\overline{C}$ problem than its two components, the SPT and WINQ rules, and a policy that randomly switches between those two rules, especially in conditions of high utilisation (Holthaus, 1997; Holthaus and Rajendran, 1997a,b; Rajendran and Holthaus, 1999).

Encouraged by these results, Holthaus and Rajendran propose a modification of the PT+WINQ rule in a follow-up paper (Holthaus and Rajendran, 2000). The name of this modification, 2PT+WINQ+NPT, is derived from the formula of the priority index of the rule, which is defined as

$$I_j^{\mathrm{2PT+WINQ+NPT}} = -(2p_{j,i} + W_{z_{j,i+1}} + p_{j,i+1}). \tag{3.44}$$

Thus, the main difference to PT+WINQ is the incorporation of the processing time of the subsequent operation of a job $p_{j,i+1}$, and the idea behind this is to give priority to jobs that can be quickly pushed through their next two operations. Holthaus and Rajendran (2000, 2002) apply 2PT+WINQ+NPT to the $Jm|r_j|\overline{C}$ problem and find that it outperforms both SPT and PT+WINQ, especially when the shop is highly utilised.

In view of the effectiveness of the PT+WINQ and 2PT+WINQ+NPT rules, one might be tempted to extend the information horizon further to include more than just the next operation. However, as Holthaus and Rajendran (2000) point out, the accuracy of the work content term is expected to decrease quickly for additional subsequent operations due to unforeseen future events, e.g. the arrival of a new job, that may happen before the job arrives at the respective work centre as well as dispatching decisions taken in between. Hence, in an ideal job shop, where jobs can arrive at any work centre from any other work centre (or from outside the shop), it does not seem sensible to look ahead for more than one operation. On the other hand, in a flow-dominant job shop, looking only at the next operation is likely to be insufficient. This is illustrated by the other extreme of a flow shop, where all jobs in a queue move on to the same work centre for their next operation. The WINQ component is the same for all jobs in this case, and the PT+WINQ rule thus reverts to the SPT rule. Furthermore, 2PT+WINQ+NPT appears to loose its advantage over SPT for the $Fm|r_j|\overline{C}$ problem, and perform worse in fact (Holthaus and Rajendran, 2002).

Another aspect that is likely to adversely affect the effectiveness of these rules is the workload balance among work centres in the shop. If the shop is perfectly balanced, as it is the case in the above studies, each work centre is equally important regarding the avoidance of idle times. However, in an unbalanced shop, the focus should be on the highly utilised work centres. Then, it is not the next operation, or the work centre on which it is to be performed, that is important but the operation on the next critical work centre and attributes relating to it.

Rajendran and Alicke (2007) develop several dispatching rules for flow shop problems with unbalanced workloads that focus on the bottleneck operations. One of their more effective rules is defined by the priority index

$$I_j^{\text{BLWKR}} = \begin{cases} -\sum_{o=i}^{h} p_{j,o} & \text{if } i \leq h, \\ -\sum_{o=i}^{l_j} p_{j,o} & \text{otherwise,} \end{cases} \tag{3.45}$$

where h is the index of the operation of a job that is to be performed on the bottleneck work centre (h is the same for all jobs in flow shop problems). The rule, referred to as Bottleneck-based LWKR (BLWKR) herein, thus makes a distinction between jobs that still have to undergo processing on the bottleneck and those that have already passed through it. In the first case, the processing times of all operations up to and including the one on the bottleneck determine the priority of a job, while in the second case the LWKR rule is simply followed. The reasoning behind this rule is that jobs that can reach the bottleneck quickly should be prioritised to support an efficient operation of the bottleneck and prevent it from starving. Rajendran and Alicke further present a variant of BLWKR that assigns more weight to the more proximate operations, with its priority index given by

$$I_j^{\text{BLWKR}^{\text{x}}} = \begin{cases} -\sum_{o=i}^{h}(h-i+1)p_{j,o} & \text{if } i \leq h, \\ -\sum_{o=i}^{l_j}(l_j-i+1)p_{j,o} & \text{otherwise.} \end{cases} \tag{3.46}$$

Both rules are tested on the $Fm||\overline{C}$ problem with one clear bottleneck work centre, for which they show a similar performance, outperforming SPT and LWKR. However, BLWKR^{x} appears to become more effective with an increasing number of work centres and when the bottleneck is located more towards the end of the flow shop (Rajendran and Alicke, 2007), which indicates that less attention should be paid to the bottleneck operation if it is farther down the line, with several operations preceding it.

In addition, Rajendran and Alicke investigate how to best adapt the BLWKR rule to problems with several bottleneck work centres. They distinguish between local and global bottlenecks, where local bottlenecks are defined by a workload that is higher than that of most other work centres but lower than that of the global bottleneck(s). Their findings indicate that local bottlenecks should be ignored and are not important in the determination of priorities. In other words, the h within the priority index of BLWKR should always refer to the next operation on a global, not local bottleneck.

El-Bouri et al. (2008) design a rule, called Cooperative Dispatching (CD), which assesses the effect of prioritising a given job on all downstream work centres before making a decision on which job to select. The priority index of CD can be expressed as

$$I_j^{\text{CD}} = -\sum_{o=i}^{l_j}(\hat{R}_{z_{j,o}}(j) \times v_{z_{j,o}}), \tag{3.47}$$

where $\hat{R}_{z_{j,o}}(j)$ represents the estimated increase of total flow time (the regret) at work centre $z_{j,o}$, required for operation o of job j, that results from prioritising job j. More specifically, the flow times refer to the set of jobs queueing at the work centre for which a decision is being made and are calculated under the assumption that, of the set, job j is processed first at every downstream work centre, and that the remaining jobs have all arrived by the time the operation of job j is completed and can be rearranged optimally (using the SPT rule). To further obtain an estimate of the time at which each work centre starts to process job j, which depends on other jobs in between, El-Bouri et al. simulate the shop using the FCFS rule, thereby assuming that the set of jobs headed by job j does not overtake any other job. The regret $\hat{R}_{z_{j,o}}(j)$ is then given by the difference between the total flow time at work centre $z_{j,o}$ that results from the prioritisation of job j, and the minimum total flow time that can be reached by choosing any of the jobs of the set. Hence, for each downstream work centre there is at least one job in the set of queueing jobs that yields a regret of 0. The regret values of the different work centres are then weighted with a factor $v_{z_{j,o}}$, which takes into account the relative workload of a work centre as well as its proximity to the local work centre, before being summed up.

Using weighting factors that give more weight to the more utilised and/or closer work centres, El-Bouri et al. apply CD to the $Fm|r_j|\overline{C}$ problem and find that it is more effective than SPT and LWKR, particularly at higher utilisation levels. However, the application of CD is limited to flow shop problems as the regret calculations are only sensible if all jobs visit the same work centres in the same order. An additional limitation of CD are the rather elaborate computations required for the determination of the flow times that include the use of simulation.

A very different approach to coordinate the interplay between up- and downstream work centres is taken by Holthaus and Ziegler (1997a,b). Their Look Ahead Job Demanding (LAJD) mechanism intervenes in the regular dispatching process once the work content of the queue of a work centre has fallen below a predefined lower bound and there are no jobs in process at other work centres that will join the queue directly afterwards. LAJD then initiates an inquiry to all work centres for jobs that have their next operation on that close-to-starving work centre and chooses the job that fits best to be processed next at the upstream work centre, overwriting the decision making of the dispatching rule. The job regarded to fit best is the one that would arrive earliest after the downstream work centre is expected to run out of work, or alternatively, if all jobs would arrive before that point in time, the one that would arrive the latest. The idea behind this is to prioritise jobs that arrive just in time to keep the work centre busy without having to wait in the queue for a long time. If a work centre receives several conflicting instructions from different downstream work centres, it randomly selects one to follow and notifies the other work centres that their instruction has been refused.

Holthaus and Ziegler test LAJD on the $Jm|r_j|\overline{C}$ problem with balanced workloads in combination with the SPT rule, and find that it achieves significant reductions of mean flow time over pure SPT dispatching. However, the results in Holthaus and Ziegler (1997b) also show that LAJD can impair the mean flow time performance of SPT if the lower bounds for the work content are set too high, leading to an excessively frequent and premature intervention of the LAJD mechanism. In other words, the effectiveness of LAJD depends largely on the values used as lower bounds, for which to find a suitable setting can be challenging. To illustrate, consider a job shop with unbalanced workloads. Clearly, a lowly utilised work centre should not be allowed to interfere with the scheduling of a bottleneck, even if its queue is empty (which could often be the case). However, this implies that each work centre should have its own bound, reflecting its workload to some extent.

Another drawback of LAJD in its current form is that it can only attract jobs that have their next operation on the (nearly) starving work centre. While this may be sufficient for ideal job shops, an extension of the attraction radius to jobs that visit the work centre in the future for some, but not necessarily their subsequent operation, could become necessary to obtain improvements in shops with dominant process flows, which in turn would increase the complexity of the approach.

3.4.2 Minimising Mean Weighted Tardiness

In Section 3.1.2, the WMDD, CR, COVERT and ATC rules are identified as the most promising rules for minimising mean (weighted) tardiness in simple single stage problems. Due to the fact that all four rules incorporate the measure of slack to determine priorities, an essential part of adapting these rules to multi-stage problems is an adequate reformulation of slack.

Since due dates are tied to jobs and not operations, the common definition of slack is job-based and is given by

$$S_j = d_j - t - \sum_{o=i}^{l_j} p_{j,o}. \tag{3.48}$$

This slack definition is used within the standard multi-stage versions of CR, WMDD and COVERT (Baker and Kanet, 1983; Russell et al., 1987; Kutanoglu and Sabuncuoglu, 1999), resulting in priority indices of

$$I_j^{\text{WMDD}} = -\frac{1}{w_j}\left(\max\left(\sum_{o=i}^{l_j} p_{j,o}, d_j - t\right)\right), \tag{3.49}$$

$$I_j^{\text{CR}} = -\frac{d_j - t}{\sum_{o=i}^{l_j} p_{j,o}} \tag{3.50}$$

and

$$I_j^{\text{COVERT}} = \frac{w_j}{p_{j,i}}\left(\max\left(1 - \frac{\max(d_j - t - \sum_{o=i}^{l_j} p_{j,o}, 0)}{k\sum_{o=i}^{l_j} \hat{q}_{j,o}}, 0\right)\right), \tag{3.51}$$

where $\hat{q}_{j,o}$ refers to the queueing time estimated for operation o of job j (see also Section 3.1.2).

On the other hand, it is possible to establish artificial operation due dates to serve as milestones in assessing the progress of a job with respect to its due date. Provided that operation due dates are set adequately, each of them then marks a reference point that indicates whether a job is behind or ahead of its planned schedule and by how much. In theory, there are an unlimited number of possibilities of setting the operation due dates $d_{j,o}$ of a job j, even when adhering to the reasonable condition that the due date of its last operation should be equal to the job due date, i.e. $d_{j,l_j} = d_j$. However, the best practice is to allocate the total initial allowance of a job among its operations proportionally to their respective processing times (Baker, 1984), as illustrated in Figure 3.3. With the operation allowances $a_{j,o}$ fixed, the due date of the imminent operation of job j is then given by

$$d_{j,i} = r_j + \sum_{o=1}^{i} a_{j,o}. \tag{3.52}$$

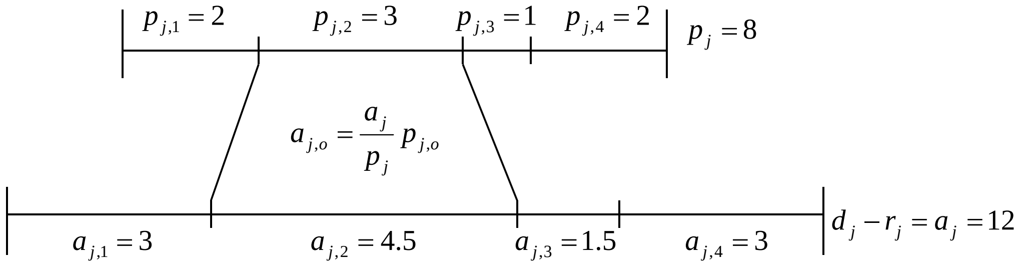


Figure 3.3: Best practice of setting operation due dates. The total allowance $a_j = d_j - r_j$ of job j is allocated to operations proportionally to their processing times $p_{j,o}$.

The availability of operation due dates allows for the computation of the slack of imminent operations as

$$S_{j,i} = d_{j,i} - t - p_{j,i}. \tag{3.53}$$

Then, substituting this operation slack for the job slack within the priority indices of WMDD, CR and COVERT (and the queueing time estimate of the imminent operation $\hat{q}_{j,i}$ for that of the remaining operations within the COVERT index) yields the operation-based variants of the rules (Baker and Kanet, 1983; Russell et al., 1987; Kutanoglu and Sabuncuoglu, 1999), called Weighted Modified Operation Due Date (WMOD), Operation Critical Ratio (OCR) and Operation Cost Over Time (OCOVERT). Their priority indices are defined as

$$I_j^{\text{WMOD}} = -\frac{1}{w_j}(\max(p_{j,i}, d_{j,i} - t)), \tag{3.54}$$

$$I_j^{\text{OCR}} = -\frac{d_{j,i} - t}{p_{j,i}} \tag{3.55}$$

and

$$I_j^{\text{OCOVERT}} = \frac{w_j}{p_{j,i}}\left(\max\left(1 - \frac{\max(d_{j,i} - t - p_{j,i}, 0)}{k\hat{q}_{j,i}}, 0\right)\right). \tag{3.56}$$

Several studies have compared the mean (weighted) tardiness performance of these rules with that of their job-based counterparts. Kanet and Hayya (1982) apply CR and OCR to a dynamic job shop and conclude that OCR achieves both a lower conditional mean tardiness and a smaller proportion of tardy jobs (and hence a lower mean tardiness) than CR across a wide range of due date tightness levels. The relative advantage of OCR over CR is confirmed by Baker and Kanet (1983) and Baker (1984) for the $Jm|r_j|\overline{T}$ problem, by Kutanoglu and Sabuncuoglu (1999) for the $Jm|r_j|\overline{wT}$ problem and by Barrett and Kadipasaoglu (1990) for the $Fm|r_j|\overline{T}$ problem, although the results in Baker and Kanet (1983) and Baker (1984) indicate that CR performs better under loose due date settings. Similarly, the operation-based MOD rule, the unweighted relative of WMOD, is generally found to be more effective in reducing mean (weighted) tardiness than the job-based MDD rule unless due dates are very loose (Baker and Kanet, 1983; Russell et al., 1987; Kutanoglu and Sabuncuoglu, 1999; El-Bouri, 2012), while the results in Raman, Talbot, and Rachamadugu (1989) for the $Jm|r_j|\overline{T}$ problem indicate that MOD also looses its advantage over MDD if the workloads of the work centres are unbalanced. Baker and Kanet (1983) evaluate COVERT and OCOVERT on the $Jm|r_j|\overline{T}$ problem and report that OCOVERT yields a lower mean tardiness at very tight due date levels, but is quickly outperformed by COVERT at looser due date settings. However, both rules are clearly dominated by MOD, which performs best overall in their study.

The general observation that the operation-based variants of due date-oriented dispatching rules tend to be more effective than their job-based counterparts motivates Vepsalainen and Morton (1987) to adapt ATC for multi-stage problems directly on the basis of operation slack. They define the priority index of the multi-stage version of ATC as

$$I_j^{\text{ATC}} = \frac{w_j}{p_{j,i}} \exp\left(- \frac{\max(d_j - t - p_{j,i} - \sum_{o=i+1}^{l_j} (\hat{q}_{j,o} + p_{j,o}), 0)}{k\overline{p}} \right) \tag{3.57}$$

and

$$d_{j,i} = d_j - \sum_{o=i+1}^{l_j} (\hat{q}_{j,o} + p_{j,o}), \tag{3.58}$$

which yields

$$d_j - t - p_{j,i} - \sum_{o=i+1}^{l_j} (\hat{q}_{j,o} + p_{j,o}) = d_{j,i} - t - p_{j,i} = S_{j,i}, \tag{3.59}$$

where $\overline{p}$ refers to the average processing time of imminent operations of the jobs in the queue.

Equation (3.58) links operation due dates, representing the time allowed to complete an operation, to operation flow times, i.e. the (estimated) time required to complete that operation. Moreover, Equation (3.59) can be rearranged to

$$S_{j,i} = d_j - t - \sum_{o=i}^{l_j} p_{j,o} - \sum_{o=i+1}^{l_j} \hat{q}_{j,o} = S_j - \sum_{o=i+1}^{l_j} \hat{q}_{j,o}, \tag{3.60}$$

which distinguishes operation slack as a conservative measure of job slack that reserves some time for delays of subsequent operations. Note that once operation due dates have been fixed, $\hat{q}_{j,o}$ in Equations (3.58) – (3.60) should be interpreted as the permitted, rather than the estimated queueing time of an operation, irrespective of whether queueing time estimates have been originally used to set the operation due dates or not. To illustrate, if operation allowances are set proportionally to processing times as in Figure 3.3, then the time an operation is allowed to queue for is $\hat{q}_{j,o} = (a_j/p_j - 1)p_{j,o}$.

The above rules have been evaluated against each other in a number of studies and on various problems. Vepsalainen and Morton (1987) apply their ATC rule along with some other rules to the $Jm|r_j|\overline{wT}$ problem and report that it yields the lowest mean weighted tardiness, irrespective of the utilisation or the tightness of due dates, thereby outperforming COVERT as the second best rule in their study. The strong performance of ATC is confirmed by Kutanoglu and Sabuncuoglu (1999), who test a number of dispatching rules on the $Jm|r_j|\overline{wT}$ problem with varying levels of workload balance, utilisation and due date tightness, and find that ATC is consistently among the best rules, together with COVERT. Similarly, results in Jensen et al. (1995) show ATC, COVERT and a WMOD rule that is based on Equation (3.6) instead of Equation (3.7) to be the most effective rules for the $Jm|r_j|\overline{wT}$ problem, irrespective of the due date tightness or distribution of job weights. Russell et al. (1987) evaluate COVERT and MOD on the $Jm|r_j|\overline{T}$ problem and conclude that MOD is the slightly better rule. However, experimental results from Anderson and Nyirenda (1990) for the same problem suggest that this holds only for moderate to tight due date settings. Similarly, El-Bouri (2012) finds that MOD is more effective than COVERT for the $Fm|r_j|\overline{T}$ problem with tight due dates, but is outperformed by COVERT in loose due date conditions. The results further indicate that COVERT performs better than MDD, which is in contrast to the results from Brah (1996) and Chiang and Fu (2007) for the $FFc||\overline{T}$ and the $Jm||\overline{T}$ problem, respectively, which show both rules to perform more or less equally well. Moreover, ATC is shown to outperform MDD, COVERT and CR on the $Jm||\overline{T}$ problem in Chiang and Fu (2007), while the studies by Philipoom and Fry (1990) and Raghu and Rajendran (1993) indicate that ATC and MOD yield a very similar performance on the $Jm|r_j|\overline{T}$ problem, in general. Overall, the mixed findings on the relative performance of the rules makes it difficult to rank them, though ATC, (W)MOD and COVERT appear to be the most promising rules.

One factor that impairs the comparability of results is the fact that different studies use different settings for the scaling parameter k and also different ways to establish the estimated or allowed queueing times $\hat{q}_{j,o}$, which affects the performance of the (O)COVERT and ATC rules. For ATC, Vepsalainen and Morton (1987) recommend to set the queueing times as a multiple of the corresponding processing times and k to a value between 1.5 and 4.5, and these guidelines are widely followed in the literature (see, e.g. Philipoom and Fry, 1990; Raghu and Rajendran, 1993; Kutanoglu and Sabuncuoglu, 1999). Russell et al. (1987), on the other hand, compare different configurations of the COVERT rule and find that it becomes more effective if queueing time estimates are based on historically observed or dynamically updated queueing times instead of being tied to operation allowances. However, some papers simply adopt the best practice for ATC and take a multiple of the respective processing time as the queueing time estimates within COVERT (e.g. Vepsalainen and Morton, 1987; Kutanoglu and Sabuncuoglu, 1999). Commonly proposed values for the scaling parameter k of COVERT are 1 or 2 (see, e.g. Baker and Kanet, 1983; Russell et al., 1987; Jensen et al., 1995; Kutanoglu and Sabuncuoglu, 1999), though the ideal value will generally depend on how the estimated queueing times are established. This discussion illustrates an important advantage of the WMOD rule, which is that it can be applied directly without having to worry about setting parameters.

Anderson and Nyirenda (1990) develop two more non-parametric rules by integrating the job-based CR and S/RPT rules with the operation-based WSPT rule. The priority indices of these rules, named CR+SPT and S/RPT+SPT can be expressed as

$$I_j^{\text{CR+SPT}} = -\frac{1}{w_j}\left(p_{j,i} \times \max\left(\frac{d_j - t}{\sum_{o=i}^{l_j} p_{j,o}}, 1\right)\right) \tag{3.61}$$

and

$$I_j^{\text{S/RPT+SPT}} = -\frac{1}{w_j}\left(p_{j,i} \times \max\left(\frac{d_j - t - \sum_{o=i}^{l_j} p_{j,o}}{\sum_{o=i}^{l_j} p_{j,o}}, 1\right)\right). \tag{3.62}$$

Note that CR+SPT is a close relative of the WMOD rule, and that both rules are equivalent for single stage problems. The main difference between CR+SPT and S/RPT+SPT is the point at which they switch to WSPT prioritisation. While CR+SPT prioritises jobs according to WSPT once they are too late to finish on time (critical ratio smaller than 1), S/RPT+SPT reverts to WSPT at an earlier point, namely when the critical ratio of a job is smaller than 2. As a consequence, S/RPT+SPT can be expected to yield a lower mean weighted tardiness than CR+SPT in conditions in which WSPT performs well, i.e. when due dates are tight, whereas CR+SPT should be the better choice for loose due date settings. This notion is largely supported by the results of Anderson and Nyirenda (1990) and Raghu and Rajendran (1993) for the $Jm|r_j|\overline{T}$ and the $Jm|r_j|\overline{wT}$ problem. However, the difference between the mean (weighted) tardiness performance of the two rules appears to be small, in general (Anderson and Nyirenda, 1990; Raghu and Rajendran, 1993; Chiang and Fu, 2007).

Both CR+SPT and S/RPT+SPT have also been tested against the other rules in various papers. Anderson and Nyirenda (1990) apply several rules to the $Jm|r_j|\overline{T}$ and the $Jm|r_j|\overline{wT}$ problem and find that CR+SPT and S/RPT+SPT dominate COVERT across a large range of due date tightness levels, and also MOD in the unit weight case unless due dates are extremely tight. The strong performance of CR+SPT and S/RPT+SPT is confirmed by the results in Raghu and Rajendran (1993) that show them to be more effective than MOD and ATC for the $Jm|r_j|\overline{T}$ problem. Similarly, Moser and Engell (1992b) report that CR+SPT yields a lower mean tardiness than MOD and COVERT for the $FJc|r_j|\overline{T}$ problem. On the other hand, Kutanoglu and Sabuncuoglu (1999) conclude that the best rules for the $Jm|r_j|\overline{wT}$ problem are S/RPT+SPT, CR+SPT, COVERT and ATC, and that performance differences between them are insignificant. Brah (1996) find no significant difference between the performance of CR+SPT, S/RPT+SPT, COVERT and MDD for the $FFc||\overline{T}$ problem, while the results in Chiang and Fu (2007) indicate that CR+SPT and S/RPT+SPT perform better than MDD and COVERT for the $Jm||\overline{T}$ problem with a moderate tightness level, but not better than ATC. In general, the relative advantage of CR+SPT and S/RPT+SPT over other rules appears to be largest for moderate tightness levels and tends to diminish as due dates become tighter (Anderson and Nyirenda, 1990; Raghu and Rajendran, 1993; Chiang and Fu, 2007). This can be expected in so far as CR+SPT, S/RPT+SPT, WMOD, COVERT and ATC all converge to the WSPT rule with increasing due date tightness (though at different rates), which could explain why some studies struggle to find any significant difference between their performance.

Consideration of Upstream Jobs

The development of look-ahead dispatching rules for problems with the objective to minimise mean weighted tardiness appears to be very promising given that the penalty for waiting for a future job may be negligible relative to the benefits of processing that job without further delay, e.g. if the future job is behind schedule and has a substantially higher weight than some non-urgent jobs in the queue.

Some ATC-based rules have been designed to deal with future jobs. Morton and Pentico (1993) extend the ATC rule by a factor that discounts the priority index of a future job for the expected waiting time that would result from its prioritisation (Pfund et al., 2008). The priority index of this X-ATC rule is given by

$$I_j^{\text{X-ATC}} = \frac{w_j}{p_{j,i}} \exp\left(-\frac{\max(d_{j,i} - t - p_{j,i}, 0)}{k_1 \overline{p}}\right)\left(1 - \frac{k_3 \times \max(r_{j,i} - t, 0)}{p_{\min}}\right), \tag{3.63}$$

where $p_{\min}$ denotes the minimum imminent operation processing time of jobs in the queue, and k_3 represents an extra scaling parameter for the waiting time discount factor. X-ATC only considers look-ahead jobs that arrive before $t + p_{\min}$, i.e. before any of the queueing jobs can complete its imminent operation, where the priority index of these jobs is linearly decreasing in their arrival time. In contrast, Pfund et al. (2008) propose an extension of the ATCS rule that incorporates an exponential discount function for the waiting time, which is normalised by the average instead of the minimum processing time in the queue. For problems without sequence-dependent setup times the priority index of this ATC with Setups and Ready Times (ATCSR) rule reduces to

$$I_j^{\text{ATCSR}} = \frac{w_j}{p_{j,i}} \exp\left(-\frac{\max(d_{j,i} - \max(r_{j,i}, t) - p_{j,i}, 0)}{k_1 \overline{p}}\right) \exp\left(-\frac{\max(r_{j,i} - t, 0)}{k_3 \overline{p}}\right). \tag{3.64}$$

Hence, another modification of the ATCSR rule concerns the computation of slack for future jobs, which is adjusted to reflect the slack they actually have at the time of their expected arrival to the considered work centre. Pfund et al. apply ATCSR and X-ATC with the additional ATCS setup time factor to the $Pm|r_j, s_{ef}|\overline{wT}$ problem, using various parameter settings for both rules, and conclude that ATCSR is the better rule. They further report that the relative advantage of the two look-ahead rules over ATCS increases for a higher s/p ratio, which is in agreement with similar findings for problems with the minimum mean flow time objective (see Section 3.4.1).

However, a disadvantage of both rules is that they require an additional scaling parameter to be set adequately in order to perform well.

Mönch and Habenicht (2003) develop a look-ahead rule for batch processing problems on the basis of the BATCS rule by Mason et al. (2002). This rule, called BATCR (BATC with Ready Times) in this work, considers only future jobs that arrive within a prespecified time window, which is a function of the average processing time in the queue. BATCR sorts these jobs and all queueing jobs in non-decreasing order of their ATCR index, which is defined as

$$I_j^{\text{ATCR}} = \frac{w_j}{p_{j,i}} \exp\left(-\frac{\max(d_{j,i} - t - p_{j,i} + \max(r_{j,i} - t, 0), 0)}{k\overline{p}}\right), \tag{3.65}$$

and selects the H jobs with the highest priorities to be included in the formation of batches, where H is a rule parameter. Using these jobs, BATCR forms a batch for every feasible job combination and prioritises batches according to the BATCR index, given by

$$I_b^{\text{BATCR}} = \frac{w_b}{p_b} \exp\left(-\frac{\max(d_b - t - p_b + \max(r_b - t, 0), 0)}{k\overline{p}}\right) \frac{1}{B_{f_b}}, \tag{3.66}$$

where the release date r_b of batch b is determined by the maximum of the times at which the jobs it contains are expected to arrive at the considered work centre. Depending on whether the batch with the highest priority contains future jobs or not, the machine either is made to wait for their arrival or starts processing that batch, which is filled up with jobs from the queue if possible.

An obvious shortcoming of the BATCR rule concerns its formation of batches. While limiting the number of eligible batches to combinations that involve only high priority jobs is certainly sensible, batches should be generally made as full as possible at the respective time, as nothing can be gained from leaving some of the machine capacity unused (see Section 3.3.2). The BATCR rule, on the contrary, tends to form and evaluate partial batches and only fills them up if they are chosen, which is likely to lead to poor decisions.

Another drawback that is common to all of the above ATC-based look-ahead rules is the inaccurate estimation of waiting time penalties, which should be based on the total number of jobs in the queue, their (average) weight and remaining allowance. Hence, there still seems to be a large potential for future work on the design of effective due date-oriented look-ahead rules.

Consideration of Downstream Work Centres

Clearly, avoiding the starvation of highly utilised work centres can also help to minimise mean (weighted) tardiness, as it may reduce the amount of allowance that jobs use up on non-productive waiting times.

Raghu and Rajendran (1993) design a rule, commonly named the RR rule after its developers, with a priority index of

$$I_j^{\text{RR}} = -\left(\frac{d_j - t - \sum_{o=i}^{l_j} p_{j,o}}{\sum_{o=i}^{l_j} p_{j,o}} \exp(-\eta) p_{j,i} + \exp(\eta) p_{j,i} + \hat{q}_{j,i+1}\right), \tag{3.67}$$

where $\hat{q}_{j,i+1}$ refers to the estimated queueing time for the next operation of job j and η to the utilisation of the shop. As shown in Equation (3.67), the RR rule uses the utilisation as a weighting factor to switch between prioritising jobs with short imminent operations at high utilisation levels and a stronger focus on the urgency of jobs at lower utilisation levels.

In several papers, RR is found to produce outstanding results for the $Jm|r_j|\overline{T}$ problem with balanced workloads, dominating rules such as MOD, COVERT, ATC, CR+SPT, and S/RPT+SPT across a wide range of utilisation and due date tightness levels (Raghu and Rajendran, 1993; Holthaus, 1997; Holthaus and Rajendran, 1997a,b; Rajendran and Holthaus, 1999; Holthaus and Rajendran, 2000). However, Raghu and Rajendran (1993) observe that RR looses some of its relative advantage with decreasing utilisation and due date tightness, which they ascribe to the reduced effectiveness of the queueing time component under these conditions. Similarly, Rajendran and Holthaus (1999) explain the finding that RR is outperformed by COVERT in flow-dominant shops by the ineffectiveness of the queueing time component for such problems. In other words, there are some strong indications that the good performance of RR is mainly due to the queueing time component, which is similar to WINQ in its prioritisation of jobs that visit a work centre with a short queue for their next operation. Hence, the RR rule appears to be specifically tailored to ideal job shop problems with a perfectly balanced workload among the work centres and, like PT+WINQ and 2PT+WINQ+NPT, cannot be expected to perform well for other problems. Moreover, it does not account for unequal job weights in its current form.

In addition to their flow time-oriented rules, Rajendran and Alicke (2007) propose two rules to minimise mean tardiness in an unbalanced flow shop, which are based on similar ideas. Their respective priority indices are given by

$$I_j^{\text{DD/BJ}} = \begin{cases} -d_{j,h} & \text{if } i \leq h, \\ -d_j & \text{otherwise,} \end{cases} \tag{3.68}$$

and

$$I_j^{\text{(TPT+DD)/BJ}} = \begin{cases} -(d_{j,h} + \sum\limits_{o=i}^{h} p_{j,o}) & \text{if } i \leq h, \\ -(d_j + \sum\limits_{o=i}^{l_j} p_{j,o}) & \text{otherwise,} \end{cases} \tag{3.69}$$

where h is again the index of the next operation of job j that is to be performed on a (global) bottleneck work centre (h is the same for all jobs in flow shop problems). Both rules thus focus on finishing bottleneck operations on time, while (TPT+DD)/BJ also prioritise jobs that can quickly reach and be pushed through the bottleneck.

Rajendran and Alicke apply DD/BJ and (TPT+DD)/BJ to the $Fm||\overline{T}$ problem with up to three bottleneck work centres, and conclude that both, and especially (TPT+DD)/BJ, perform significantly better than the other rules tested, which includes EDD and its operation-based variant, ODD. Hence, there appears to be some potential in concentrating on the due date of the bottleneck operation. However, the design of a rule that can outperform more effective rules, such as ATC, is likely to require a more sophisticated integration of this information with other attributes than just a simple addition of the remaining processing time and absolute due date, which is further unsuitable for problems with dynamic job arrivals (see Section 3.1.2).

El-Bouri (2012) presents an adjusted version of CD for the minimisation of mean tardiness that computes the regret $\hat{R}_{z_{j,o}}(j)$ as the increase of local tardiness instead of local flow time, where local tardiness is measured with respect to the due dates of the operations to be performed on the respective work centre. To determine the minimum local tardiness for a given work centre, the ODD rule is used in combination with a procedure that tries to improve the ODD sequence by pairwise interchanges of adjacent jobs. This tardiness-variant of CD is shown to outperform benchmark rules such as MDD, MOD and COVERT on the $Fm|r_j|\overline{T}$ problem, irrespective of the shop size, utilisation or tightness of due dates (El-Bouri, 2012).

A drawback of this CD version is that the determination of appropriate weighting factors $v_{z_{j,o}}$ seems to become rather difficult. In particular, El-Bouri finds that, for conditions of tight due dates and/or high utilisation, it is beneficial to give a higher weight to more proximate work centres, as in the flow time-oriented version of CD, while the opposite is true if due dates are relatively loose, which El-Bouri ascribes to the fact that tardiness only arises at work centres farther downstream in the latter case. This illustrates a main problem of the tardiness-variant of CD, namely that the tardiness-based regret values strongly depend on the estimated operation completion times and the used operation due dates, which thus affect the effectiveness of the rule. On the other hand, the suitability of regrets based on local tardiness is generally questionable given that the global mean tardiness, unlike the global mean flow time, is not the sum of its local components. Hence, it may be more effective to use the original CD to reduce queueing times and combine it with a dispatching rule that concentrates on due dates (and weights) of jobs.

Holthaus and Ziegler (1997a,b) show that their LAJD mechanism can also be used to improve the performance of the COVERT rule for the $Jm|r_j|\overline{T}$ problem with balanced workloads. The improvements, which are obtained for a wide range of due date tightness levels, seem to be due to the reductions of waiting times achieved by LAJD. This is supported by the observation that COVERT and SPT show a similar mean flow time performance when combined with

LAJD, performing significantly better than SPT by itself. However, the application of LAJD is still hindered by the requirement to determine work content bounds and an attraction radius that are suitable for the problem at hand.

In summary, there are a number of dispatching rules for multi-stage problems that effectively minimise mean flow time and mean weighted tardiness, respectively. A general finding is that operation-based rules, such as SPT and WMOD, tend to perform better than the corresponding job-based rules, i.e. LWKR and WMDD, though rules that incorporate both job and operation attributes, e.g. CR+SPT, also appear to perform well. Moreover, some rules that make use of upstream or downstream information are found to be very effective, which illustrates the potential of incorporating such information into the design of dispatching rules.

On the other hand, the development of sophisticated rules for complex scheduling problems is usually a tedious trial-and-error process, with candidate rules tested in a simulation model of the considered shop, modified and retested until they meet the demands for actual implementation, which requires a significant amount of expertise, time and coding-effort (Geiger et al., 2006). This motivates the development of methods to support and automate the design of dispatching rules, which is the main concern of this thesis. The following chapter describes the methodology of this work, specifically the tools and techniques used in the process of developing the methods.

Chapter 4

Methodology

An essential part of this thesis consists in the actual development of methods and algorithms, covering all stages from the conceptual design to the implementation into computer code. Once implemented, the procedures need to be tested on some example problem to establish whether they are effective, which is another important part of this work.

This chapter presents the tools and techniques used for the development and evaluation of methods. The first section provides a short introduction to evolutionary algorithms, which are a key component of the developed methods. This is followed by another section describing the simulation experiments and statistical tests to assess the effectiveness of methods.

4.1 Development of Methods

The methods designed in this work are all based on Evolutionary Algorithms (EAs), which are stochastic search procedures, inspired by natural evolution, that maintain a set of candidate solutions, the so-called *population*. The general process of EAs can be described as follows. In each iteration, the promising *individuals*, i.e. candidate solutions with good objective function values or *fitness* values, are selected from the population to *reproduce*, in the spirit of 'survival of the fittest'. Reproduction typically involves the operations of *crossover*, in which two individuals, the *parents*, are recombined to create new individuals, the *children*, and *mutation*, which makes a random modification to an individual. The newly created individuals, often together with the best original individuals, then form the new population, also referred to as the next *generation*. This cycle, which is illustrated in Figure 4.1, is repeated until a specified termination criterion is met, e.g. a limit on runtime or the number of iterations is reached. Hence, this iterative process evolves better and better solutions over time, just like in nature, where the individuals become better and better adapted to their environment. Finally, the best individual of the last generation is returned as the solution of the EA.

A key advantage of EAs is that their general design is largely abstract from the given problem as the interface is mainly established by the definition of an adequate problem-specific

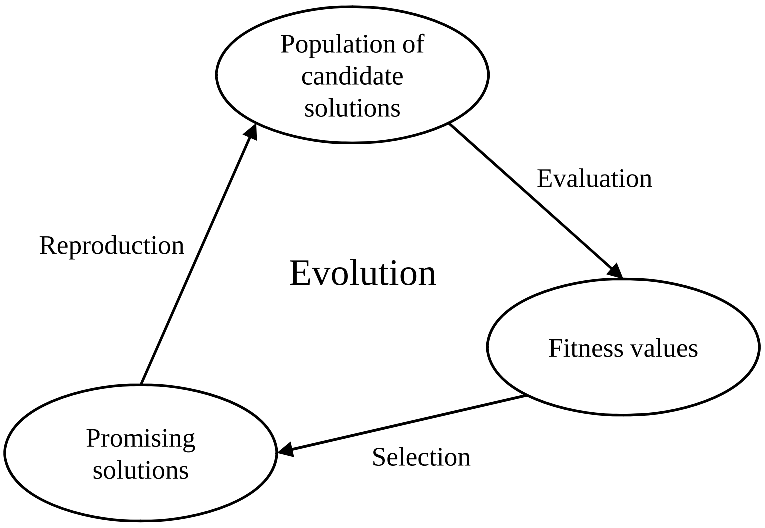


Figure 4.1: Simplified diagram of the evolutionary cycle.

fitness or evaluation function. Hence, EAs can be applied to any type of problem with relative ease, in principle, since most components are reusable. This is reflected by the numerous successful applications of EAs that can be found in the literature, including in the area of scheduling (Hart et al., 2005), and motivates their use in this work. The specific design of each developed EA, in particular the encoding of solutions, fitness function, selection scheme and reproduction operators, is detailed within the presentation of the respective method, which are discussed in the following chapters (for an extensive review of EAs, see Eiben and Smith, 2003).

All algorithms are implemented in Java programming language, using the library ECJ (Luke et al., 1998), which contains a collection of generic EA structures and components. This allows the programming effort to be concentrated on the implementation of newly developed components such as fitness functions, or specifically designed encodings and operators.

4.2 Evaluation of Methods

The primary aim of the methods proposed in this work is to lead to the design of better dispatching rules. Hence, the effectiveness of a method can be gauged by the performance of the dispatching rules that result from its application relative to some benchmark rules from the literature. Due to the lack of suitable analytical methods, dispatching rules are generally evaluated by means of simulation (Blackstone et al., 1982), which is adopted herein.

Prior to the execution of the actual simulation experiments, the warm-up period for a given simulation model, i.e. the time it takes the simulation to reach steady state from an initial empty state, is determined using Welch's procedure (see also Law, 2007, Chapter 9). This is a graphical method, in which the moving average for a characteristic output quantity such as the throughput or flow time, averaged over a number of replications, is plotted over time or number of completed jobs, respectively. Figure 4.2 shows such a graph for one of the job shop models from Chapter 5. The simulation is considered to be warmed up after the graph has levelled out, and some (conservative) value beyond this point is selected as the end of the warm-up period, e.g. 1000 job completions in the case of Figure 4.2. For the purpose of generating these graphs, the simulation is run with the First Come First Served (FCFS) rule.

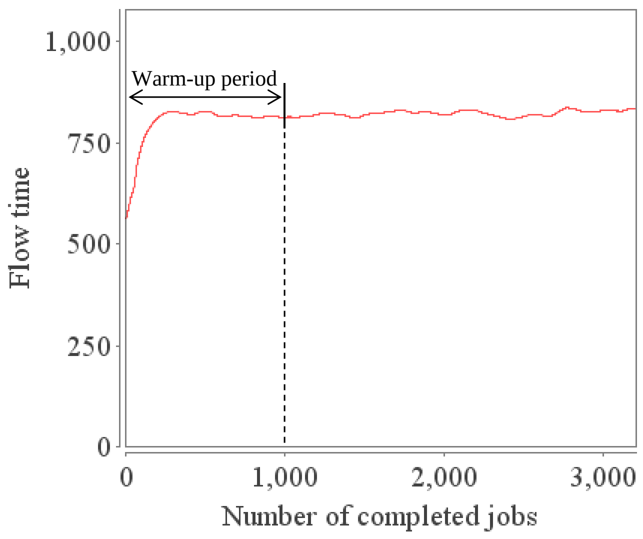


Figure 4.2: Moving averages for job flow time for a job shop model from Chapter 5. The warm-up period is determined to comprise the completion of 1000 jobs, at which point the flattened out graph indicates that the simulation has reached steady state (see also Law, 2007, Chapter 9).

With the warm-up period determined, the evaluation of dispatching rules is done by running the simulation with the corresponding rule for some longer period, where data collection starts at the end of the warm-up. More specifically, data is collected for the first n jobs arriving after the end of the warm-up period, and the shop is continuously loaded with new jobs until the last of these n jobs is completed, which marks the end of the simulation run. Each experiment, defined by the combination of dispatching rule and simulation model, is replicated 20 times with different random number streams to yield a precise measure of the performance of a given rule. Moreover, the variance reduction technique of common random numbers is used, i.e. each dispatching rule is tested with the same 20 random number streams (Law, 2007, Chapter 11).

All simulation experiments in this work are carried out with Jasima (Hildebrandt, 2012), a discrete-event simulation environment that has been developed in cooperation with the BIBA institute at the University of Bremen, Germany. Jasima is very roughly based on a Java-port of the SIMLIB library as described by Huffman (2001). Given that it is Java-based, Jasima can be easily integrated with ECJ, which facilitates the implementation of the methods proposed in this work. Another advantage of using a self-developed, rather than a commercial simulation software is the increased flexibility with regard to the implementation of specific dispatching rules or shop environments. The correct operation of Jasima and the implemented simulation models has been widely ensured by verification tests, including a high-level analysis of the simulation output as well as a thorough examination of the simulation trace files.

To establish whether the performance differences between dispatching rules are statistically significant, simulation results are subjected to a single-factor within-subjects Analysis Of Variance (ANOVA), and the Bonferroni-Dunn test in particular (Sheskin, 2007, Test 24). This parametric test aims to detect differences in population means and assumes an experimental design with dependent samples that result from testing a number of different treatments (dispatching rules) on the same subjects (random number streams), which qualifies it to be used in conjunction with common random numbers. The test further requires that

- subjects are randomly selected,
- the underlying population of each treatment follows a normal distribution,
- the sphericity assumption is satisfied.

Clearly, the first condition is fulfilled by the use of random numbers in creating the seeds for each replication of a simulation experiment, which corresponds to a random selection of subjects. Moreover, since all of the measures examined in this work, e.g. the mean flow time or mean weighted tardiness (of a simulation run), are averages of a large number of individual (though correlated) observations, they can be expected to closely follow a normal distribution (Law, 2007, Chapter 9). Despite this, Anderson-Darling and Kolmogorov-Smirnov tests (see Law, 2007, Chapter 6) are conducted in this work to check for any considerable violations of the normality assumption before applying ANOVA to any data. The sphericity assumption, on the other hand, which relates to the nature of underlying population variances and covariances, is difficult to verify (Sheskin, 2007, Test 24). Instead, the Geisser-Greenshouse method, as described in Sheskin (2007, Test 24), is followed, which involves employing a more conservative tabled critical F-value so that the sphericity assumption no longer has to hold. Hence, the suitability of the test for its intended purpose is ensured on the whole.

The Bonferroni-Dunn test is based on the Bonferroni inequality, which adjusts the significance level of an individual comparison α_{ind} for the number of conducted comparisons v according to $\alpha_{\text{ind}} = \alpha/v$, where α is the overall significance level of the test (Sheskin, 2007, Test 21). The adjustment implies that a set of comparisons, e.g. underlying a complete ranking of some compared dispatching rules, can be considered to be correct with $(1 - \alpha)\%$ probability. However, the drawback of the Bonferroni-Dunn test is that it is rather conservative so that performance differences between dispatching rules are only detected if there is very strong evidence for it (Sheskin, 2007, Test 21).

In this work, an overall significance level of $\alpha = 0.05$ is used for all tests, where the p-values of the pairwise comparisons are provided in Appendix A for further reference. All tests have been implemented in Java, following the descriptions in Sheskin (2007, Tests 21 and 24) and Law (2007, Chapter 6), and verified with examples from these books.

This concludes the description of the general methodology used to develop and test methods. The actual design of the methods as well as results on their performance are presented in the following chapters.

Chapter 5

Evolutionary Search for Difficult Problem Instances [1]

A possible approach to the development of better dispatching rules, and algorithms in general, is to study the performance of current benchmark algorithms, especially in situations where they fail. Any insights gained from this analysis can then be incorporated into the design of a new, improved algorithm, making the development process more systematic.

This chapter presents a method to support the process of developing dispatching rules by employing an EA to search for job shop problem instances on which a given rule performs poorly, thus helping to reveal its weaknesses. First, some related work is reviewed in Section 5.1, followed by a detailed description of the EA in Section 5.2. The method is applied to a scenario from the literature in Section 5.3, resulting in two new dispatching rules, which are evaluated against benchmark rules in Section 5.4. The chapter concludes with a critical appraisal of the method and some suggestions for future work.

5.1 Related Work

In areas other than scheduling, metaheuristics, and EAs in particular, have been used to generate problem instances that are hard to solve for a particular algorithm, where hardness is measured in terms of the quality of the solution the algorithm generates (relative to some benchmark) or the amount of time it requires to (optimally) solve the problem. Cotta and Moscato (2003) employ an EA to search for permutations of natural numbers that are hard to rearrange for two well-known sorting algorithms. They evolve instances of different sizes and use them to derive lower bounds for the worst-case runtime complexity of the two algorithms as a function of problem size, which turn out to be very close to their actual respective

[1]This method is published in J. Branke and C. W. Pickardt (2011), Evolutionary search for difficult problem instances to support the design of job shop dispatching rules, *European Journal of Operational Research* 212(1), 22–32. However, the experimental results presented in this chapter deviate from those reported in the paper due to a slightly different experimental setup.

worst-case complexity. Van Hemert (2006) presents several applications of EAs to generate instances of combinatorial problems that are difficult to solve for a corresponding solution algorithm in terms of runtime. The evolved instances are analysed for common patterns, which reveals some structural properties that characterise challenging instances within each problem domain. Smith-Miles and Van Hemert (2011) extend this work in two ways. First, they do not only let their EA search for hard (and easy) instances, but also for instances that are hard for one solution procedure relative to another. Second, they use machine learning to extract structural properties from the evolved instances to predict which algorithm is most suitable for a given instance. Smith-Miles and Van Hemert test their approach on two solution algorithms for the travelling salesman problem and discover some problem features that reliably predict the relative performance of the algorithms. Ahammed and Moscato (2011) also generate instances of the travelling salesman problem that take a state-of-the-art solver a long time to solve, but use a simple local search algorithm instead of an EA.

Some researchers have employed metaheuristics to evolve objective or fitness functions that are challenging for another, problem-solving metaheuristic. Using a common EA and Genetic Programming (GP), which is a special kind of EA, Oltean (2004) generate one-dimensional functions for which one metaheuristic performs badly relative to another, where the tested algorithms include some simple local search procedures with restarts as well as random search. Similarly, Langdon and Poli (2007) develop a GP algorithm to evolve two-dimensional functions that are comparatively hard for one metaheuristic relative to another regarding the time required to find a global optimum. They apply this algorithm to a variety of metaheuristics, particularly different variants of particle swarm optimisation, and reveal some of their strengths and weaknesses by a visual analysis of their search behaviour on the evolved functions. Shirakawa et al. (2010) employ (Cartesian) GP to generate two-dimensional functions for which two metaheuristics show a large performance difference, but use a performance measure that takes into account the whole search process of a metaheuristic, specifically the best objective function value found at each point in time. They further develop a multiobjective version of the algorithm that considers the complexity of functions as a secondary criterion. The algorithm is applied to a number of EAs, based on different crossovers, as well as random search and shown to find some simple functions that illustrate advantages and shortcomings of each crossover operator.

Simple heuristics such as dispatching rules are by definition designed to produce solutions of acceptable quality in a short time. Hence, in the case of heuristics it is only sensible to search for problem instances for which the tested algorithm yields poor (or good) solutions, as their runtime is generally not an issue. Kosoresow and Johnson (2002) develop an EA that searches for instances of combinatorial problems that are hard for a given heuristic, where hardness of an instance is measured by the ratio of the objective function value of the heuristic solution to that of the optimal solution. They apply the EA to some heuristics for the taxicab problem and obtain lower bounds on their worst-case performance. Similarly, Julstrom (2009) uses an EA to generate instances of the quadratic knapsack problem that are badly solved by a simple heuristic, and discovers some structural properties in those instances that explain why the heuristic fails.

These studies show the potential of using EAs to evolve problem instances that reveal weaknesses of solution methods, and heuristics in particular. However, this work appears to be the first to apply the idea to the area of scheduling. Moreover, the proposed EA is designed to search for instances where the poor performance of a tested dispatching rule can be attributed to a single suboptimal decision, which is a key difference to previous work. The advantage of focussing on single decision points is that it greatly facilitates the analysis of problem instances and the identification of conditions that cause the rule to fail, as illustrated by the application in Section 5.3.

5.2 Algorithm Design

In general, the EA operates as described in Section 4.1, and the focus of this section is to detail the algorithm-specific components, namely the encoding of candidate solutions and the evaluation, selection and reproduction of individuals.

5.2.1 Encoding of Solutions

Since the EA is designed to generate difficult instances of the job shop problem, candidate solutions are in fact problem instances in this case. The encoding of job shop problem instances is illustrated in Figure 5.1. Each instance is represented by an $n \times 2l$ two-dimensional array containing solely non-negative integer values. The n rows correspond to the processing orders

Job	Op. 1		Op. 2		Op. 3		Op. 4	
1	3	20	4	48	1	26	2	16
2	3	15	1	26	2	41	1	25
3	2	21	3	10	1	37	3	44
4	3	15	4	30	1	1	2	32
5	1	18	4	4	3	30	2	6
6	4	44	2	9	4	34	3	13

Figure 5.1: Example of the encoding of job shop problem instances as $n \times 2l$ two-dimensional arrays for an instance with $n = 6$ jobs, consisting of $l = 4$ operations (grey parts are added for illustration purposes).

of n jobs consisting of l operations. Each operation is defined by the values in two successive columns, which refer to the index of the required work centre and the processing time, e.g. the first operation of Job 1 in the shown problem instance is to be executed on Work Centre 3 and lasts 20 time units. Values for the operation processing times are restricted to the (problem-specific) interval $[p_{\min}, p_{\max}]$, while work centre indices can take values in the range of 1 to c, where c denotes the number of work centres in the shop. The encoding thus only specifies job attributes, other parameters of the scheduling problem such as the number of work centres or the objective function are not part of the encoding and are shared among all individuals. Although revisitations of work centres are allowed, the EA is prevented to generate instances in which two consecutive operations of the same job have the same work centre index as this technically corresponds to a violation of the no preemption assumption (see Section 2.1.2), given that the two operations can be considered as one.

5.2.2 Evaluation

Figure 5.2 presents the procedure to evaluate individuals, which comprises the following steps. First, individual x is decoded into the associated problem instance $OP(x)$, which is then converted into a mathematical programme and solved to optimality. Pan and Chen (2005) formulate a mixed integer programme for the static job shop problem, which is adopted for this purpose, and the SCIP solver (Achterberg et al., 2003) is used to solve the programmes. After determining the optimum with objective function value $Z^*(OP(x))$, the tested dispatching rule is applied to the problem and the resulting solution is obtained from simulation. Since dispatching rules typically produce active schedules, which possess the property that it is impossible to start any operation earlier without having to postpone another (Pinedo, 2008, Chapter 2), the optimal schedule is transformed into an active schedule before comparing both solutions. In this comparison, the operation starting times are checked one by one in chronological order until an operation is encountered for which the starting times deviate in the two schedules. This reflects the first point in time at which the dispatching rule takes a decision that is presumably not optimal. Then, a constrained scheduling problem $CP(x)$ is generated by fixing the starting times of all examined operations, including the first one for which the starting times differ, to the corresponding values of the rule solution. This problem is again optimally solved, resulting in a solution that contains exactly one suboptimal decision, as all subsequent decisions have been reoptimised. The fitness of the individual is then given by the difference of the objective function value of the optimal solution to the constrained problem to the objective function value of the optimal solution to the original problem, i.e. $Z^*(CP(x)) - Z^*(OP(x))$. Hence, the individuals with the highest fitness values are those problem instances for which the rule takes a single suboptimal decision with large negative consequences on the achievable objective function value. In contrast, a problem instance that is optimally solved by the tested

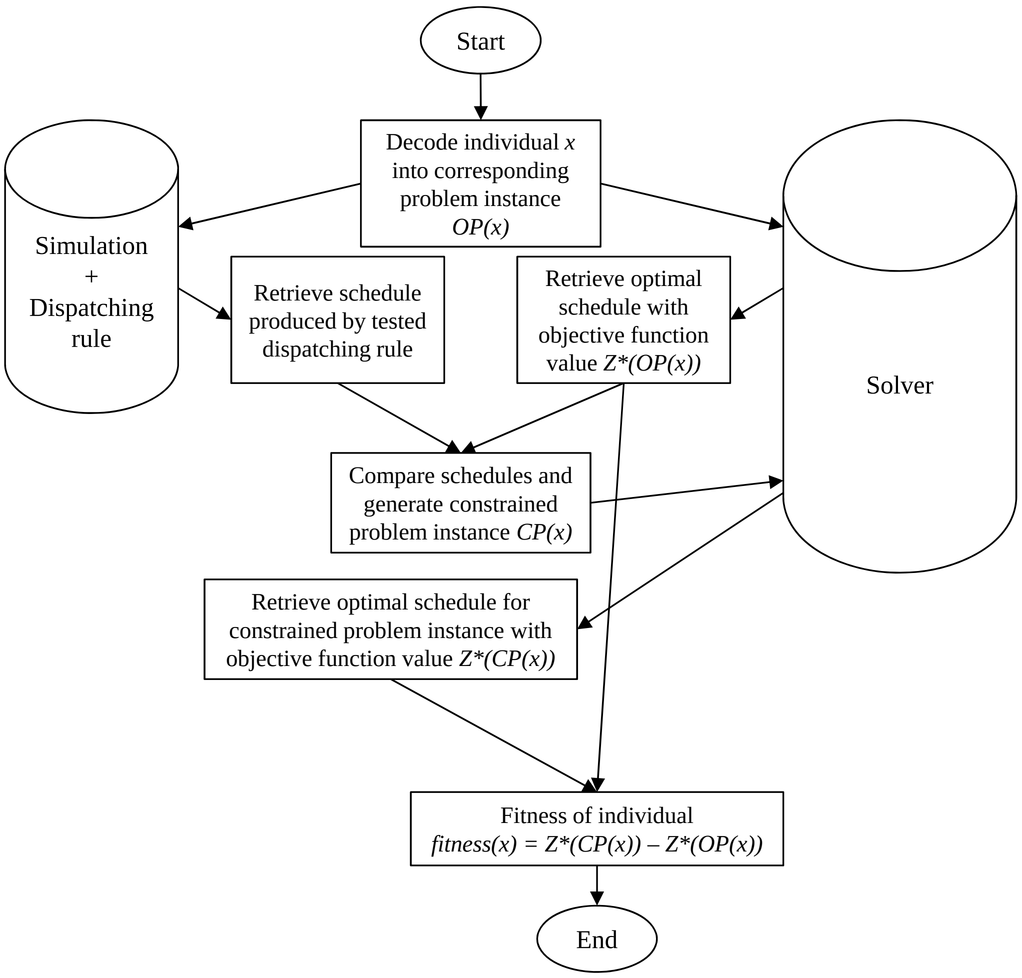

Start
Decode individual x into corresponding problem instance OP(x)
Simulation + Dispatching rule
Retrieve schedule produced by tested dispatching rule
Retrieve optimal schedule with objective function value Z*(OP(x))
Solver
Compare schedules and generate constrained problem instance CP(x)
Retrieve optimal schedule for constrained problem instance with objective function value Z*(CP(x))
Fitness of individual
fitness(x) = Z*(CP(x)) – Z*(OP(x))
End

Figure 5.2: Evaluation of job shop problem instances.

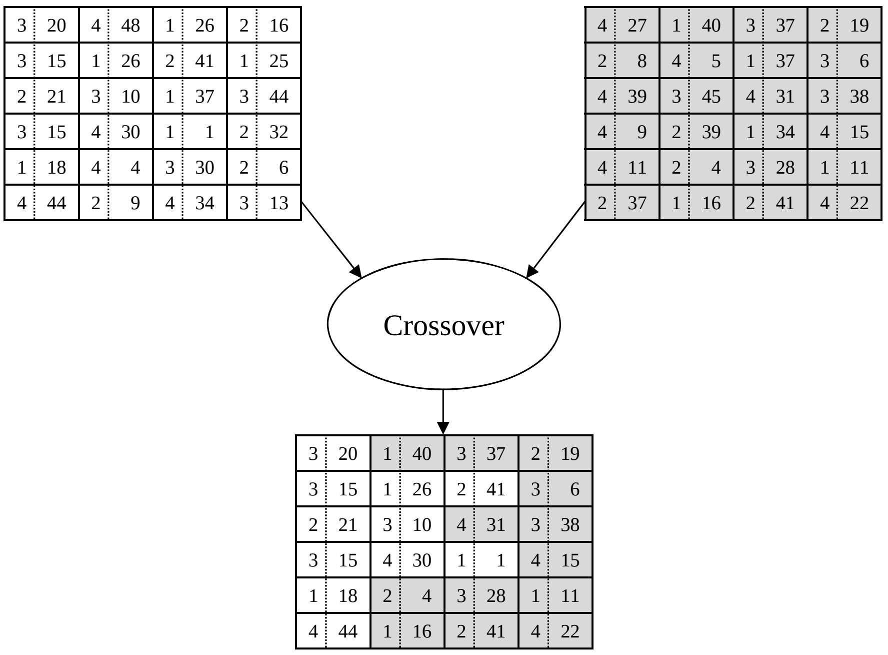
dispatching rule receives the worst possible fitness value of 0, since $Z^*(CP(x)) = Z^*(OP(x))$ in this case.

5.2.3 Selection

For the selection of promising individuals, the μ+λ breeder of ECJ is employed (Luke et al., 1998). This selection scheme chooses the μ individuals with the highest fitness values from the current population to produce λ children, with which together they form the population of the next generation. The population size is therefore $\mu + \lambda$. However, only one of two parents required for the crossover operation is taken from the set of the μ best individuals, the other is determined as the better of two randomly sampled individuals from the current population. Note that this selection scheme is elitist, i.e. the best individuals of each generation are guaranteed to survive into the next generation.

5.2.4 Reproduction

New individuals are created by successively applying the crossover and mutation operators described in the following. In general, it is the early and later operations of jobs that respectively compete for machine time within a problem instance. This motivates the design of a crossover that cuts through the processing orders of every job (at a random point) rather than through the job set, which would result in the exchange of whole jobs. Figure 5.3 presents an example of the crossover, where the white and grey part within the child originates from the first and second parent, respectively. Only crossover points that separate entries of two different operations are permitted. Moreover, a cut is never made at a point that would lead to a processing order with two consecutive operations on the same work centre, since this is infeasible. Thus, the only possible crossover point for Job 6 in the example is the one shown, as the other two options would result in Job 6 having two consecutive operations on Work Centre 2 or 4. If no feasible crossover point can be found at all for a job, its whole processing order is taken over from the first parent.



3	20	4	48	1	26	2	16
3	15	1	26	2	41	1	25
2	21	3	10	1	37	3	44
3	15	4	30	1	1	2	32
1	18	4	4	3	30	2	6
4	44	2	9	4	34	3	13

4	27	1	40	3	37	2	19
2	8	4	5	1	37	3	6
4	39	3	45	4	31	3	38
4	9	2	39	1	34	4	15
4	11	2	4	3	28	1	11
2	37	1	16	2	41	4	22

3	20	1	40	3	37	2	19
3	15	1	26	2	41	3	6
2	21	3	10	4	31	3	38
3	15	4	30	1	1	4	15
1	18	2	4	3	28	1	11
4	44	1	16	2	41	4	22

Figure 5.3: Example of the crossover operation for job shop problem instances. Each processing order is cut at a random point, and the parts before and after the crossover points of the two parents (illustrated in white and grey) are recombined to produce a new individual.

The mutation operator, illustrated in Figure 5.4, modifies individuals by adding a normally distributed random variable to operation processing times. The altered entries are encircled in Figure 5.4, e.g. the processing time of the third operation of Job 4 is changed from 1 to 0. Any operation with a processing time of 0 is treated as nonexistent by the EA and is removed during the decoding of the individual into a problem instance. Similarly, a previously nonexistent operation may be added if its processing time of 0 is increased to a positive value. Hence, the mutation operator is capable of changing the number of operations of a job. A final step further assures that all processing times of a mutated individual lie inside the permitted interval of $[p_{\min}, p_{\max}]$, setting values that are smaller than the lower bound to 0, or $p_{\min}$ if the former would create an infeasible processing order (with two subsequent operations on the same work centre), and values that exceed the upper bound to $p_{\max}$. Both crossover and mutation are applied with a certain probability, which are parameters of the EA that are specified in the following section.

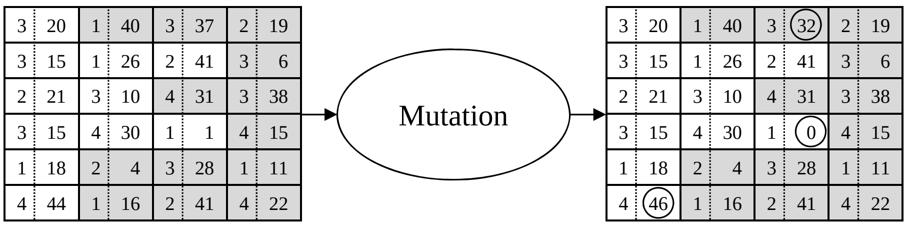


Figure 5.4: Example of the mutation operation for job shop problem instances. The processing times of operations are modified by adding a normally distributed random variable.

5.3 Application to a Benchmark Rule from the Literature

To illustrate the benefits of the proposed method, a step-by-step application is presented in this section. This starts with the selection of a problem and a benchmark rule from the literature, which the EA is configured to evolve difficult instances for. The resulting instances are analysed via Gantt charts for conditions that explain the failure of the rule. Based on the identified weaknesses of the benchmark rule, a new rule is developed that is able to overcome these limitations. At this point, the process can be started once again with the new rule being the one tested as demonstrated in Section 5.3.5.

5.3.1 Scenario Description

The method is applied to the PT+WINQ rule, which is one of the best rules for the $Jm|r_j|\overline{C}$ problem with balanced workloads (see Section 3.4.1). PT+WINQ is particularly suitable to show the potential of the proposed method as it is a very effective rule, which makes the task of coming up with a better rule generally challenging. Moreover, it is also a relatively simple rule, which facilitates the analysis of its decisions in the evolved instances.

The PT+WINQ rule is found to be one of the best rules for minimising mean flow time in the study by Rajendran and Holthaus (1999), who test a number of rules on several dynamic job shop problems, which are adopted as the scenario herein. Their shops contain ten, equally utilised work centres, each consisting of a single machine, where new jobs arrive at shops according to a Poisson process. In one shop, every job consists of ten operations that imply visiting each work centre exactly once, while in the other the number of operations per job varies between two and ten, and is drawn from a uniform distribution. The processing times of operations are generally drawn from a discrete uniform distribution ranging from 1 to 49.

5.3.2 Parameter Setting

The EA is configured to search for instances that resemble the given problem, and operation processing times are thus limited to the range of $p_{\min} = 1$ to $p_{\max} = 49$. Furthermore, the number of work centres c (and machines) is set to 4 and the number of jobs (which are all ready for processing from the start) n to 6, where each job consists of up to $l = 4$ operations. Such small problem sizes do not only reduce the running time of the solver (which is the most time-consuming part of the algorithm), but also facilitate the analysis of evolved instances. The other parameters of the EA are set in line with commonly used EA parameter settings, as indicated in Table 5.1.

Table 5.1: Parameter setting of the EA to generate difficult-to-solve job shop problem instances.

Parameter	
Number of generations	25
Population size	100
Initial population	Randomly generated
Selection parameter μ	20
Selection parameter λ	80
Crossover probability	100%
Mutation probability	5%
Number of work centres c	4
Number of jobs n	6
Maximum number of operations per job l	4
Minimum operation processing time $p_{\min}$	1
Maximum operation processing time $p_{\max}$	49
Scheduling objective	Minimise mean flow time

Since EAs are stochastic in nature, each run potentially returns a different problem instance. They may exploit different weaknesses of the dispatching rule and a user might want to look at the results of several runs. Hence, ten replications of the EA are run.

5.3.3 Analysis of Problem Instances

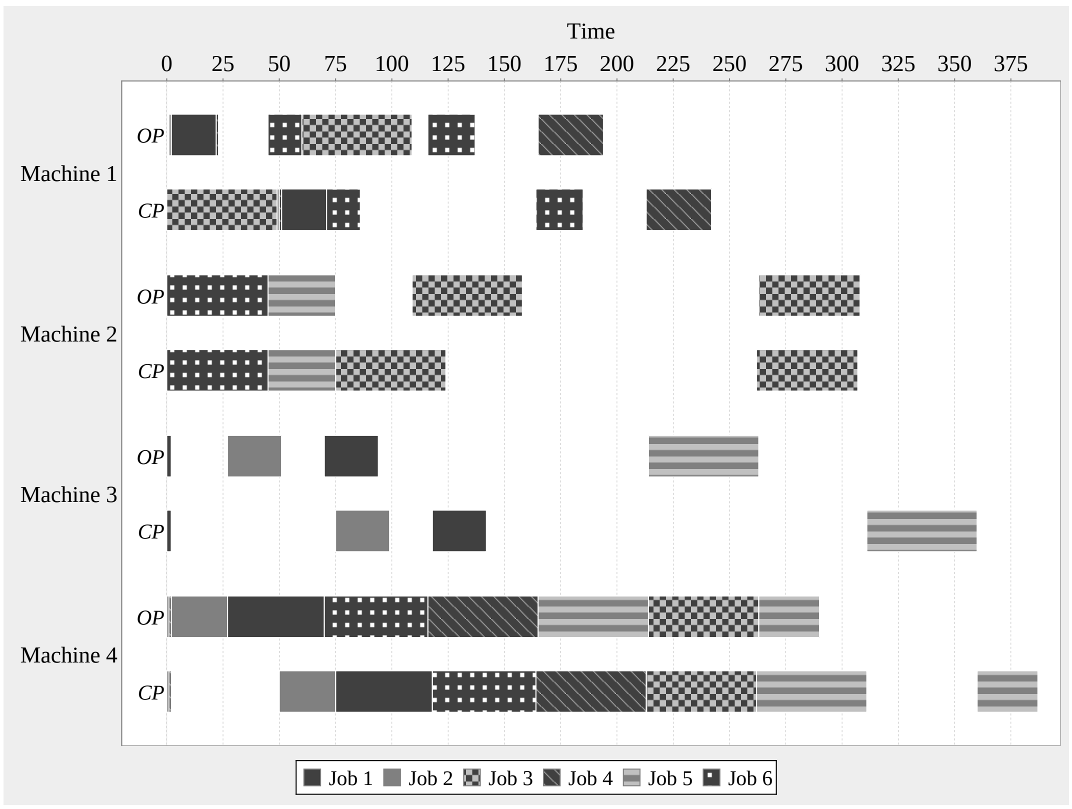
The evolved instances are analysed with the help of a Gantt chart generator that has been specifically designed for this purpose. Both the optimal machine schedule for the original problem *OP* and the one for the constrained problem *CP*, as described in Section 5.2.2, are displayed in a single chart. This facilitates a machine-by-machine comparison of the schedules and the identification of the suboptimal decision made by the tested rule. The decision point can be easily found by checking the schedules along the time axis for the first difference. The effects of the decision of the rule can then be gathered from an analysis of the two schedules following this first difference, after which both schedules represent an optimal solution to the residual problem.

Figure 5.5 depicts one of the problem instances evolved by the EA. The earliest difference between the two schedules for this instance can be observed at time 0 on Machine 1, where PT+WINQ starts processing Job 3 (see *CP* schedule) as it is the only one waiting in the queue (Job 1 requires Machine 3 for its first operation, Job 2 and Job 4 both require Machine 4 first, while Job 5 and Job 6 require Machine 2). However, this decision leads to substantial delays for Job 1 and Job 2, which complete their first operations on Machine 3 and Machine 4 after only two and one unit(s) of time, respectively, arrive at Machine 1 (which they both require for their second operation) shortly after the start of the operation of Job 3, and have to wait for that operation (which occupies the machine for 49 time units) to finish. The delayed processing of Job 1 and Job 2 at Machine 1 in turn results in a temporary starvation of Machine 4, i.e. the machine with the highest workload by far, and consequently a large increase of mean flow time. In this case, the workload of a machine is measured by the sum of processing times of all operations to be performed on it, which respectively add up to 136, 169, 99 and 290 for the four machines in the given problem instance. The optimal decision (indicated by the *OP* schedule), on the other hand, is to wait for the completion of the first operations of Job 1 and Job 2 and prioritise them at Machine 1 so that the bottleneck, i.e. Machine 4, does not starve.

An issue with this problem instance and other evolved instances is that the distribution of workloads among the machines is not representative of the given problem, which is characteristed by a balanced workload. In consequence, insights gained from the analysis of these instances are unlikely to be of any value for the development of a rule that performs better on the actual problem. To focus the search of the EA on more balanced problem instances, its fitness function is modified such that the imbalance of workloads is penalised. More specifically, a penalty that is equal to a tenth of the variance of workloads over machines is introduced. To illustrate, the variance of workloads of the problem instance shown in Figure 5.5 is 5137.25, resulting in a penalty of $0.1 \times 5137.25 = 513.725$, which is subtracted from the original fitness value to give the final fitness value of the individual. Using this modified fitness function, another ten replications of the EA are run.

Job	Op. 1		Op. 2		Op. 3		Op. 4	
1	3	2	1	20	4	43	3	24
2	4	1	1	1	4	25	3	24
3	1	49	2	49	4	49	2	45
4	4	1	1	1	4	49	1	29
5	2	30	4	49	3	49	4	27
6	2	45	1	15	4	46	1	21

(a) Problem instance



(b) Optimal and constrained machine schedules

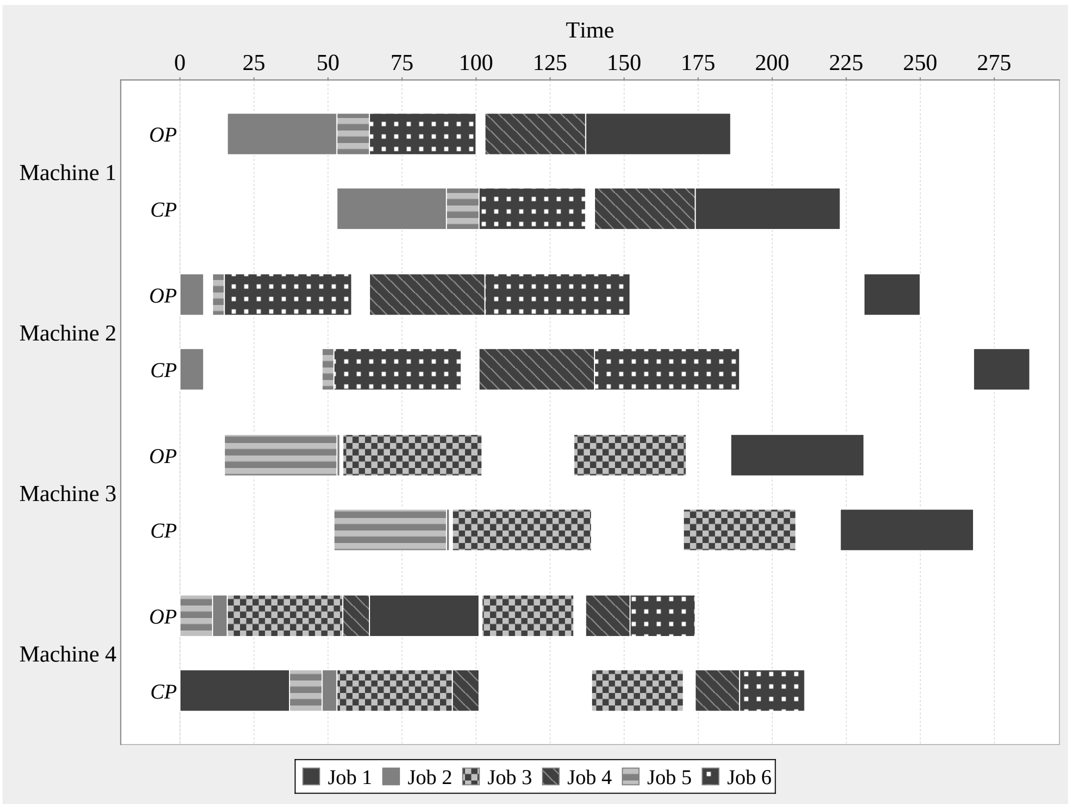
Figure 5.5: Problem instance solved ineffectively by PT+WINQ with respect to the minimum mean flow time objective — a premature start of a long operation delays arrival of other jobs at the bottleneck machine, which is starving consequently.

Figure 5.6 presents the individual with the highest fitness value, discovered in these ten additional replications. The first thing noticable is the nearly perfect balance of machine workloads, which is also apparent in the other, subsequently discussed problem instances, as a result of the introduced penalty. The first difference between the two schedules for this instance can be observed at time 0 on Machine 4. The PT+WINQ rule chooses to process Job 1 despite its long operation processing time ($p_{1,i} = 37$) because it subsequently proceeds to Machine 1, which has an empty queue at the time ($W_{z_{1,i+1}} = 0$). In the optimal solution, the shorter operation of Job 5 ($p_{5,i} = 11$) is processed first, which is not prioritised by the rule due to the large work content of Machine 2, required for the next operation of Job 5 ($W_{z_{5,i+1}} = 51$). The decision of the rule follows from a comparison of the priority indices of the two jobs, as $I_1^{\text{PT+WINQ}} = -(37 + 0) = -37 > -62 = -(51 + 11) = I_5^{\text{PT+WINQ}}$ (see Section 3.4.1), and it is a bad one for two reasons. First of all, apart from Job 5 there are two more jobs with short operations awaiting service on Machine 4, with another arriving shortly after. All three jobs are largely delayed as the machine is occupied with the operation of Job 1 for a significant amount of time, thereby increasing the local mean flow time at Machine 4. Second, the postponement of Job 2 and Job 5 leads to extensive idle times on Machine 1 and Machine 3 later on, as there is no other job available that could be processed on these machines. Both factors increase the (global) mean flow time.

Another problem instance evolved by the EA for the PT+WINQ rule is shown in Figure 5.7. The suboptimal scheduling decision of PT+WINQ again occurs on Machine 4 at time 0. As in the previous instance, the rule assigns a higher priority to the longer job, i.e. Job 3 ($p_{3,i} = 49$ compared to $p_{4,i} = 30$), due to a large difference in the WINQ values ($W_{z_{3,i+1}} = 18$ for Machine 3, $W_{z_{4,i+1}} = 45$ for Machine 2), which leads to some short jobs being delayed at Machine 4. Moreover, the WINQ component shows to be incapable, in this case, of balancing the work content among machines in order to avoid idle times, which is its original purpose (see Section 3.4.1). This is due to the fact that the work contents of Machine 2 and Machine 3 change to a great extent while Machine 4 is busy with processing Job 3. When the operation is finished, i.e. after 49 time units, it is Machine 2 that is now facing an empty queue after its completion of Job 1. Job 1 has in turn proceeded to Machine 3, thereby largely increasing its work content. The WINQ component does not consider these developments and in fact turns out to be counterproductive in its attempt to balance work contents in this case. The optimal decision, on the other hand, is to process Job 4 on Machine 4 first, which consequently keeps Machine 2 busy.

Job	Op. 1		Op. 2		Op. 3		Op. 4	
1	4	37	1	49	3	45	2	19
2	2	8	4	5	1	37	3	1
3	4	39	3	47	4	31	3	38
4	4	9	2	39	1	34	4	15
5	4	11	2	4	3	38	1	11
6	2	43	1	36	2	49	4	22

(a) Problem instance

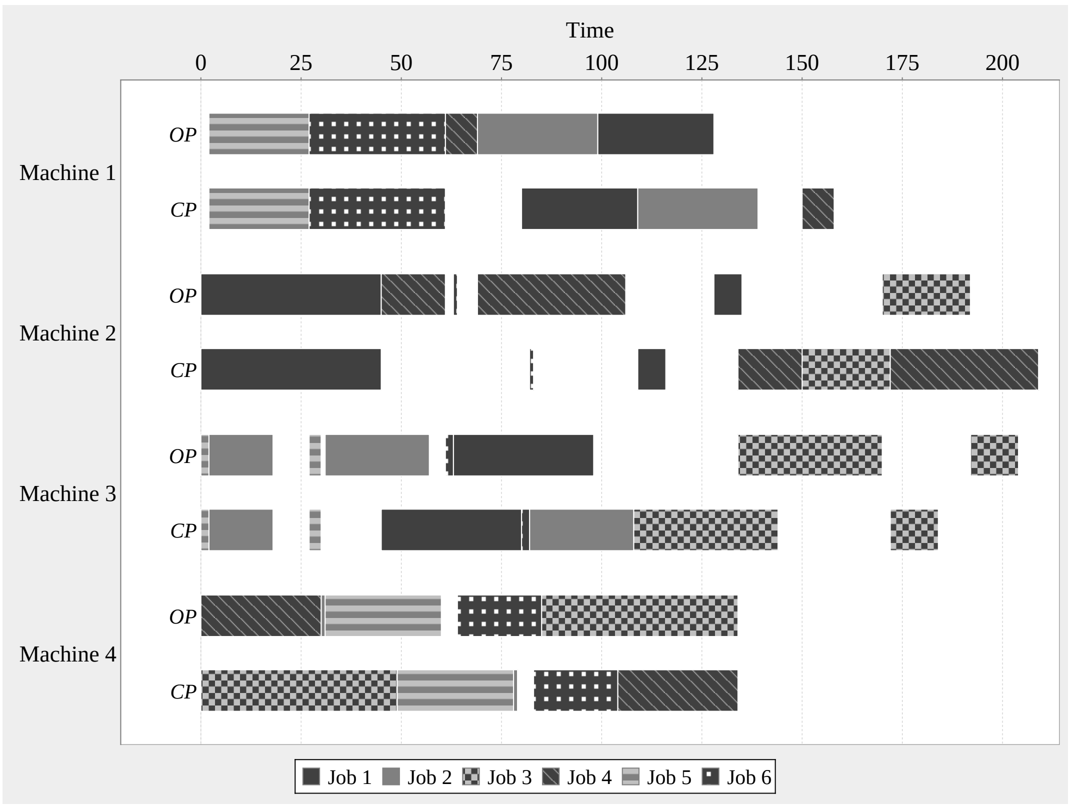


(b) Optimal and constrained machine schedules

Figure 5.6: Problem instance solved ineffectively by PT+WINQ with respect to the minimum mean flow time objective — prioritisation of a job with long operation delays jobs with short operations and leads to idle times at other machines.

Job	Op. 1		Op. 2		Op. 3		Op. 4	
1	2	45	3	35	1	29	2	7
2	3	16	4	1	3	26	1	30
3	4	49	3	36	2	22	3	12
4	4	30	2	16	1	8	2	37
5	3	2	1	25	3	3	4	29
6	1	34	3	2	2	1	4	21

(a) Problem instance



(b) Optimal and constrained machine schedules

Figure 5.7: Problem instance solved ineffectively by PT+WINQ with respect to the minimum mean flow time objective — focus of WINQ component on current work contents causes a decision that subsequently leads to unbalanced work contents and machine idle times.

Based on the findings from the analysis of the above problem instances, the following conclusions are drawn for the design of a better rule:

1. The increase of the local mean flow time at a given work centre that results from the priorisation of a long operation should be measured more precisely, taking into account all shorter operations that are delayed.
2. The component that assesses the conditions at downstream work centres should assess the expected work content at the time the respective job arrives at that work centre, not the current work content.
3. The focus should be on idle time avoidance rather than on balancing the work content of the different work centres.

5.3.4 Design of a New Dispatching Rule

A new dispatching rule, developed on the basis of the above conclusions, is presented in this section. The rule consists of two components. The first component, called Increased Flow Time (IFT), is derived from the first conclusion, and its index is defined as

$$IFT_j = \sum_{y \in N^{\mathrm{Q}}, p_{y,i} < p_{j,i}} \left(p_{j,i} - p_{y,i} \right), \tag{5.1}$$

where $p_{j,i}$ denotes the processing time of the imminent operations of job j, and N^{Q} the set of queueing jobs. The IFT_j index measures the increase of total flow time at the local work centre that would result from prioritising job j, which is equal to 0 for the job with the shortest operation.

The second component, called Utilised Idle Time (UIT), is derived from the other two conclusions and estimates how much machine idle time a job can fill in with its subsequent operation if its imminent operation is started immediately. It incorporates an extended WINQ component, in which the operation processing times of all look-ahead jobs, which are expected to arrive at the respective work centre during the execution of the imminent operation of a considered job j, are added to the current work content of the work centre. This refers to jobs that require the same work centre for their next operation as job j and take less time to finish their current operation. The complete index of the UIT component is given by

$$UIT_j = \begin{cases} p_{j,i+1} & \text{if } W^{\mathrm{x}}_{z_{j,i+1}} - p_{j,i} \leq 0 \ \wedge \ t + p_{j,i} + p_{j,i+1} \leq r_{y,i}, \\ r_{y,i} - \left(t + p_{j,i} \right) & \text{if } W^{\mathrm{x}}_{z_{j,i+1}} - p_{j,i} \leq 0 \ \wedge \ t + p_{j,i} + p_{j,i+1} > r_{y,i}, \\ p_{j,i+1} - \left(W^{\mathrm{x}}_{z_{j,i+1}} - p_{j,i} \right) & \text{if } 0 < W^{\mathrm{x}}_{z_{j,i+1}} - p_{j,i} \leq p_{j,i+1} \ \wedge \ t + p_{j,i} + p_{j,i+1} \leq r_{y,i}, \\ r_{y,i} - \left(t + W^{\mathrm{x}}_{z_{j,i+1}} \right) & \text{if } 0 < W^{\mathrm{x}}_{z_{j,i+1}} - p_{j,i} \leq p_{j,i+1} \ \wedge \ t + p_{j,i} + p_{j,i+1} > r_{y,i}, \\ 0 & \text{otherwise,} \end{cases} \tag{5.2}$$

where $W^{\mathrm{x}}_{z_{j,i+1}}$ denotes the extended work content of the work centre $z_{j,i+1}$ required for the next operation of job j, as defined above, $r_{y,i}$ indicates the arrival time of the look-ahead job y that is expected to arrive first at that work centre after job j, $p_{j,i+1}$ denotes the processing time of the next operation of job j, and t the current time. Note that $r_{y,i}$ is equal to infinity if there is no such job y that satisfies the criteria. The UIT_j index accounts for the fact that other machines decrease their work content by up to $p_{j,i}$ time units while the current operation of job j is being processed. Thus, the next machine on the route of job j will possibly be starving at the time of its arrival if $W^{\mathrm{x}}_{z_{j,i+1}} - p_{j,i} \leq 0$, in which case the whole processing time of the next operation of job j can be utilised to fill in idle time, represented by the first case in Equation (5.2). However, if another job is expected to arrive before the subsequent operation of job j is completed, only the gap between the arrival of these two jobs may be filled in by job j (the second case). On the other hand, if $W^{\mathrm{x}}_{z_{j,i+1}}$ exceeds $p_{j,i}$, there will be work left of the amount $W^{\mathrm{x}}_{z_{j,i+1}} - p_{j,i}$ when job j arrives at its next machine and, depending on the time of the following job arrival, only $p_{j,i+1} - W^{\mathrm{x}}_{z_{j,i+1}} + p_{j,i}$ (the third case) or $r_{y,i} - t - W^{\mathrm{x}}_{z_{j,i+1}}$ (the fourth case) units of idle time can be used. Finally, if the remaining work at the point of arrival of job j is greater than the duration of its operation on that machine, i.e. $W^{\mathrm{x}}_{z_{j,i+1}} - p_{j,i} > p_{j,i+1}$, no idle time can be utilised by job j at all (the fifth case). Although UIT does not give a guarantee on the amount of idle time that is actually used by the subsequent operation of a job, it is certainly a more suitable measure than WINQ to prevent machines from running idle.

With its components defined, the priority index of the new rule, named IFT−UIT, is simply computed as

$$I_j^{\mathrm{IFT-UIT}} = -\left(IFT_j - UIT_j\right). \tag{5.3}$$

Thus, the rule prioritises jobs that lead to a smaller increase of local flow time and can utilise more idle time, and therefore seeks to efficiently resolve queues and avoid idle times at the same time.

To illustrate the operation of the IFT−UIT rule, it is applied to the decision point at time 0 at Machine 4 of the problem instance from Figure 5.6. The available jobs and the attributes that are relevant for the determination of their priority indices are listed in Table 5.2. The IFT_j index of each job is calculated according to Equation (5.1), e.g. the index of Job 1 is

Table 5.2: Determination of job priorities according to the IFT−UIT rule for the problem instance from Figure 5.6 ($r_{y,i} = \infty$ for all jobs).

j	$p_{j,i}$	IFT_j	$W^{\mathrm{x}}_{z_{j,i+1}}$	$p_{j,i+1}$	UIT_j	$I_j^{\mathrm{IFT-UIT}}$
1	37	54	0	49	49	-5
3	39	60	0	47	47	-13
4	9	0	51	39	0	0
5	11	2	51	4	0	-2

equal to the sum of the differences of the processing time of its imminent operation and the processing time of the shorter operations of Job 4 ($37 - 11 = 26$) and Job 5 ($37 - 9 = 28$), i.e. $IFT_1 = 26 + 28 = 54$. The UIT_j indices are determined using Equation (5.2). To illustrate, Job 1 requires Machine 1 for its subsequent operation (see Figure 5.6), which has an empty queue at the time, with no look-ahead jobs expected to arrive during the execution of the imminent operation of Job 1 ($W^{\mathrm{x}}_{z1,i+1} = 0$) or after ($r_{y,i} = \infty$). Thus, it is expected that Machine 1 will be starving at the time of the arrival of Job 1 and that the whole 49 units of processing time of its next operation can be utilised to fill in idle time if its imminent operation is started immediately ($UIT_1 = 49$). In contrast, Job 5 requires Machine 2 for its subsequent operation, which has a work content of 51, with Job 2 and Job 6 both waiting to be processed (see Figure 5.6). Since the remaining work at the prospective point of arrival of Job 5 (after 11 time units) will still be greater than the duration of its operation on that machine ($51 - 11 > 4$), no idle time can be utilised by Job 5 at all ($UIT_5 = 0$). Subtracting the IFT_j from the UIT_j indices, it is easy to see that Job 4 has the highest $I_j^{\mathrm{IFT-UIT}}$ index and is therefore chosen by the IFT−UIT rule.

5.3.5 Applying the Method to the New Rule

Although the IFT−UIT rule already outperforms PT+WINQ (see Section 5.4), the process does not have to stop here. Instead, it can continue with testing the newly developed rule to check for smaller or newly introduced flaws. The EA is thus applied again, but this time to search for problem instances that the IFT−UIT rule cannot solve properly. All other parameter settings remain the same as before, and ten replications are run again.

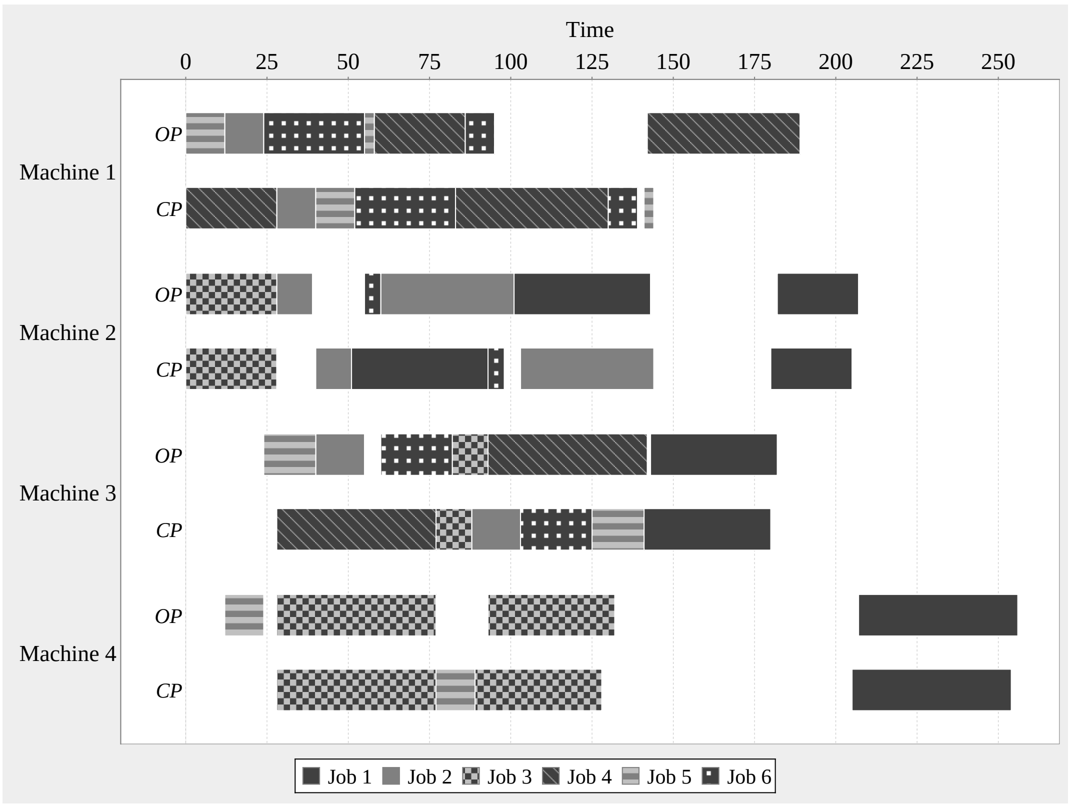
One of the evolved problem instances is depicted in Figure 5.8. Of the four jobs queueing at Machine 1 at time 0, IFT−UIT chooses Job 4 since its subsequent operation is estimated to utilise 49 units of idle time of Machine 3. However, Job 4 consists of three long operations and favouring a single job with long operations over several jobs with shorter operations increases the mean flow time. Furthermore, the potential benefit of utilising an extensive amount of idle time of Machine 3 is not realised as this period can be largely filled with several other jobs, provided that they are prioritised at Machine 1 as in the optimal schedule.

One quick fix to deal with this weakness of the IFT−UIT rule is to add a term to its priority index that penalises jobs that have a long next operation, similar to how Holthaus and Rajendran (2000) extend the PT+WINQ to the 2PT+WINQ+NPT rule (see Section 3.4.1). The priority index of this modified rule, named IFT−UIT+NPT, is then given by

$$I_j^{\mathrm{IFT-UIT+NPT}} = -\left(IFT_j - UIT_j + p_{j,i+1}\right). \tag{5.4}$$

Job	Op. 1		Op. 2		Op. 3		Op. 4	
1	2	42	3	39	2	25	4	49
2	1	12	2	11	3	15	2	41
3	2	28	4	49	3	11	4	39
4	1	28	3	49	1	47	4	0
5	1	12	4	12	3	16	1	3
6	1	31	2	5	3	22	1	9

(a) Problem instance



(b) Optimal and constrained machine schedules

Figure 5.8: Problem instance solved ineffectively by IFT−UIT with respect to the minimum mean flow time objective — IFT−UIT delays short jobs by prioritising a job with a long subsequent operation due to the large amount of idle time it is estimated to fill in, which turns out to be inaccurate.

5.4 Experimental Results

The newly developed rules are evaluated by replicating the experiments of Rajendran and Holthaus (1999). In particular, the rules are applied to the two job shops described in Section 5.3.1, operating under two different levels of utilisation, namely 80% and 95%. For these utilisation levels the warm-up period has been determined to comprise the completion of 1000 and 7000 jobs, respectively. The performance measurements are generally based on a sample size of $n = 50000$ jobs per simulation run.

Regarding the selection of benchmarks, it is obviously necessary to include PT+WINQ in the performance comparison to assess whether the analysis of its weaknesses has actually led to the development of a better rule. Moreover, SPT and 2PT+WINQ+NPT are selected as benchmark rules due to their reported strong performance for the $Jm|r_j|\overline{C}$ problem (see Section 3.4.1). The PT+WINQ and 2PT+WINQ+NPT rules are further modified by replacing the WINQ component by the extended version that is used within the UIT component, providing two more benchmarks. The job-based relative of SPT, the LWKR rule, and the simple FCFS rule complete the set of benchmark rules.

Table 5.3 presents the results of the simulation experiments. The values in bold indicate the best performance achieved by any rule for the respective problem, while those marked with the same superscript, e.g. the mean flow time values of SPT and PT+WINQ for the problem with 2–10 operations per job and 80% utilisation, are not significantly different from each other according to the Bonferroni-Dunn test, at the overall 5% significance level. The results are consistent with previous findings in the literature in that SPT outperforms LWKR and that

Table 5.3: Mean flow time performance of newly designed and benchmark rules for ideal job shops at different utilisation levels. Differences between performance values marked with the same superscript are not significant at the overall 5% significance level. The best performance values are in bold.

Dispatching rule	Mean flow time			
Shop type	2–10 ops. per job		10 ops. per job	
Utilisation	80%	95%	80%	95%
FCFS	509.4	1752.5	819.0	2592.9
LWKR	492.9	1903.3	770.3	2672.6
SPT	384.8^a	1000.9	616.7^a	1425.0^a
PT+WINQ	384.6^a	961.6	616.2^a	1421.8^a
PT+WINQ^x	383.4	955.6	613.5	1409.3
2PT+WINQ+NPT	381.4	898.2	608.9	1317.4
2PT+WINQ^x+NPT	378.5	889.1^a	602.7	1296.5
IFT−UIT	372.9	887.0^a	596.5	1274.7
IFT−UIT+NPT	**363.3**	**835.9**	**582.5**	**1211.7**

PT+WINQ is at least as effective as SPT, with 2PT+WINQ+NPT dominating both. Furthermore, the extended WINQ component, denoted by WINQ^x in Table 5.3, is shown to generally improve the mean flow time performance of rules incorporating WINQ, though improvements are relatively minor (around 1%). Most notably, both IFT−UIT and IFT−UIT+NPT prove to dominate all other rules, where IFT−UIT+NPT is the best rule in every single case, achieving a mean flow time reduction of 4–8% over the previous best rule, 2PT+WINQ+NPT. Hence, it can be concluded that the approach of employing an EA to search for difficult problem instances has been successful in prompting the design of a superior rule for the given problem, i.e. the $Jm|r_j|\overline{C}$ problem with balanced workloads.

In their paper, Rajendran and Holthaus (1999) further study the impact of the flow pattern on the relative effectiveness of dispatching rules by altering their problems so that jobs visit work centres in increasing order of the work centre index, with other problem parameters left unchanged. Consequently, the shop in which each job visits each work centre exactly once becomes a flow shop, while the other one becomes a flow-dominant job shop. Although the IFT−UIT and IFT−UIT+NPT rules are designed for ideal job shops and thus cannot be expected to perform well for these problems, the experiments are replicated herein to assess the robustness of the rules to different flow patterns.

Table 5.4 presents the results of these additional experiments, which show both rules to be surprisingly robust. In fact, the IFT−UIT+NPT rule continues to yield the lowest mean flow time, even for the flow shop problems, which is remarkable considering that its design is based on the analysis of instances of the job shop problem.

Table 5.4: Mean flow time performance of newly designed and benchmark rules for flow-dominant job shop and flow shop at different utilisation levels. Differences between performance values marked with the same superscript are not significant at the overall 5% significance level. The best performance values are in bold.

Dispatching rule	Mean flow time			
Shop type	Flow.-dom. job shop		Flow shop	
Utilisation	80%	95%	80%	95%
FCFS	464.6	1656.0	619.8	2051.6
LWKR	424.8	1408.8	553.3	1545.5
SPT	358.8	946.6	510.1^a	1202.5^b
PT+WINQ	357.1^b	912.4	510.1^a	1202.5^b
PT+WINQ^x	356.5^a	907.6	510.1^a	1202.5^b
2PT+WINQ+NPT	357.6	891.1	512.7^b	1275.5^c
2PT+WINQ^x+NPT	356.2^a	881.5	512.7^b	1275.5^c
IFT−UIT	356.4^{a,b}	868.0	523.5	**1157.0**^{a}
IFT−UIT+NPT	**345.6**	**842.6**	**503.0**	**1154.3**^{a}

5.5 Critical Appraisal and Future Work

In summary, the method presented in this chapter has led to the development of superior dispatching rules for a well-studied scheduling problem, which demonstrates the potential of using an EA to reveal the weaknesses of current benchmark rules. However, there are a few issues with regard to both the generation and the analysis of problem instances that limit the usefulness of the method.

The first point concerns the computational time needed to generate a problem instance. While in the above application the runtime for a replication of the EA has been less than two hours (on a 2.66 GHz Intel Core 2 Duo with 2 GB RAM), which is certainly acceptable, the runtime will quickly become a restrictive factor if the integrated solver is required to solve more complex scheduling problems, e.g. involving sequence-dependent setup times or batch processing. One way to deal with this issue would be to stop the solver after a certain maximum time (if it has not returned the optimum yet) and return the best feasible solution found so far. Alternatively, some other type of solution method such as a metaheuristic could be used to solve the problem instances. In any case, the scale of the problem instances have to be rather small to be able to generate good solutions in a reasonable amount of time, raising the question of the representativeness of these instances for the kind of large-scale dynamic problems to which dispatching rules are typically applied.

In Section 5.3, problem instances with only four work centres and six jobs have revealed weak points in the design of dispatching rules, whose elimination has resulted in rules with a better performance on dynamic problems with ten work centres. This is an indication for that the conditions found in the static instances are likewise encountered in the dynamic problems. On the other hand, as illustrated by the instance depicted in Figure 5.5 and the need to introduce a penalty for workload imbalance, the generation of representative and insightful instances is not trivial. In fact, there is no guarantee that modifying the logic of a dispatching rules based on the insights gained from a specific instance will actually improve the performance of the rule. Thus, the development of dispatching rules remains a trial-and-error process with this method, though becoming more systematic.

Another reason for configuring the EA to search for small problem instances is that larger instances are harder to analyse for a human expert. Given that the potential to develop more effective dispatching rules appears to be bigger for complex problems, the requirement for instances to be easy to analyse clearly presents another limitation of the method in its current form. Hence, better user support in the analysis phase, e.g. in form of a user interface that automatically identifies critical operations that cannot be delayed without impact on the objective function value, is an important area of future work. A step further would be to automate the analysis of instances by employing a machine learning algorithm. However, such an algorithm would need to be able to extract the context of suboptimal rule decisions that explains both the causes and effects of the decisions, which seems rather challenging.

Chapter 6

Automatic Rule Generation by Genetic Programming [1]

The characterisation of the development of dispatching rules as an iterative process suggests the employment of EAs to automate this search for an effective rule. Such methods that operate on a search space of heuristics are commonly referred to as hyper-heuristics (Burke et al., 2010). Since dispatching rules are essentially defined by their priority indices, i.e. mathematical expressions, a particularly suitable type of EA for the generation of dispatching rules appears to be Genetic Programming (GP) as it is well designed for evolving mathematical expressions (see, e.g. Poli et al., 2008, Chapter 4).

This chapter presents a GP hyper-heuristic that generates composite dispatching rules from basic job and machine attributes. The next section reviews related work, followed by a detailed description of the algorithm in Section 6.2. The hyper-heuristic is applied to some complex problems from semiconductor manufacturing as described in Section 6.3, and the performance and robustness of the evolved rules is assessed in Sections 6.4 and 6.5, respectively. The chapter concludes with a critical appraisal of the method and some suggestions for future work.

6.1 Related Work

The basic idea to automate the development of dispatching rules by means of a hyper-heuristic has been around for a long time. An early paper on this topic is the one by Hershauer and Ebert (1975), who combine a deterministic pattern search algorithm with simulation to generate a composite rule that minimises a specific cost function of a dynamic job shop. They test their procedure with different cost functions and find that its effectiveness strongly depends

[1]This method is published in C. W. Pickardt, T. Hildebrandt, J. Branke, J. Heger, and B. Scholz-Reiter (2013), Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems, *International Journal of Production Economics* 145(1), 67–77. However, the experimental results presented in this chapter deviate from those reported in the paper due to a slightly different experimental setup.

on aspects such as the predefined format of the priority indices of the composite rules and the starting solution of the search. These findings further motivate the use of GP for the automatic generation of dispatching rules as it is flexible to evolve priority indices of different formats and lengths, and like any EA operates on a set of (starting) solutions.

Several researchers have developed GP-based hyper-heuristics to generate dispatching rules for various scheduling problems. Atlan et al. (1994) use GP to evolve rules for particular instances of the $Jm||C_{\max}$ problem. They find that the evolved rules yield (near-)optimal solutions to the respective (training) instance and outperform a simple benchmark rule on a wider set of instances, given that they are not too different from the training instance. Miyashita (2000) evolves dispatching rules for a dynamic job shop problem by means of GP, which show to be more effective than some common rules from the literature. Dimopoulos and Zalzala (2001) address the $1||\overline{T}$ problem in this way and obtain rules that prove to outperform EDD and SPT. Their results further highlight the importance of ensuring that the training instances are representative of the problem(s) that the rules are supposed to solve. Geiger et al. (2006) develop a GP hyper-heuristic to generate dispatching rules for a range of single machine problems and report that it is able to discover the optimal rules where they are known, e.g. SPT for the $1||\overline{C}$ problem and EDD for the $1||T_{\max}$ problem, and very competitive ones for all other problems. In particular, the rules evolved for the $1|r_j|\overline{C}$ problem are generally found to reproduce the performance of SPT, while those evolved for the $1||\overline{T}$ and $1|r_j|\overline{T}$ problem show to be similarly effective as ATC, MDD and COVERT. In a follow-up paper, Geiger and Uzsoy (2008) apply their hyper-heuristic to some single batch processing machine problems. Again, the hyper-heuristic is shown to be effective, evolving rules that perform as good as the optimal BSPT rule for the $1|B_f|\overline{C}$ problem, as well as rules that outperform BSPT on the $1|r_j, B_f|\overline{C}$ problem, and BATC and BEDD on the $1|B_f|\overline{T}$ and $1|r_j, B_f|\overline{T}$ problem. Jakobović and Budin (2006) employ GP to evolve dispatching rules for the $1|r_j|\overline{wT}$ and $Jm||\overline{wT}$ problem and find that the evolved rules achieve a better performance than some variants of the ATC rule amongst others. In another paper, Jakobović et al. (2007) address several parallel machine problems, where the GP rules again show to outperform the benchmark rules in every case, including ATC for the $Qm||\overline{wT}$ problem, X-ATC for the $Qm|r_j|\overline{wT}$ problem and ATCS for the $Qm|s_{ef}|\overline{wT}$ problem. Jakobović and Marasović (2012) extend this work by examining different parameter settings of the hyper-heuristic and by testing it on a wider range of scheduling problems. Their results largely confirm the earlier findings in that the hyper-heuristic is generally able to evolve rules that are more effective than the best benchmark rules, i.e. ATC and its variants for the $1||\overline{wT}$, $1|r_j|\overline{wT}$, $1|s_{ef}|\overline{wT}$ and $Jm||\overline{wT}$ problem. Moreover, the performance of the hyper-heuristic is found to improve if a higher number of training instances is used in the evaluation of individuals. Tay and Ho (2007, 2008) use GP to generate dispatching rules for some flexible job shop problems and report that the evolved rules perform slightly better than several simple rules from the literature.

In some of these studies, GP has been further employed to evolve several dispatching rules simultaneously, where each rule is tailored to a certain (group of) work centre(s). Geiger et al. (2006) propose a simple extension of their hyper-heuristic for multi-stage problems that involves the generation of a different rule for every work centre in the shop. They apply this hyper-heuristic to the $F2||C_{\max}$ problem, for which it shows to generate rule sets that yield close-to-optimal solutions. Miyashita (2000) develops a GP hyper-heuristic that is based on a predetermined classification of work centres into bottlenecks and non-bottlenecks and evolves one rule for each class of work centres. The hyper-heuristic is tested on two job shop problems with five work centres, of which one or two, respectively, are bottlenecks, and found to be more effective than a GP hyper-heuristic that evolves a single rule for all work centres or one that evolves a different rule for every work centre. Encouraged by these results, Jakobović and Budin (2006) design a GP hyper-heuristic that optimises the classification of work centres while searching for good dispatching rules. More specifically, each individual consists of three mathematical expressions, where one of them is a function of attributes relating to the workload of a work centre that determines which of the two dispatching rules, encoded by the other two expressions, to apply. Jakobović and Budin report that the best rule set generated for the $Jm||\overline{wT}$ problem in 20 replications of this hyper-heuristic performs significantly better than the best rule evolved in 20 replications of a hyper-heuristic that generates a single rule for all work centres.

GP has also been successfully employed for the automatic generation of other types of scheduling heuristics. Yin et al. (2003) use GP to evolve so-called predictive heuristics, which aim to construct schedules that are robust to unpredictable breakdowns of machines. Vázquez-Rodríguez and Ochoa (2011) create variants of the NEH flow shop heuristic by letting GP modify the rule that is used for sorting the jobs in the initial step of the heuristic. Nguyen et al. (2012) develop a GP hyper-heuristic for the simultaneous generation of dispatching and due date assignment rules that is based on the principles of cooperative coevolution, in which one population of dispatching rules coevolves with another population of due date assignment rules.

Other hyper-heuristics that are not based on GP have also been proposed for the generation of dispatching rules. Nie et al. (2010) propagate the use of Gene Expression Programming (GEP), a close relative of GP, to evolve dispatching rules and find it to be slightly more effective than GP. However, they apply both hyper-heuristics with a parameter setting that is optimised for the GEP hyper-heuristic, thus introducing a bias in favour of the latter. Eguchi et al. (2008) design a hyper-heuristic that is composed of a neural network and a simulated annealing metaheuristic, but hint that its effectiveness is strongly dependent on a suitable parameter setting and starting solution. Olafsson and Li (2010) combine an EA with a decision tree algorithm, where the role of the EA is to determine a subset of training instances so that the dispatching rule that results from the application of the decision tree algorithm to this subset performs well on the test instances. This implies that the rules are optimised on the test instances and are therefore likely to be overfitted to them. El-Bouri et al. (2000) employ a neural network algorithm to learn dispatching rules from optimal schedules, while Ingimundardottir and Runarsson (2011) use a linear classification algorithm in a similar manner. However, such an approach presupposes the availability of (near-)optimal solutions, and thus is not practical for dynamic problems.

Overall, this literature review indicates that the generation of dispatching rules by means of GP is very promising. However, a shortcoming of the previous work is that it predominantly focusses on problems of small to moderate size, which are further often static. In fact, some of the above studies provide rules with future information such as the number of jobs still to arrive and their release dates, which is unavailable in a truly dynamic environment. Consequently, the rules evolved in most of these papers are tailored to static problem instances of moderate size. However, such problems are likely to be better solved by some global solution method, whereas the advantage of dispatching rules lies in their ability to deal with complex and dynamic problems, as discussed in Section 2.2. This work investigates the potential of GP hyper-heuristics to generate effective rules for complex, dynamic problems.

6.2 Algorithm Design

In general, the GP hyper-heuristic operates as described in Section 4.1, and the focus of this section is to detail the algorithm-specific components, namely the encoding of candidate solutions and the evaluation, selection and reproduction of individuals.

6.2.1 Encoding of Solutions

In GP, candidate solutions are typically encoded as syntax trees, where candidate solutions are dispatching rules in this case. Each dispatching rule is thus represented by a syntax tree, which encodes the priority index that the rule applies to determine the job to process next on a given machine, i.e. the job with the highest index. If the considered machine is a batch processing machine, the priority index further determines the composition of batches, which are filled with (other compatible) jobs in non-increasing order of their indices. The encoding of dispatching rules is illustrated in Figure 6.1, where GP trees are read recursively, starting from the root node, and from left to right, which yields the priority index $I_j = w_j(2 + p_j) - p_j$ for the tree shown in the figure. The priority index can be any function of attributes that are provided to the hyper-heuristic, e.g. the weight of a job w_j or its processing time p_j, and an important design decision thus concerns the selection of adequate terminals (attributes) and functions (operators) as they define the search space and the rules that can be generated.

One aspect that is not specified by the encoding is which jobs are to be considered in the decision making of a dispatching rule. By default, the information horizon of dispatching

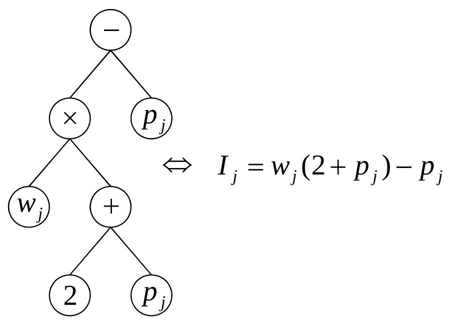


Figure 6.1: Example of the encoding of dispatching rules as GP trees, where rules prioritise jobs in non-increasing order of their indices I_j.

rules is restricted to jobs that are already queueing at the respective work centre. However, as discussed in Section 3.4, in can be beneficial to also take into account look-ahead jobs that arrive at the work centre in the near future. The effect of extending the horizon to look-ahead jobs that are in process at an upstream work centre and visit the local work centre for their next operation is briefly discussed in Section 6.6.

6.2.2 Evaluation

Following the decoding of an individual into the corresponding dispatching rule, a simulation experiment is run to evaluate the performance of the rule for the given objective, which measures the fitness of the individual. However, there are some particular challenges involved in this approach, commonly known as simulation(-based) optimisation (Fu et al., 2005).

First of all, simulation optimisation is computationally expensive and there is a need to use simulation time efficiently. In this work, only one replication of the simulation is run to evaluate an individual, with the same random number stream for all individuals belonging to the same generation. The use of common random numbers helps to reduce the variance of the relative performance of dispatching rules, which is essential for selecting the best individuals in each generation of the hyper-heuristic. In contrast, different streams are used across generations to avoid that the resulting rules are overfitted to one specific stream. While this setting is shown to be favourable in Branke (2001), it implies that all individuals have to be reevaluated in every generation. The actual length of the individual simulation runs is determined by experimentation (see Section 6.3.2).

Another issue that arises in dynamic problems, especially with the presence of work centres with sequence-dependent setup times or batch processing capabilities, is that some rules or individuals, which may be present particularly at the start of the run of the hyper-heuristic, can lead to an unstable system (see also Sections 3.2 and 3.3). To prevent wasting excessive time on the evaluation of such inferior rules, the number of jobs in the shop is monitored during the simulation run and the run is aborted if a preset threshold value for the number of jobs is exceeded. The corresponding individual then receives a high penalty on its fitness value, which ensures that it is quickly discarded. The threshold value is simply set to a value that is well above the number of jobs in the shop observed for stable benchmark rules.

6.2.3 Selection

Individuals are selected for reproduction using tournament selection. This scheme, which is the most commonly used in GP (Poli et al., 2008, Chapter 2), randomly samples a number of individuals from the current population, called the *tournament size*, of which the one with the highest fitness is chosen as a parent. The thusly selected parents then produce the individuals that form the population of the next generation. It follows that every population consists of entirely new individuals with this selection scheme. However, to ensure that the best individuals of each generation survive into the next generation, an elitist strategy is employed in addition, which directly inserts the best individual(s) of every generation into the next one, irrespective of whether they are selected for reproduction or not.

6.2.4 Reproduction

New individuals are created by applying one of two standard GP operators, called subtree crossover and subtree mutation. The subtree crossover recombines two individuals by randomly picking a node in each individual and swapping over the connecting subtrees, thereby producing two new individuals. Figure 6.2 presents an example of the crossover, where the white and grey parts within the two children originate from the first and second parent, respectively. The subtree mutation operator, on the other hand, modifies an individual by replacing a randomly selected node and the connecting subtree with another, randomly generated tree. Figure 6.3 illustrates this operation, where altered parts of the tree are indicated in grey. In both operations, non-leaf nodes are chosen with 90% and leaf nodes with 10% probability, which ensures that larger subexpression are exchanged more frequently (see also Poli et al., 2008, Chapter 2). The probability of selecting either subtree crossover or mutation for reproduction is a parameter of the hyper-heuristic, which are specified in the following section.

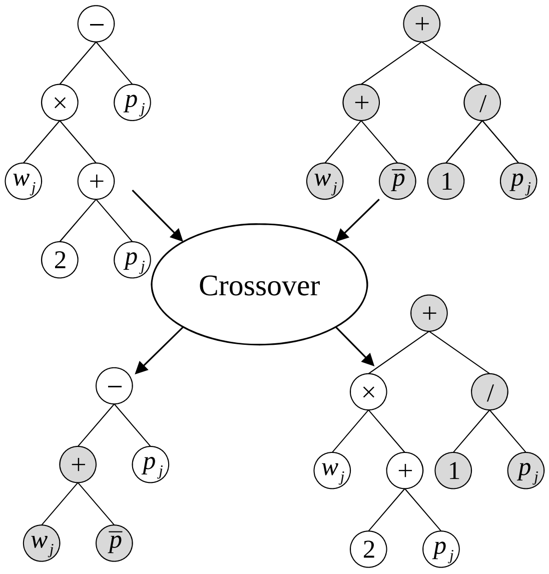


Figure 6.2: Example of a subtree crossover operation. A node is randomly selected in each parent (the leftmost '×' and '+' nodes in this case) and the connecting subtrees are swapped.

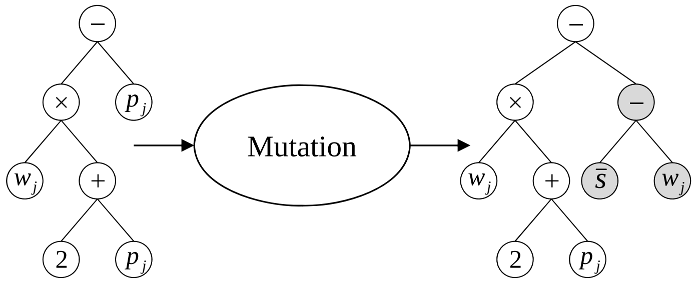


Figure 6.3: Example of a subtree mutation operation. A randomly selected subtree of the individual (the rightmost 'p_j' leaf node in this case) is replaced with a randomly generated tree.

6.3 Application to Scheduling Problems from Semiconductor Manufacturing

To assess its effectiveness, the GP hyper-heuristic is tested on several scheduling problems, as described in this section.

6.3.1 Scenario Description

The hyper-heuristic is employed to generate dispatching rules for two shops from the MASM testbed (Feigin et al., 1994), which includes data sets for a number of simulation models that are based on real-world semiconductor manufacturing systems. The shops, which correspond to the Fab4r and the Fab6 data set of the testbed, feature most of the complex characteristics that are typical of the industry, such as (identical) parallel machines, sequence-dependent setup times, batch processing and revisitations of work centres by jobs, making the manual design of effective dispatching rules very challenging. Hence, they represent a typical scenario in which one may want to resort to a hyper-heuristic for the development of dispatching rules.

Table 6.1 lists some characteristic features of the two shops, which differ largely in terms of their size and the number of operations required per job. Each of the two shops produces a limited set of different products, where jobs of the same product type have exactly the same processing requirements, which implies that they follow the same route. New jobs arrive continuously at the shops according to a Poisson process, with arrival rates as specified in the MASM testbed in case of the Fab6 shop, which yield a bottleneck utilisation (or shop utilisation) of 92.3%. The arrival rates for the Fab4r shop, on the other hand, are altered to 4.5 and 10.5 jobs per day of product type A and B, respectively, to create a more challenging problem with several highly utilised work centres and a bottleneck utilisation of 93.8%. Upon their arrival, each job is assigned a weight, drawn from a discrete uniform distribution on the $[1, 10]$ interval, and a due date, generated by means of the Total Work Content (TWK) method. The TWK method, which is widely used in the literature (Baker, 1984), simply assigns each job an allowance that is a multiple of its total processing time, i.e. $d_j = r_j + K \times \sum_{o=1}^{l_j} p_{j,o}$. In this work, the allowance factor K for a job is randomly drawn from a continuous uniform distribution on the $[2, 5]$ or $[1.2, 1.6]$ interval for the Fab4r and the Fab6 shop, respectively. These distributions yield a moderate to tight due date setting for both shops operating under the aforementioned job arrival rates. For each shop, the given objective is either to minimise mean flow time or to minimise mean weighted tardiness, resulting in four different scheduling problems altogether, which are denoted by $FJc|r_j, s_{ef}, B_f|\overline{C}$ and $FJc|r_j, s_{ef}, B_f|\overline{wT}$, respectively. The warm-up period for the Fab4r and the Fab6 problems has been determined to comprise the completion of 3000 and 1500 jobs, respectively.

Table 6.1: Main features of the two semiconductor manufacturing shops.

	Fab4r	Fab6
Number of work centres (machines)	31 (36)	104 (228)
Number of products	2	9
Number of operations per job	92 or 19	234–355
Total processing time per job	7145 or 1175	15624–25298

6.3.2 Parameter Setting

The hyper-heuristic is provided with a number of terminals and functions that are used within effective dispatching rules from the literature, including operation due dates, which are set as illustrated in Figure 3.3. The terminal set is further supplemented with some constants to enable the hyper-heuristic to weight attributes differently, especially since the latter have different units and can be of different magnitude, and a newly designed attribute called setup time regret. The setup time regret measures the amount of setup time that can be saved for a considered job if its imminent operation is processed on another parallel machine, given their current setups. The function set, on the other hand, contains some basic arithmetic and mathematical operations, and a ternary if-then-else (ifte) operator, which checks whether the first argument is larger than or equal to 0 and returns the second argument if that is the case, and the third otherwise.

To determine an adequate setting for the simulation experiments used to evaluate individuals, some preliminary experiments have been conducted on the Fab4r problem with the minimum mean weighted tardiness objective. In particular, ten replications of the hyper-heuristic have been run for each of several settings for the run length of the simulation experiments. In general, the performance of the hyper-heuristic can be expected to degrade with shorter simulation runs as they yield less precise measures of the actual performance of a rule, and are therefore more likely to mislead the selection of good individuals. However, the box plot of the performance of the ten respective dispatching rules evolved with different run length settings, shown in Figure 6.4, indicates no significant difference between the tested settings. Hence, an intermediate setting is used and the simulation run length is set to 15000 job completions, or

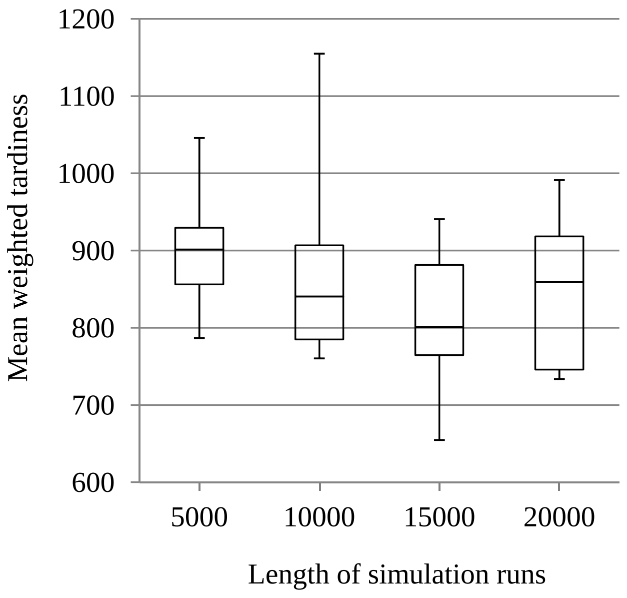


Figure 6.4: Effect of simulation run length, used in the evaluation of individuals, on the performance of the GP hyper-heuristic, measured by the performance of evolved rules.

five times the warm-up period rather, which corresponds to 7500 job completions in the case of the Fab6 problems.

With the run length fixed to 15000 job completions, another set of experiments has been carried out to examine whether the evaluation of individuals should account for the warm-up period or whether the fitness values should be based on all jobs completed in a simulation run. Figure 6.5 presents the box plot for the four different settings tested, which cover the whole range from including all jobs in the fitness calculations to discarding the first 3000 jobs, i.e. accounting for the whole warm-up period. The plot, which again is based on ten respective replications of the hyper-heuristic, does not indicate any significant difference between the different settings, and the fitness function is thus simply defined to incorporate all jobs.

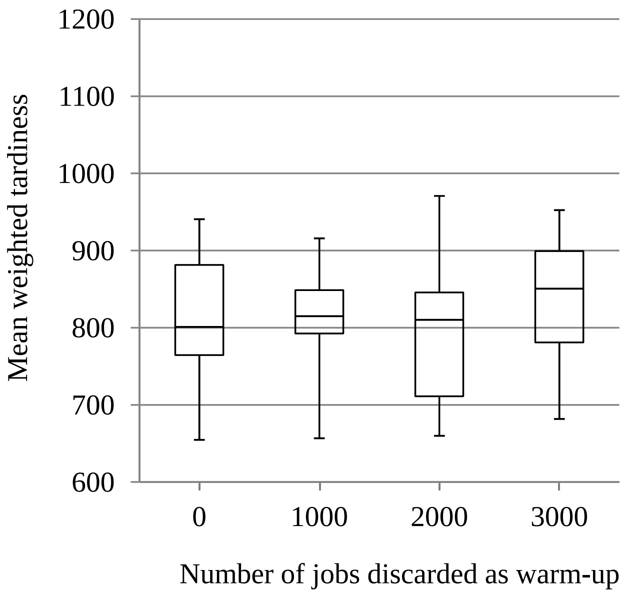


Figure 6.5: Effect of accounting for warm-up in the evaluation of individuals on the performance of the GP hyper-heuristic, measured by the performance of evolved rules.

Table 6.2 summarises the parameter setting of the GP hyper-heuristic, which follows general GP guidelines, including the generation of the initial population by means of the ramped half-and-half method with a depth range of 2 to 6 (Poli et al., 2008, Chapter 3). Ramped half-and-half fills half the population with trees whose leaves are all at the same depth and the other half with trees whose leaves are at various depths up to a predetermined limit, where the depth (limit) of a tree is randomly sampled from the specified range. Using this setting, the GP hyper-heuristic is applied to the four scheduling problems, running 20 (new) replications for each of the two Fab4r problems and 10 replications for each of the two Fab6 problems.

Table 6.2: Parameter setting of the GP hyper-heuristic.

Parameter	
Number of generations	50
Population size	1000
Initial population	Randomly generated using ramped half-and-half (depth 2 to 6)
Tournament size	7
Elitism	1%
Crossover probability	90%
Mutation probability	10%
Maximum tree depth	17
Simulation run length	5 × warm-up period
Terminals	Number of remaining operations of job ($l_j - i + 1$),
	remaining job processing time ($\sum_{o=i}^{l_j} p_{j,o}$),
	imminent operation processing time ($p_{j,i}$),
	average operation processing time of queueing jobs ($\overline{p}$),
	setup time for imminent operation ($s_{ef_{j,i}}$),
	setup time regret if job is processed on current machine,
	average setup time for queueing jobs ($\overline{s}$),
	number of queueing jobs of the same setup/batch family ($n^{\mathrm{Q}}_{f_{j,i}}$),
	batch size ($\min(n^{\mathrm{Q}}_{f_{j,i}}, B_{f_{j,i}})$),
	number of jobs in the queue (n^{Q}),
	0, 1, 2, 10,
(only $\overline{wT}$ problems)	job weight (w_j),
	remaining allowance of job ($d_j - t$),
	remaining allowance of imminent operation ($d_{j,i} - t$)
Functions	+, −, ×, ÷, max, min, ifte

6.4 Experimental Results

The evolved rules are evaluated on the respective problem they have been generated for, by running additional simulation experiments with different random number streams than those used in their generation. Moreover, data collection in these experiments only starts after the end of the warm-up period, as usual, where performance measurements for the Fab4r and the Fab6 problems are based on a sample size of $n = 50000$ and $n = 30000$ jobs per simulation run, respectively. The performance of the rules evolved for the two $FJc|r_j, s_{ef}, B_f|\overline{C}$ problems is discussed first, followed by a presentation of the results for the $FJc|r_j, s_{ef}, B_f|\overline{wT}$ problems.

6.4.1 Minimising Mean Flow Time

Since the problems at hand involve sequence-dependent setup times and batch processing, the best benchmark rules are likely to be rules that explicitly account for these characteristics. Various extensions of the effective SPT rule have been developed to address problems with these processing characteristics (see Sections 3.2 and 3.3). Two rules that integrate SNSPT and the exhaustive FSASPT rule with BSPT in such a way that they reduce to SNSPT (or FSASPT-SPT) for work centres with setups, to BSPT for batch processing work centres, and to SPT for simple serial processing work centres are included as benchmarks, where the rule name succeeding the '-' identifies the rule used to prioritise jobs within a family chosen by FSASPT, i.e. SPT in this case. These rules are denoted by BSNSPT and FSASPT-BSPT in the following. The SNSPT and FSASPT-SPT rules are further combined with the MCB rule so that the former serve as tie-breaking rules for MCB at batch processing work centres, providing two more benchmarks. The non-parallel machine version of DJAH, which reduces to SPT for serial processing work centres, BLWKR, PT+WINQ, 2PT+WINQ+NPT and FCFS complete the set of benchmark rules (see Section 3.4 for their definitions). However, since none of the latter rules take into account setup times, they are all implemented as exhaustive rules, i.e. jobs of the same family always receive highest priority at work centres with sequence-dependent setup times. Moreover, BLWKR, PT+WINQ and 2PT+WINQ+NPT are combined with MCB in the same way as SNSPT and FSASPT-SPT to ensure an efficient operation of batch processing work centres (see also Section 3.3).

Table 6.3 presents the results of the simulation experiments for the Fab4r problem, comparing the mean flow time (and mean queueing time) performance of the best and the worst of the 20 rules, generated in 20 replications of the GP hyper-heuristic, with that of the benchmark rules. In the following tables, the values in bold indicate the best performance achieved by any rule for the respective problem, while those marked with the same superscript, e.g. the mean flow time values of BSNSPT and FSASPT-BSPT in Table 6.3, are not significantly different from each other according to the Bonferroni-Dunn test, at the overall 5% significance level. To distinguish the different GP rules, the shop and objective that a rule has been evolved for are given in brackets in the tables. Table 6.3 shows that the hyper-heuristic is well able to generate rules that perform significantly better than the manually developed rules for this problem, with even the worst of the 20 rules outperforming the best benchmark rule, MCB-PT+WINQ. It follows that the mean flow time produced by the evolved rules on average is also lower (equal to 9635.7). The average performance of the evolved rules is an unbiased measure of the effectiveness of the hyper-heuristic, i.e. in a single run the hyper-heuristic can be expected to generate a rule that yields a mean flow time close to 9635.7 in this case. However, the relative advantage of the evolved rules is comparatively small, with the best GP rule yielding a 3% reduction of mean flow time, or a 4% decrease of its reducible component, the mean queueing time, over MCB-PT+WINQ. Table 6.3 further shows that the best benchmark rules are all based on the

Table 6.3: Performance of evolved and benchmark rules for the Fab4r shop regarding mean flow time and mean queueing time. Differences between performance values marked with the same superscript are not significant at the overall 5% significance level. The best performance values are in bold.

Dispatching rule	Mean flow time	Mean queueing time
FCFS	12661.9	9698.8
DJAH	10818.7	7855.6
BSNSPT	10469.2^{a}	7506.1^{a}
FSASPT-BSPT	10437.1^{a}	7474.0^{a}
MCB-BLWKR	10152.2	7189.1
MCB-SNSPT	10091.7	7128.6
MCB-FSASPT-SPT	10049.8	7086.7
MCB-2PT+WINQ+NPT	9903.9	6940.8
MCB-PT+WINQ	9849.0	6885.9
GP Worst(Fab4r, $\overline{C}$)	9670.4	6707.3
GP Best(Fab4r, $\overline{C}$)	**9581.2**	**6618.1**

MCB rule, which strongly suggests that the efficient operation of batch processing work centres is key to achieving a low mean flow time in this shop. The superior performance of the PT+WINQ and 2PT+WINQ+NPT rules, on the other hand, is surprising given that they are specifically designed for balanced job shop problems, whereas the workloads of work centres in this shop are highly unbalanced. It indicates that further improvements might be realised by the hyper-heuristic if it is provided with the WINQ attribute.

Table 6.4 lists the mean flow time produced by the best GP rule and the best benchmark rules, classified according to the product type of a job. It shows that the evolved rule does not gain its advantage by favouring one product over the other, but that it achieves a lower mean flow time for both products, thus increasing the overall efficiency of the shop.

Table 6.4: Mean flow time performance of the best evolved rule and the best benchmark rules for the Fab4r shop, subdivided by product type. The best performance values are in bold.

Dispatching rule	Mean flow time	
Product type	A	B
MCB-2PT+WINQ+NPT	22541.2	4500.3
MCB-PT+WINQ	22375.8	4492.7
GP Best(Fab4r, $\overline{C}$)	**22192.0**	**4188.9**

In many ways, the results for the Fab6 problem resemble those for the Fab4r problem. Table 6.5 compares the mean flow time (and mean queueing time) performance of the best and the worst of the ten rules, generated in ten replications of the hyper-heuristic, with that of the benchmark rules. Again, the GP rules prove to yield a significantly lower mean flow time (27585.7 on average) than the benchmark rules, with the best GP rule achieving a reduction of 1%, or 4% if measured in mean queueing time. However, the ranking of the benchmark rules indicates that batch processing work centres are less critical than in the Fab4r shop. This notion is particularly supported by the observation that the rules that apply BSPT at batch processing work centres tend be slightly more effective than their counterparts based on MCB. Moreover, there is some evidence that avoiding setup times is very important in this shop as the best benchmark rules are based on the FSASPT rule, which has a strong focus on the minimisation of setup times. This illustrates a general strength of hyper-heuristics, namely that they automatically adapt to the specific structure of a given problem by generating (or selecting) a heuristic that is particularly suitable for it.

Table 6.5: Performance of evolved and benchmark rules for the Fab6 shop regarding mean flow time and mean queueing time. Differences between performance values marked with the same superscript are not significant at the overall 5% significance level. The best performance values are in bold.

Dispatching rule	Mean flow time	Mean queueing time
MCB-BLWKR	31611.5	12155.8
FCFS	30179.7	10724.1
MCB-2PT+WINQ+NPT	28387.7^{d}	8932.0^{d}
DJAH	28330.1^{d}	8874.4^{d}
MCB-PT+WINQ	$28250.7^{b,c}$	$8795.1^{b,c}$
MCB-SNSPT	28232.3^{c}	8776.6^{c}
BSNSPT	28188.8^{b}	8733.2^{b}
MCB-FSASPT-SPT	27903.2^{a}	8447.6^{a}
FSASPT-BSPT	27871.3^{a}	8415.7^{a}
GP Worst(Fab6, $\overline{C}$)	27614.7	8159.0
GP Best(Fab6, $\overline{C}$)	**27546.6**	**8091.0**

Figure 6.6 displays the mean flow time of jobs by product type, resulting from the application of the best GP rule and the best benchmark rules, where the names of the products are taken from the MASM data set (Feigin et al., 1994). Again, the evolved rule is shown to reduce the mean flow time of every single product and increase the overall efficiency of the shop.

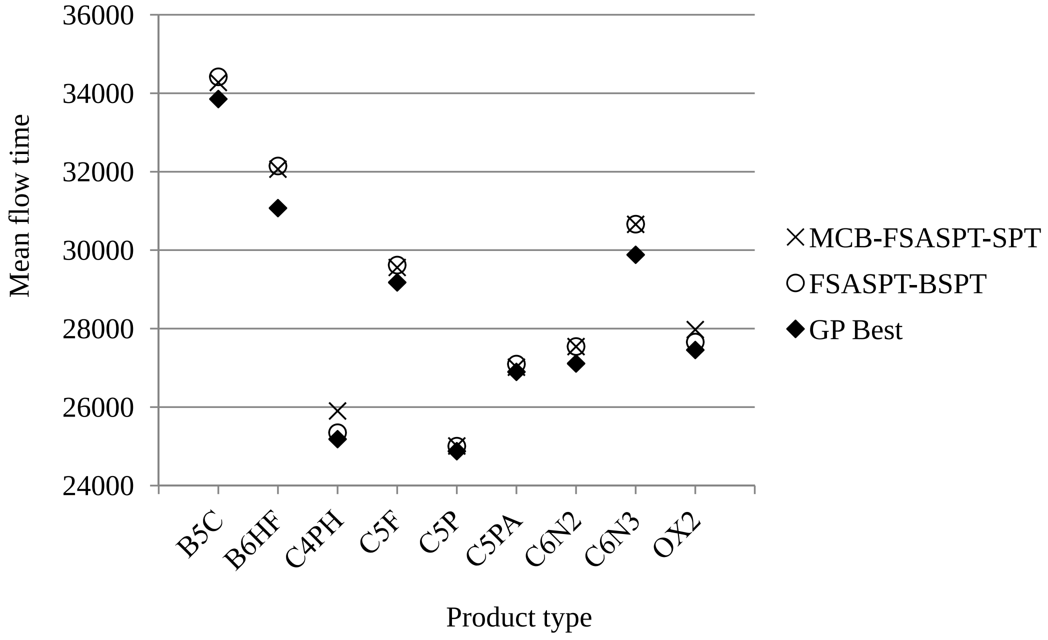


Figure 6.6: Mean flow time performance of the best evolved rule and the best benchmark rules for the Fab6 shop, subdivided by product type.

6.4.2 Minimising Mean Weighted Tardiness

Regarding the selection of suitable benchmark rules for the $FJc|r_j, s_{ef}, B_f|\overline{wT}$ problems, a rule that integrates the BATC^x and ATCS rules in the same way as illustrated by Mason et al. (2002) in the design of their BATCS rule is used as a benchmark. This rule, called BATCS^x in this work, accounts for sequence-dependent setup times and batch processing, and thus can be expected to perform well for the given problems. In addition, the WMOD, ATCS, CR+SPT and S/RPT+SPT rules are selected as benchmarks due to their reported effectiveness for multi-stage problems with the minimum mean weighted tardiness objective (see Section 3.4). The simple FCFS rule and a hierarchical rule, referred to as LW-OCR, that prioritises jobs in non-increasing order of their weights and uses the OCR priority index to break ties complete the set of benchmark rules. The motivation for including the LW-OCR rule in the performance comparison is that such rules are widely used in practice (Pfund et al., 2006). Again, all rules that do not explicitly consider setup times, i.e. all but BATCS^x and ATCS, are implemented as exhaustive rules, while WMOD, ATCS, CR+SPT, S/RPT+SPT and LW-OCR are combined with the MCB rule as described above.

To determine a good parameter setting for ATCS and BATCS^x, some preliminary experiments have been run for each of the two problems, in which both rules have been tested with 100 different settings using values of 1.0, 1.5, 2.0, ..., 5.0, 5.5 for k_1 and 0.01, 0.05, 0.1, 0.25, 0.5, ..., 1.25, 1.5, 2.0 for k_2, covering the ranges of typical values (Lee, Bhaskaran, and

Pinedo, 1997; Perez, Fowler, and Carlyle, 2005). The ATCS and BATCS^x rules are subsequently applied with the parameter setting found to be best in those experiments.

Table 6.6 presents the results of the simulation experiments for the Fab4r problem, comparing the performance of the best and the worst of the 20 evolved rules with that of the benchmark rules regarding mean weighted tardiness and other related criteria. The best GP rule is shown to clearly outperform the best benchmark rules, MCB-WMOD, MCB-ATCS and MCB-S/RPT+SPT, reducing mean weighted tardiness by about 43%. On the other hand, the worst GP rule is shown to produce a significantly higher mean weighted tardiness than MCB-WMOD and MCB-ATCS. Hence, it can be established that the hyper-heuristic has the potential to generate outstanding rules for this problem, but is not always able to tap its potential. This can be partly attributed to the additional factors that have to be taken into account in minimising mean weighted tardiness, making the design of or search for effective dispatching rules generally more difficult than for the mean flow time criterion. This is also reflected by the huge performance differences between relatively simple rules, such as LW-OCR, and the more sophisticated manually developed rules. However, the mean weighted tardiness achieved by the 20 evolved rules on average is 789.4, which is clearly lower than that produced by the best benchmark rules, and the hyper-heuristic can thus be expected to generate a better rule for this problem in a single run, which illustrates its effectiveness. Table 6.6 further shows that the best benchmark rules are all based on the MCB rule, while the performance of the BATCS^x rule, which does not explicitly prioritise fuller batches, is disappointing. This lends additional support to the notion that the efficient operation of batch processing work centres is critical in the Fab4r shop.

Table 6.6: Performance of evolved and benchmark rules for the Fab4r shop regarding mean weighted tardiness and other due date-related measures. Differences between performance values marked with the same superscript are not significant at the overall 5% significance level. The best performance values are in bold.

Dispatching rule	Mean weight. tardiness	Mean tardiness	Proportion tardy (in %)	Cond. mean tardiness
FCFS	15022.3	2734.3	83.1	3287.2
MCB-LW-OCR	3437.3	1666.9	37.0	4499.0
BATCS^x(k_1 = 5.5, k_2 = 0.1)	2295.0	1269.3	41.5	3053.2
MCB-CR+SPT	1084.4^c	609.2	33.2	1825.4
MCB-S/RPT+SPT	1031.3^{a,b}	634.6	28.6^b	2208.7
MCB-ATCS(k_1 = 4.0, k_2 = 0.01)	1017.5^a	442.6	**22.0**	1992.3
MCB-WMOD	1007.5^a	429.0	26.9^a	1581.3
GP Worst(Fab4r, $\overline{wT}$)	1102.4^{b,c}	373.0	31.4	1169.9
GP Best(Fab4r, $\overline{wT}$)	**576.1**	**286.6**	27.3^{a,b}	**1030.2**

Although the evolved rules are specifically optimised for the mean weighted tardiness criterion, Table 6.6 indicates that the dominance of the best rule carries over to other common measures of due date performance. For the minimisation of mean tardiness and conditional mean tardiness, the best GP rule is found to remain best, and is only found to be outperformed by the MCB-ATCS rule with respect to the proportion of tardy jobs. In other words, the rule effectively minimises the average delay of all jobs, irrespective of their weight, and tends to avoid extreme due date violations. This can also be seen in Figure 6.7, which displays the mean weighted tardiness of jobs, classified by weight, that results from the application of the best evolved rule and the best benchmark rules. The evolved rule is shown to reduce the mean (weighted) tardiness of jobs with smaller weights while accepting a marginally higher delay of jobs with larger weights than MCB-S/RPT+SPT.

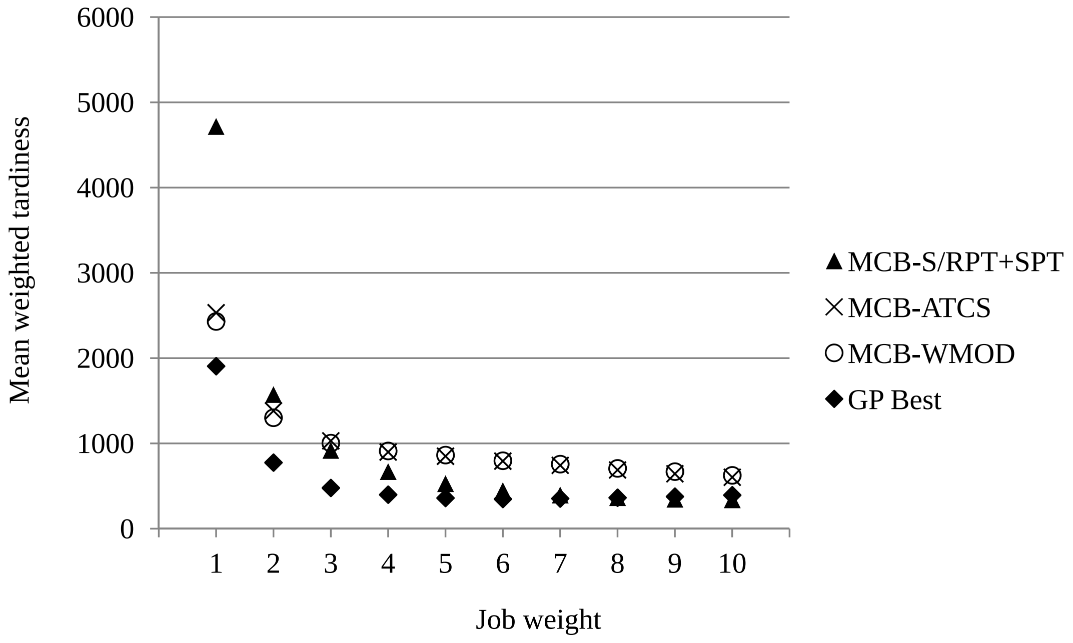


Figure 6.7: Mean weighted tardiness performance of the best evolved rule and the best benchmark rules for the Fab4r shop, subdivided by job weight.

Table 6.7 presents the results for the Fab6 problem, comparing the performance of the best and the worst of the ten rules, generated by the hyper-heuristic, with that of the benchmark rules. For this problem, the hyper-heuristic proves to generate rules that generally yield a significantly lower mean weighted tardiness (2672.2 on average) than any of the manually developed rules, with the best GP rule achieving a 23% reduction over MCB-WMOD, the best benchmark rule. The BATCS^x rule, on the other hand, again fails to produce a competitive performance and is clearly outperformed by its base rule ATCS (in combination with MCB), indicating that its design is flawed to some extent.

Table 6.7: Performance of evolved and benchmark rules for the Fab6 shop regarding mean weighted tardiness and other due date-related measures. Differences between performance values marked with the same superscript are not significant at the overall 5% significance level. The best performance values are in bold.

Dispatching rule	Mean weight. tardiness	Mean tardiness	Proportion tardy (in %)	Cond. mean tardiness
FCFS	19159.6	3491.3[d]	73.4	**4744.9**
MCB-LW-OCR	5569.6[c]	2772.5	42.8	6470.4
MCB-S/RPT+SPT	4970.4[b]	2489.6	39.7	6262.1
BATCS[x](k_1 = 5.5, k_2 = 0.1)	4823.2[a,b,c]	3776.5[d]	**29.2**	12825.5
MCB-CR+SPT	4255.4[a]	2308.5	38.6	5967.4
MCB-ATCS(k_1 = 1.5, k_2 = 0.01)	3362.0	**1613.7**[a]	32.7	4918.2
MCB-WMOD	3290.1	1672.9[b,c]	31.8	5242.0
GP Worst(Fab6, $\overline{wT}$)	2863.4	1725.9[c]	30.9	5561.8[a]
GP Best(Fab6, $\overline{wT}$)	**2538.4**	**1668.3**[a,b]	30.5	5447.4[a]

The best GP rule further shows to effectively minimise other due date-related measures. More specifically, it is found to be among the best rules for mean tardiness and to be only slightly outperformed by the (otherwise non-competitive) BATCS[x] rule with respect to the proportion of tardy jobs. However, the fact that the evolved rule does not achieve a (significant) reduction of mean tardiness over the best benchmark rules indicates that it gains its advantage by trading off a larger delay of jobs with small weights for a decrease of mean (weighted) tardiness for jobs with large weights. This is also illustrated by Figure 6.8, which compares the mean weighted tardiness performance of the evolved rule and the best benchmark rules for jobs classified by weight. It shows that the evolved rule reduces the mean (weighted) tardiness for all jobs with weight larger than 1 at the expense of a substantial increase of mean (weighted) tardiness for jobs with weight 1.

In summary, for each of the four examined problems the GP hyper-heuristic proves to generate dispatching rules that are significantly better than manually developed benchmark rules, which demonstrates the effectiveness of the method. Moreover, the relative advantage of the best rules evolved for the $\overline{wT}$ problems is larger than that of the best rules evolved for the corresponding $\overline{C}$ problems. This suggests that the potential of the GP-based generation of dispatching rules is generally higher for more complex problems, for which the manual design of effective dispatching rules is more difficult. On the other hand, the larger variances of the performance of rules evolved in different replications of the hyper-heuristic for the $\overline{wT}$ problems indicate that the larger (three additional terminals) and more complex search space also poses a challenge to the hyper-heuristic, implying that it may have to be run several times to tap its full potential.

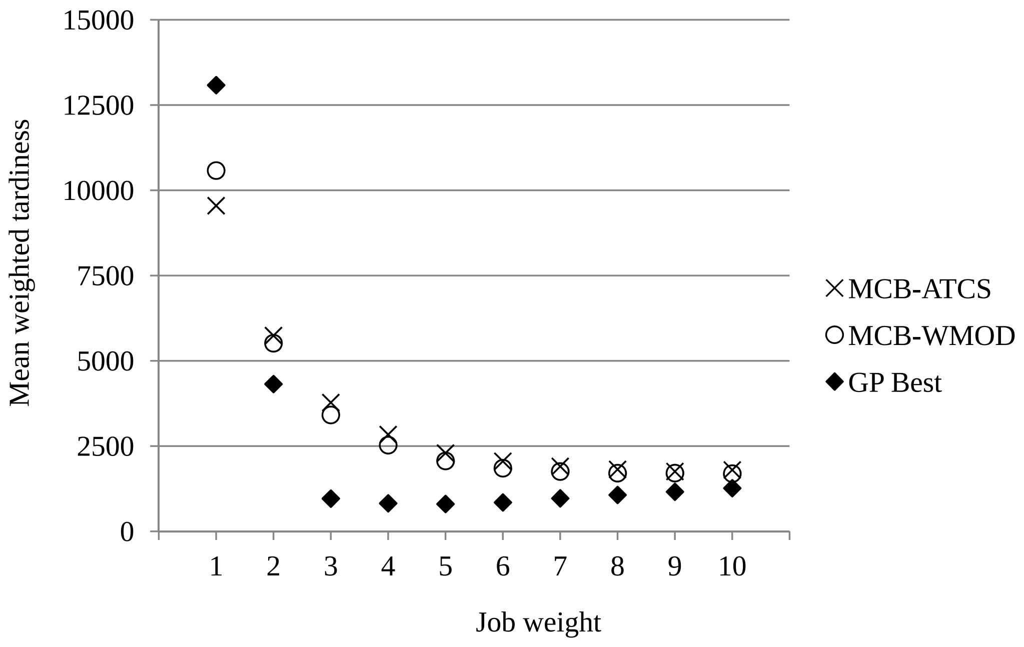


Figure 6.8: Mean weighted tardiness performance of the best evolved rule and the best benchmark rules for the Fab6 shop, subdivided by job weight.

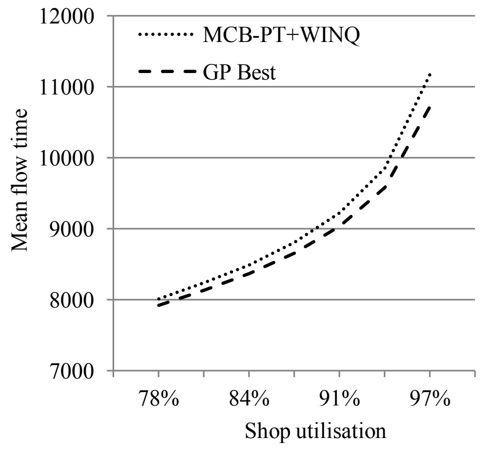
Since the hyper-heuristic optimises on a simulation model with a specific set of parameter values, the generated rules could be expected to be overly sensitive to changing conditions in the shop. If these alterations are the result of a deliberate decision, e.g. the introduction of a new product or the purchase of a new machine, the hyper-heuristic could be simply rerun with an adapted simulation model to generate a new rule that fits the new conditions. However, some changes, e.g. in the job arrival pattern, might be a very subtle and difficult to detect. If the performance of the evolved rules were to strongly deteriorate in such a case, this would certainly question the practicality of the whole approach, and the following section thus examines the robustness of the evolved rules.

6.5 Sensitivity Analysis

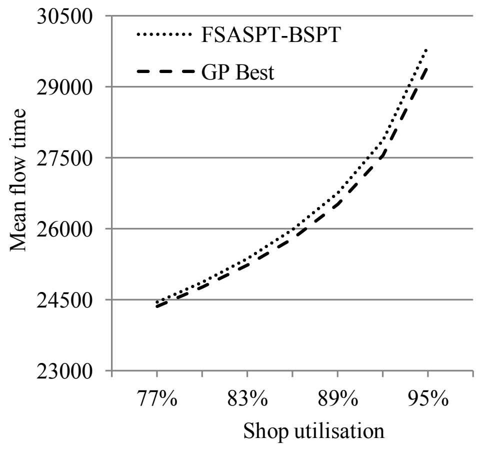
The sensitivity analysis is carried out for the best of the rules, evolved for each of the four scheduling problems. These four rules are tested for their robustness to changes in the job arrival rates, while those evolved for the $\overline{wT}$ problems are also subjected to different due date settings. A suitable benchmark for the robustness of an evolved rule is the performance of the rule that one would employ without access to the hyper-heuristic, i.e. MCB-PT+WINQ and FSASPT-BSPT for the Fab4r and the Fab6 problem with the minimum mean flow time objective, respectively, and MCB-WMOD for both $\overline{wT}$ problems. These rules are likely to be robust as, contrary to the ATCS-based rules, they do not possess any parameters that have been optimised for the original problem.

6.5.1 Arrival Rates

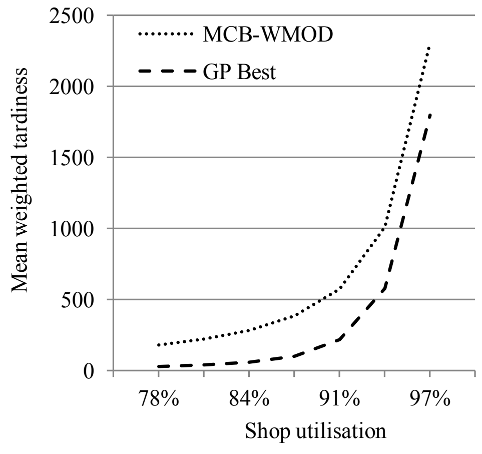
As described in Section 6.3.1, the default arrival rates lead to a shop utilisation of 93.8% and 92.3% for the Fab4r and Fab6 shop, respectively. Figure 6.9 compares the performance of the best evolved and the best benchmark rule for the four scheduling problems under different utilisation levels, resulting from an increase or decrease of the total job arrival rate. In every of the four cases, the GP rule proves to dominate the benchmark rule across the whole range



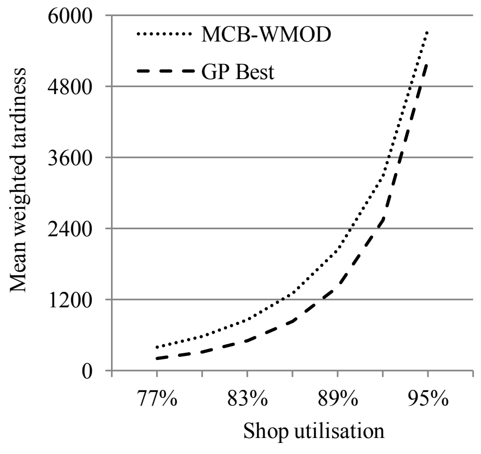
(a) Fab4r, $\overline{C}$ problem



(b) Fab6, $\overline{C}$ problem



(c) Fab4r, $\overline{wT}$ problem



(d) Fab6, $\overline{wT}$ problem

Figure 6.9: Performance of the best evolved and the best benchmark rule for the four scheduling problems under varying levels of utilisation.

of examined utilisation levels, which indicates that the rules evolved by the hyper-heuristic are generally robust to changes in the total job arrival rate.

In addition to an increase or decrease in the total arrival rate, the share of job arrivals that belong to a certain product type may also change, thus resulting in a different product mix. A side effect of this is a possible change in the location of the bottleneck. To test the sensitivity of rules to variations in the product mix, the utilisation of the bottleneck, whereever its location, is held constant. Figure 6.10 presents the results of these experiments for the Fab4r problems, where the share of product A in the original mix is 30% (see Section 6.3.1). It shows that both GP rules, evolved for the two different objectives, outperform the best benchmark rule across a wide range of product mixes.

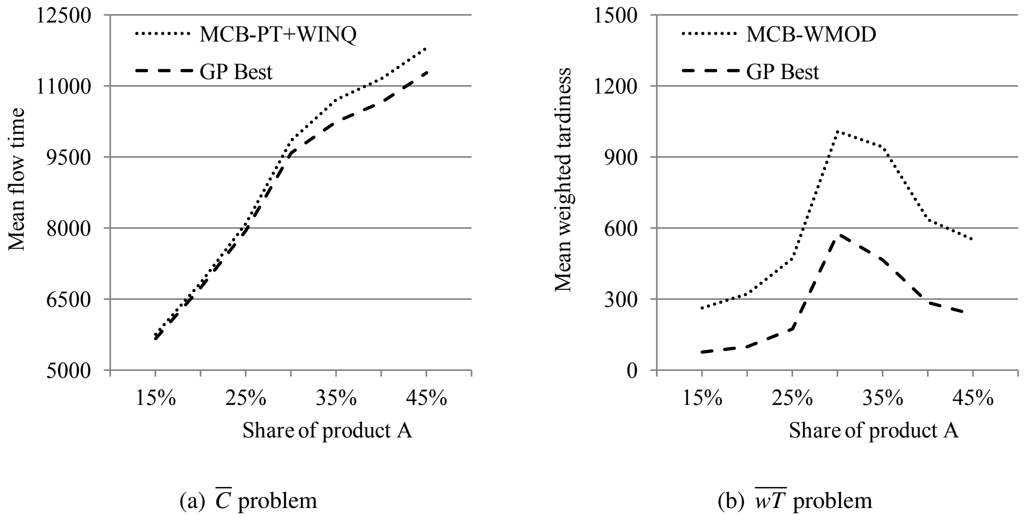


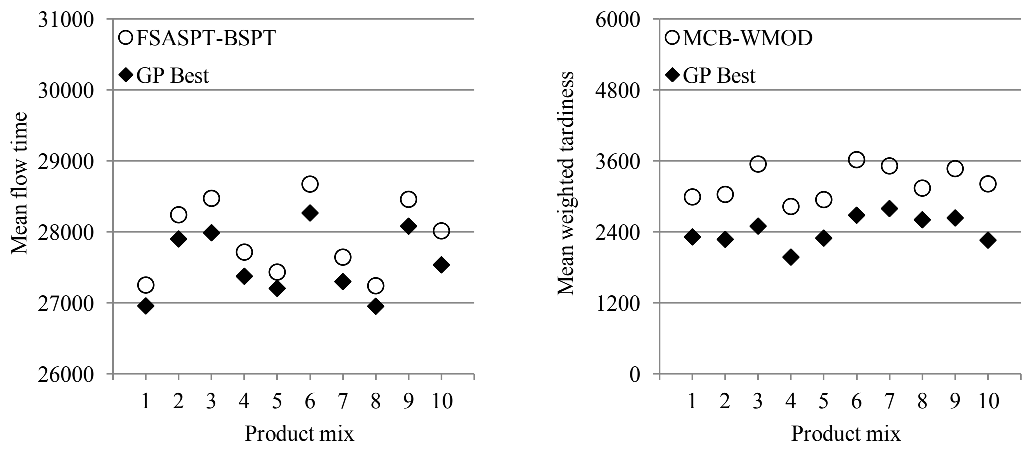
Figure 6.10: Performance of the best evolved and the best benchmark rule for the two Fab4r problems under varying product mixes.

Given that there are nine products in the Fab6 shop, it is impractical to systematically examine all variations from the original mix. Instead, a random sampling scheme is used to investigate whether rules are robust with respect to changes in the product mix. In this scheme, a mix is generated by randomly assigning each product to one of two groups with 50% probability. The share of the first group in the product mix is then increased by 10% and the share of the second group decreased by 10%, where the arrival rates of the individual products within each group are changed in proportion to their original arrival rates. Using this scheme, ten different product mixes are generated, which are listed in Table 6.8.

Figure 6.11 displays the performance of the best GP rules, evolved for the two Fab6 problems, and the best benchmark rules under these ten product mixes. Both GP rules prove to consistently maintain their relative advantage, despite comparatively big fluctuations in the absolute performance values. Hence, it can be concluded that the hyper-heuristic generates rules that are generally robust to changes in the product mix.

Table 6.8: Product mixes generated for the Fab6 shop by means of the random sampling scheme. A ‘+’ and ‘−’ indicates that the corresponding product is included in the group whose share is increased and decreased, respectively.

Product mix	B5C	B6HF	C4PH	C5F	C5P	C5PA	C6N2	C6N3	OX2
1	−	−	−	+	+	−	+	+	−
2	+	+	+	+	−	+	−	+	+
3	−	+	−	+	−	−	+	+	−
4	−	+	−	−	−	−	+	−	+
5	+	+	−	−	+	+	−	+	−
6	+	+	−	+	−	−	+	+	+
7	−	+	+	−	+	−	+	+	−
8	−	−	+	+	+	−	−	−	−
9	+	+	+	+	−	−	−	+	+
10	−	+	+	−	−	−	+	+	−



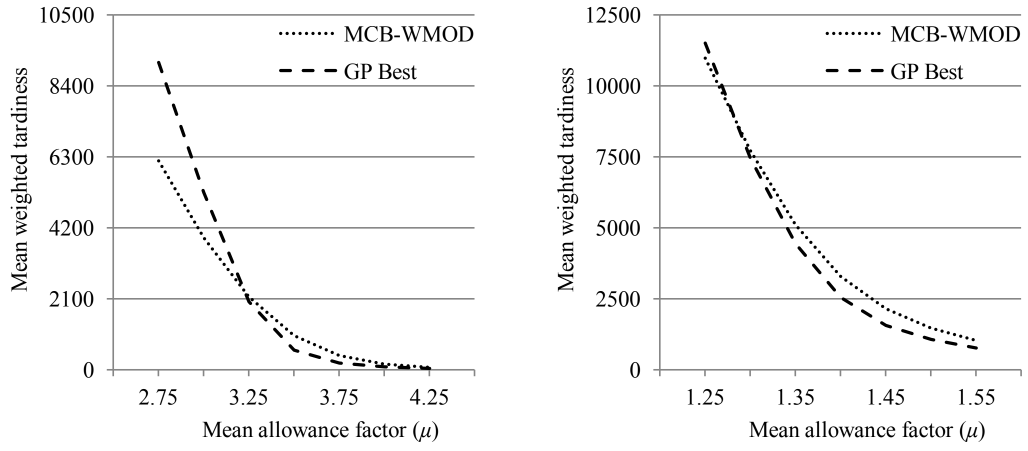
(a) $\overline{C}$ problem (b) $\overline{wT}$ problem

Figure 6.11: Performance of the best evolved and the best benchmark rule for the two Fab6 problems under varying product mixes.

6.5.2 Due Date Setting

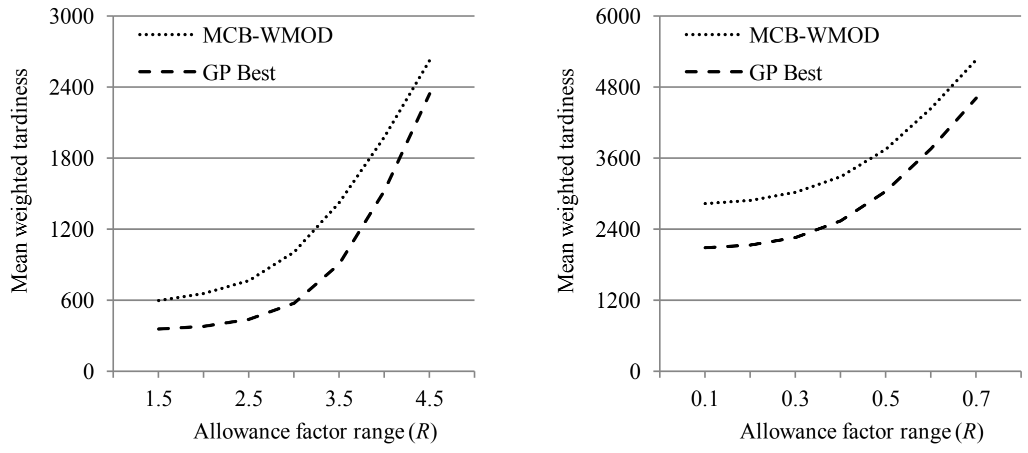
The application of the TWK due date assignment method with uniformly distributed allowance factors involves two parameters that control the tightness and the range of job due dates, respectively. In particular, the mean allowance factor μ reflects the due date tightness, where a higher mean allowance factor corresponds to looser due dates, while the range of allowance factors R measures the variability of due dates in terms of their tightness. Both parameters are varied in the following to assess whether the evolved rules are sensitive to changes in them.

Figures 6.12 and 6.13 compare the performance of the best GP rule and the best benchmark rule for the two $\overline{wT}$ problems for different values of μ and R, respectively, where the default setting corresponds to $\mu = 3.5$ and $R = 3$ for the Fab4r shop, and to $\mu = 1.4$ and $R = 0.4$ for the Fab6 shop (see Section 6.3.1). For both problems, the evolved rule is shown to remain superior to the benchmark rule for larger mean allowance factors, but to loose its relative advantage with a decreasing mean allowance factor until it is eventually outperformed (see Figure 6.12).



(a) Fab4r problem (b) Fab6 problem

Figure 6.12: Performance of the best evolved and the best benchmark rule for the two $\overline{wT}$ problems under varying levels of due date tightness. Due dates are generated by means of the TWK method with allowance factors drawn from a uniform distribution on the interval $[\mu - \frac{R}{2}, \mu + \frac{R}{2}]$.



(a) Fab4r problem (b) Fab6 problem

Figure 6.13: Performance of the best evolved and the best benchmark rule for the two $\overline{wT}$ problems under varying levels of due date range. Due dates are generated by means of the TWK method with allowance factors drawn from a uniform distribution on the interval $[\mu - \frac{R}{2}, \mu + \frac{R}{2}]$.

In other words, the GP rules are not robust to tighter due date settings. On the other hand, Figure 6.13 indicates that the GP rules are able to maintain their relative advantage for different ranges of allowance factors. Overall, these results suggest that the hyper-heuristic evolves rules that are robust to changes in the due date setting, with the exception of an increased level of due date tightness.

6.6 Critical Appraisal and Future Work

In summary, the GP hyper-heuristic presented in this chapter has shown to generate superior dispatching rules for several complex and dynamic scheduling problems, which are generally robust to changes in the job arrival pattern and due date setting. The relative performance of the evolved rules is only found to deteriorate considerably if due dates become tighter. However, this weakness can be addressed in two ways. First of all, a large change in the due date setting could probably be detected by monitoring the due dates of incoming jobs, in which case the hyper-heuristic could be run again to generate a rule suitable for the new conditions. Secondly, the hyper-heuristic could be configured to evolve more robust rules, e.g. by basing the fitness of individuals on several simulation experiments with different settings for the due date tightness level. The evolved rules are then less tailored to one setting, i.e. they can be expected to be more robust to changes, though possibly at the expense of a slightly worse performance for the default tightness level. While the above results demonstrate the potential of using GP to generate dispatching rules for complex and dynamic scheduling problems, there are several areas in which the hyper-heuristic could be improved.

A drawback of the hyper-heuristic in its current form are its high computational requirements. The runtime of one replication has been around 5 hours (on a 2.9 GHz Intel Quad-Core Xeon with 8 GB RAM) for the Fab4r problems, and has shown to increase to approximately 16 hours (on a similarly powerful 2.8 GHz Intel Quad-Core i5 with 4 GB RAM) for the Fab6 problems due to the larger shop (see Section 6.3.1), which is computationally more expensive to simulate. Even though one could argue that such a long runtime is acceptable given that the hyper-heuristic is supposed to be only run once and offline, and that the manual development of an effective dispatching rule typically takes a lot longer, improving the efficiency of the hyper-heuristic should clearly be a main aim of future work.

Such efforts to make the hyper-heuristic more efficient should particularly focus on the reduction of the number and length of simulation runs used for the evaluation of individuals, which take up the largest amount of computational time by far. Apart from tuning the GP parameters, specifically the number of generations and the population size, one way to reduce the number of simulation runs could be to retain the random number seed for a number of consecutive generations before changing it, or to (re)use a limited number of seeds in a round-robin fashion. This would save simulation runs for individuals that have survived from previous generations and have already been evaluated with a given seed. Concerning the length of simulation runs, a trade-off can be expected between the runtime and performance of the hyper-heuristic due to the reduced precision of performance measurements associated with shorter simulation runs. However, it may be possible to save some time without compromising the effectiveness of the hyper-heuristic by starting with relatively short runs that allow the hyper-heuristic to focus on the promising areas of the search space, and then increasing the run length, and therefore the precision, as the search proceeds. In general, the effects of using a certain simulation setting will depend on a number of factors, such as the nature of the search space, the selection scheme and the variance of the relative performance of rules that is due to the random number seed, making the optimisation of the number and length of simulation runs a difficult task.

A different approach to saving simulation runs could be to incorporate a procedure to detect duplicates in the population that is run prior to the evaluation of individuals. Then, all individuals that are found to represent the same rule can be evaluated by a single simulation run. The motivation for such a procedure arises from the observation that one dispatching rule can be encoded in numerous ways, e.g. $-p_{j,i}$, $-(p_{j,i} + \min(s_{ef_{j,i}}, -10))$ and $1/(p_{j,i} \times p_{j,i})$ are three different priority indices representing the SPT rule as setup times, by definition, cannot be negative (and thus be smaller than -10) and prioritising jobs in non-increasing order of $-p_{j,i}$, $-(p_{j,i} - 10)$ or $1/(p_{j,i} \times p_{j,i})$ all corresponds to SPT. The detection of duplicates, however, is not easy and a simple comparison of the syntax not sufficient in general. To illustrate, even a procedure that eliminates redundant and ineffective subexpressions that do not affect the ranking of priorities, such as constants, before comparing the (remaining) expressions can only establish the identity of the first two of the above indices, but not that they are equal to the third. Hence, an effective duplicate detection procedure should go beyond a simple syntax comparison.

Besides making the hyper-heuristic more efficient, there is some potential for improving the effectiveness of the hyper-heuristic by providing it with up- and downstream information. In fact, some experiments have been conducted in which the eligible job set and the queue terminals have been extended to include look-ahead jobs that have started their current operation and require the considered work centre for their next operation. Furthermore, the terminal set of the hyper-heuristic has been supplemented with the arrival time attribute $\max(r_{j,i} - t, 0)$, which allows for queueing jobs ($r_{j,i} - t \leq 0$) and look-ahead jobs ($r_{j,i} - t > 0$) to be distinguished (see also Section 3.4). On the one hand, the results of these experiments have been encouraging in that the best rules evolved by this extended hyper-heuristic for the two $FJc|r_j, s_{ef}, B_f|\overline{C}$ problems have shown to clearly outperform the best rules evolved with the default setting. On the other hand, the hyper-heuristic has been found to become less reliable, generating very poor or even unstable rules in about a third of the runs. This illustrates that, while the incorporation of up- and downstream information within the GP hyper-heuristic is promising, it also creates new challenges such as a more complex search space, which require further attention.

Finally, another issue concerns the nature of the evolved priority indices, which tend to be composed of hundreds of (terminal and functional) symbols. This makes it impossible for a practitioner to understand the decision logic of an evolved rule, who is required to blindly trust that the decisions of the rule are justified and will lead to the desired outcome in the long term. This is likely to adversely affect the acceptance of the method in practice, and should therefore be addressed in some way in the future.

Chapter 7

Automatic Generation of Sets of Work Centre-Specific Rules [1]

In multi-stage problems, different dispatching rules may be used at different work centres, which has been found to be beneficial for some problems where work centres vary with respect to their position in the shop (Barrett and Barman, 1986; LaForge and Barman, 1989; Mahmoodi et al., 1996; Sarper and Henry, 1996; Barman, 1997, 1998) and/or workload (Raman, Talbot, and Rachamadugu, 1989; Ruben and Mahmoodi, 1998; Bokhorst, Nomden, and Slomp, 2008). In consequence, some researchers have moved on to designing a set of work centre-specific rules instead of one rule for the whole shop (e.g. Cigolini et al., 1999; Lee et al., 2003). However, this substantially increases the development effort as each rule has to match the other rules in the set so that together they operate the interdependent work centres effectively.

This chapter presents a hyper-heuristic that (automatically) generates sets of work centre-specific rules by selecting a (potentially) different rule for each work centre from a number of given rules. The next section reviews related work, followed by a detailed description of the algorithm in Section 7.2. The hyper-heuristic is applied to some complex problems from semiconductor manufacturing as described in Section 7.3, and the performance and robustness of the evolved rule sets is assessed in Sections 7.4 and 7.5, respectively. The chapter concludes with a critical appraisal of the method and some suggestions for future work.

[1]An earlier version of this method is published in C. W. Pickardt, T. Hildebrandt, J. Branke, J. Heger, and B. Scholz-Reiter (2013), Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems, *International Journal of Production Economics* 145(1), 67–77. Apart from the extensions made to the method, a slightly different experimental setup is used in this chapter, and experimental results thus deviate from those reported in the paper.

7.1 Related Work

In the literature, various types of hyper-heuristics have been proposed for the generation of sets of work centre-specific dispatching rules. Ishii and Talavage (1994) design a heuristic search algorithm that proceeds by selecting work centres roughly in decreasing order of their utilisation and choosing the best of a number of given rules for the respective work centre, where work centres not yet optimised operate under some default base rule. They test this hyper-heuristic on several dynamic job shop problems, for which it shows to discover rule sets that are more effective than applying any of the given rules at all work centres. Baek et al. (1998) extend this work in several ways. First, their hyper-heuristic, which proceeds in a similar sequential fashion, operates on a search space of composite rules, whose priority indices are weighted linear combinations of priority indices of some common rules. Second, they examine how its performance is affected by employing a different base rule or a different criterion for the optimisation order of work centres. Baek et al. apply their hyper-heuristic to two $FJc|r_j|\overline{C}$ problems and find that it generates rule sets that yield a significantly lower mean flow time than all benchmark rules, including those that form the components of the composite rules. Moreover, they observe that the best rule sets are obtained if the hyper-heuristic optimises work centres in decreasing order of their utilisation and starts with a good, rather than a bad base rule. One drawback of hyper-heuristics that select or generate work centre-specific rules one-by-one is that they implicitly assume that the relative effectiveness of a candidate rule for a considered work centre is more or less independent of the rules used at the other work centres in the shop, which is generally not the case.

Some researchers have developed hyper-heuristics that optimise all rules simultaneously, which are predominantly based on EAs. Caskey and Storch (1996) use an EA, which operates on the search space of all possible combinations of a number of given rules, to search for good sets of work centre-specific rules, but do not report on the performance of the evolved rule sets. Lee, Piramuthu, and Tsai (1997) embed a similar EA in a decision tree algorithm that is designed to learn effective job release policies. While their results show the overall method to be effective, it is unclear to what extent the EA contributes to these results. Yang et al. (2007) also develop an EA-based hyper-heuristic that selects a (potentially) different rule for each work centre from a number of given rules. The hyper-heuristic is tested on the $FFc|s_{ef}|\overline{T}$ problem and shown to evolve rule sets that perform better than some simple benchmark rules. In contrast, Azadeh et al. (2012) propose a neural network algorithm for the selection of different rules for different work centres but do not establish whether it is actually effective in guiding the search towards promising rule sets. Following up their previous work, Baek and Yoon (2002) design a coevolutionary hyper-heuristic that maintains one population of composite rules for each work centre and coevolves these populations in parallel. They apply this hyper-heuristic to the $FJc|r_j|\overline{T}$ problem and find that it evolves rule sets that outperform single benchmark rules and yield a similar mean tardiness as the rule set generated by the sequential hyper-heuristic from their earlier paper (Baek et al., 1998). Geiger et al. (2006) address the $F2||C_{\max}$ problem with a Genetic Programming (GP) hyper-heuristic that evolves a separate rule for each of the two work centres, which is shown to generate rule sets that yield close-to-optimal solutions. However, a disadvantage of hyper-heuristics that simultaneously generate or select a separate rule for every work centre in the shop is that their search space grows exponentially in the number of work centres, which is likely to impair their effectiveness for larger problems.

A possible way to limit the size of the search space is to classify similar work centres and let the hyper-heuristic evolve only one rule for each class of work centres. In this regard, Miyashita (2000) compares three different GP hyper-heuristics — one that evolves a single rule for all work centres, one that evolves a different rule for every work centre, and one that is based on a predetermined classification of work centres into bottlenecks and non-bottlenecks and evolves one rule for bottlenecks and another for non-bottlenecks. The hyper-heuristics are tested on two job shop problems with five work centres, of which one or two, respectively, are bottlenecks. Overall, the classification approach is found to be most effective, while the hyper-heuristic that evolves one rule per work centre is shown to be susceptible to overfitting, i.e. the resulting rule sets perform very well on the training instances but poorly on the test instances. On the other hand, it can be difficult to classify work centres beforehand in terms of how to operate them, especially if they differ not only with respect to their workload, but also with regard to other aspects such as their setup requirements, batch processing capacities and their relative position in the shop. Then, an alternative could be to use a hyper-heuristic that undertakes the classification of work centres while searching for good dispatching rules, such as the one proposed by Jakobović and Budin (2006). In their GP-based hyper-heuristic, an individual consists of three mathematical expressions, where one of them is a function of attributes relating to the workload of a work centre that determines which of the two dispatching rules, encoded by the other two expressions, to apply. Jakobović and Budin report that the best rule set generated for the $Jm||\overline{wT}$ problem in 20 replications of this hyper-heuristic performs significantly better than the best rule evolved in 20 replications of a GP hyper-heuristic that generates a single rule for all work centres. However, it is not clear whether such a hyper-heuristic is able to generate larger sets of specialised rules as its search space becomes extremely complex with an increasing number of rules it is supposed to evolve.

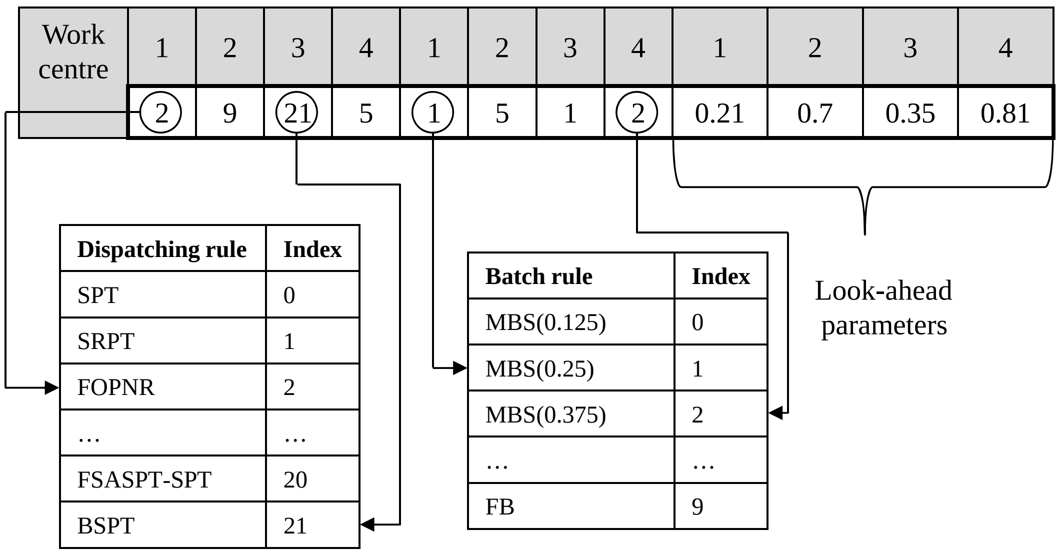
Overall, this literature review indicates that despite the apparent potential of using different rules at different work centres, there are not many papers that study the performance of hyper-heuristics for the generation of sets of work centre-specific dispatching rules. Moreover, a shortcoming of the existing studies is that they focus on small problems with ten or fewer work centres, which implicate a small search space. This work investigates the potential of EA-based hyper-heuristics to generate sets of work centre-specific rules for complex, dynamic problems that feature a large set of heterogeneous work centres. The proposed hyper-heuristic is specifically designed for shops that contain both serial and batch processing work centres, and further extends previously developed hyper-heuristics by optimising the information horizon of each work centre, i.e. the degree to which jobs that are expected to arrive in the near future are considered in the decision making. The motivation for the latter extension is that it may be beneficial to wait for future jobs at some work centres, e.g. to save a long setup time or to support the operation of a downstream bottleneck, while it may be counterproductive at others, e.g. at highly utilised work centres (see also Section 3.4).

7.2 Algorithm Design

In general, the hyper-heuristic operates as described in Section 4.1, and the focus of this section is to detail the algorithm-specific components, namely the encoding of candidate solutions and the evaluation, selection and reproduction of individuals.

7.2.1 Encoding of Solutions

The encoding of candidate solutions, i.e. rule sets in this case, is illustrated in Figure 7.1. Each rule set is represented by a one-dimensional array (or genome), consisting of three segments of length c, where c denotes the number of work centres in the shop. The first segment contains non-negative integer values that refer to the indices of the dispatching rules used at the respective work centres. However, most dispatching rules are designed to select jobs, not



Dispatching rule	Index
SPT	0
SRPT	1
FOPNR	2
…	…
FSASPT-SPT	20
BSPT	21

Batch rule	Index
MBS(0.125)	0
MBS(0.25)	1
MBS(0.375)	2
…	…
FB	9

Figure 7.1: Example of the encoding of dispatching rule sets as one-dimensional arrays for a problem with c = 4 work centres (grey parts are added for illustration purposes).

batches, and have to be combined with an additional rule to constitute a dispatching rule for batch processing work centres. These batch rules, which do not have any effect on the operation of serial processing work centres, are specified in the second segment. Finally, the third segment contains the values for the so-called look-ahead parameters, i.e. real values on the $[0, 1)$ interval that determine the information horizon of each work centre. More precisely, a look-ahead value close to 1.0 implies that all jobs that are expected to arrive at the work centre before $t + 1.0 \times \max_{j \in N^{\mathrm{Q}}}(s_{ef_{j,i}} + p_{j,i} + s_{f_{j,i}e})$ are considered in the decision making, where t denotes the current time, $p_{j,i}$ the processing time of the imminent operation of job j, $s_{ef_{j,i}}$ and $s_{f_{j,i}e}$ the setup time for changing over to the setup required for that operation and back to the current setup, and N^{Q} refers to the set of queueing jobs. The motivation for this upper bound is that all jobs arriving within this time horizon are potentially affected by the next decision of a given dispatching rule, which may well prioritise the job with the longest operation in the queue. Moreover, the upper bound takes into account that, in the worst case, the quickest way of changing over to the setup required by the subsequent job may be by changing back to the original setup first. On the other hand, a natural lower bound for the information horizon is to restrict the decision making to jobs in the queue, which corresponds to a value of 0 for the look-ahead parameter. To illustrate, the individual in Figure 7.1 encodes a rule set for a problem with four work centres, where Work Centre 1 operates according to the MBS(0.25)-FOPNR rule and considers all jobs that arrive before $t + 0.21 \times \max_{j \in N^{\mathrm{Q}}}(s_{ef_{j,i}} + p_{j,i} + s_{f_{j,i}e})$. Clearly, a rule set can only consist of rules that are provided to the hyper-heuristic, and an important design decision thus concerns the selection of adequate rules as they define the search space and the rule sets that can be generated.

7.2.2 Evaluation

Following the decoding of an individual into the corresponding rule set, a simulation experiment is executed to evaluate the performance of the rule set for the given objective, which measures the fitness of the individual. The simulation setting used within the GP hyper-heuristic (see Section 6.2.2) is adopted for this purpose, i.e. one replication is run to evaluate an individual, the same random number stream is used for all individuals belonging to the same generation, and the random number streams are changed from generation to generation. Furthermore, to detect individuals that lead to an unstable system and save time on their evaluation, the number of jobs in the shop is monitored again and the simulation run aborted, if the number of jobs exceeds a certain threshold value, resulting in a high penalty on the fitness value of the corresponding individual.

7.2.3 Selection

For the selection of promising individuals, the μ+λ breeder of ECJ is employed (Luke et al., 1998). This selection scheme chooses the μ individuals with the highest fitness values from the current population to produce λ children, with which together they form the population of the next generation. The population size is therefore $\mu + \lambda$. However, only one of two parents required for the crossover operation is taken from the set of the μ best individuals, the other is determined as the better of two randomly sampled individuals from the current population. Note that this selection scheme is elitist, i.e. the best individuals of each generation are guaranteed to survive into the next generation.

7.2.4 Reproduction

New individuals are created by successively applying two standard EA operators, called uniform crossover and uniform mutation. The uniform crossover recombines two individuals by randomly swapping over genes between them, where each gene is swapped with a 50% probability. Figure 7.2 presents an example of the crossover, where the white and grey parts within the two children originate from the first and second parent, respectively. Though the uniform crossover yields two new individuals, only one of them is kept by the hyper-heuristic. This individual is then subjected to the uniform mutation operator, which modifies individuals by replacing the value of a gene with a random value generated from a uniform distribution on the respective range of feasible values. Figure 7.3 illustrates this operation, where altered entries are encircled. Both crossover and mutation are applied with a certain probability, which are parameters of the hyper-heuristic that are specified in the following section.

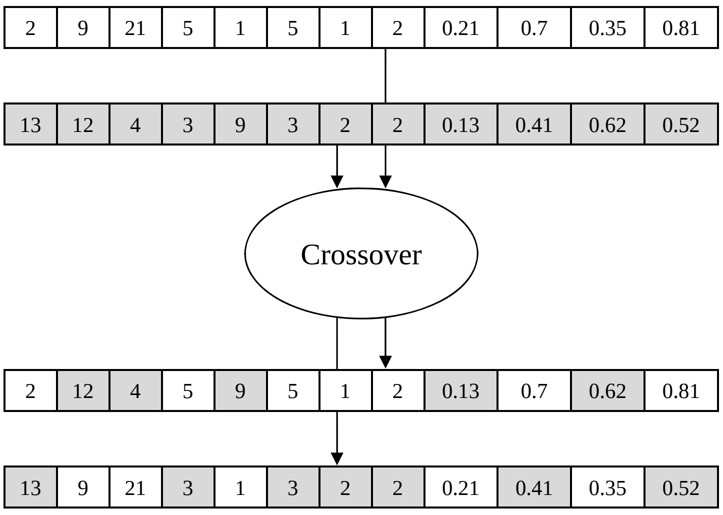


Figure 7.2: Example of a uniform crossover operation. Genes are swapped between the parents with a 50% probability.

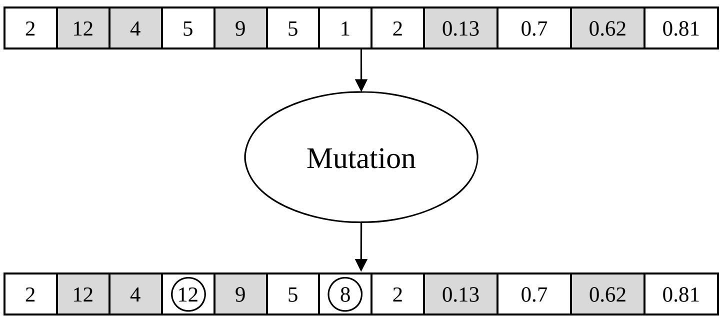


Figure 7.3: Example of a uniform mutation operation. Gene values are replaced with random values generated from a uniform distribution.

7.3 Application to Scheduling Problems from Semiconductor Manufacturing

To assess its effectiveness, the hyper-heuristic is tested on several scheduling problems, as described in this section.

7.3.1 Scenario Description

The hyper-heuristic is applied to the same four scheduling problems as the GP hyper-heuristic from Chapter 6, which are denoted by $FJc|r_j, s_{ef}, B_f|\overline{C}$ and $FJc|r_j, s_{ef}, B_f|\overline{wT}$, respectively (see Section 6.3.1). These problems feature different types of work centres that vary with regard to their setup requirements, batch processing capacities, number of parallel machines, workload and their position in the shop. Hence, they represent a typical scenario in which one may want to use a set of work centre-specific rules, and are therefore well suited to test the proposed hyper-heuristic. Moreover, by applying the rule set hyper-heuristic to the same problems as the GP hyper-heuristic, the rules evolved by the latter can be provided to the former in the attempt to generate even better rule sets.

As described in Section 6.3.1, the problems involve two shops, referred to as Fab4r and Fab6, that contain 31 and 104 work centres, respectively. However, some of these work centres have such a low utilisation that their impact on the overall performance of the shop is likely to be negligible. Table 7.1 lists the (most) critical work centres of the two shops, which have been identified by examination of the average queue lengths that arise at the various work centres if the respective shop operates according to the FCFS rule, where the names of the work centres are taken from the MASM data set (Feigin et al., 1994). Note that the utilisation of work centres involving sequence-dependent setup times and batch processing generally depends on the dispatching rule used. Hence, a lower bound is given in Table 7.1 for the utilisation of these work centres, which is based on the assumption that only full batches are

Table 7.1: Critical work centres of the two semiconductor manufacturing shops.

Shop	Name	Special properties	Utilisation	Position index
Fab4r	AME135	–	93.8%	0.89
	D1-9	Parallel machines	91.1%	0.53
	DFC4	–	89.1%	0.91
	PE1-5	Parallel machines; Setups	$\geq$ 74.3%	0.51
	QLESS	Batch processing	$\geq$ 64.8%	0.52
Fab6	AME 4+5+7+8	Parallel machines; Setups	$\geq$ 56.2%	0.64
	ASM B2	Batch processing	$\geq$ 90.5%	0.13
	CAN 0.43/MII	Parallel machines	87.6%	0.54
	LTS 3	Parallel machines	92.3%	0.64

processed and setups can always be avoided. Clearly, these work centres can quickly become the bottleneck if operated in an inefficient way, e.g. by regularly conducting time-consuming setups or processing near-empty batches.

The position of a work centre is captured by an index that is computed as follows. First, the index of each operation is normalised to the length of the associated route (in terms of number of operations) so that the indices of the first and last operation of each route take the values of 0 and 1, respectively. Then, the position index of a work centre is calculated as the weighted average of the normalised indices of operations to be performed on that work centre, where operations are weighted by their relative frequency, which is determined by the job arrival rates. Hence, a higher position index indicates that the work centre is further downstream. Moreover, in the extreme case of a flow shop, the position indices are uniformly distributed on the $[0, 1]$ interval, where the first work centre has an index of 0 and the last an index of 1, while in the other extreme of an ideal job shop all work centres have an index of 0.5.

7.3.2 Parameter Setting

As shown in Table 7.2, the rule set hyper-heuristic is provided with a number of dispatching rules from the literature, including those that are known to be effective for similar problems. A description of these rules can be found in Chapter 3, except for LW, FOPNR and S/OPN, which prioritise jobs in non-increasing order of their weight, and in non-decreasing order of their number of remaining operations and slack per remaining operation, respectively. Some of the rules, especially those whose application often results in two or more jobs having the same priority, incorporate a secondary rule to break ties. These rules are identified by a '-' in the table, separating the primary rule from the tie-breaking rule. BATCS^{x}, which reverts to ATCS for serial processing work centres, and MBS are further provided with a range of common parameter values (Lee, Bhaskaran, and Pinedo, 1997; Perez, Fowler, and Carlyle, 2005), where $B_{\max}$ denotes the maximum possible batch size for the given problem.

Table 7.2: Parameter setting of the rule set hyper-heuristic.

Parameter	
Number of generations	250
Population size	200
Initial population	Randomly generated
Selection parameter μ	100
Selection parameter λ	100
Crossover probability	100%
Mutation probability	5%
Simulation run length	5 $\times$ warm-up period
Dispatching rules	SPT, LWKR, FOPNR, WINQ, PT+WINQ,
	2PT+WINQ+NPT, BLWKR, BLWKR^{x}
	(all as exhaustive and non-exhaustive rules),
	SSPT, SNSPT, FSAST-SPT, (exhaustive) FSASPT-SPT, BSPT,
(only $\overline{wT}$ problems)	EDD, ODD, S/OPN, CR, OCR, MDD, MOD,
	LW-EDD, LW-ODD, LW-S/OPN, LW-CR, LW-OCR,
	LW-MDD, LW-MOD, WMDD, WMOD, CR+SPT,
	S/RPT+SPT (all as exhaustive and non-exhaustive rules),
	BATCS^{x}(k_1 = 1.0, 1.5, . . . , 5.5, k_2 = 0.01, 0.05, . . . , 2.0)
Batch rules	MBS($L = 1/B_{\max}, 2/B_{\max}, \ldots, 1$), MCB, FB

The other parameters of the hyper-heuristic are set in line with commonly used EA parameter settings, as indicated in Table 7.2, where the simulation setting for the evaluation of individuals is adopted from the GP hyper-heuristic (see Section 6.3.2). Using this parameter setting, the hyper-heuristic is applied to the four scheduling problems, running 20 replications for each of the two Fab4r problems and 10 replications for each of the two Fab6 problems.

In addition, another series of experiments is conducted in which the rule set hyper-heuristic is also provided with a rule evolved by the GP-hyper-heuristic to assess the potential of combining the two hyper-heuristics. The GP rule, i.e. an individual encoding a rule set that implies the use of the GP rule at every work centre, is then further included in the initial population of the rule set hyper-heuristic. Again, 20 and 10 replications of the hyper-heuristic are run for each of the two Fab4r and Fab6 problems, respectively, using the same random number streams.

7.4 Experimental Results

The evolved rule sets are evaluated on the respective problem they have been generated for, by running additional simulation experiments with different random number streams than those used in their generation. The performance measurements for the Fab4r and the Fab6 problems are based on a sample size of $n = 50000$ and $n = 30000$ jobs per simulation run, where the warm-up period has been determined to comprise the completion of 3000 and 1500 jobs, respectively. The performance of the rule sets evolved for the two $FJc|r_j, s_{ef}, B_f|\overline{C}$ problems is discussed first, followed by a presentation of the results for the $FJc|r_j, s_{ef}, B_f|\overline{wT}$ problems.

7.4.1 Minimising Mean Flow Time

Since the hyper-heuristic is applied to the same problems as the GP hyper-heuristic, the same rules, particularly those that are found to be most effective in Section 6.4, are used as benchmarks. Table 7.3 presents the results of the simulation experiments for the Fab4r problem, comparing the mean flow time (and mean queueing time) performance of the best and the worst of the 20 rule sets, generated in 20 replications of the hyper-heuristic (without access to a GP rule), with that of the best benchmark rule, MCB-PT+WINQ. In the following tables, values in bold indicate the best performance achieved by any rule (set) for a given objective, while those marked with the same superscript, e.g. the mean flow time values of MCB-FSASPT-SPT and FSASPT-BSPT in Table 7.5, are not significantly different from each other according to the Bonferroni-Dunn test, at the overall 5% significance level. To distinguish the different rule sets, the shop and objective that a Rule Set (RS) has been evolved for are given in brackets in the tables. Table 7.3 shows that the hyper-heuristic is well able to generate rule sets that outperform the best benchmark rule for this problem, with even the worst rule set yielding a significantly lower mean flow time than MCB-PT+WINQ. On average, the evolved rule sets produce a mean flow time of 9544.2, which is an unbiased measure of the effectiveness of the hyper-heuristic, i.e. in a single run the hyper-heuristic can be expected to generate a rule set that yields a mean flow time close to 9544.2. However, the relative advantage of the evolved rule sets is comparatively small, with the best rule set reducing mean flow time over MCB-PT+WINQ by 4%, and mean queueing time by 5%.

Table 7.3: Performance of evolved rule sets and benchmark rules for the Fab4r shop regarding mean flow time and mean queueing time. The best performance values are in bold.

Dispatching rule (set)	Mean flow time	Mean queueing time
MCB-PT+WINQ	9849.0	6885.9
RS Worst(Fab4r, $\overline{C}$)	9580.6	6617.6
RS Best(Fab4r, $\overline{C}$)	**9490.4**	**6527.4**

To determine whether the combination of the two hyper-heuristics is beneficial, the performance of the rule sets evolved with access to a GP rule, subsequently referred to as GP rule sets, is compared to the performance of the better of the respective GP rule and rule set that result from applying the two hyper-heuristics separately from each other, using the same random number streams. This is considered a fair benchmark as the computational time for running both hyper-heuristics separately is similar to running them in sequence, which is necessary to generate a GP rule set. Figure 7.4 illustrates this replication-wise comparison of GP rules and rule sets, evolved for the Fab4r shop with the minimum mean flow time objective, with the GP rule sets. The comparison reveals that, despite achieving a lower mean flow time on average (9499.7 compared to 9544.2), only 14 of the 20 GP rule sets perform significantly better than the corresponding conventional rule set (which are generally better than the GP rules) at the overall 5% significance level. In other words, while there are some indications that providing the rule set hyper-heuristic with a GP rule may allow for the generation of a superior rule set, it cannot be concluded that the combination of the two hyper-heuristics is generally beneficial for this problem.

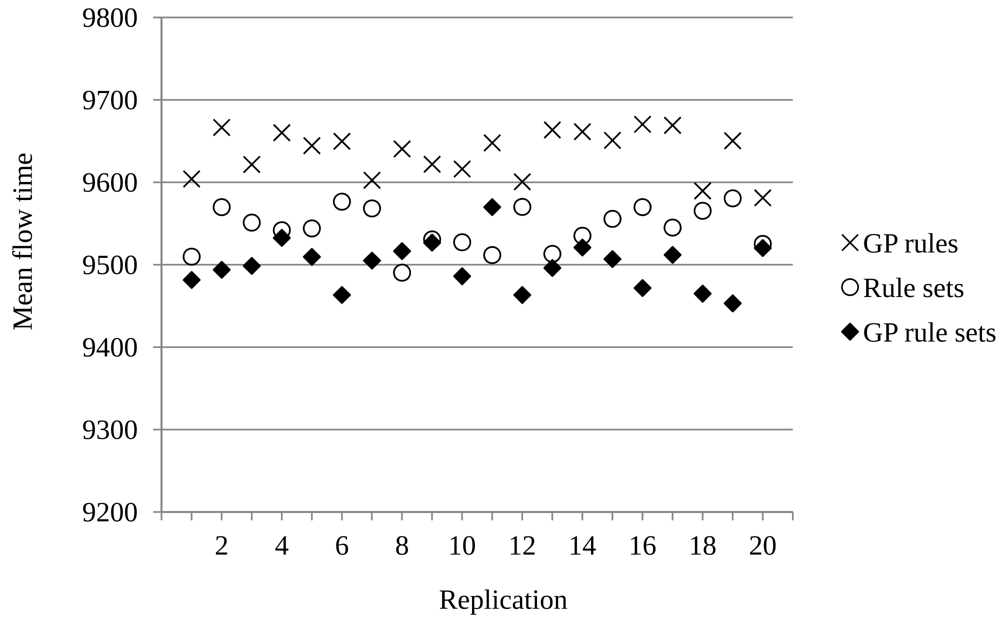


Figure 7.4: Performance of GP rules and rule sets, evolved with and without access to the respective GP rule, for the Fab4r shop with the minimum mean flow time objective.

Table 7.4 lists the mean flow time produced by the best overall rule set, which results from a run in which the hyper-heuristic is provided with a GP rule, and the best single rule, i.e. the best GP rule, for jobs classified by their product type. It shows that the GP Rule Set (GPRS) gains its relative advantage by decreasing the mean flow time for the shorter and more frequent product B at the expense of a slightly higher mean flow time for product A.

Table 7.5 presents the results for the Fab6 problem, comparing the performance of the best and the worst of the ten rule sets, evolved without access to a GP rule, with that of the best benchmark rules. They indicate that the hyper-heuristic is rather ineffective for this problem as the best rule set is found to perform only marginally better than the benchmark rules, decreasing mean flow time and mean queueing time by less than 1%. Moreover, the

Table 7.4: Mean flow time performance of the best rule set and the best single rule for the Fab4r shop, subdivided by product type. The best performance values are in bold.

Dispatching rule (set)	Mean flow time	
Product type	A	B
GP Best(Fab4r, $\overline{C}$)	**22192.0**	4188.9
GPRS Best(Fab4r, $\overline{C}$)	22309.8	**3955.8**

Table 7.5: Performance of evolved rule sets and benchmark rules for the Fab6 shop regarding mean flow time and mean queueing time. Differences between performance values marked with the same superscript are not significant at the overall 5% significance level. The best performance values are in bold.

Dispatching rule (set)	Mean flow time	Mean queueing time
MCB-FSASPT-SPT	27903.2^{a}	8447.6^{a}
FSASPT-BSPT	27871.3^{a}	8415.7^{a}
RS Worst(Fab6, $\overline{C}$)	28052.2	8596.6
RS Best(Fab6, $\overline{C}$)	**27800.2**	**8344.6**

evolved rule sets yield a mean flow time of 27916.1 on average, which is higher than that achieved by the FSASPT-BSPT or the MCB-FSASPT-SPT rule. In other words, the hyper-heuristic cannot be expected to generate a better rule set in a single run.

In view of these disappointing results, it is surprising to find that the hyper-heuristic is able to generate very effective rule sets for this problem if it is provided with a GP rule, with the best GP rule set achieving a 1% reduction of mean flow time, or 4% reduction of mean queueing time, over the best GP rule. In fact, the replication-wise comparison, illustrated in Figure 7.5, shows that all ten GP rule sets, which yield a mean flow time of 27355.1 on average, perform significantly better than the respective GP rule and conventional rule set (and therefore better than the better of the two) at the overall 5% significance level, which indicates that the hyper-heuristics complement one another very well in this case.

Figure 7.6 displays the mean flow time of jobs by product type, resulting from the application of the best (GP) rule set and the best single (GP) rule. As in the case of the Fab4r problem, the rule set is shown to gain its relative advantage by accelerating the processing of the shorter products, thereby delaying the completion of the longer ones.

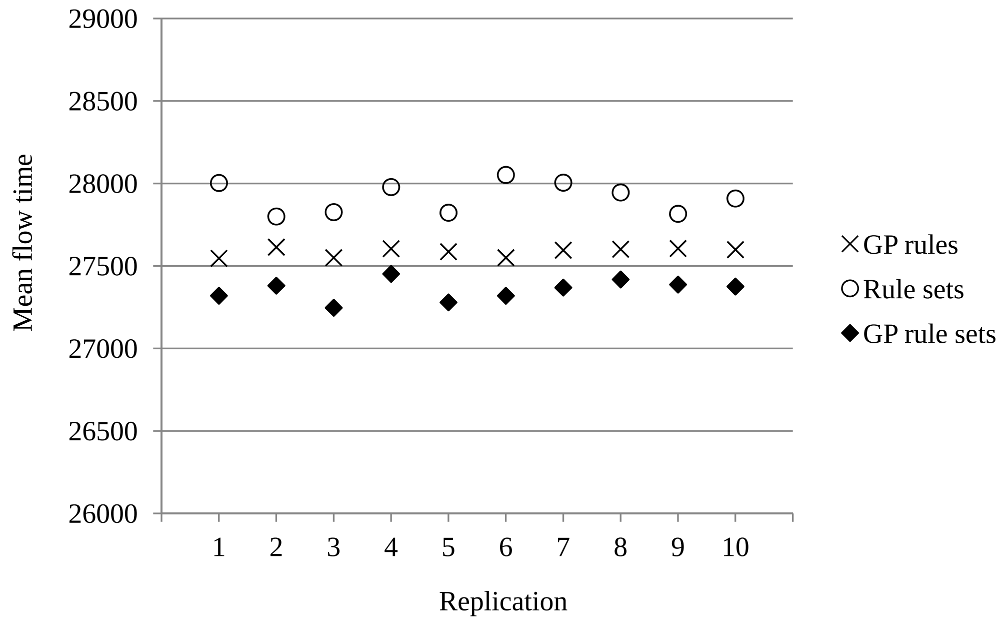


Figure 7.5: Performance of GP rules and rule sets, evolved with and without access to the respective GP rule, for the Fab6 shop with the minimum mean flow time objective.

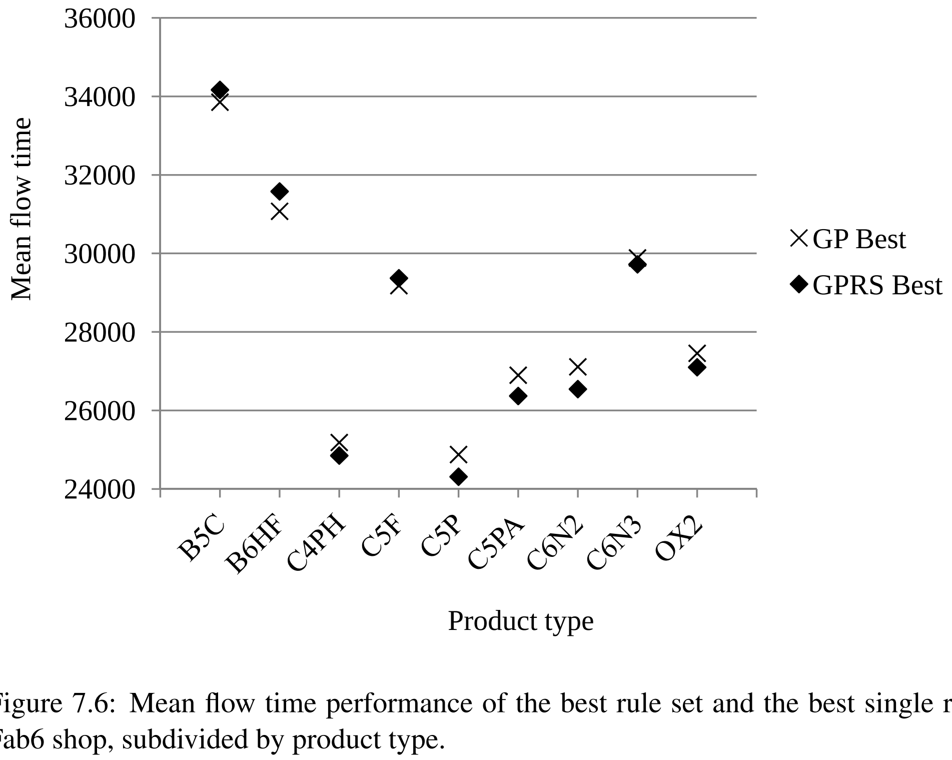


Figure 7.6: Mean flow time performance of the best rule set and the best single rule for the Fab6 shop, subdivided by product type.

7.4.2 Minimising Mean Weighted Tardiness

Table 7.6 presents the results for the Fab4r problem with the minimum mean weighted tardiness objective, comparing the performance of the best and the worst of the 20 rule sets, evolved for this problem, with that of the best benchmark rules. It shows that the hyper-heuristic has the potential to generate rule sets that significantly outperform the benchmark rules, reducing mean weighted tardiness by about 25%, but that it is not always able to tap its potential, as demonstrated by the relatively poor performance of the worst rule set. On the other hand, the mean weighted tardiness achieved by the rule sets on average is 861.1, which is clearly lower than that produced by the benchmark rules. Thus, the hyper-heuristic can be expected to generate a better rule set in a single run, which illustrates its effectiveness.

Table 7.6: Performance of evolved rule sets and benchmark rules for the Fab4r shop regarding mean weighted tardiness and other due date-related measures. Differences between performance values marked with the same superscript are not significant at the overall 5% significance level. The best performance values are in bold.

Dispatching rule (set)	Mean weight. tardiness	Mean tardiness	Proportion tardy (in %)	Cond. mean tardiness
MCB-S/RPT+SPT	1031.3[a]	634.6	28.6	2208.7
MCB-ATCS(k_1 = 4.0, k_2 = 0.01)	1017.5[a]	442.6	22.0	1992.3
MCB-WMOD	1007.5[a]	429.0	26.9	**1581.3**[a]
RS Worst(Fab4r, $\overline{wT}$)	1106.2	602.3	35.6	1685.5
RS Best(Fab4r, $\overline{wT}$)	**754.2**	**311.9**	**20.1**	**1531.2**[a]

In addition, providing the hyper-heuristic with a GP rule proves to be highly effective for this problem. The replication-wise comparison reveals that all GP rule sets yield a significantly lower mean weighted tardiness than the better of the GP rule and the rule set evolved without access to the GP rule (see Figure 7.7), which strongly suggests that combining the hyper-heuristics is generally beneficial in this case. This conclusion is further supported by the big difference between the average performance of the GP rule sets, which achieve a mean weighted tardiness of 633.6, and the mean weighted tardiness of 768.9 that results from averaging the 20 performance values of the better of the GP rule and the conventional rule set of each replication. In particular, the best GP rule set is found to reduce mean weighted tardiness over the best GP rule by another 15%.

Regarding other measures of due date performance, the best (GP) rule set is found to decrease the proportion of tardy jobs over the best single (GP) rule by 7.1 percentage points, but to increase mean tardiness by 5%. This indicates that the rule set gains its relative advantage by focussing on the timely completion of jobs with larger weights, while accepting relatively large due date violations for jobs with small weights. Figure 7.8, which displays the mean weighted tardiness that results from applying the best GP rule set (GPRS Best) and the best

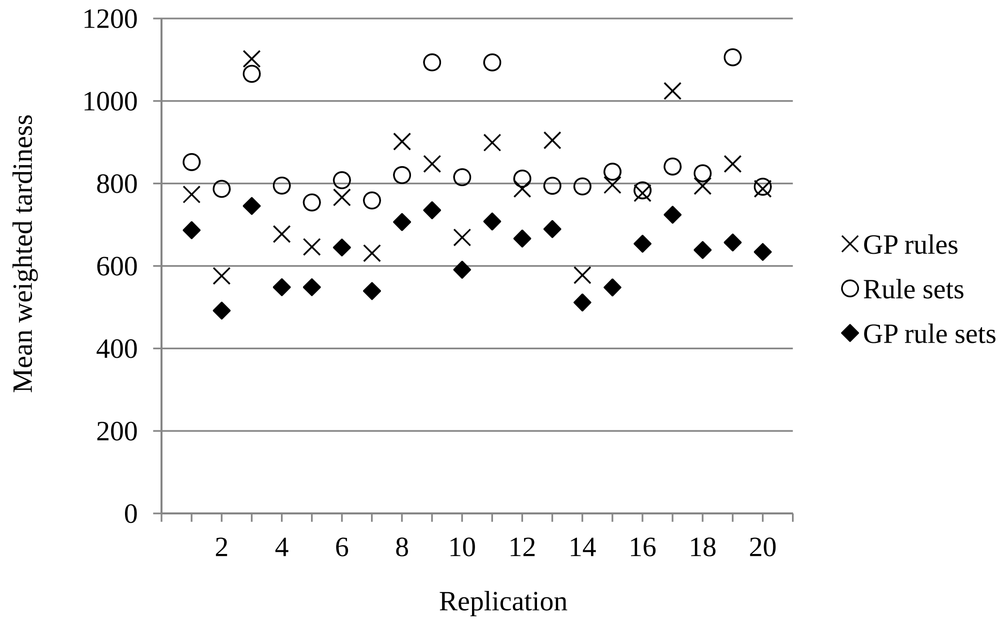


Figure 7.7: Performance of GP rules and rule sets, evolved with and without access to the respective GP rule, for the Fab4r shop with the minimum mean weighted tardiness objective.

GP rule (GP Best) for jobs classified by weight, confirms this notion. It shows that the rule set yields a lower mean (weighted) tardiness for jobs with weight equal to or larger than 3 at the expense of a further delay of jobs with smaller weights.

Table 7.7 compares the performance of the best and the worst of the ten rule sets, evolved for the Fab6 problem, with that of MCB-WMOD, the best benchmark rule, regarding mean weighted tardiness and other related criteria. Again, the hyper-heuristic proves to be effective, generating rule sets that generally yield a significantly lower mean weighted tardiness (2872.3 on average) than the benchmark rule, with the best rule set achieving a reduction of 15%.

Table 7.7: Performance of evolved rule sets and benchmark rules for the Fab6 shop regarding mean weighted tardiness and other due date-related measures. Differences between performance values marked with the same superscript are not significant at the overall 5% significance level. The best performance values are in bold.

Dispatching rule (set)	Mean weight. tardiness	Mean tardiness	Proportion tardy (in %)	Cond. mean tardiness
MCB-WMOD	3290.1	**1672.9**	31.8	**5242.0**
RS Worst(Fab6, $\overline{wT}$)	2995.4	1803.4	27.5	6539.4[a]
RS Best(Fab6, $\overline{wT}$)	**2796.3**	1699.9	**26.0**	6494.4[a]

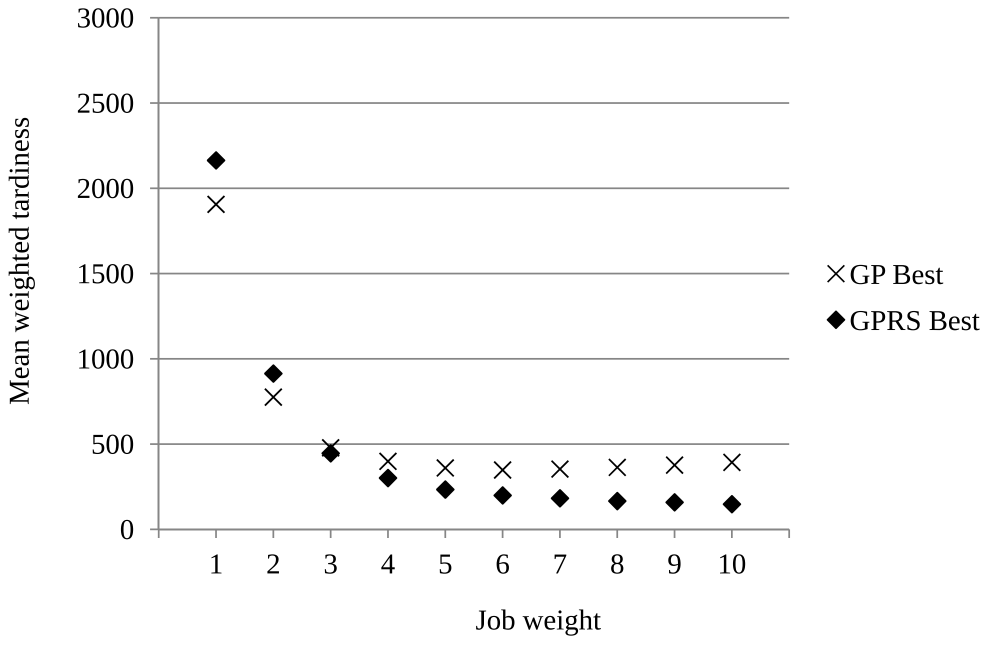


Figure 7.8: Mean weighted tardiness performance of the best rule set and the best single rule for the Fab4r shop, subdivided by job weight.

Moreover, Figure 7.9 shows that providing the hyper-heuristic with GP rules allows for the generation of even better rule sets, with all GP rule sets, which yield a mean weighted tardiness of 2409.0 on average, performing significantly better than the respective GP rule and conventional rule set (and therefore better than the better of the two). Hence, it can be concluded that the combination of the two hyper-heuristics is beneficial for this problem. In particular, the best GP rule set is found to reduce mean weighted tardiness over the best GP rule by another 12%.

The dominance of the best GP rule set over the best GP rule proves to carry over to other measures of due date performance. More specifically, the rule set achieves a 7% lower mean tardiness and decreases proportion of tardy jobs by 4.5 percentage points, while increasing conditional mean tardiness by 10%. In other words, it minimises the average delay of all jobs, irrespective of their weight, by completing more jobs on time. This is also reflected by Figure 7.10, which shows that the rule set yields a lower mean (weighted) tardiness than the single rule for almost every category of jobs, classified by weight.

In summary, for each of the four examined problems the hyper-heuristic proves to generate sets of work centre-specific rules that are significantly better than single benchmark rules from the literature, which demonstrates the effectiveness of the method. For three of the four problems, the hyper-heuristic further shows to generate even better rule sets if it is provided with a GP rule, which also outperform the respective GP rule, indicating that there is some potential

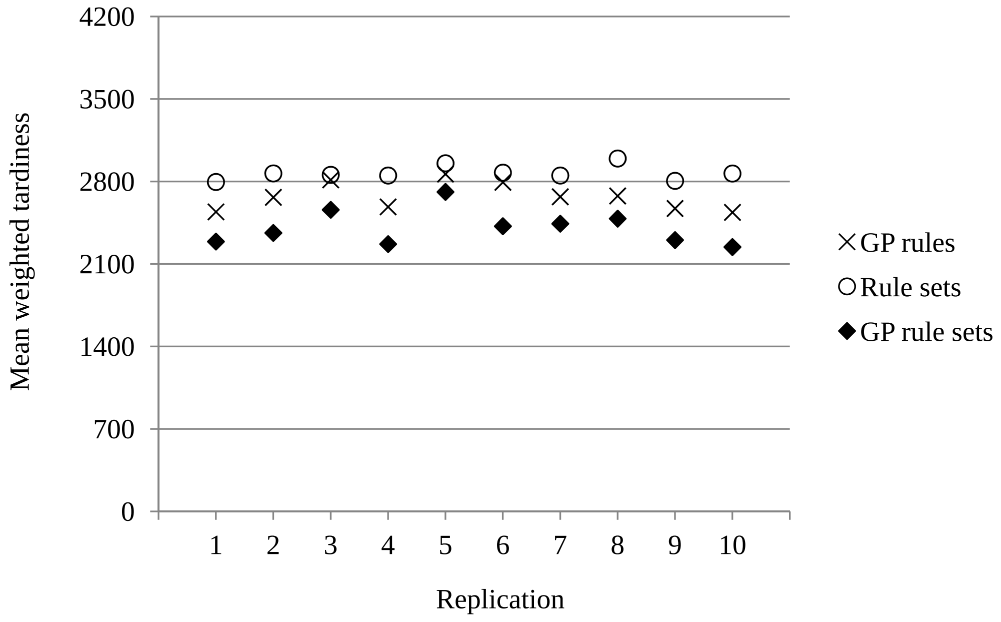


Figure 7.9: Performance of GP rules and rule sets, evolved with and without access to the respective GP rule, for the Fab6 shop with the minimum mean weighted tardiness objective.

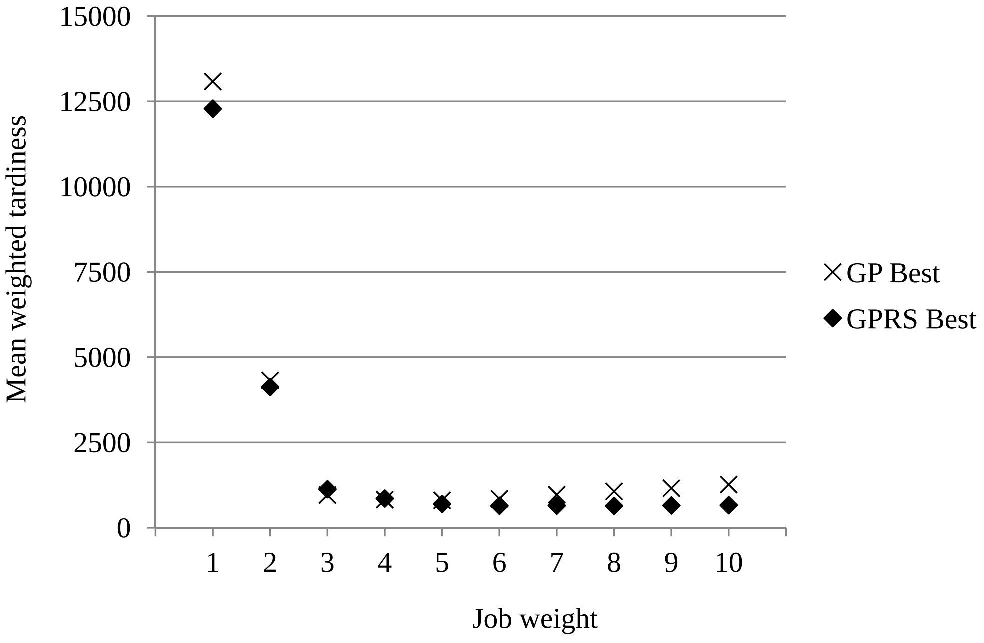


Figure 7.10: Mean weighted tardiness performance of the best rule set and the best single rule for the Fab6 shop, subdivided by job weight.

in combining the two hyper-heuristics. Moreover, the relative advantage of the best rule sets evolved for the $\overline{wT}$ problems is larger than that of the best rule sets evolved for the corresponding $\overline{C}$ problems, which is in line with the findings for the GP hyper-heuristic and supports the conjecture that the automatic generation of (work centre-specific) dispatching rules by means of hyper-heuristics is more promising for more complex problems.

Table 7.8 specifies the rules and look-ahead values of the best rule set, evolved for each of the four scheduling problems, for the critical work centres. As expected, the best rule sets, which all result from running the hyper-heuristic with access to a GP rule, consist of a mixture of the respective GP rule and some common rules. More specifically, the GP rules are primarily selected for work centres with complex properties such as batch processing capabilities or sequence-dependent setups (see also Table 7.1), for which it is more difficult to develop good rules manually. Given that these rule sets are the result of a twofold optimisation on the simulation model representing the original problem, they can be considered to be even more tailored to this problem than the single GP rules. Hence, they could be expected to be very sensitive to changing conditions in the shop, which is examined in the following section.

Table 7.8: Dispatching rules and look-ahead values used at critical work centres as part of the respectively best rule set.

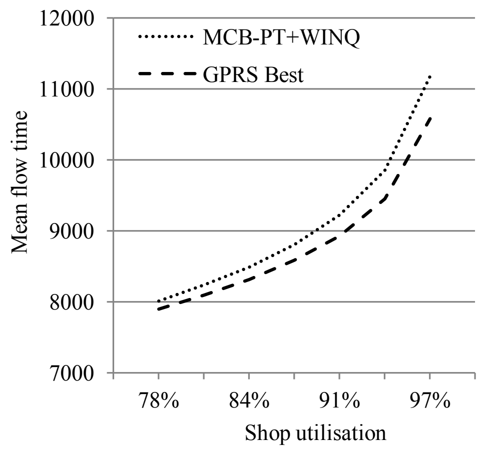
Shop	Work centre	($\overline{C}$ problem)		($\overline{wT}$ problem)	
		Rule	Look-ahead	Rule	Look-ahead
Fab4r	AME135	BLWKR	0.38	ATC(k_1 = 4.0)	0.07
	D1-9	BLWKR	0.00	WMDD	0.00
	DFC4	BLWKR^x	0.31	GP rule	0.00
	PE1-5	GP rule	0.02	GP rule	0.12
	QLESS	MCB-GP rule	0.14	FB-GP rule	0.04
Fab6	AME 4+5+7+8	GP rule	0.65	GP rule	0.08
	ASM B2	BSPT	0.09	GP rule	0.14
	CAN 0.43/MII	GP rule	0.20	ATC(k_1 = 1.0)	0.00
	LTS 3	SPT	0.00	GP rule	0.00

7.5 Sensitivity Analysis

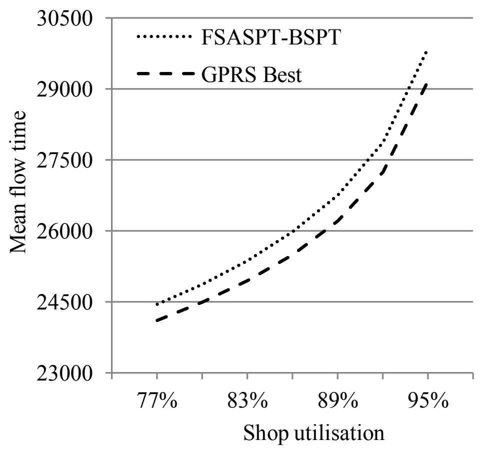
The sensitivity analysis carried out for the GP rule sets is similar to the one conducted for the GP rules in Section 6.5, i.e. all four rule sets are tested under different job arrival rates, while those evolved for the $\overline{wT}$ problems are also subjected to different due date settings. Again, the best rules from the literature, which are considered to be robust, are used as benchmarks for the robustness of a rule set.

7.5.1 Arrival Rates

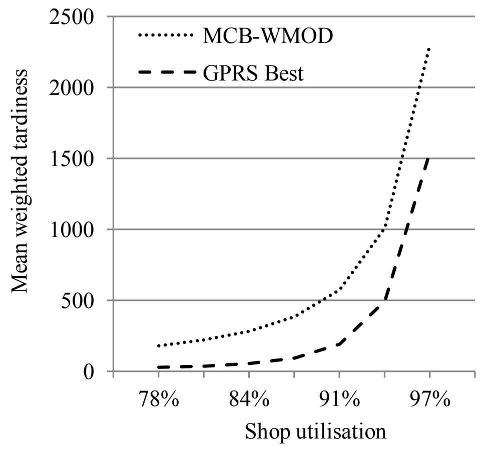
Figure 7.11 compares the performance of the best rule set and the best benchmark rule for the four scheduling problems under different utilisation levels, resulting from a different total job arrival rate, where the default arrival rates lead to a utilisation of 93.8% and 92.3% for the Fab4r and Fab6 shop, respectively. In every of the four cases, the rule set proves to dominate the benchmark rule across the whole range of examined utilisation levels, which indicates that the rule sets evolved by the hyper-heuristic are generally robust to changes in the total job arrival rate.



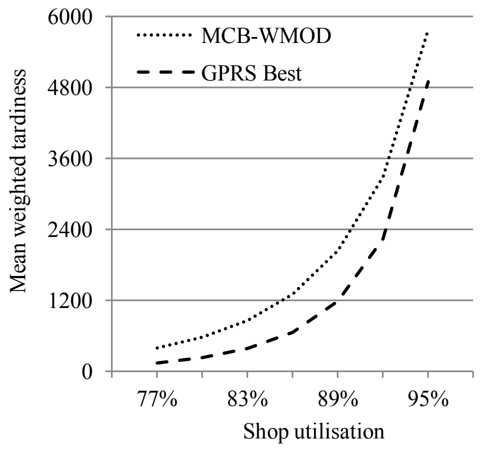
(a) Fab4r, $\overline{C}$ problem



(b) Fab6, $\overline{C}$ problem



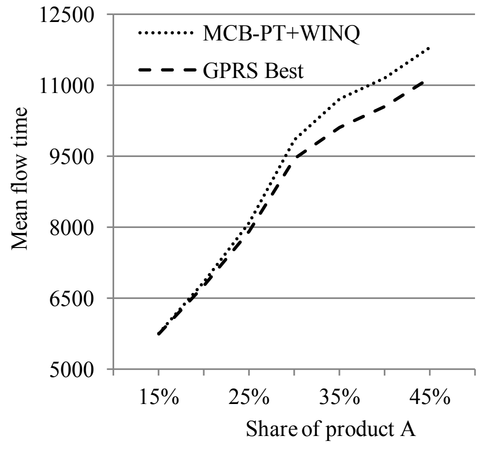
(c) Fab4r, $\overline{wT}$ problem



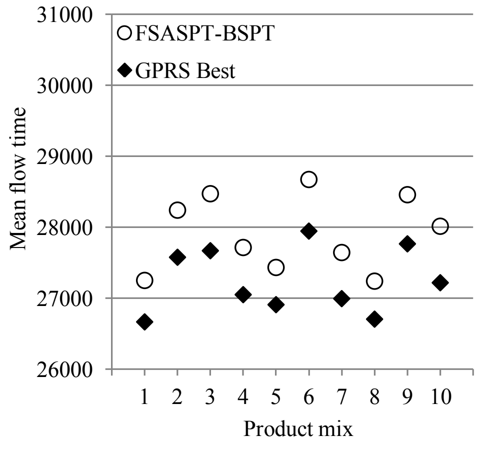
(d) Fab6, $\overline{wT}$ problem

Figure 7.11: Performance of the best rule set and the best benchmark rule for the four scheduling problems under varying levels of utilisation.

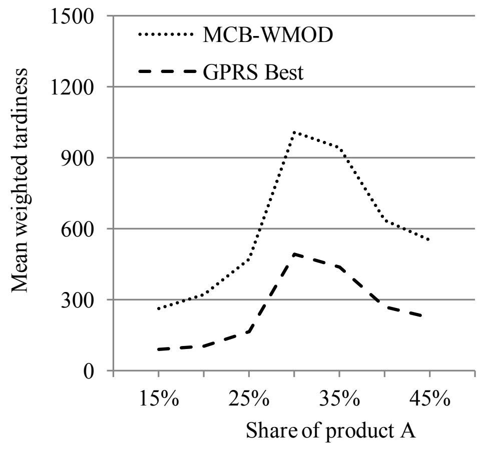
The effect of a change in the product mix, resulting from an increase and/or decrease of the arrival rate of certain products, on the relative performance of the evolved rule sets is shown in Figure 7.12, where the original mix for the Fab4 shop corresponds to a 30% share of product A and the product mixes for the Fab6 shop are derived from the original mix as described in Section 6.5.1. Again, the utilisation of the bottleneck, whereever its location, is held constant in these experiments. Despite comparatively big differences in the absolute performance values, the rule sets show to more or less maintain their relative advantage, which indicates that they are generally robust to changes in the product mix.



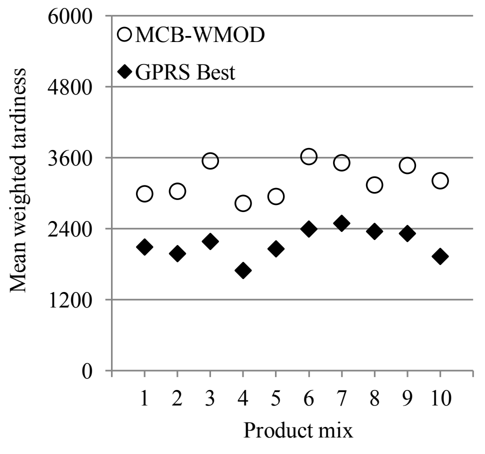
(a) Fab4r, $\overline{C}$ problem



(b) Fab6, $\overline{C}$ problem



(c) Fab4r, $\overline{wT}$ problem

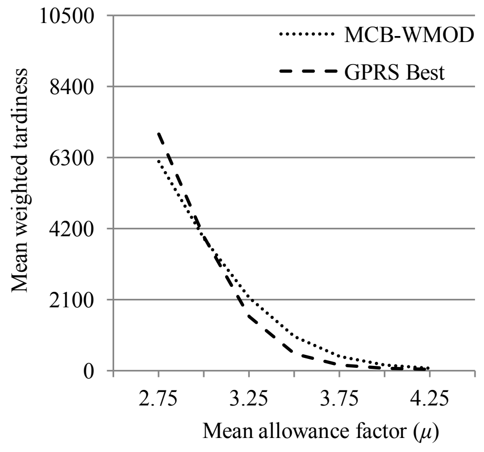


(d) Fab6, $\overline{wT}$ problem

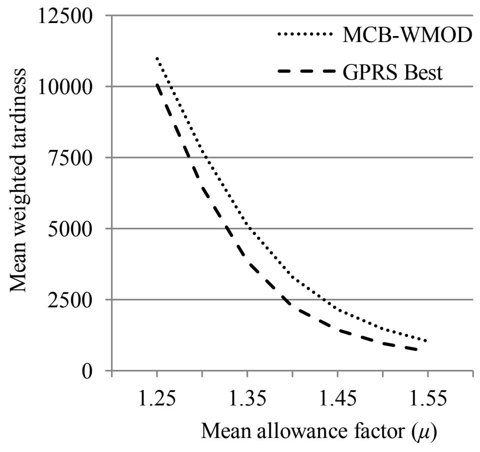
Figure 7.12: Performance of the best rule set and the best benchmark rule for the four scheduling problems under varying product mixes.

7.5.2 Due Date Setting

The GP rule sets evolved for the $\overline{wT}$ problems are assessed for their robustness to two parameters of the due date setting, namely the tightness and range of due dates, which are respectively controlled by the mean μ and the range R of the uniform distribution used to generate the allowance factors for the TWK due date assignment method (see Section 6.3.1). Figures 7.13 and 7.14 compare the performance of the best rule set and the best benchmark rule for the two $\overline{wT}$ problems for different values of μ and R, respectively, where the default due date setting corresponds to $\mu = 3.5$ and $R = 3$ for the Fab4r shop, and to $\mu = 1.4$ and $R = 0.4$ for the Fab6 shop. Similar to the single rules evolved by the GP hyper-heuristic, the rule sets are found to be robust to changes in the due date setting apart from an increasing tightness of due dates, i.e. a decreasing mean allowance factor, for which they show to loose their relative advantage. However, the comparatively smaller gradient indicates that they are less sensitive than the single GP rules (see Section 6.5.2), which is surprising given that the GP rule sets can be considered to be more tailored to the original setting.



(a) Fab4r problem



(b) Fab6 problem

Figure 7.13: Performance of the best rule set and the best benchmark rule for the two $\overline{wT}$ problems under varying levels of due date tightness. Due dates are generated by means of the TWK method with allowance factors drawn from a uniform distribution on the interval $[\mu - \frac{R}{2}, \mu + \frac{R}{2}]$.

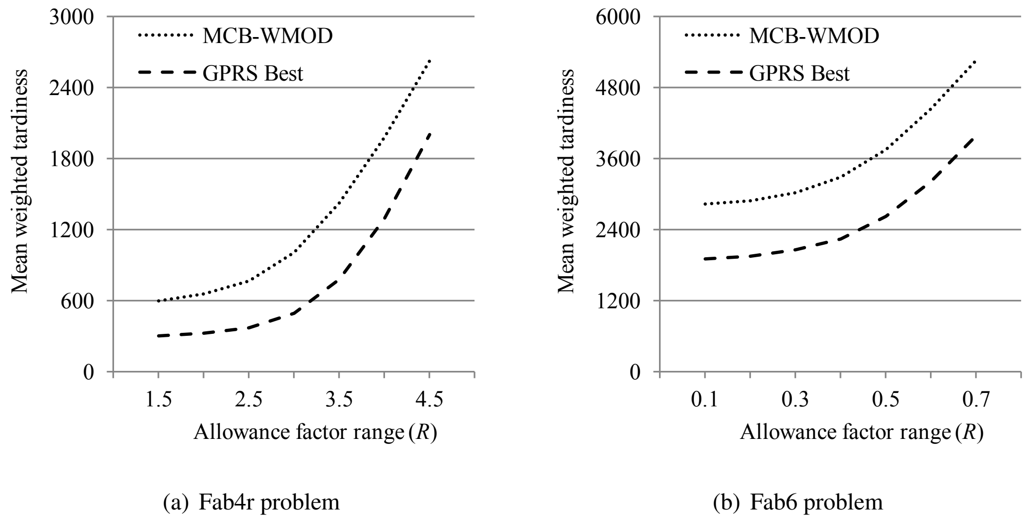


Figure 7.14: Performance of the best rule set and the best benchmark rule for the two $\overline{wT}$ problems under varying levels of due date range. Due dates are generated by means of the TWK method with allowance factors drawn from a uniform distribution on the interval $[\mu - \frac{R}{2}, \mu + \frac{R}{2}]$.

7.6 Critical Appraisal and Future Work

In summary, the hyper-heuristic presented in this chapter has shown to generate sets of work centre-specific dispatching rules for several complex and dynamic scheduling problems that are superior to single rules and generally robust to changes in the job arrival pattern and due date setting. The relative performance of the evolved rule sets is only found to deteriorate considerably if due dates become tighter (though at a slower rate than the GP rules from Chapter 6), which could be addressed in the same way as described in Section 6.6 for the GP hyper-heuristic. While these results demonstrate the potential of using a hyper-heuristic to generate sets of work centre-specific rules for complex and dynamic scheduling problems, the hyper-heuristic could be improved and extended in several ways.

First of all, like the GP hyper-heuristic from Chapter 6, the hyper-heuristic is characterised by extensive runtimes that are mainly due to the simulation-based evaluation of individuals. The runtime of one replication has been around 5 hours (on a 2.9 GHz Intel Quad-Core Xeon with 8 GB RAM) for the Fab4r problems, and has shown to increase to around 30 hours (on a similarly powerful 2.8 GHz Intel Quad-Core i5 with 4 GB RAM) for the Fab6 problems because of the larger shop (see Section 6.3.1), which is computationally more expensive to simulate. One main aim of future work should therefore be to improve the efficiency of the hyper-heuristic, e.g. by implementing some of the measures suggested in Section 6.6.

A possible way to further improve the effectiveness of the hyper-heuristic could be by a more sophisticated design of the look-ahead parameter so that the information horizon is set proportionally to the operation processing time of the queueing job with the highest priority (according to the given dispatching rule), rather than the processing time of the longest operation in the queue. The rationale behind this is that the job with the highest priority is the one that would be processed next if future job arrivals were not considered, and waiting for a future job should thus be assessed as an alternative to starting the operation of this candidate job. In particular, this modified look-ahead parameter would prevent the consideration of jobs that arrive after the processing of the candidate job would complete, which makes sense as such jobs are not affected by the decision to start the operation.

In view of the superior performance of the rule sets that result from the (sequential) combination of the two hyper-heuristics, another promising area for future work appears to be the development of a GP hyper-heuristic that generates several work centre-specific composite rules in parallel. However, in designing such a hyper-heuristic, one has to deal with the problem that the search space grows exponentially both in the number of rules and in the (maximum) number of parameters or symbols that define a rule. Hence, evolving a separate GP rule, i.e. priority index, of arbitrary complexity for every single work centre in the shop will generally not be feasible for shops of realistic size. Instead, the size of the search space has to be limited by restricting the complexity and/or the number of rules to be evolved.

Restricting the complexity of the GP rules, which is controlled by the maximum tree depth parameter, is based on the assumption that rules designed for specific work centres do not have to be as complex as rules designed for a whole shop in order to be effective. It has the positive side effect that simpler rules may be intelligible to practitioners, thereby increasing the acceptance of the method. On the other hand, setting the tree depth parameter to a low value does not only reduce the size of the search space but may also adversely affect the effectiveness of the GP search operators. The challenge is therefore to determine a (problem-specific) parameter value that sufficiently reduces the size of the search space without impairing the ability of the hyper-heuristic to generate good rules, which may not even be possible.

In contrast, the restriction of the number of rules to be evolved appears to be relatively straightforward. Often, the performance of a shop is mainly determined by a small subset of the present work centres, in which case the hyper-heuristic should obviously be configured to evolve rules for these important work centres while the others can be simply operated with some default base rule. However, the identification of important work centres may not always be easy. In fact, some experiments that involved running the critical work centres of the Fab4r and the Fab6 shop with the rules of the best rule sets (see Table 7.8) and all other work centres with the simple FCFS rule have shown that it is generally not sufficient to focus only on the highly utilised work centres. To further illustrate, consider a lowly utilised work centre that is located directly downstream of a batch processing work centre. Jobs are likely to arrive at the considered work centre in batches, in which case its dispatching rule can select among a number of jobs, from time to time, and may have a larger effect on the shop performance than the rule of another, moderately utilised work centre at which jobs arrive one by one. Hence, the importance of work centres generally depends on a number of factors, not just the utilisation. An additional way to reduce the number of rules to be evolved is to group similar work centres and generate only one rule per group. However, this creates the problem of determining an adequate classification of work centres, which, as briefly discussed in Section 7.1, can itself be very challenging.

Chapter 8

Conclusion

In this thesis, three methods, based on Evolutionary Algorithms (EAs), have been proposed for the design of dispatching rules for complex and dynamic scheduling problems. The first method employs an EA to search for job shop problem instances on which a given dispatching rule performs badly, which can be analysed to reveal weaknesses of the rule, thereby providing guidelines for the development of a better rule. The method has been applied to a benchmark rule for a well-studied dynamic job shop problem with the objective of minimising mean flow time, prompting the design of a new rule that does not only outperform all benchmark rules on the job shop problem but also on the corresponding flow shop problem, across different levels of utilisation. The other two methods fall into the category of hyper-heuristics and automate the rule development process by employing an EA directly to search for effective dispatching rules. The first hyper-heuristic is based on Genetic Programming (GP) and generates a single rule from basic job and machine attributes, whereas the second hyper-heuristic generates a set of work centre-specific rules by selecting a (potentially) different rule for each work centre from a number of existing rules. Both hyper-heuristics have been applied to some dynamic, flexible job shop problems from semiconductor manufacturing with the objectives of minimising mean flow time and mean weighted tardiness, respectively, for which they both have shown to generate (sets of) dispatching rules that outperform the benchmark rules from the literature and are generally robust to changes in the job arrival pattern and due date setting. Moreover, a sequential combination of the two hyper-heuristics, in which a rule evolved by the GP hyper-heuristic is provided to the rule set hyper-heuristic, has been found to be beneficial in most cases, with the resulting rule sets achieving an even better performance than the (sets of) rules evolved separately by the hyper-heuristics. In summary, the application of each method has resulted in dispatching rules that perform better than the best benchmark rules, which demonstrates the effectiveness of the methods.

Although the potential of these methods is generally higher for more complex problems, for which it is more difficult to design effective dispatching rules manually, a high problem complexity also poses a challenge to them. In particular, the method that evolves problem instances requires a solution method that can generate schedules of a high solution quality in a reasonable amount of time, which can quickly become a restrictive factor for more complex problems. In addition, an expert has to fully understand the generated schedules, especially the causes and effects of a bad decision taken by a tested rule, to be able to come up with suggestions on how to design a better rule, which may be too demanding if the problem is very complex. In contrast, the suitability of hyper-heuristics appears to be less affected by the problem complexity as they operate largely abstract from the considered problem. On the other hand, a more complex problem is often associated with a larger and more complex search space, which does affect the performance of a hyper-heuristic, as indicated by the results for the two proposed hyper-heuristics. Then, it becomes crucial to design the hyper-heuristic in an intelligent way, e.g. by providing it only with the most important attributes or by focussing its search on a subset of important work centres, which again requires a certain expertise and understanding of the problem. In other words, any of the proposed methods may prove to be impractical for extremely complex scheduling problems. However, in such a case it is also highly questionable whether an effective dispatching rule (set) can be designed without any methodic support.

Clearly, there are a number of issues that could be addressed in the future. Each of the methods can be further developed in various ways and some directions on future work have respectively been given in Chapters 5–7. There may also be additional potential in combining the methods, e.g. the method that evolves problem instances could be used to identify important information that could then be provided as an (aggregate) attribute to the GP hyper-heuristic. Above all, scheduling problems occur in many other areas that are characterised by complex and dynamic environments, e.g. airport traffic control, project management or health care operations, which could be addressed in a similar way. Thus, there are numerous opportunities to extend the scope of the methods by adapting and applying them to other problem areas in the future.

Appendix A

p-Values of Pairwise Comparisons

p-Values relating to Chapter 5

Table A.1: *p*-values of pairwise comparisons relating to Table 5.3 (job shop with 2–10 ops. per job and 80% utilisation).

	[2]	[3]	[4]	[5]	[6]	[7]	[8]	IFT−UIT+NPT
[1] FCFS	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] LWKR		< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[3] SPT			0.5665	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[4] PT+WINQ				< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[5] PT+WINQ^x					< 0.0001	< 0.0001	< 0.0001	< 0.0001
[6] 2PT+WINQ+NPT						< 0.0001	< 0.0001	< 0.0001
[7] 2PT+WINQ^x+NPT							< 0.0001	< 0.0001
[8] IFT−UIT								< 0.0001

Table A.2: *p*-values of pairwise comparisons relating to Table 5.3 (job shop with 2–10 ops. per job and 95% utilisation).

	[2]	[3]	[4]	[5]	[6]	[7]	[8]	IFT−UIT+NPT
[1] FCFS	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] LWKR		< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[3] SPT			< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[4] PT+WINQ				0.0002	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[5] PT+WINQ^x					< 0.0001	< 0.0001	< 0.0001	< 0.0001
[6] 2PT+WINQ+NPT						< 0.0001	< 0.0001	< 0.0001
[7] 2PT+WINQ^x+NPT							0.3419	< 0.0001
[8] IFT−UIT								< 0.0001

Table A.3: p-values of pairwise comparisons relating to Table 5.3 (job shop with 10 ops. per job and 80% utilisation).

	[2]	[3]	[4]	[5]	[6]	[7]	[8]	IFT−UIT+NPT
[1] FCFS	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] LWKR		< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[3] SPT			0.1891	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[4] PT+WINQ				< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[5] PT+WINQ^x					< 0.0001	< 0.0001	< 0.0001	< 0.0001
[6] 2PT+WINQ+NPT						< 0.0001	< 0.0001	< 0.0001
[7] 2PT+WINQ^x+NPT							< 0.0001	< 0.0001
[8] IFT−UIT								< 0.0001

Table A.4: p-values of pairwise comparisons relating to Table 5.3 (job shop with 10 ops. per job and 95% utilisation).

	[2]	[3]	[4]	[5]	[6]	[7]	[8]	IFT−UIT+NPT
[1] FCFS	0.0008	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] LWKR		< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[3] SPT			0.1871	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[4] PT+WINQ				< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[5] PT+WINQ^x					< 0.0001	< 0.0001	< 0.0001	< 0.0001
[6] 2PT+WINQ+NPT						< 0.0001	< 0.0001	< 0.0001
[7] 2PT+WINQ^x+NPT							< 0.0001	< 0.0001
[8] IFT−UIT								< 0.0001

Table A.5: p-values of pairwise comparisons relating to Table 5.4 (flow-dominant job shop with 80% utilisation).

	[2]	[3]	[4]	[5]	[6]	[7]	[8]	IFT−UIT+NPT
[1] FCFS	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] LWKR		< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[3] SPT			< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[4] PT+WINQ				< 0.0001	0.0009	< 0.0001	0.0029	< 0.0001
[5] PT+WINQ^x					< 0.0001	0.0293	0.7122	< 0.0001
[6] 2PT+WINQ+NPT						< 0.0001	< 0.0001	< 0.0001
[7] 2PT+WINQ^x+NPT							0.1557	< 0.0001
[8] IFT−UIT								< 0.0001

Table A.6: *p*-values of pairwise comparisons relating to Table 5.4 (flow-dominant job shop with 95% utilisation).

	[2]	[3]	[4]	[5]	[6]	[7]	[8]	IFT−UIT+NPT
[1] FCFS	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] LWKR		< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[3] SPT			< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[4] PT+WINQ				< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[5] PT+WINQ^x					< 0.0001	< 0.0001	< 0.0001	< 0.0001
[6] 2PT+WINQ+NPT						< 0.0001	< 0.0001	< 0.0001
[7] 2PT+WINQ^x+NPT							0.0001	< 0.0001
[8] IFT−UIT								< 0.0001

Table A.7: *p*-values of pairwise comparisons relating to Table 5.4 (flow shop with 80% utilisation).

	[2]	[3]	[4]	[5]	[6]	[7]	[8]	IFT−UIT+NPT
[1] FCFS	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] LWKR		< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[3] SPT			1.0000	1.0000	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[4] PT+WINQ				1.0000	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[5] PT+WINQ^x					< 0.0001	< 0.0001	< 0.0001	< 0.0001
[6] 2PT+WINQ+NPT						1.0000	< 0.0001	< 0.0001
[7] 2PT+WINQ^x+NPT							< 0.0001	< 0.0001
[8] IFT−UIT								< 0.0001

Table A.8: *p*-values of pairwise comparisons relating to Table 5.4 (flow shop with 95% utilisation).

	[2]	[3]	[4]	[5]	[6]	[7]	[8]	IFT−UIT+NPT
[1] FCFS	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] LWKR		< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[3] SPT			1.0000	1.0000	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[4] PT+WINQ				1.0000	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[5] PT+WINQ^x					< 0.0001	< 0.0001	< 0.0001	< 0.0001
[6] 2PT+WINQ+NPT						1.0000	< 0.0001	< 0.0001
[7] 2PT+WINQ^x+NPT							< 0.0001	< 0.0001
[8] IFT−UIT								0.2670

p-Values relating to Chapter 6

Table A.9: p-values of pairwise comparisons relating to Table 6.3.

	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	GP Best(Fab4r, $\overline{C}$)
[1] FCFS	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] DJAH		< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[3] BSNSPT			0.0135	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[4] FSASPT-BSPT				< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[5] MCB-BLWKR					0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[6] MCB-SNSPT						< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[7] MCB-FSASPT-SPT							< 0.0001	< 0.0001	< 0.0001	< 0.0001
[8] MCB-2PT+WINQ+NPT								< 0.0001	< 0.0001	< 0.0001
[9] MCB-PT+WINQ									< 0.0001	< 0.0001
[10] GP Worst(Fab4r, $\overline{C}$)										< 0.0001

Table A.10: p-values of pairwise comparisons relating to Table 6.5.

	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	GP Best(Fab6, $\overline{C}$)
[1] MCB-BLWKR	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] FCFS		< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[3] MCB-2PT+WINQ+NPT			0.0180	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[4] DJAH				0.0003	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[5] MCB-PT+WINQ					0.3169	0.0040	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[6] MCB-SNSPT						0.0003	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[7] BSNSPT							< 0.0001	< 0.0001	< 0.0001	< 0.0001
[8] MCB-FSASPT-SPT								0.0082	< 0.0001	< 0.0001
[9] FSASPT-BSPT									< 0.0001	< 0.0001
[10] GP Worst(Fab6, $\overline{C}$)										< 0.0001

Table A.11: p-values of pairwise comparisons relating to Table 6.6 (mean weighted tardiness).

	[2]	[3]	[4]	[5]	[6]	[7]	[8]	GP Best(Fab4r, $\overline{wT}$)
[1] FCFS	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] MCB-LW-OCR		< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[3] BATCS^x(k_1 = 5.5, k_2 = 0.1)			< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[4] MCB-CR+SPT				< 0.0001	< 0.0001	< 0.0001	0.3936	< 0.0001
[5] MCB-S/RPT+SPT					0.2145	0.0232	0.0046	< 0.0001
[6] MCB-ATCS(k_1 = 4.0, k_2 = 0.01)						0.0250	< 0.0001	< 0.0001
[7] MCB-WMOD							< 0.0001	< 0.0001
[8] GP Worst(Fab4r, $\overline{wT}$)								< 0.0001

Table A.12: p-values of pairwise comparisons relating to Table 6.6 (mean tardiness).

	[2]	[3]	[4]	[5]	[6]	[7]	[8]	GP Best(Fab4r, $\overline{wT}$)
[1] FCFS	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] MCB-LW-OCR		< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[3] BATCS^x(k_1 = 5.5, k_2 = 0.1)			< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[4] MCB-CR+SPT				< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[5] MCB-S/RPT+SPT					< 0.0001	< 0.0001	< 0.0001	< 0.0001
[6] MCB-ATCS(k_1 = 4.0, k_2 = 0.01)						< 0.0001	< 0.0001	< 0.0001
[7] MCB-WMOD							< 0.0001	< 0.0001
[8] GP Worst(Fab4r, $\overline{wT}$)								< 0.0001

Table A.13: p-values of pairwise comparisons relating to Table 6.6 (proportion tardy).

	[2]	[3]	[4]	[5]	[6]	[7]	[8]	GP Best(Fab4r, $\overline{wT}$)
[1] FCFS	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] MCB-LW-OCR		< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[3] BATCS^x(k_1 = 5.5, k_2 = 0.1)			< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[4] MCB-CR+SPT				< 0.0001	< 0.0001	< 0.0001	0.0012	< 0.0001
[5] MCB-S/RPT+SPT					< 0.0001	< 0.0001	< 0.0001	0.0186
[6] MCB-ATCS(k_1 = 4.0, k_2 = 0.01)						< 0.0001	< 0.0001	< 0.0001
[7] MCB-WMOD							< 0.0001	0.2350
[8] GP Worst(Fab4r, $\overline{wT}$)								< 0.0001

Table A.14: p-values of pairwise comparisons relating to Table 6.6 (conditional mean tardiness).

	[2]	[3]	[4]	[5]	[6]	[7]	[8]	GP Best(Fab4r, $\overline{wT}$)
[1] FCFS	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] MCB-LW-OCR		< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[3] BATCS^x(k_1 = 5.5, k_2 = 0.1)			< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[4] MCB-CR+SPT				< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[5] MCB-S/RPT+SPT					< 0.0001	< 0.0001	< 0.0001	< 0.0001
[6] MCB-ATCS(k_1 = 4.0, k_2 = 0.01)						< 0.0001	< 0.0001	< 0.0001
[7] MCB-WMOD							< 0.0001	< 0.0001
[8] GP Worst(Fab4r, $\overline{wT}$)								< 0.0001

Table A.15: p-values of pairwise comparisons relating to Table 6.7 (mean weighted tardiness).

	[2]	[3]	[4]	[5]	[6]	[7]	[8]	GP Best(Fab6, $\overline{wT}$)
[1] FCFS	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] MCB-LW-OCR		< 0.0001	0.0014	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[3] MCB-S/RPT+SPT			0.4773	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[4] BATCS^x(k_1 = 5.5, k_2 = 0.1)				0.0143	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[5] MCB-CR+SPT					< 0.0001	< 0.0001	< 0.0001	< 0.0001
[6] MCB-ATCS(k_1 = 1.5, k_2 = 0.01)						< 0.0001	< 0.0001	< 0.0001
[7] MCB-WMOD							< 0.0001	< 0.0001
[8] GP Worst(Fab6, $\overline{wT}$)								< 0.0001

Table A.16: p-values of pairwise comparisons relating to Table 6.7 (mean tardiness).

	[2]	[3]	[4]	[5]	[6]	[7]	[8]	GP Best(Fab6, $\overline{wT}$)
[1] FCFS	< 0.0001	< 0.0001	0.1392	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] MCB-LW-OCR		< 0.0001	0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[3] MCB-S/RPT+SPT			< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[4] BATCS^x(k_1 = 5.5, k_2 = 0.1)				< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[5] MCB-CR+SPT					< 0.0001	< 0.0001	< 0.0001	< 0.0001
[6] MCB-ATCS(k_1 = 1.5, k_2 = 0.01)						< 0.0001	< 0.0001	0.0041
[7] MCB-WMOD							0.0243	0.7754
[8] GP Worst(Fab6, $\overline{wT}$)								0.0005

Table A.17: p-values of pairwise comparisons relating to Table 6.7 (proportion tardy).

	[2]	[3]	[4]	[5]	[6]	[7]	[8]	GP Best(Fab6, $\overline{wT}$)
[1] FCFS	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] MCB-LW-OCR		< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[3] MCB-S/RPT+SPT			< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[4] BATCS^x(k_1 = 5.5, k_2 = 0.1)				< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[5] MCB-CR+SPT					< 0.0001	< 0.0001	< 0.0001	< 0.0001
[6] MCB-ATCS(k_1 = 1.5, k_2 = 0.01)						< 0.0001	< 0.0001	< 0.0001
[7] MCB-WMOD							< 0.0001	< 0.0001
[8] GP Worst(Fab6, $\overline{wT}$)								< 0.0001

Table A.18: p-values of pairwise comparisons relating to Table 6.7 (conditional mean tardiness).

	[2]	[3]	[4]	[5]	[6]	[7]	[8]	GP Best(Fab6, $\overline{wT}$)
[1] FCFS	< 0.0001	< 0.0001	< 0.0001	< 0.0001	0.0002	< 0.0001	< 0.0001	< 0.0001
[2] MCB-LW-OCR		< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[3] MCB-S/RPT+SPT			< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[4] BATCS^x(k_1 = 5.5, k_2 = 0.1)				< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[5] MCB-CR+SPT					< 0.0001	< 0.0001	< 0.0001	< 0.0001
[6] MCB-ATCS(k_1 = 1.5, k_2 = 0.01)						< 0.0001	< 0.0001	< 0.0001
[7] MCB-WMOD							0.0002	0.0003
[8] GP Worst(Fab6, $\overline{wT}$)								0.0255

p-Values relating to Chapter 7

Table A.19: *p*-values of pairwise comparisons relating to Table 7.3.

	[2]	RS Best(Fab4r, $\overline{C}$)
[1] MCB-PT+WINQ	< 0.0001	< 0.0001
[2] RS Worst(Fab4r, $\overline{C}$)		< 0.0001

Table A.20: *p*-values of pairwise comparisons relating to Figure 7.4.

Replication	GP rule set vs. GP rule	GP rule set vs. Rule set
1	< 0.0001	< 0.0001
2	< 0.0001	< 0.0001
3	< 0.0001	< 0.0001
4	< 0.0001	0.3282
5	< 0.0001	0.0004
6	< 0.0001	< 0.0001
7	< 0.0001	< 0.0001
8	< 0.0001	0.0044
9	< 0.0001	0.6468
10	< 0.0001	0.0004
11	< 0.0001	< 0.0001
12	< 0.0001	< 0.0001
13	< 0.0001	0.0389
14	< 0.0001	0.1722
15	< 0.0001	< 0.0001
16	< 0.0001	< 0.0001
17	< 0.0001	0.0002
18	< 0.0001	< 0.0001
19	< 0.0001	< 0.0001
20	< 0.0001	0.4461

Table A.21: *p*-values of pairwise comparisons relating to Table 7.5.

	[2]	[3]	RS Best(Fab6, $\overline{C}$)
[1] MCB-FSASPT-SPT	0.0082	< 0.0001	< 0.0001
[2] FSASPT-BSPT		< 0.0001	< 0.0001
[3] RS Worst(Fab6, $\overline{C}$)			< 0.0001

Table A.22: p-values of pairwise comparisons relating to Figure 7.5.

Replication	GP rule set vs. GP rule	GP rule set vs. Rule set
1	< 0.0001	< 0.0001
2	< 0.0001	< 0.0001
3	< 0.0001	< 0.0001
4	< 0.0001	< 0.0001
5	< 0.0001	< 0.0001
6	< 0.0001	< 0.0001
7	< 0.0001	< 0.0001
8	< 0.0001	< 0.0001
9	< 0.0001	< 0.0001
10	< 0.0001	< 0.0001

Table A.23: p-values of pairwise comparisons relating to Table 7.6 (mean weighted tardiness).

	[2]	[3]	[4]	RS Best(Fab4r, $\overline{wT}$)
[1] MCB-S/RPT+SPT	0.2145	0.0232	< 0.0001	< 0.0001
[2] MCB-ATCS(k_1 = 4.0, k_2 = 0.01)		0.0250	< 0.0001	< 0.0001
[3] MCB-WMOD			< 0.0001	< 0.0001
[4] RS Worst(Fab4r, $\overline{wT}$)				< 0.0001

Table A.24: p-values of pairwise comparisons relating to Table 7.6 (mean tardiness).

	[2]	[3]	[4]	RS Best(Fab4r, $\overline{wT}$)
[1] MCB-S/RPT+SPT	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] MCB-ATCS(k_1 = 4.0, k_2 = 0.01)		< 0.0001	< 0.0001	< 0.0001
[3] MCB-WMOD			< 0.0001	< 0.0001
[4] RS Worst(Fab4r, $\overline{wT}$)				< 0.0001

Table A.25: p-values of pairwise comparisons relating to Table 7.6 (proportion tardy).

	[2]	[3]	[4]	RS Best(Fab4r, $\overline{wT}$)
[1] MCB-S/RPT+SPT	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] MCB-ATCS(k_1 = 4.0, k_2 = 0.01)		< 0.0001	< 0.0001	< 0.0001
[3] MCB-WMOD			< 0.0001	< 0.0001
[4] RS Worst(Fab4r, $\overline{wT}$)				< 0.0001

Table A.26: p-values of pairwise comparisons relating to Table 7.6 (conditional mean tardiness).

	[2]	[3]	[4]	RS Best(Fab4r, $\overline{wT}$)
[1] MCB-S/RPT+SPT	< 0.0001	< 0.0001	< 0.0001	< 0.0001
[2] MCB-ATCS(k_1 = 4.0, k_2 = 0.01)		< 0.0001	< 0.0001	< 0.0001
[3] MCB-WMOD			< 0.0001	0.0266
[4] RS Worst(Fab4r, $\overline{wT}$)				< 0.0001

Table A.27: p-values of pairwise comparisons relating to Figure 7.7.

Replication	GP rule set vs. GP rule	GP rule set vs. Rule set
1	< 0.0001	< 0.0001
2	< 0.0001	< 0.0001
3	< 0.0001	< 0.0001
4	< 0.0001	< 0.0001
5	< 0.0001	< 0.0001
6	< 0.0001	< 0.0001
7	< 0.0001	< 0.0001
8	< 0.0001	< 0.0001
9	< 0.0001	< 0.0001
10	< 0.0001	< 0.0001
11	< 0.0001	< 0.0001
12	< 0.0001	< 0.0001
13	< 0.0001	< 0.0001
14	< 0.0001	< 0.0001
15	< 0.0001	< 0.0001
16	< 0.0001	< 0.0001
17	< 0.0001	< 0.0001
18	< 0.0001	< 0.0001
19	< 0.0001	< 0.0001
20	< 0.0001	< 0.0001

Table A.28: p-values of pairwise comparisons relating to Table 7.7 (mean weighted tardiness).

	[2]	RS Best(Fab6, $\overline{wT}$)
[1] MCB-WMOD	< 0.0001	< 0.0001
[2] RS Worst(Fab6, $\overline{wT}$)		< 0.0001

Table A.29: p-values of pairwise comparisons relating to Table 7.7 (mean tardiness).

	[2]	RS Best(Fab6, $\overline{wT}$)
[1] MCB-WMOD	< 0.0001	0.0004
[2] RS Worst(Fab6, $\overline{wT}$)		< 0.0001

Table A.30: p-values of pairwise comparisons relating to Table 7.7 (proportion tardy).

	[2]	RS Best(Fab6, $\overline{wT}$)
[1] MCB-WMOD	< 0.0001	< 0.0001
[2] RS Worst(Fab6, $\overline{wT}$)		< 0.0001

Table A.31: p-values of pairwise comparisons relating to Table 7.7 (conditional mean tardiness).

	[2]	RS Best(Fab6, $\overline{wT}$)
[1] MCB-WMOD	< 0.0001	< 0.0001
[2] RS Worst(Fab6, $\overline{wT}$)		0.0347

Table A.32: *p*-values of pairwise comparisons relating to Figure 7.9.

Replication	GP rule set vs. GP rule	GP rule set vs. Rule set
1	< 0.0001	< 0.0001
2	< 0.0001	< 0.0001
3	< 0.0001	< 0.0001
4	< 0.0001	< 0.0001
5	< 0.0001	< 0.0001
6	< 0.0001	< 0.0001
7	< 0.0001	< 0.0001
8	< 0.0001	< 0.0001
9	< 0.0001	< 0.0001
10	< 0.0001	< 0.0001

References

Achterberg, T., T. Berthold, G. Gamrath, S. Heinz, T. Koch, M. Pfetsch, S. Vigerske, and K. Wolter (2003). SCIP: a MIP solver and constraint integer programming framework. http://scip.zib.de/ (last accessed 14 January 2013).

Adam, N. R. and J. Surkis (1980). Priority update intervals and anomalies in dynamic ratio type job shop scheduling rules. *Management Science* 26(12), 1227–1231.

Ahammed, F. and P. Moscato (2011). Evolving L-systems as an intelligent design approach to find classes of difficult-to-solve traveling salesman problem instances. In C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. I. Esparcia-Alcázar, J. J. Merelo, F. Neri, M. Preuss, H. Richter, J. Togelius, and G. N. Yannakakis (Eds.), *Applications of Evolutionary Computation*, pp. 1–11. Berlin and Heidelberg: Springer.

Akçalı, E., R. Uzsoy, D. G. Hiscock, A. L. Moser, and T. J. Teyner (2000). Alternative loading and dispatching policies for furnace operations in semiconductor manufacturing: a comparison by simulation. In J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick (Eds.), *Proceedings of the 2000 Winter Simulation Conference*, Volume 2, pp. 1428–1435. Piscataway, NJ: IEEE Press.

Anderson, E. J. and J. C. Nyirenda (1990). Two new rules to minimize tardiness in a job shop. *International Journal of Production Research* 28(12), 2277–2292.

Arzi, Y. and D. Raviv (1998). Dispatching in a workstation belonging to a re-entrant production line under sequence dependent set-up times. *Production Planning & Control* 9(7), 690–699.

Atlan, L., J. Bonnet, and M. Naillon (1994). Learning distributed reactive strategies by genetic programming for the general job shop problem. In D. D. Dankel, II and J. H. Stewman (Eds.), *Proceedings of the Seventh Florida Artificial Intelligence Research Symposium.*

Aytug, H., M. A. Lawley, K. McKay, S. Mohan, and R. Uzsoy (2005). Executing production schedules in the face of uncertainties: a review and some future directions. *European Journal of Operational Research* 161(1), 86–110.

Azadeh, A., A. Negahban, and M. Moghaddam (2012). A hybrid computer simulation-artificial neural network algorithm for optimisation of dispatching rule selection in stochastic job shop scheduling problems. *International Journal of Production Research* 50(2), 551–566.

Baek, D. H. and W. C. Yoon (2002). Co-evolutionary genetic algorithm for multi-machine scheduling: coping with high performance variability. *International Journal of Production Research* 40(1), 239–254.

Baek, D. H., W. C. Yoon, and S. C. Park (1998). A spatial rule adaptation procedure for reliable production control in a wafer fabrication system. *International Journal of Production Research* 36(6), 1475–1491.

Baker, K. R. (1984). Sequencing rules and due-date assignments in a job shop. *Management Science* 30(9), 1093–1104.

Baker, K. R. and J. W. M. Bertrand (1982). A dynamic priority rule for scheduling against due-dates. *Journal of Operations Management* 3(1), 37–42.

Baker, K. R. and J. J. Kanet (1983). Job shop scheduling with modified due dates. *Journal of Operations Management* 4(1), 11–22.

Barman, S. (1997). Simple priority rule combinations: an approach to improve both flow time and tardiness. *International Journal of Production Research* 35(10), 2857–2870.

Barman, S. (1998). The impact of priority rule combinations on lateness and tardiness. *IIE Transactions* 30(5), 495–504.

Barrett, R. T. and S. Barman (1986). A SLAM II simulation study of a simplified flow shop. *Simulation* 47(5), 181–189.

Barrett, R. T. and S. N. Kadipasaoglu (1990). Dispatching rules for a dynamic flow shop. *Production and Inventory Management Journal* 31(1), 54–58.

Blackstone, J. H., D. T. Phillips, and G. L. Hogg (1982). A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *International Journal of Production Research* 20(1), 27–45.

Bokhorst, J. A. C., G. Nomden, and J. Slomp (2008). Performance evaluation of family-based dispatching in small manufacturing cells. *International Journal of Production Research* 46(22), 6305–6321.

Brah, S. A. (1996). A comparative analysis of due date based job sequencing rules in a flow shop with multiple processors. *Production Planning & Control* 7(4), 362–373.

Brah, S. A. and G. E. Wheeler (1998). Comparison of scheduling rules in a flow shop with multiple processors: a simulation. *Simulation* 71(5), 302–311.

Branke, J. (2001). Reducing the sampling variance when searching for robust solutions. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke (Eds.), *GECCO 2001: Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 235–242. San Francisco: Morgan Kaufmann.

Burke, E. K., M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward (2010). A classification of hyper-heuristic approaches. In M. Gendreau and J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (2nd ed.), pp. 449–468. New York: Springer.

Carroll, D. C. (1965). Heuristic sequencing of jobs with single and multiple components. PhD thesis, MIT.

Caskey, K. and R. L. Storch (1996). Heterogeneous dispatching rules in job and flow shops. *Production Planning & Control* 7(4), 351–361.

Chiang, T. C. and L. C. Fu (2007). Using dispatching rules for job shop scheduling with due date-based objectives. *International Journal of Production Research* 45(14), 3245–3262.

Chiang, T.-C. and L.-C. Fu (2009). Using a family of critical ratio-based approaches to minimize the number of tardy jobs in the job shop with sequence dependent setup times. *European Journal of Operational Research* 196(1), 78–92.

Cigolini, R., A. Comi, A. Micheletti, M. Perona, and A. Portioli (1999). Implementing new dispatching rules at SGS-Thomson Microelectronics. *Production Planning & Control* 10(1), 97–106.

Cotta, C. and P. Moscato (2003). A mixed evolutionary-statistical analysis of an algorithm's complexity. *Applied Mathematics Letters* 16(1), 41–47.

Deb, R. K. and R. F. Serfozo (1973). Optimal control of batch service queues. *Advances in Applied Probability* 5(2), 340–361.

Dimopoulos, C. and A. M. S. Zalzala (2001). Investigating the use of genetic programming for a classic one-machine scheduling problem. *Advances in Engineering Software* 32(6), 489–498.

Dobson, G. and R. S. Nambimadom (2001). The batch loading and scheduling problem. *Operations Research* 49(1), 52–65.

Du, J. and J. Y.-T. Leung (1990). Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research* 15(3), 483–495.

Duenyas, I. and J. J. Neale (1997). Stochastic scheduling of a batch processing machine with incompatible job families. *Annals of Operations Research* 70, 191–220.

Eguchi, T., F. Oba, and S. Toyooka (2008). A robust scheduling rule using a neural network in dynamically changing job-shop environments. *International Journal of Manufacturing Technology and Management* 14(3–4), 266–288.

Eiben, A. E. and J. E. Smith (2003). *Introduction to Evolutionary Computing*. Berlin and Heidelberg: Springer.

El-Bouri, A. (2012). A cooperative dispatching approach for minimizing mean tardiness in a dynamic flowshop. *Computers & Operations Research* 39(7), 1305–1314.

El-Bouri, A., S. Balakrishnan, and N. Popplewell (2000). Sequencing jobs on a single machine: a neural network approach. *European Journal of Operational Research* 126(3), 474–490.

El-Bouri, A., S. Balakrishnan, and N. Popplewell (2008). Cooperative dispatching for minimizing mean flowtime in a dynamic flowshop. *International Journal of Production Economics* 113(2), 819–833.

Feigin, G., J. Fowler, and R. Leachman (1994). Modeling and Analysis for Semiconductor Manufacturing. http://masmlab.engineering.asu.edu/ftp.htm (last accessed 14 January 2013).

Fowler, J. W., G. L. Hogg, and D. T. Phillips (2000). Control of multiproduct bulk server diffusion/oxidation processes. Part 2: multiple servers. *IIE Transactions* 32(2), 167–176.

Fowler, J. W., D. T. Phillips, and G. L. Hogg (1992). Real-time control of multiproduct bulk-service semiconductor manufacturing processes. *IEEE Transactions on Semiconductor Manufacturing* 5(2), 158–163.

Frazier, G. V. (1996). An evaluation of group scheduling heuristics in a flow-line manufacturing cell. *International Journal of Production Research* 34(4), 959–976.

Fu, M. C., F. W. Glover, and J. April (2005). Simulation optimization: a review, new developments, and applications. In M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines (Eds.), *Proceedings of the 2005 Winter Simulation Conference*, pp. 83–95. Piscataway, NJ: IEEE Press.

Garey, M. R., D. S. Johnson, and R. Sethi (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research* 1(2), 117–129.

Gavett, J. W. (1965). Three heuristic rules for sequencing jobs to a single production facility. *Management Science* 11(8), B166–B176.

Geiger, C. D. and R. Uzsoy (2008). Learning effective dispatching rules for batch processor scheduling. *International Journal of Production Research* 46(6), 1431–1454.

Geiger, C. D., R. Uzsoy, and H. Aytuğ (2006). Rapid modeling and discovery of priority dispatching rules: an autonomous learning approach. *Journal of Scheduling* 9(1), 7–34.

Gere, W. S. (1966). Heuristics in job shop scheduling. *Management Science* 13(3), 167–190.

Glassey, C. R. and W. W. Weng (1991). Dynamic batching heuristic for simultaneous processing. *IEEE Transactions on Semiconductor Manufacturing* 4(2), 77–82.

Graham, R. L., E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* 5, 287–326.

Hart, E., P. Ross, and D. Corne (2005). Evolutionary scheduling: a review. *Genetic Programming and Evolvable Machines* 6(2), 191–220.

Haupt, R. (1989). A survey of priority rule-based scheduling. *OR Spektrum* 11(1), 3–16.

Hershauer, J. C. and R. J. Ebert (1975). Search and simulation selection of a job-shop sequencing rule. *Management Science* 21(7), 833–843.

Hildebrandt, T. (2012). Jasima — an efficient Java Simulator for Manufacturing and Logistics. http://code.google.com/p/jasima/ (last accessed 14 January 2013).

Holthaus, O. (1997). Design of efficient job shop scheduling rules. *Computers & Industrial Engineering* 33(1–2), 249–252.

Holthaus, O. and C. Rajendran (1997a). Efficient dispatching rules for scheduling in a job shop. *International Journal of Production Economics* 48(1), 87–105.

Holthaus, O. and C. Rajendran (1997b). New dispatching rules for scheduling in a job shop — an experimental study. *The International Journal of Advanced Manufacturing Technology* 13(2), 148–153.

Holthaus, O. and C. Rajendran (2000). Efficient jobshop dispatching rules: further developments. *Production Planning & Control* 11(2), 171–178.

Holthaus, O. and C. Rajendran (2002). A study on the performance of scheduling rules in buffer-constrained dynamic flowshops. *International Journal of Production Research* 40(13), 3041–3052.

Holthaus, O. and H. Ziegler (1997a). Improving job shop performance by coordinating dispatching rules. *International Journal of Production Research* 35(2), 539–549.

Holthaus, O. and H. Ziegler (1997b). Look ahead job demanding for improving job shop performance. *OR Spektrum* 19(1), 23–29.

Huffman, B. J. (2001). An object-oriented version of SIMLIB (a simple simulation package). *INFORMS Transactions on Education* 2(1), 1–15.

Hunsucker, J. L. and J. R. Shah (1994). Comparative performance analysis of priority rules in a constrained flow shop with multiple processors environment. *European Journal of Operational Research* 72(1), 102–114.

Ingimundardottir, H. and T. P. Runarsson (2011). Supervised learning linear priority dispatch rules for job-shop scheduling. In C. A. Coello Coello (Ed.), *Learning and Intelligent Optimization*, pp. 263–277. Berlin and Heidelberg: Springer.

Ishii, N. and J. J. Talavage (1994). A mixed dispatching rule approach in FMS scheduling. *The International Journal of Flexible Manufacturing Systems* 6(1), 69–87.

Jackson, J. R. (1955). Scheduling a production line to minimize maximum tardiness. Technical report, University of California, Los Angeles.

Jakobović, D. and L. Budin (2006). Dynamic scheduling with genetic programming. In P. Collet, M. Tomassini, M. Ebner, S. Gustafson, and A. Ekárt (Eds.), *Genetic Programming*, pp. 73–84. Berlin and Heidelberg: Springer.

Jakobović, D., L. Jelenković, and L. Budin (2007). Genetic programming heuristics for multiple machine scheduling. In M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, and A. I. Esparcia-Alcázar (Eds.), *Genetic Programming*, pp. 321–330. Berlin and Heidelberg: Springer.

Jakobović, D. and K. Marasović (2012). Evolving priority scheduling heuristics with genetic programming. *Applied Soft Computing* 12(9), 2781–2789.

Jensen, J. B., M. K. Malhotra, and P. R. Philipoom (1998). Family-based scheduling of shops with functional layouts. *International Journal of Production Research* 36(10), 2687–2700.

Jensen, J. B., P. R. Philipoom, and M. K. Malhotra (1995). Evaluation of scheduling rules with commensurate customer priorities in job shops. *Journal of Operations Management* 13(3), 213–228.

Julstrom, B. A. (2009). Evolving heuristically difficult instances of combinatorial problems. In F. Rothlauf (Ed.), *GECCO '09: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pp. 279–285. New York: ACM Press.

Kanet, J. J. (1982). On anomalies in dynamic ratio type scheduling rules: a clarifying analysis. *Management Science* 28(11), 1337–1341.

Kanet, J. J. and J. C. Hayya (1982). Priority dispatching with operation due dates in a job shop. *Journal of Operations Management* 2(3), 167–175.

Kanet, J. J. and X. Li (2004). A weighted modified due date rule for sequencing to minimize weighted tardiness. *Journal of Scheduling* 7(4), 261–276.

Kim, S. C. and P. M. Bobrowski (1994). Impact of sequence-dependent setup time on job shop scheduling performance. *International Journal of Production Research* 32(7), 1503–1520.

Kim, Y.-D., J.-G. Kim, B. Choi, and H.-U. Kim (2001). Production scheduling in a semiconductor wafer fabrication facility producing multiple product types with distinct due dates. *IEEE Transactions on Robotics and Automation* 17(5), 589–598.

Kiran, A. S. and M. L. Smith (1984). Simulation studies in job shop scheduling — II: performance of priority rules. *Computers & Industrial Engineering* 8(2), 95–105.

Kochhar, S. and R. J. T. Morris (1987). Heuristic methods for flexible flow line scheduling. *Journal of Manufacturing Systems* 6(4), 299–314.

Kosoresow, A. P. and M. P. Johnson (2002). Finding worst-case instances of, and lower bounds for, online algorithms using genetic algorithms. In B. McKay and J. Slaney (Eds.), *AI 2002: Advances in Artificial Intelligence*, pp. 344–355. Berlin and Heidelberg: Springer.

Kouvelis, P., R. L. Daniels, and G. Vairaktarakis (2000). Robust scheduling of a two-machine flow shop with uncertain processing times. *IIE Transactions* 32(5), 421–432.

Kutanoglu, E. and I. Sabuncuoglu (1999). An analysis of heuristics in a dynamic job shop with weighted tardiness objectives. *International Journal of Production Research* 37(1), 165–187.

LaForge, R. L. and S. Barman (1989). Performance of simple priority rule combinations in a flow-dominant shop. *Production and Inventory Management Journal* 30(3), 1–4.

Langdon, W. B. and R. Poli (2007). Evolving problems to learn about particle swarm optimizers and other search algorithms. *IEEE Transactions on Evolutionary Computation* 11(5), 561–578.

Law, A. M. (2007). *Simulation Modeling and Analysis* (4th ed.). Singapore: McGraw-Hill.

Lee, C.-Y., S. Piramuthu, and Y.-K. Tsai (1997). Job shop scheduling with a genetic algorithm and machine learning. *International Journal of Production Research* 35(4), 1171–1191.

Lee, G.-C., Y.-D. Kim, J.-G. Kim, and S.-H. Choi (2003). A dispatching rule-based approach to production scheduling in a printed circuit board manufacturing system. *Journal of the Operational Research Society* 54(10), 1038–1049.

Lee, Y. H., K. Bhaskaran, and M. Pinedo (1997). A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions* 29(1), 45–52.

Lee, Y. H. and M. Pinedo (1997). Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research* 100(3), 464–474.

Lenstra, J. K., A. H. G. Rinnooy Kan, and P. Brucker (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics* 1, 343–362.

Little, J. D. C. (1961). A proof for the queuing formula: $L = \lambda W$. *Operations Research* 9(3), 383–387.

Luke, S., L. Panait, G. Balan, S. Paus, Z. Skolicki, E. Popovici, K. Sullivan, J. Harrison, J. Bassett, R. Hubley, A. Chircop, J. Compton, W. Haddon, S. Donnelly, B. Jamil, J. Zelibor, E. Kangas, F. Abidi, H. Mooers, and J. O'Beirne (1998). ECJ: a Java-based Evolutionary Computation Research System. http://cs.gmu.edu/~eclab/projects/ecj/ (last accessed 14 January 2013).

Mahmoodi, F. and K. J. Dooley (1991). A comparison of exhaustive and non-exhaustive group scheduling heuristics in a manufacturing cell. *International Journal of Production Research* 29(9), 1923–1939.

Mahmoodi, F., K. J. Dooley, and P. J. Starr (1990). An investigation of dynamic group scheduling heuristics in a job shop manufacturing cell. *International Journal of Production Research* 28(9), 1695–1711.

Mahmoodi, F., C. T. Mosier, and R. E. Guerin (1996). The effect of combining simple priority heuristics in flow-dominant shops. *International Journal of Production Research* 34(3), 819–839.

Mahmoodi, F., E. J. Tierney, and C. T. Mosier (1992). Dynamic group scheduling heuristics in a flow-through cell environment. *Decision Sciences* 23(1), 61–85.

Mason, S. J., J. W. Fowler, and W. M. Carlyle (2002). A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops. *Journal of Scheduling* 5(3), 247–262.

Mason, S. J., J. W. Fowler, W. M. Carlyle, and D. C. Montgomery (2005). Heuristics for minimizing total weighted tardiness in complex job shops. *International Journal of Production Research* 43(10), 1943–1963.

Mehta, S. V. and R. Uzsoy (1998). Minimizing total tardiness on a batch processing machine with incompatible job families. *IIE Transactions* 30(2), 165–178.

Miyashita, K. (2000). Job-shop scheduling with genetic programming. In D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer (Eds.), *GECCO 2000: Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 505–512. San Francisco: Morgan Kaufmann.

Mönch, L. and I. Habenicht (2003). Simulation-based assessment of batching heuristics in semiconductor manufacturing. In S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice (Eds.), *Proceedings of the 2003 Winter Simulation Conference*, Volume 2, pp. 1338–1345. Piscataway, NJ: IEEE Press.

Monma, C. L. and C. N. Potts (1989). On the complexity of scheduling with batch setup times. *Operations Research* 37(5), 798–804.

Morton, T. E. and D. W. Pentico (1993). *Heuristic Scheduling Systems: With Applications to Production Systems and Project Management.* Hoboken, NJ: John Wiley & Sons.

Morton, T. E. and R. M. V. Rachamadugu (1982). Myopic heuristics for the single machine weighted tardiness problem. Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University.

Moser, M. and S. Engell (1992a). Comprehensive evaluation of priority rules for on-line scheduling: the single machine case. In L. Gerhardt, M. Silva, and A. Desrochers (Eds.), *Proceedings of the Third International Conference on Computer Integrated Manufacturing*, pp. 403–412. Los Alamitos, CA: IEEE Computer Society Press.

Moser, M. and S. Engell (1992b). A survey of priority rules for FMS scheduling and their performance for the benchmark problem. In T. Basar and S. Verdu (Eds.), *Proceedings of the 31st IEEE Conference on Decision and Control*, Volume 1, pp. 392–397. Piscataway, NJ: IEEE Press.

Mosier, C. T., D. A. Elvers, and D. Kelly (1984). Analysis of group technology scheduling heuristics. *International Journal of Production Research* 22(5), 857–875.

Neuts, M. F. (1967). A general class of bulk queues with Poisson input. *The Annals of Mathematical Statistics* 38(3), 759–770.

Nguyen, S., M. Zhang, M. Johnston, and K. C. Tan (2012). A coevolution genetic programming method to evolve scheduling policies for dynamic multi-objective job shop scheduling problems. In H. Abbass, D. Essam, and R. Sarker (Eds.), *2012 IEEE Congress on Evolutionary Computation (CEC)*, pp. 3261–3268. Piscataway, NJ: IEEE Press.

Nie, L., X. Shao, L. Gao, and W. Li (2010). Evolving scheduling rules with gene expression programming for dynamic single-machine scheduling problems. *The International Journal of Advanced Manufacturing Technology* 50(5–8), 729–747.

Nomden, G., D.-J. Van der Zee, and J. Slomp (2008). Family-based dispatching: anticipating future jobs. *International Journal of Production Research* 46(1), 73–97.

Olafsson, S. and X. Li (2010). Learning effective new single machine dispatching rules from optimal scheduling data. *International Journal of Production Economics* 128(1), 118–126.

Oltean, M. (2004). Searching for a practical evidence of the no free lunch theorems. In A. J. Ijspeert, M. Murata, and N. Wakamiya (Eds.), *Biologically Inspired Approaches to Advanced Information Technology*, pp. 472–483. Berlin and Heidelberg: Springer.

Ouelhadj, D. and S. Petrovic (2009). A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling* 12(4), 417–431.

Ovacik, I. M. and R. Uzsoy (1994). Rolling horizon algorithms for a single-machine dynamic scheduling problem with sequence-dependent setup times. *International Journal of Production Research* 32(6), 1243–1263.

Pan, J. C.-H. and J.-S. Chen (2005). Mixed binary integer programming formulations for the reentrant job shop scheduling problem. *Computers & Operations Research* 32(5), 1197–1212.

Panwalkar, S. S. and W. Iskander (1977). A survey of scheduling rules. *Operations Research* 25(1), 45–61.

Perez, I. C., J. W. Fowler, and W. M. Carlyle (2005). Minimizing total weighted tardiness on a single batch process machine with incompatible job families. *Computers & Operations Research* 32(2), 327–341.

Pfund, M., J. W. Fowler, A. Gadkari, and Y. Chen (2008). Scheduling jobs on parallel machines with setup times and ready times. *Computers & Industrial Engineering* 54(4), 764–782.

Pfund, M. E., S. J. Mason, and J. W. Fowler (2006). Semiconductor manufacturing scheduling and dispatching: state of the art and survey of needs. In J. W. Herrmann (Ed.), *Handbook of Production Scheduling*, pp. 213–241. New York: Springer.

Philipoom, P. R. and T. D. Fry (1990). The robustness of selected job-shop dispatching rules with respect to load balance and work-flow structure. *Journal of the Operational Research Society* 41(10), 897–906.

Pinedo, M. L. (2008). *Scheduling: Theory, Algorithms, and Systems* (3rd ed.). New York: Springer.

Poli, R., W. B. Langdon, and N. F. McPhee (2008). *A Field Guide to Genetic Programming*. http://www.gp-field-guide.org.uk/ (last accessed 14 January 2013).

Potts, C. N. and M. Y. Kovalyov (2000). Scheduling with batching: a review. *European Journal of Operational Research* 120(2), 228–249.

Rachamadugu, R. M. V. and T. E. Morton (1983). Myopic heuristics for the weighted tardiness problem on identical parallel machines. Technical report, The Robotics Institute, Carnegie-Mellon University.

Raghu, T. S. and C. Rajendran (1993). An efficient dynamic dispatching rule for scheduling in a job shop. *International Journal of Production Economics* 32(3), 301–313.

Rajendran, C. and K. Alicke (2007). Dispatching in flowshops with bottleneck machines. *Computers & Industrial Engineering* 52(1), 89–106.

Rajendran, C. and O. Holthaus (1999). A comparative study of dispatching rules in dynamic flowshops and jobshops. *European Journal of Operational Research* 116(1), 156–170.

Raman, N., R. V. Rachamadugu, and F. B. Talbot (1989). Real-time scheduling of an automated manufacturing center. *European Journal of Operational Research* 40(2), 222–242.

Raman, N., F. B. Talbot, and R. V. Rachamadugu (1989). Due date based scheduling in a general flexible manufacturing system. *Journal of Operations Management* 8(2), 115–132.

Robinson, J. K., J. W. Fowler, and J. F. Bard (1995). The use of upstream and downstream information in scheduling semiconductor batch operations. *International Journal of Production Research* 33(7), 1849–1869.

Ruben, R. A. and F. Mahmoodi (1998). Lot splitting in unbalanced production systems. *Decision Sciences* 29(4), 921–949.

Ruben, R. A., C. T. Mosier, and F. Mahmoodi (1993). A comprehensive analysis of group scheduling heuristics in a job shop cell. *International Journal of Production Research* 31(6), 1343–1369.

Russell, G. R. and P. R. Philipoom (1991). Sequencing rules and due date setting procedures in flow line cells with family setups. *Journal of Operations Management* 10(4), 524–545.

Russell, R. S., E. M. Dar-El, and B. W. Taylor, III (1987). A comparative analysis of the COVERT job sequencing rule using various shop performance measures. *International Journal of Production Research* 25(10), 1523–1540.

Sarper, H. and M. C. Henry (1996). Combinatorial evaluation of six dispatching rules in a dynamic two-machine flow shop. *Omega: The International Journal of Management Science* 24(1), 73–81.

Sheskin, D. J. (2007). *Handbook of Parametric and Nonparametric Statistical Procedures* (4th ed.). Boca Raton, FL: Chapman & Hall/CRC.

Shirakawa, S., N. Yata, and T. Nagao (2010). Evolving search spaces to emphasize the performance difference of real-coded crossovers using genetic programming. In P. Sobrevilla, J. Aranda, and S. Xambo (Eds.), *2010 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1657–1664. Piscataway, NJ: IEEE Press.

Smith, W. E. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly* 3(1–2), 59–66.

Smith-Miles, K. and J. Van Hemert (2011). Discovering the suitability of optimisation algorithms by learning from evolved instances. *Annals of Mathematics and Artificial Intelligence* 61(2), 87–104.

Tay, J. C. and N. B. Ho (2007). Designing dispatching rules to minimize total tardiness. In K. P. Dahal, K. C. Tan, and P. I. Cowling (Eds.), *Evolutionary Scheduling*, pp. 101–124. Berlin and Heidelberg: Springer.

Tay, J. C. and N. B. Ho (2008). Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering* 54(3), 453–473.

Toptal, A. and I. Sabuncuoglu (2010). Distributed scheduling: a review of concepts and applications. *International Journal of Production Research* 48(18), 5235–5262.

Uzsoy, R. (1994). Scheduling a single batch processing machine with non-identical job sizes. *International Journal of Production Research* 32(7), 1615–1635.

Uzsoy, R. (1995). Scheduling batch processing machines with incompatible job families. *International Journal of Production Research* 33(10), 2685–2708.

Uzsoy, R., C.-Y. Lee, and L. A. Martin-Vega (1992). A review of production planning and scheduling models in the semiconductor industry part I: system characteristics, performance evaluation and production planning. *IIE Transactions* 24(4), 47–60.

Van der Zee, D.-J. (2003). Look-ahead strategies for controlling batch operations in industry — an overview. In S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice (Eds.), *Proceedings of the 2003 Winter Simulation Conference*, Volume 2, pp. 1480–1487. Piscataway, NJ: IEEE Press.

Van der Zee, D.-J. (2010). Non-exhaustive family based dispatching heuristics — exploiting variances of processing and set-up times. *International Journal of Production Research* 48(13), 3783–3802.

Van der Zee, D. J., A. Van Harten, and P. C. Schuur (1997). Dynamic job assignment heuristics for multi-server batch operations — a cost based approach. *International Journal of Production Research* 35(11), 3063–3093.

Van Hemert, J. I. (2006). Evolving combinatorial problem instances that are difficult to solve. *Evolutionary Computation* 14(4), 433–462.

Vázquez-Rodríguez, J. A. and G. Ochoa (2011). On the automatic discovery of variants of the NEH procedure for flow shop scheduling using genetic programming. *Journal of the Operational Research Society* 62(2), 381–396.

Vepsalainen, A. P. J. and T. E. Morton (1987). Priority rules for job shops with weighted tardiness costs. *Management Science* 33(8), 1035–1047.

Vinod, V. and R. Sridharan (2008). Scheduling a dynamic job shop production system with sequence-dependent setups: an experimental study. *Robotics and Computer-Integrated Manufacturing* 24(3), 435–449.

Webster, S. and K. R. Baker (1995). Scheduling groups of jobs on a single machine. *Operations Research* 43(4), 692–703.

Wemmerlöv, U. and A. J. Vakharia (1991). Job and family scheduling of a flow-line manufacturing cell: a simulation study. *IIE Transactions* 23(4), 383–393.

Weng, W. W. and R. C. Leachman (1993). An improved methodology for real-time production decisions at batch-process work stations. *IEEE Transactions on Semiconductor Manufacturing* 6(3), 219–225.

Wilbrecht, J. K. and W. B. Prescott (1969). The influence of setup time on job shop performance. *Management Science* 16(4), B274–B280.

Yamamoto, M. and S. Y. Nof (1985). Scheduling/rescheduling in the manufacturing operating system environment. *International Journal of Production Research* 23(4), 705–722.

Yang, T., Y. Kuo, and C. Cho (2007). A genetic algorithms simulation approach for the multi-attribute combinatorial dispatching decision problem. *European Journal of Operational Research* 176(3), 1859–1873.

Yin, W.-J., M. Liu, and C. Wu (2003). Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming. In R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon (Eds.), *The 2003 Congress on Evolutionary Computation (CEC 2003)*, Volume 2, pp. 1050–1055. Piscataway, NJ: IEEE Press.

List of Dispatching Rules

Abbreviation	Full name	Description on page(s)
2PT+WINQ+NPT	(variant of the PT+WINQ rule)	45
ATC	Apparent Tardiness Cost	22, 51
ATCS	Apparent Tardiness Cost with Setups	29
ATCSR	Apparent Tardiness Cost with Setups and Ready Times	54
BATC	Batch Apparent Tardiness Cost	34
BATC^x	(variant of the BATC rule)	34
BATCR	Batch Apparent Tardiness Cost with Ready Times	55
BATCS	Batch Apparent Tardiness Cost with Setups	34
BEDD	Batch Earliest Due Date	33
BLWKR	Bottleneck-based Least Work Remaining	46
BLWKR^x	(variant of the BLWKR rule)	46–47
BSPT	Batch Shortest Processing Time	31
BWSPT	Batch Weighted Shortest Processing Time	33
CD	Cooperative Dispatching	47–48
COVERT	Cost Over Time	22, 49
CR	Critical Ratio	20, 49
CR+SPT	(combination of the CR and SPT rules)	53
DBH	Dynamic Batching Heuristic	39
DD/BJ		56–57
DJAH	Dynamic Job Assignment Heuristic	43
ECR	Enhanced Critical Ratio	29–30
EDD	Earliest Due Date	18–19
FB	Full Batch	32
FSASPT	Family Shortest Average Setup and Processing Time	26–27
FSASPT_AD	Adaptive Family Shortest Average Setup and Processing Time	27–28
FSASPT_GA	Gated Family Shortest Average Setup and Processing Time	27
FSASPT_HY	Hybrid Family Shortest Average Setup and Processing Time	27–28
FSAST	Family Shortest Average Setup Time	26
LWKR	Least Work Remaining	37
MCB	Most Complete Batch	33
MCR	Minimum Cost Rate	41–42
MBS	Minimum Batch Size	32
MBSBSPT	(combination of the MBS and BSPT rules)	32
MDD	Modified Due Date	19

Abbreviation	Full name	Description on page(s)
MOD	Modified Operation Due Date	50
MS	Minimum Slack	21
NACH	Next Arrival Control Heuristic	40–41
OCOVERT	Operation Cost Over Time	50
OCR	Operation Critical Ratio	50
ODD	Earliest Operation Due Date	57
PT+WINQ	Processing Time plus Work In Next Queue	45
PUCH	Processing Urgency Classification Heuristic	35–36
RR	Raghu and Rajendran	56
SASPT	Shortest Average Setup and Processing Time	25–26
SNSPT	Shortest Normalised Setup and Processing Time	25
SPT	Shortest Processing Time	18, 37
SSPT	Shortest Setup and Processing Time	25
SST	Shortest Setup Time	24
S/RPT	Slack per Remaining Processing Time	21
S/RPT+SPT	(combination of the S/RPT and SPT rules)	53
(TPT+DD)/BJ		56–57
WMDD	Weighted Modified Due Date	20, 21, 49
WMOD	Weighted Modified Operation Due Date	50
WSPT	Weighted Shortest Processing Time	18
X-ATC	(variant of the ATC rule)	54

List of Symbols

B_f	Maximum batch size of family f
C_j	Completion time of job j
c	Number of work centres
d_j (d_b)	Due date of job j (of batch b)
$d_{j,o}$ ($d_{j,i}$)	Due date of operation o (of imminent operation) of job j
l_j	Number of operations of job j
m	Number of machines
N_b	Set of jobs in batch b
N^{L} (N_f^{L})	Set of look-ahead jobs (of family f)
N^{Q} (N_f^{Q})	Set of queueing jobs (of family f)
n	Total number of jobs
n_b	Number of jobs in batch b
n^{Q} (n_f^{Q})	Number of queueing jobs (of family f)
p_j (p_b)	Processing time of job j (of batch b)
$p_{j,o}$ ($p_{j,i}$)	Processing time of operation o (of imminent operation) of job j
$\overline{p}$	Average operation processing time of queueing jobs
$\hat{q}_j$	Estimated queueing time of job j
$\hat{q}_{j,o}$ ($\hat{q}_{j,i}$)	Estimated queueing time of operation o (of imminent operation) of job j
r_j (r_b)	Release date of job j (of batch b)
$r_{j,i}$	Arrival time of job j at the work centre required for its imminent operation
S_j	Slack of job j
s_{fg} (s_{ef})	Setup time for switching from family f to family g (e represents current setup)
$\overline{s}$	Average setup time for queueing jobs given the current machine setup
T_j	Tardiness of job j
t	Current time
w_j (w_b)	Weight of job j (of batch b)