Int. J. Production Economics 145 (2013) 67–77

Contents lists available at SciVerse ScienceDirect

Int. J. Production Economics

journal homepage: www.elsevier.com/locate/ijpe

Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems

Christoph W. Pickardt [a], Torsten Hildebrandt [b], Jürgen Branke [a,*], Jens Heger [b], Bernd Scholz-Reiter [b]

[a] Warwick Business School, The University of Warwick, Coventry CV4 7AL, UK
[b] BIBA—Bremen Institute of Production and Logistics, Hochschulring 20, 28359 Bremen, Germany

ARTICLE INFO

Article history:
Received 17 February 2012
Accepted 20 October 2012
Available online 30 October 2012

Keywords:
Hyper-heuristics
Dispatching rules
Production scheduling
Semiconductor manufacturing
Evolutionary algorithms
Genetic programming

ABSTRACT

We propose a two-stage hyper-heuristic for the generation of a set of work centre-specific dispatching rules. The approach combines a genetic programming (GP) algorithm that evolves a composite rule from basic job attributes with an evolutionary algorithm (EA) that searches for a good assignment of rules to work centres. The hyper-heuristic is tested against its two components and rules from the literature on a complex dynamic job shop problem from semiconductor manufacturing. Results show that all three hyper-heuristics are able to generate (sets of) rules that achieve a significantly lower mean weighted tardiness than any of the benckmark rules. Moreover, the two-stage approach proves to outperform the GP and EA hyper-heuristic as it optimises on two different heuristic search spaces that appear to tap different optimisation potentials. The resulting rule sets are also robust to most changes in the operating conditions.

1. Introduction

Production scheduling is concerned with the allocation of resources, e.g. machines, to the processing of a number of jobs. The task is to determine a schedule that optimises a given performance criterion such as the makespan. One of the most complex scheduling problems is the job shop problem, in which each job consists of a number of operations that need to be performed on distinct work centres in a prescribed order, where the order in which a job visits the work centres is job-specific. In this work, a work centre is defined as a set of (1 to m) identical machines with the same functionality.

A widely used approach to real-world scheduling, where problems are often characterised by a highly complex and dynamic environment, are dispatching rules. Dispatching rules are simple heuristics that, whenever a machine is available, determine the job with the highest priority of the jobs waiting to be processed next on that machine. The computation of priorities is typically based on local information, which allows dispatching rules to be executed quickly, irrespective of the complexity of the overall problem. Moreover, because each scheduling decision is made at the latest possible moment, i.e. immediately before its implementation, dispatching rules naturally possess the ability to react to dynamic changes. Other advantages of dispatching rules include their simple and intuitive nature, their ease of implementation within practical settings, and their flexibility to incorporate domain knowledge and expertise (Aytug et al., 2005; Geiger et al., 2006).

On the other hand, the lack of a global perspective on the problem of dispatching rules is also their biggest drawback. They take scheduling decisions on the basis of current local conditions without assessing the negative impact a decision might have on the decision-making at other work centres in the future. The limited horizon of dispatching rules also explains the absence of a single rule that outperforms all others across different shop configurations, operating conditions and objective functions (Blackstone et al., 1982; Haupt, 1989; Holthaus and Rajendran, 1997; Rajendran and Holthaus, 1999). The decision which rule to select generally depends on the specific problem at hand. In addition, some researchers have shown that it can be beneficial to select different rules at different work centres within a shop. This appears to be particularly true for problems where work centres vary with respect to their relative position in the system (LaForge and Barman, 1989; Mahmoodi et al., 1996; Barman, 1997) or their utilisation (Raman et al., 1989; Ruben and Mahmoodi, 1998; Bokhorst et al., 2008), or possess different characteristics altogether (Cigolini et al., 1999; Lee et al., 2003). In summary, the employed rules typically have to be customised to the problem in order to tap the full potential of a dispatching rule-based approach.

The development of customised dispatching rules is usually a tedious procedure requiring a significant amount of expertise, coding-effort and time. The challenge is to design local, decentralised rules which result in a good global performance of a complex production environment. Generally, this is achieved by a trial-and-error procedure, with candidate rules tested in a simulation model of the considered manufacturing system, modified,

* Corresponding author.
E-mail addresses: christoph.pickardt@warwick.ac.uk (C.W. Pickardt), hil@biba.uni-bremen.de (T. Hildebrandt), juergen.branke@wbs.ac.uk (J. Branke), heg@biba.uni-bremen.de (J. Heger), bsr@biba.uni-bremen.de (B. Scholz-Reiter).

0925-5273/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ijpe.2012.10.016

and retested until they fulfill the requirements for actual implementation (Geiger et al., 2006). This process can be automated by a hyper-heuristic. Hyper-heuristics are optimisation methods that operate on a search space of heuristics (Burke et al., 2010). In this work, evolutionary algorithms (EAs) are employed as hyper-heuristics to search for effective dispatching rules, and discrete-event simulation is used to evaluate the evolved rules.

In a previous paper (Pickardt et al., 2010), we apply a hyper-heuristic that is based on a special type of EA called genetic programming (GP) to create a single dispatching rule for a complex and dynamic job shop from semiconductor manufacturing. Here, we extend the method with another EA that, in a second stage, assigns a different dispatching rule to each work centre in the shop. This two-stage hyper-heuristic is tested by comparing its performance to that of the original GP hyper-heuristic and the standard rule-assignment hyper-heuristic without access to evolved rules.

The paper is organised as follows. Section 2 reviews the related literature, followed by a presentation of the three hyper-heuristics in Section 3. These are applied to a scenario from semiconductor manufacturing, described in Section 4 and results are reported in Section 5. Some investigations of the robustness of the generated dispatching rule sets are done in Section 6, and the paper concludes with a summary and some suggestions for future work.

2. Literature review

An early paper related to the generation of composite dispatching rules whose priority indices are mathematical functions of several job attributes is the one by Hershauer and Ebert (1975). They define composite rules as the weighted sum or product of common priority indices, and use Hooke–Jeeves pattern search to find the best weights for a job shop problem. They find that the effectiveness of their method strongly depends on aspects such as the format of the composite rules and the starting solution of the search. In face of this, GP seems very suitable as it is flexible to create rules of different formats and lengths, and like any EA operates on a set of (starting) solutions. Atlan et al. (1994) employ GP to compose a dispatching rule for a particular job shop instance. The resulting rules obtain (near-)optimal solutions and are robust with respect to perturbations in processing times.

Several recent studies have used GP as a hyper-heuristic to learn a new composite rule that outperforms manually developed benchmark rules on a class of problems. Dimopoulos and Zalzala (2001) and Jakobović and Budin (2006) address various single machine problems with due date-related measures, and report that the rules evolved by GP in most cases outperform the rules from the literature. Geiger et al. (2006) apply a GP hyper-heuristic to a range of single machine problems, which finds optimal dispatching rules where they are known and yields competitive rules for all other problems. In a follow-up paper, Geiger and Uzsoy (2008) use their hyper-heuristic to evolve dispatching rules for a batch processing machine that can process several jobs together in a batch, and again manage to generate good rules that are optimal in some cases. More complicated problem environments are considered by Jakobović et al. (2007), who use GP to create rules for several parallel machine problems with and without sequence-dependant setup times and Tay and Ho (2008), who address a flexible job shop problem, where each work centre contains several machines in parallel. Both studies report the approach to use a GP hyper-heuristic for the generation of composite rules to be successful. Interestingly, the dominance of automatically generated rules seems to become more pronounced for more complex problems (Jakobović et al., 2007), supporting the intuition that the potential of hyper-heuristics is highest when it is difficult to design effective rules manually. Some researchers have proposed alternative methods for the automatic creation of dispatching rules. Olafsson and Li (2010) combine an EA with a decision tree algorithm to learn new rules. Nie et al. (2010) propagate the use of gene expression programming, which is based on similar ideas as GP, for this purpose. Both papers address only single machine problems.

GP-based rule generation has also been successfully used to improve scheduling algorithms of which dispatching rules form an integral part. Yin et al. (2003) apply this technique to evolve predictive scheduling heuristics which produce schedules that are robust to unpredictable breakdowns of machines. In a similar fashion, Vázquez-Rodríguez and Ochoa (2011) create effective variants of the NEH heuristic for different permutation flow shop problems by modifying the dispatching rule underlying the heuristic.

The above publications indicate that GP hyper-heuristics for the generation of dispatching rules is a promising approach. However, they predominantly investigate relatively simple and static problems and allow rules to access future information that is typically unavailable, such as the number of jobs still to arrive or their release dates. Since dispatching rules are more likely to be employed where they are most beneficial, namely in complex, dynamically changing environments, we have previously applied the approach to such problem environments with promising results (Hildebrandt et al., 2010; Pickardt et al., 2010).

The optimisation problem of assigning each work centre the best of a set of given rules so that the rule combination results in a good shop performance has been addressed in various ways. Pierreval and Ralambondrainy (1990), Pierreval (1992) and El-Bouri and Shah (2006) use machine learning techniques, in particular neural networks, that base the choice of an appropriate rule on properties such as the workload distribution or relative position of a work centre. Ishii and Talavage (1994) design a heuristic search algorithm that, starting from a base rule, sequentially selects the best of a given set of dispatching rules for each individual work centre, where work centres are considered roughly in decreasing order of their utilisation. They apply the algorithm to various job shop problems and find combinations of rules that dominate the individual rules. A drawback of this procedure is that it is not always easy to identify the extent to which work centres are critical, especially in the presence of sequence-dependant setups and batch processing. Yang et al. (2007) develop an EA hyper-heuristic that selects different rules for different work centres. They apply the hyper-heuristic to the problem of minimising mean tardiness in a flexible flow shop, for which it discovers rule combinations that clearly outperform some benchmark rules.

A natural extension to the studies above is the design of hyper-heuristics that exploit the two potentials of composite rules and sets of work centre-specific rules at the same time. Baek et al. (1998) propose a procedure that sequentially generates a composite rule for each work centre one-by-one. They apply it to two flexible flow shops, for which it is able to learn rules that yield a significantly lower mean flow time than all benchmark rules. In a follow up paper, Baek and Yoon (2002) develop a coevolutionary algorithm (CoEA) that evolves the work-centre specific composite rules in parallel. Applied to a flexible job shop problem with the mean tardiness objective, they find the CoEA hyper-heuristic to be more effective and efficient than the sequential procedure suggested in their earlier paper. Geiger et al. (2006) use GP to generate one composite rule for each machine to address the two machine-flow shop makespan problem. They report that the evolved rule set resembles the behaviour of the optimal Johnson algorithm.

The main problem with the development of hyper-heuristics for the automatic generation of entire sets of work centre-specific composite rules is the size of the search space, which grows exponentially both in the number of components that define a

rule and the number of rules to be created. Consequently, Geiger et al. (2006) consider a shop with only two work centres, while Baek et al. (1998) and Baek and Yoon (2002) keep the format of composite rules very simple by defining them as weighted sums of common priority indices. For larger shops, an alternative could be to classify work centres and generate only one rule for each group. To this effect, Miyashita (2000) compares three different GP hyper-heuristics—the first evolves a single rule for all work centres as usual, the second generates a different rule for every work centre, while the third uses a predetermined classification of work centres into bottlenecks and non-bottlenecks to evolve one rule for each of the two categories. Applied to a job shop with five work centres and one or two bottlenecks, the classification approach is shown to produce the best results, overall. Jakobović and Budin (2006) present a GP hyper-heuristic that optimises the classification of work centres while searching for good dispatching rules. In their three-tree encoding, two of the trees correspond to a composite dispatching rule, and the third tree is a decision tree that uses attributes related to the workload of a work centre to decide which of the two composite rules to apply. They find this hyper-heuristic to be more effective than a GP that evolves only one rule for all work centres for several job shop problems. The drawback of these hyper-heuristics is that the grouping of work centres in terms of which dispatching rule to apply becomes very difficult when work centres differ not only with respect to their level of utilisation, but also their setup requirements, their batch capacities, their position within the system, etc., which will affect the effectiveness of such an approach.

We design a two-stage hyper-heuristic that combines a GP to generate a composite rule with an EA to select the best possible combination of common and evolved rules. The examined problem is taken from semiconductor manufacturing, which possesses various complex properties as described in Section 4. We have previously applied a standard GP hyper-heuristic to the same problem, being the first to test the approach on a complex dynamic job shop (Pickardt et al., 2010). Here, we extend our work by appending the second optimisation phase that forms the two-stage hyper-heuristic, which we compare to its two components, the GP and an EA hyper-heuristic similar to the one proposed by Yang et al. (2007).

3. Algorithm design

The implemented hyper-heuristics are all based on EAs, which are iterative, stochastic search procedures inspired by natural evolution. They maintain a set (population) of candidate solutions (individuals), and, in each iteration (generation), select good solutions from the population (survival of the fittest), and generate new solutions (children) by recombining (crossing over) two old solutions (parents) and/or randomly modifying (mutating) a solution. Over time, this process 'evolves' better and better solutions, just like in nature, where the individuals become better and better adapted to their environment. Fig. 1 depicts the general procedure of EAs. The interested reader is referred to Eiben and Smith (2003) for a detailed introduction. The following sections detail the specific components of the EAs developed in this work. The actual implementation is based on the Java library ECJ (Luke et al., 1998).

3.1. GP-based generation of composite rules

GP is a special kind of EA that is characterised by its ability to evolve individuals of variable length, where solution candidates are encoded as tree structures (Koza, 1992). Typical applications of GP are the automatic creation of mathematical formulas or computer programmes. Therefore, it is rather obvious to use GP for the generation of composite dispatching rules, which are nothing more than simple algorithms defined by a formula to calculate job priorities.

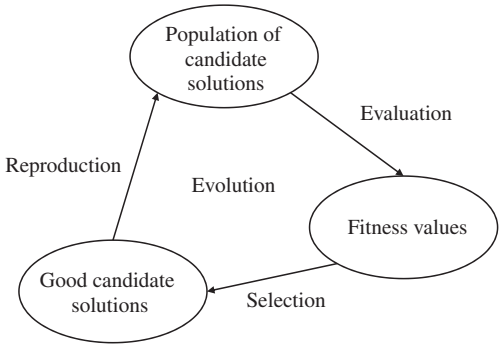

Fig. 1. Simplified diagram of evolutionary cycle.

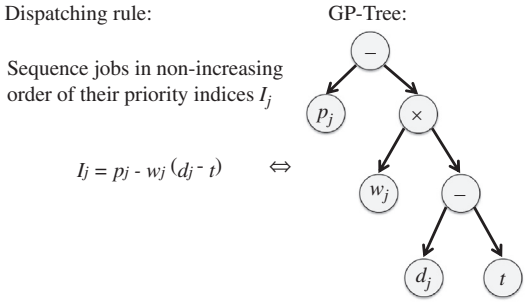

Fig. 2. GP-tree representation of an exemplary dispatching rule with a priority index I_j that is a function of some attributes p_j, w_j, d_j and t.

3.1.1. Encoding of individuals

Fig. 2 shows an example of the encoding of a dispatching rule as tree structure, on which GP operates. A rule is represented by its priority index I_j, which is used to determine the job with the highest priority that is to be processed next. The priority index itself is a function of some of the attributes of the job, e.g. its due date d_j, and can be represented as a GP-tree, which is decoded recursively, starting from the root node, and from left to right.

An important design aspect of GP is the selection of adequate terminals (attributes) and operators as they define the functions (dispatching rules) that can be generated. The selection of suitable terminals is generally problem-specific, and is thus discussed in Section 4.3.1.

3.1.2. Reproduction

The reproduction step is realised using standard GP operators. Individuals from the current population are selected using tournament selection, i.e. a number of individuals equal to the tournament size are randomly sampled from the population and the best is chosen to undergo the respective reproduction operation(s). Here, new individuals are created using either the standard subtree crossover or subtree mutation with a certain probability. 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. The mutation operator modifies an individual by exchanging parts of it with a randomly generated subtree. Furthermore, an elitist strategy is employed that ensures that the best individuals of the current population survive into the next generation.

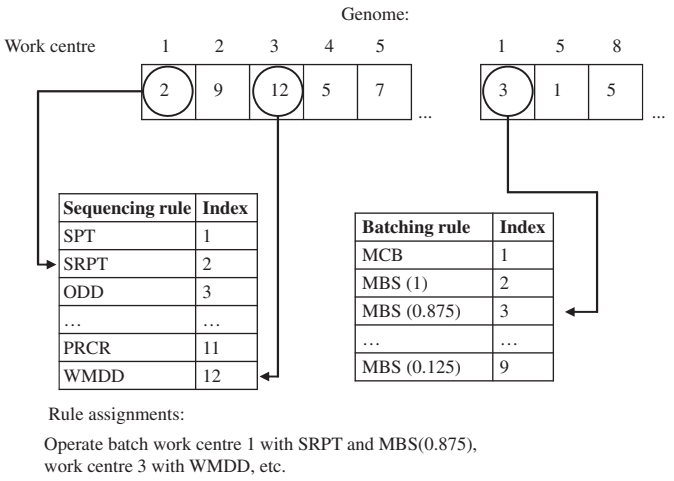

Fig. 3. Representation of an exemplary rule combination as a genome.

3.2. EA-based selection of rules for individual work centres

A standard EA is used to search for a good combination of work centre-specific rules for a given shop, which are encoded as strings of integer values.

3.2.1. Encoding of individuals

The encoding of rule combinations is similar to the one proposed by Yang et al. (2007), where each gene corresponds to a particular work centre and its value specifies the sequencing rule to be applied to it. However, an additional gene is required for work centres with batch processing machines to fully define the dispatching of batches (see Section 4.2). Clearly, the effectiveness of the EA is largely determined by the set of rules provided, which should include the rules that are known to perform well for the particular problem (see also Section 4.3.2). Fig. 3 illustrates the encoding of the EA.

3.2.2. Reproduction

The $\mu+\lambda$ breeder of ECJ is employed in this algorithm. Roughly, the individuals with the μ highest fitness values are chosen from the current population of size $\mu+\lambda$ to produce λ children. These are created by successively applying the uniform crossover and mutation, each with a certain probability. The uniform crossover operator recombines two parents by swapping genes between them. Each gene is swapped at random with a probability of 0.5. The mutation simply overwrites a gene with a random value generated from the uniform distribution on the feasible range of values. The λ children and the μ parents then form the population of the next generation.

3.3. Two-stage approach

In the two-stage hyper-heuristic, a composite rule is first evolved with GP. This rule is then provided to the EA amongst other rules from the literature to search for further improvements. Fig. 4 shows a diagram of the procedure. By optimising on two different heuristic search spaces, the two-stage hyper-heuristic is expected to find combinations of rules that outperform both the composite rules and the rule sets evolved by the GP and EA hyper-heuristic when applied on their own.

3.4. Fitness evaluation using discrete-event simulation

Due to the lack of analytical methods for more complex shop configurations, the prevalent method of evaluating dispatching rules is by simulation. This implies that every rule (combination) created during the run of an EA has to be tested in a simulation environment to determine its fitness. A particular challenge of this approach, which is commonly known as simulation optimisation (Fu et al., 2005), is that it tends to require tremendous computing power. We address this problem in several ways:

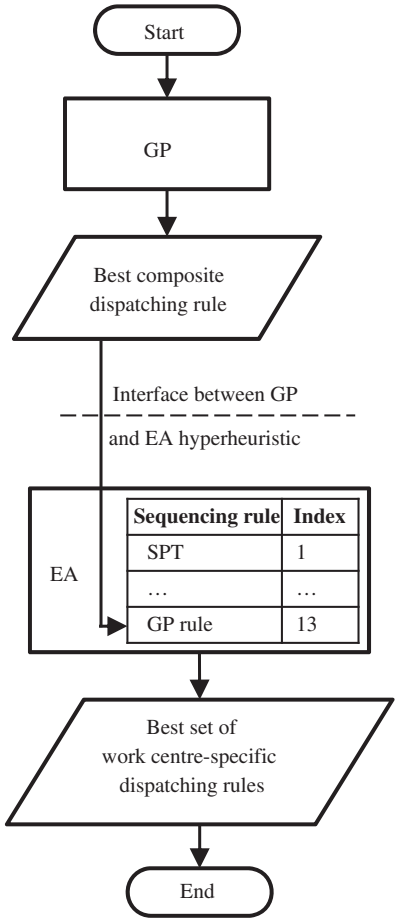

Fig. 4. Process diagram of the two-stage approach.

1. The algorithms are designed to utilise multiple cores in parallel for the execution of simulation experiments.
2. Only one replication of the simulation is conducted in every generation with the same random number seed for all individuals. Using common random numbers provides a good basis for a relative comparison between system configurations, e.g. dispatching rules, which is needed for the selection of the (relatively) best individuals of each generation. However, in order to avoid overfitting to one specific random number stream, different seeds are used across generations. This setting is shown to be favourable by Branke (2001).
3. During the optimisation stage, individuals are evaluated using a relatively short simulation run, and all data, even that collected during the warm-up period, are included in the calculation of performance measures. The motivation for this approach is to save simulation time while at the same time use all available data. This is likely to result in imprecise estimates of the performance of individuals, but will work well as long as these estimates are accurate enough to drive the optimisation into the right direction. On the other hand, a longer simulation run is executed for the evaluation of individuals of the last generation, with data from the warm-up phase being discarded. This results in more precise estimates of the performance of these individuals,

which increases the chance of correctly selecting the best as the best individual of the run.

4. The number of jobs in the system is closely monitored during the simulation run, and the run is aborted if the number of jobs exceeds a certain threshold value. This saves time in the evaluation of inferior individuals that lead to an unstable system, which may be present particularly at the start of an EA run. Any individual for which the simulation experiment did not finish receives a very low fitness value and hence is quickly discarded.

We use a self-implemented discrete-event simulation, called Jasima (Hildebrandt, 2012), for the evaluation of dispatching rules. Jasima is very roughly based on a Java-port of the SIMLIB library (Law, 2007), as described by Huffman (2001). It is an efficient simulation that utilises multiple cores, and offers a variety of dispatching rules and the flexibility to implement more complex scenarios such as the one used in this paper. By making the simulation kernel publicly available under an open source license, we want to encourage researchers to use the package as a test bed when comparing scheduling methods in the future.

4. Application to a problem from semiconductor manufacturing

We apply the three hyper-heuristics to a complex scheduling problem from semiconductor manufacturing with dynamic, stochastic job arrivals. Shops encountered in that industry typically possess characteristics such as reentrant material flows, and machines with sequence-dependant setup times or batch processing capabilities (Uzsoy et al., 1992; Pfund et al., 2006; Wu et al., 2008), increasing the complexity of scheduling even beyond the classic job shop problem, which is already NP-hard (Garey et al., 1976). The fact that there are different types of work centres in a semiconductor manufacturing system motivates a work centre-specific generation and selection of dispatching rules, which makes the problem suitable for testing the effectiveness of the two-stage hyper-heuristic.

4.1. Scenario description

The model shop is based on the dataset 4r of the MASM testbed (Feigin et al., 1994). The model contains 36 machines in 31 work centres, required to produce seven products. The products follow one of the two possible routes and products sharing the same route do not differ in any other way. It is therefore more sensible to speak of two distinct product categories instead of seven products. Jobs of product category A consist of 92 operations, while jobs of product category B are comprised of 19 operations, and both routes involve revisitations of work centres. Work centres vary with regard to their setup requirements, batch processing capabilities, number of parallel machines, workload and relative position in the system. In the following, jobs that can be processed together as a batch or require the same machine setup are said to belong to the same (batch or setup) family.

The mean job arrival rates are set to 4.5 jobs per day of product category A and 10.5 jobs per day of product category B, resulting in a mix of 30%/70%. This leads to a challenging problem with several critical work centres. Table 1 provides a description of the highly utilised work centres and their properties. Explicit attention has to be paid to the two work centres involving sequence-dependant setups and batching as their utilisation depends on the dispatching strategy used. If those work centres are controlled in an inefficient way, e.g. by regularly conducting time-consuming setups or by processing near empty batches, these work centres can easily become the bottleneck.

Table 1
Critical work centres in the shop.

Name	Special properties	Utilisation
AME135	–	93.8%
D1-9	Parallel machines	91.1%
DFC4	–	89.1%
PE1-5	Parallel machines; setups	74.3% (excluding setups)
QLESS	Batching	64.8% (with full batches)

Job arrivals follow a Poisson process. On arrival of a job j, it is assigned a weight w_j that is generated from a discrete uniform distribution on the interval [1,10]. Furthermore, its due date d_j is set with the total work content method, i.e. $d_j = r_j + a_j \times p_j$, where r_j and p_j refer to the release date and total processing time of the job, respectively. The allowance factor a_j is sampled from a continuous uniform distribution on the interval [2,5]. The objective is to minimise the mean weighted tardiness $(1/n)\sum_{j=1}^{n} w_j T_j$, where n is the number of completed jobs and the tardiness T_j of a job j is defined as the non-negative difference between its completion time C_j and its due date, i.e. $T_j = \max(C_j - d_j, 0)$. The mean weighted tardiness is a widely used measure of the delivery performance in a situation where some customers are considered more important than others (Pinedo, 2008, Chapter 2).

4.2. Benchmarks

In order to evaluate the effectiveness of the rule (sets) generated by the hyper-heuristics, some well-known rules from the literature are used as benchmarks. Most of these dispatching or sequencing rules have been designed for unit-capacity machines and their definition does not cover the formation of batches. They therefore have to be combined with a so-called batching rule to fully specify the dispatching regime for batch processing machines. Batching rules are discussed after the presentation of the sequencing rules.

4.2.1. Sequencing rules

The following six sequencing rules are included in the study.

(1) PRFCFS: The Priority First Come-First Served (PRFCFS) rule schedules jobs in non-increasing order of their weights and breaks ties by selecting the job that arrived at the work centre first. This is an easily implemented rule, which explains the popularity of PRFCFS within the semiconductor industry (Pfund et al., 2006).

(2) PRCR: The Priority Critical Ratio (PRCR) rule uses the CR rule to break ties between jobs with equal weights. The CR rule sequences jobs in non-decreasing order of their critical ratio, which is the ratio of the remaining allowance (the time till it is due) of a job to its remaining processing time $p_{rem,j}$, i.e. $(d_j - t)/p_{rem,j}$, where t refers to the current time. It tends to prioritise jobs that have less time available before they become late, but also acknowledges the need to quickly finish tardy jobs that are close to their completion. Consequently, PRCR is often employed by semiconductor manufacturers with the focus on on-time delivery (Pfund et al., 2006).

(3) WMDD: The Modified Due Date (MDD) rule is due to Baker and Bertrand (1982) in the attempt to integrate the strengths of the SPT and EDD rule, which both perform well for mean tardiness under different levels of due date tightness. Kanet and Li (2004) extend MDD to the Weighted MDD (WMDD) rule that accounts for different job weights to reduce the mean weighted tardiness. The MDD formulation for job shops by Baker and Kanet

(1983) is adopted here for WMDD to give a priority index of

$$I_j = -\frac{1}{w_j}\max(p_{rem,j}, d_j - t), \quad (1)$$

where a larger index I_j corresponds to a higher priority of job j. Although this extension of WMDD to a job shop is self-evident, we have been the first to test it to our knowledge (Pickardt et al., 2010).

(4) WMOD: In the Modified Operation Due Date (MOD) rule (Baker and Kanet, 1983), the remaining processing time and due date of the job in the priority index of MDD are replaced by the processing time $p_{imt,j}$ and due date $d_{imt,j}$ of its imminent operation. MOD is extended here to the Weighted MOD (WMOD) rule with its priority index given by

$$I_j = -\frac{1}{w_j}\max(p_{imt,j}, d_{imt,j} - t). \quad (2)$$

Although Kanet and Li (2004) mention this rule as a possible extension of WMDD to job shops, we seem to have been the first to test this rule in a previous paper (Pickardt et al., 2010).

None of the above rules account for sequence-dependant setup times, which can become a problem if they cause critical machines to run inefficiently by conducting many unnecessary setups. Hence, all the above rules are equipped with a simple setup avoidance mechanism which ensures that jobs requiring no setup always receive highest priority.

(5) ATCS: Another rule specifically designed to minimise the mean weighted tardiness is the Apparent Tardiness Cost (ATC) rule by Vepsalainen and Morton (1987). Lee et al. (1997) extend the rule to ATC with Setups (ATCS), which explicitly considers sequence-dependant setup times in the prioritisation of jobs. The priority index I_j of a job j is calculated with ATCS as

$$I_j = \frac{w_j}{p_{imt,j}} \exp\left(-\frac{\max(d_{imt,j} - p_{imt,j} - t, 0)}{k_1 \overline{p}_{imt}}\right) \exp\left(-\frac{s_j}{k_2 \overline{s}}\right). \quad (3)$$

In Eq. (3), s_j refers to the time required to set up the machine from the current state to the state required for processing job j while $\overline{p}_{imt}$ and $\overline{s}$ indicate the average operation processing time and the setup time of queueing jobs, respectively. ATCS also possesses two scaling parameters k_1 and k_2 that have to be set. In general, the best parameter values depend on the problem at hand. We tested ATCS with 192 different parameter settings using values of 0.0001, 0.001, 0.01, 0.1, 0.5, 1.0, 1.5, …, 6.5, 7, 8, 9, 10, 20, 50, 100 for k_1 and 0.0001, 0.001, 0.01, 0.1, 0.25, 0.5, 1.0, 1.5 for k_2, covering more than the ranges of values suggested by Lee et al. (1997). The rule is subsequently applied with the best parameter setting for the given problem.

(6) BATCS: An extension of the ATCS rule to cope with batch processing machines is proposed by Mason et al. (2002). The Batch ATCS (BATCS) rule operates on batches instead of jobs, and takes into account the fullness of a batch in addition to sequence-dependant setup times. If applied on a unit-capacity machine, the rule handles all jobs as full 1-job batches and hence reverts to ATCS. The priority index I_b of BATCS for a batch b is given by

$$I_b = \frac{w_b}{p_{imt,b}} \exp\left(-\frac{\max(d_{imt,b} - p_{imt,b} - t, 0)}{k_1 \overline{p}_{imt}}\right) \exp\left(-\frac{s_b}{k_2 \overline{s}}\right) \left(\frac{n_b}{c_b}\right), \quad (4)$$

where n_b indicates the number of jobs in the batch and c_b the maximum number of jobs that could be batched together for that operation. The average weight of the jobs in the batch determines the batch weight w_b, while the batch due date $d_{imt,b}$ corresponds to the minimum operation due date of the contained jobs. In this work, the BATCS rule is first applied to form a batch for every family with jobs in the queue. This is done by filling the batch up with jobs of the given family in decreasing order of their priority indices (BATCS reverts to ATCS) up to the maximum number. BATCS is then applied again to select the batch with the highest priority index for processing. Similar to ATCS, BATCS was tested with the same k_1 and k_2 values and is applied with the best parameter setting in the following comparisons.

4.2.2. Batching rules

Except for BATCS, all sequencing rules are combined with two different batching rules, resulting in 11 $(5 \times 2 + 1)$ benchmark rules altogether.

(1) MCB: The Most Complete Batch (MCB) rule selects the fullest batch, i.e. the one with the highest ratio of the number of jobs a batch contains to the number of jobs that could be processed together in a full batch n_b/c_b. For every family with jobs in the queue, the sequencing rule is used to select jobs in decreasing order of their priority to form a batch of maximum size. MCB then chooses the fullest batch and resolves tie-breaks by selecting the one that contains the job with the highest priority. The MCB rule is similar to the principle of setup avoidance in its focus on maximising the utilisation of machine capacity.

(2) MBS: The Minimum Batch Size (MBS) rule, which originates from a paper by Neuts (1967), allows for a batch to be started only if its fullness n_b/c_b is equal to or larger than a critical number L. MBS restrains the sequencing rule to select only among those jobs for which there are enough jobs of the same family waiting to form a batch of the required fullness. The batch is then filled with compatible jobs in decreasing order of their priority up to the maximum possible number. All batch operations in the examined shop can be run with up to eight jobs and the best of the eight possible values for L is subsequently used for each of the benchmark rules.

4.3. Parameter settings

A number of parameters have to be set in the hyper-heuristics. In particular, the respective search spaces have to be defined by selecting appropriate terminals and functions for GP, and the set of rules available to the EA.

4.3.1. GP

The terminals and functions for the GP hyper-heuristic include predominantly those attributes that make up the good rules from the literature. The terminal set is supplemented with some additional variables that appear promising, and constants that allow GP to normalise generated terms. The former include two terminals that assess the number of queueing jobs of the same batch family to allow a rule to prioritise fuller batches. Beside basic arithmetic and mathematical operations, the function set contains a ternary version of ifte, i.e. if $a \geq 0$ then b else c, where a,b,c are sub-expressions evolved by GP. Table 2 summarises the parameter setting used for GP, which is based on recommendations for default values (Poli et al., 2008).

4.3.2. EA

Table 3 shows the parameter setting of the EA hyper-heuristic. The EA is provided with a set of common dispatching rules (Haupt, 1989; Kutanoglu and Sabuncuoglu, 1999), which includes all those used as benchmarks. Specifically, ATCS, BATCS and MBS are provided with different parameter values that include the best ones found for the given problem. Moreover, when the EA is run as part of the two-stage hyper-heuristic, the rule set is extended with the best rule generated by the GP hyper-heuristic in the first stage.

To allow for a fair comparison, the number of generations and population size is set to the same values as for the GP algorithm. Although only $\lambda = 750$ new individuals are created in every generation of the EA, the remaining individuals have to be re-evaluated due to the generation-specific seed which strongly

Table 2
Parameter setting of the GP hyper-heuristic.

Parameter	
Generations	50
Population size	1000
Initial population	Ramped half-and-half (min depth 2, max depth 6)
Selection	Tournament (size 7)
Crossover probability	90%
Max. depth for crossover	17
Mutation probability	10%
Elitism	1%
Terminals (17)	Remaining number of uncompleted operations, remaining proc. time, imminent operation proc. time, average proc. time in queue, weight, remaining allowance, rem. allowance of imminent op., setup time for imminent operation, max. reduced setup time if processed on parallel machine, number of queueing jobs of same setup family, average setup time in queue, fullness of batch, number of queueing jobs of same batch family, 0, 1, 2, 10
Functions (7)	+, −, ×, ÷, max, min, ifte

Table 3
Parameter setting of the EA hyper-heuristic. Detailed descriptions of the sequencing rules can be found in the surveys by Haupt (1989) and Kutanoglu and Sabuncuoglu (1999).

Parameter	
Generations	50
Population size	1000
Initial population	Randomly generated [+GP rule]
Selection	$\mu(=250)+\lambda(=750)$
Crossover probability	100%
Mutation probability	5%
Sequencing rules (58+1)	PRFCFS, PRCR, WSPT, WMDD, WMOD, CR, SPT, LWKR, EDD, MDD, ODD, MOD, SLACK (all with and without setup avoidance), ATCS, BATCS (both with $k_1=2.0,3.0,5.0,6.0$ and $k_2=0.005,0.01,0.1,1.0$), [GP rule]
Batching rules (9)	MBS ($L=0.125,0.25,0.375,0.5,0.625,0.75,0.875,1$), MCB

affects the absolute fitness values. Hence, both hyper-heuristics perform exactly 50×1000 simulation experiments during a run. Other parameters of the EA, such as the crossover and mutation probabilities, are fixed to typically encountered values. The initial population of the EA is generated randomly. However, an individual that corresponds to applying the GP rule to every work centre is deliberately inserted into the initial population, if the EA is run as the second stage of the two-stage hyper-heuristic.

In each hyper-heuristic, the shop is simulated for 2 years to evaluate an individual in every but the last generation. In the last generation, individuals are evaluated with a longer simulation experiment of six simulated years (see Section 3.4), where the first year serves as the warm-up period as determined using the Welch procedure (Law, 2007, Chapter 9). The best individual of the last generation is returned by a hyper-heuristic as the best rule (set) of the run.

5. Experimental results

The three hyper-heuristics are run for 20 times, which results in 20 different evolved (sets of) rules. Common random numbers are used across the hyper-heuristics to give the same set of simulation experiments for the fitness evaluations of a replication. The performance of evolved and benchmark rules is measured by conducting 50 replications of the simulation experiment encompassing six years, where data collection starts after the 1 year of warm-up. Again, common random numbers are used in these experiments. All performance values are tested for statistically significant differences using the Bonferroni–Dunn test, with complex comparisons when composite means of groups are involved (Sheskin, 2007, Test 24). The test adjusts the overall significance level $\alpha=0.05$ for the number of comparisons made by applying the Bonferroni inequality (Law, 2007, Chapter 10).

Table 4 compares the performance of the best of the 20 (set of) rules, generated by each of the hyper-heuristics, with that of the benchmark rules for the mean weighted tardiness and other criteria. The values in bold indicate the best performance achieved for a given measure. Table 4 shows that all three of the hyper-heuristics are able to find (sets of) dispatching rules that significantly outperform the best benchmark rules WMOD_MCB and ATCS_MCB with respect to mean weighted tardiness. Moreover, the rule set discovered by the two-stage hyper-heuristic is the most effective overall, yielding a 46% reduction of mean weighted tardiness over the best manually developed rule, WMOD_MCB. The good performance of the WMOD rule is another remarkable result of this study, considering that it has been largely ignored in the literature. Irrespective of the batching rule in use, WMOD achieves a significantly lower mean weighted tardiness than the PRCR rule, which according to Pfund et al. (2006) is what due date-oriented practitioners employ in the semiconductor industry.

Although the evolved (sets of) rules are specifically optimised for the mean weighted tardiness, their dominance carries over to other common measures of due date performance. For the mean tardiness and the conditional mean tardiness (the mean tardiness of tardy jobs), the rule set generated by the two-stage approach again is best. It is only outperformed by the best rule (set) of the other two hyper-heuristics with respect to the percentage of tardy jobs.

In summary, these results illustrate the benefits of applying hyper-heuristics in general within this context. To assess the effectiveness of each of the hyper-heuristics, the average performance of replications is considered rather than that of the best replication, which could have been a 'lucky punch'. Table 5

Table 4
Comparison of performance of the automatically generated rule (sets) with benchmark rules regarding mean weighted tardiness (MWT), mean tardiness (MT), percentage of tardy jobs (PT) and conditional mean tardiness (CMT). All rules but ATCS and BATCS are applied with setup avoidance. The best performance values are in bold, and values marked with the same letter are not significantly different from each other at the 5% significance level.

Dispatching rule (set)	MWT (in time units)	MT (in time units)	PT (in %)	CMT (in time units)
Two-stage best	**548.8**	**315.6**[a]	22.2	**1404.8**
GP best	641.2	356.8	21.5	1640.2[b]
EA best	750.2	**319.4**[a]	**20.0**	1579.1[a]
WMOD_MCB	1016.7[a]	433.2[b]	27.0	1593.1[a,b]
ATCS_MCB(3.0, 0.005)	1035.7[a]	431.2[b]	22.8	873.5
BATCS(5.0, 0.01)	1669.8	921.3	30.9	2972.1
WMDD_MCB	1833.3	875.5	25.9	3368.9
ATCS(6.0, 0.01)_ MBS(0.625)	2296.0	1036.0[c]	39.5[a]	2614.0
WMOD_MBS(0.625)	2491.1	1025.8[c]	49.9[b]	2049.5
PRCR_MCB	3405.9	1642.4	37.2	4416.3[c]
PRFCFS_MCB	3641.9	1722.9[d]	37.4	4606.0
WMDD_MBS(0.625)	4141.6	1752.4[d]	39.8[a]	4396.0[c]
PRCR_MBS(0.625)	6481.4	2814.8	49.9[b]	5644.1
PRFCFS_MBS(0.625)	6703.2	2874.7	49.8[b]	5770.2

Table 5
Evaluation of hyper-heuristics, based on the mean weighted tardiness performance (in time units) of the best rule (set) discovered, averaged over the twenty hyper-heuristic replications. In addition, the standard deviation and the performance of the best and worst rule (set) from the 20 replications are given.

Hyper-heuristic	Avg.	Std.	Best	Worst
GP	797.6	110.4	641.2	1009.2
EA	797.5	26.5	750.2	844.3
min(GP, EA)	749.7	52.1	641.2	834.0
Two-stage	637.5	55.2	548.8	744.5

presents the mean weighted tardiness performance averaged over the twenty (sets of) rules for each hyper-heuristic, together with the standard deviation and the performance of the best and worst of the respective twenty (sets of) rules. Although the best GP rule fares significantly better than the best EA rule set, both hyper-heuristics produce (sets of) rules that, on average, are similarly effective. As indicated by the higher standard deviation, the performance of rules evolved in different replications of the GP hyper-heuristic fluctuates more than in the case of the EA rule sets, which could be a result of the larger and more complex search space of GP. Then, which of the two hyper-heuristics to select becomes a matter of the number of replications or computational time available, and it can not be concluded that one is better than the other.

A valid benchmark for the two-stage approach is to run its two components, the GP and EA hyper-heuristic, separately from each other, and take the better of the two resulting individuals as the basis for a comparison. It implies that the same number of simulation experiments (1000∗50∗2) are conducted as for one replication of the two-stage hyper-heuristic. In Table 5, this benchmark is indicated by the notation min(GP, EA) and is shown to lead to a higher mean weighted tardiness on average than rule sets generated by the two-stage approach. Since this difference is statistically significant, the combined hyper-heuristic is more effective than running the GP or EA separately and selecting the better rule (set). This statement is supported by Fig. 5, which displays the performance of the (sets of) rules resulting from each of the 20 runs of the three hyper-heuristics. For every single replication, the two-stage hyper-heuristic finds a rule set that yields a lower mean weighted tardiness than either the GP rule or the EA rule set by itself. Moreover, there is a clear positive correlation (coefficient of 0.812) between the performance of the GP rule and the corresponding rule set from the two-stage hyper-heuristic, indicating that a good GP rule provides the basis and is part of a more effective rule set. The figure also confirms the earlier observation that the effectiveness of rules generated with GP seem to vary a lot more between runs of the hyper-heuristic than that of the rule sets evolved by the EA, although none of the 20 GP rules is worse than WMOD_MCB, the best benchmark from the literature.

Now that the superiority of the two-stage hyper-heuristic is established, the aim is to gain an understanding of what makes it so effective. There must be a certain optimisation potential that is not tapped by the benchmark rules, nor by the two single-stage hyper-heuristics. As a starting point, the jobs can be categorised by their weight to see which are delayed and by how much. The results in Table 4 indicate that the best GP rule is more effective in minimising the mean weighted tardiness than the best EA rule set, but not when weights are disregarded as illustrated by their respective mean tardiness performances. In other words, the former focusses on the timely completion of jobs that have a high weight while the latter tries to keep the overall delay at a low level.

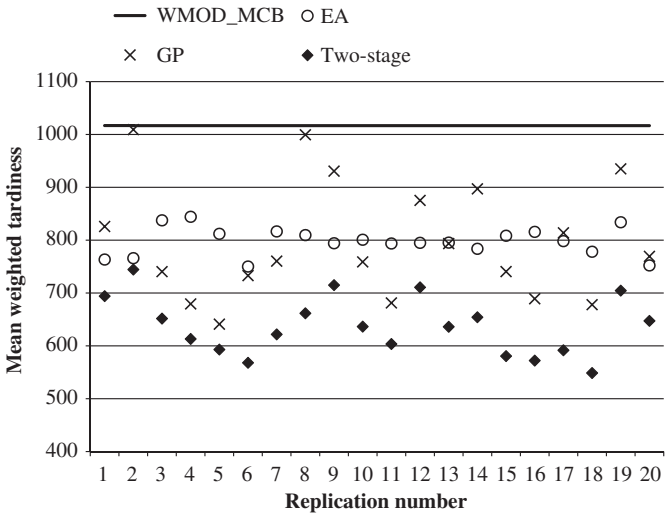

Fig. 5. Mean weighted tardiness performance (in time units) of the rule or rule set from each replication of the three hyper-heuristics. The performance of WMOD_MCB as the best rule from the literature is included as a reference line.

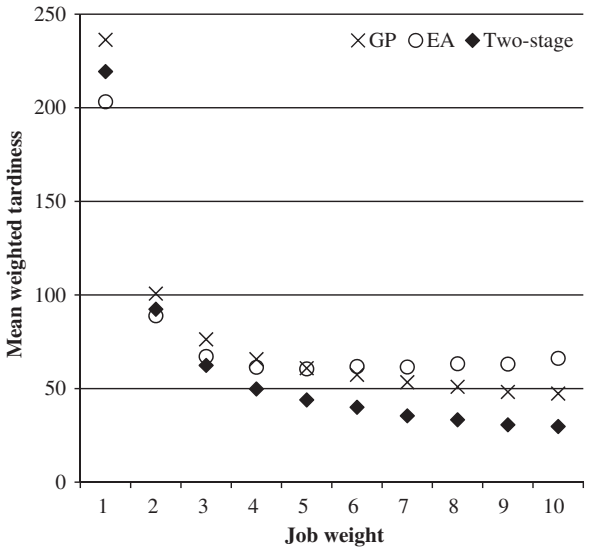

Fig. 6. Mean weighted tardiness (in time units) for jobs classified by weight, achieved on average by the three hyper-heuristics.

This behaviour of the best rule (set) is actually typical of the rules and rule sets evolved by the two respective hyper-heuristics. Fig. 6 presents the weighted tardiness incurred by the three hyper-heuristics on average, for each of the 10 subsets of jobs classified by their weight. The graph shows that the GP rules are generally more effective in minimising the tardiness of jobs with high weights, whereas the EA rule sets achieve a smaller delay of jobs with low weights at the expense of an increased tardiness of jobs with high weights. For jobs with a weight of 5, both hyper-heuristics generate (sets of) rules that result in a similar delay on average. The rule sets evolved by the two-stage approach reduce tardiness of jobs with a weight of 3 or greater even more than both the GP and EA, accepting in exchange a higher tardiness than the EA rule sets for jobs with a weight of 2 or less. From this discussion it appears that the GP and EA hyper-heuristic concentrate on different optimisation potentials, which both can be exploited and traded-off by the two-stage hyper-heuristic.

More insights into what the hyper-heuristics optimise can be gained by an analysis of the resulting rule sets for common assignments of rules to the individual work centres. Table 6 lists, for each of

the critical work centres (see Section 4.1), the rule that is assigned to it most often within the twenty rule sets, generated by the EA and the two-stage hyper-heuristics, and the associated frequency of selection. As can be seen, the GP rule is always applied to the work centre QLESS, and also to the work centre PE1-5 within a majority of the rule sets created by the two-stage approach. Since these work centres involve batching and sequence-dependant setup times, respectively, they need to be controlled in an efficient way in order to prevent them from becoming bottlenecks. This is supported by the fact that the EA without access to the GP rule consistently evolves rule sets where both work centres are run with rules that focus on efficiency. In the case of PE1-5, this means minimising unnecessary setup operations, i.e. setup avoidance, where for QLESS it is equivalent to maximising the use of capacity, which is the strategy of the MCB batching rule. The GP hyper-heuristic seems to recognise the criticality of these two work centres, and concentrates its efforts on the generation of dispatching rules that ensure their effective operation. It also appears that available manually developed rules are insufficient for these work centres, which supports the intuition that the presence of complex properties such as sequence-dependant setups makes the design of effective rules considerably more difficult. In contrast, the GP rule is not the first choice for other work centres. The two-stage hyper-heuristic generally applies an ATC-based rule (BATCS and ATCS revert to ATC for machines without setups or batching) to the work centre AME135, with $k_1 = 3.0$ in. nine instances and $k_1 = 5.0$ in. another six, and WMDD to the work centre DFC4. Both rules are also most frequently assigned by the EA. For the work centre D1-9, there appears to be no clear preference for any rule, which could indicate that none of the rules fit to the work centre particularly well.

A surprising result of the analysis of assignments of rules to critical work centres is the nearly complete absence of the WMOD rule. Despite outperforming almost all the other benchmark rules when applied to all work centres (see Table 4), it occurs only once for one of the five work centres in the 40 evolved rule sets. This indicates that WMOD is a compromise rule that is not very effective for any specific work centre, but one that works reasonably well overall.

Since the hyper-heuristics optimise on simulation experiments with one given parameter setting, the generated (set of) rules could be sensitive to changing conditions in the system. If these changes are overt and 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 on an adopted simulation model. However, some changes, e.g. in the job arrival pattern, might be a lot more subtle and less obvious. If the performance of the evolved (sets of) rules strongly deteriorated in such a case, the practicality of the whole approach would be disputable.

Table 6
Dispatching rules assigned most frequently to critical work centres by the EA and the two-stage hyper-heuristic.

Work centre	EA		Two-stage	
	Rule	Frequency	Rule	Frequency
AME135	ATC(5.0)	11/20	ATC(3.0)	9/20
D1-9	MDD	7/20	GP rule	5/20
DFC4	WMDD	20/20	WMDD	10/20
PE1-5	CR (setup avoidance)	10/20	GP rule	15/20
QLESS	CR_MCB	15/20	GP rule	20/20

6. Sensitivity analysis

The benchmark for the robustness of an evolved rule (set) is the rule that would be employed without the availability of a hyper-heuristic, i.e. WMOD_MCB. This rule is also likely to be robust as it does not require any parameters to be set. The sensitivity analysis is conducted for the best rule set from the two-stage hyper-heuristic with respect to changes in the arrival rates of the different products and the due date setting.

6.1. Arrival rates

As described in Section 4.1, the total arrival rate of the original scenario is 15 jobs per day with a mix of 30%/70% for the two distinct product categories, leading to a bottleneck utilisation of 93.8%. Fig. 7 shows how the best rule set of the two-stage hyper-heuristic fares in relation to WMOD_MCB for other job arrival patterns. The comparison is done for varying levels of bottleneck utilisation due to changes in the total arrival rate, as well as for different mixes of the product categories with the bottleneck utilisation kept at the same level. Varied product shares also have the side effect of a changing location of the bottleneck.

Clearly, the rule set dominates WMOD_MCB across all examined utilisation levels (Fig. 7a) and product mixes (Fig. 7b), and it can be concluded that the rule set is robust to changes in the arrival rates.

6.2. Due date setting

The original allowance factors are uniformly distributed within the interval [2,5] (see Section 4.1) with a range $R=3$ and

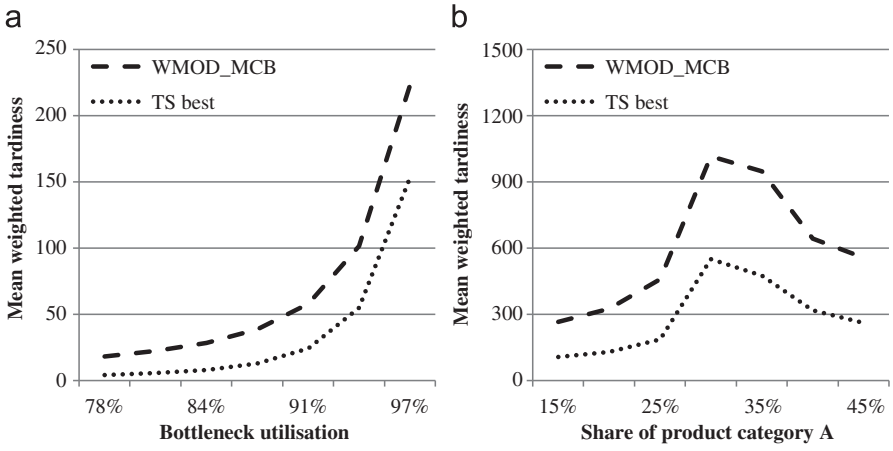

Fig. 7. Mean weighted tardiness performance (in time units) of the best rule set of the two-stage hyper-heuristic and WMOD_MCB for different job arrival patterns. (a) Variation in total arrival rate. (b) Variation in product category mix.

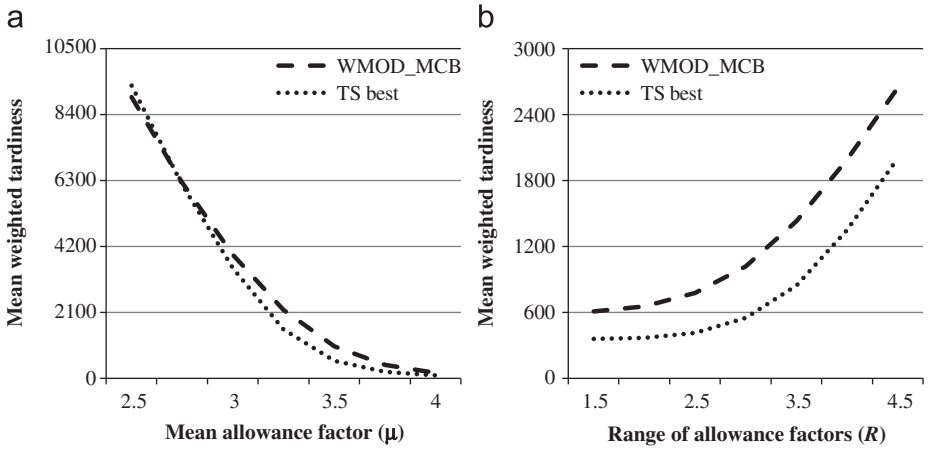

Fig. 8. Mean weighted tardiness performance (in time units) of the best rule set of the two-stage hyper-heuristic and WMOD_MCB for different due date settings. Due dates are generated from a uniform distribution on the interval $[\mu-(R/2),\mu+(R/2)]$. (a) Variation in due date tightness ($R=3$). (b) Variation in due date variability ($\mu=3.5$).

a mean $\mu=3.5$. Reducing the mean leads to tighter due dates on average, while a higher mean allowance factor corresponds to looser due dates. The allowance factor range is a measure of the overall variability of due dates in terms of their tightness. Fig. 8 compares the performance of the best rule set of the two-stage hyper-heuristic with that of WMOD_MCB across different levels of these two variables.

As shown in Fig. 8(a), the rule set is never worse than WMOD_MCB for larger allowance factors. However, WMOD_MCB consistently gains on the rule set with a decreasing mean allowance factor, until it eventually outperforms the rule set. Therefore, the rule set is not very robust to tighter due date settings. In contrast, the relative advantage of the rule set over WMOD_MCB is maintained across varying allowance factor ranges and insensitive to a change (Fig. 8(b)).

In summary, the best rule set generated by the two-stage hyper-heuristic is better than the best benchmark rule WMOD_MCB for most of the examined conditions. It therefore appears to be sufficiently robust to changes in the arrival rates and the due date setting. The only exception is if due dates become very tight in which case WMOD_MCB becomes more effective. However, such a large change could probably be detected by monitoring the due dates of incoming jobs and restarting the hyper-heuristic to generate a rule (set) suitable for the new conditions.

In order to obtain an estimate on how much could be gained by reoptimising in case of small changes, we ran 20 additional replications of the hyper-heuristic for each of the two scenarios with 90.6% and 96.9% utilisation (everything else unchanged). The results show that the optimised rule set for the case of 96.9% utilisation reduces the mean weighted tardiness by 34% relative to WMOD_MCB, in comparison to using the rule set evolved for 93.8% which reduces the mean weighted tardiness by 31%. In case of 90.6% utilisation, the specially evolved rule set reduces the mean weighted tardiness by 67%, compared to the 58% reduction by the rule set evolved for 93.8% utilisation. This shows that, as expected, specially evolved rule (sets) indeed perform better than rule (sets) evolved for a different scenario, but compared to the great improvements over the best benchmark rule, the additional benefit is small and the rule (sets) can be regarded as robust to small changes in the environment.

7. Conclusion

In this paper, we have developed a two-stage hyper-heuristic to automatically generate sets of dispatching rules for complex and dynamic scheduling problems. The approach combines a GP hyper-heuristic that evolves a composite rule from basic attributes (Hildebrandt et al., 2010; Pickardt et al., 2010), with an EA hyper-heuristic that assigns different rules to different work centres as proposed by Yang et al. (2007). We tested the two-stage procedure on a scenario from semiconductor manufacturing with the objective of minimising mean weighted tardiness against its two components, the GP and EA hyper-heuristics, and a wide range of manually designed rules from the literature. While all three hyper-heuristics were able to create (sets of) rules that substantially outperform the manually designed rules, the two-stage approach produced a significantly better performance than the other two hyper-heuristics. Among the benchmark rules from the literature, the relatively simple, but widely ignored WMOD rule turned out to be extremely effective for reducing mean weighted tardiness, and should be included in future performance studies of dispatching rules.

A closer analysis of the results has revealed that the GP and EA hyper-heuristic appear to tap different optimisation potentials and that the two-stage approach is capable of exploiting both to some extent. An investigation of the evolved rule sets has further indicated that the GP hyper-heuristic concentrates on two work centres with setup requirements and batching capabilities that have to be controlled efficiently, and for which it is harder to develop effective rules manually. The sensitivity analysis showed that the rule sets resulting from the two-stage hyper-heuristic are robust to most changes in the job arrival pattern and due date setting.

An obvious direction for future research is the application of the approach to other shop configurations, operating conditions and objective functions to see whether it performs equally well. Since the effectiveness of the two-stage hyper-heuristic depends on a good composite rule that is generated in the first stage, efforts should be put into making the GP algorithm more reliable as this would most likely benefit the two-stage approach. Another general issue is the large amount of computational time spent on the simulation-based evaluations of individuals. This could be addressed by using surrogate functions that are able to approximate the results of simulation experiments.

Acknowledgements

We would like to thank Robert Sarney for implementing a first version of the EA hyper-heuristic and the four anonymous referees for their valuable comments. This research has been supported by the German Research Council (DFG) under Grants BR 1592/7-1 and SCHO 540/17-1.

References

Atlan, L., Bonnet, J., Naillon, M., 1994. Learning distributed reactive strategies by genetic programming for the general job shop problem. In: Proceedings of the Seventh Annual Florida Artificial Intelligence Research Symposium.

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

Baek, D.H., Yoon, W.C., 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., Yoon, W.C., Park, S.C., 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., Bertrand, J.W.M., 1982. A dynamic priority rule for scheduling against due-dates. Journal of Operations Management 3 (1), 37–42.

Baker, K.R., Kanet, J.J., 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.

Blackstone, J.H., Phillips, D.T., Hogg, G.L., 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., Nomden, G., Slomp, J., 2008. Performance evaluation of family-based dispatching in small manufacturing cells. International Journal of Production Research 46 (22), 6305–6321.

Branke, J., 2001. Reducing the sampling variance when searching for robust solutions. In: GECCO-2001: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 235–242.

Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.R., 2010. A classification of hyper-heuristic approaches. In: Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol. 146, second ed. Springer, New York, pp. 449–468.

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

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

Eiben, A.E., Smith, J.E., 2003. Introduction to Evolutionary Computing Natural Computing, first ed. Springer, Berlin, Heidelberg.

El-Bouri, A., Shah, P., 2006. A neural network for dispatching rule selection in a job shop. The International Journal of Advanced Manufacturing Technology 31 (3–4), 342–349.

Feigin, G., Fowler, J., Leachman, R., 1994. Modeling and analysis for semiconductor manufacturing. Last accessed 26 October 2011 http://www.eas.asu.edu/~masmlab.

Fu, M.C., Glover, F.W., April, J., 2005. Simulation optimization: a review, new developments, and applications. In: Proceedings of the 2005 Winter Simulation Conference, pp. 83–95.

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

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

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

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

Hershauer, J.C., Ebert, R.J., 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. Last accessed 16 February 2012 http://code.google.com/p/jasima/.

Hildebrandt, T., Heger, J., Scholz-Reiter, B., 2010. Towards improved dispatching rules for complex shop floor scenarios—a genetic programming approach. In: GECCO '10: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 257–264.

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

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

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

Jakobović, D., Budin, L., 2006. Dynamic scheduling with genetic programming. In: Genetic Programming. Lecture Notes in Computer Science, vol. 3905. Springer, Berlin, Heidelberg, pp. 73–84.

Jakobović, D., Jelenkovic, L., Budin, L., 2007. Genetic programming heuristics for multiple machine scheduling. In: Genetic Programming. Lecture Notes in Computer Science, vol. 4445. Springer, Berlin, Heidelberg, pp. 321-330.

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

Koza, J.R., 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA.

Kutanoglu, E., Sabuncuoglu, I., 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., Barman, S., 1989. Performance of simple priority rule combinations in a flow-dominant shop. Production and Inventory Management Journal 30 (3), 1–4.

Law, A.M., 2007. Simulation Modeling and Analysis, fourth ed. McGraw-Hill, New York, USA.

Lee, G.-C., Kim, Y.-D., Kim, J.-G., Choi, S.-H., 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., Bhaskaran, K., Pinedo, M., 1997. A heuristic to minimize the total weighted tardiness with sequence-dependent setups. IIE Transactions 29 (1), 45–52.

Luke, S., Panait, L., Balan, G., Paus, S., Skolicki, Z., Popovici, E.,Sullivan, K., Harrison, J., Bassett, J., Hubley, R., Chircop, A., Compton, J., Haddon, W., Donnelly, S., Jamil, B., Zelibor, J., Kangas, E., Abidi, F., Mooers, H., O'Beirne, J., 1998. A Java-Based Evolutionary Computation Research System. Last accessed 24 January 2012 〈http://cs.gmu.edu /~eclab/projects/ecj/〉.

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

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

Miyashita, K., 2000. Job-shop scheduling with genetic programming. In: GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 505–512.

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

Nie, L., Shao, X., Gao, L., Li, W., 2010. Evolving scheduling rules with gene expression programming for dynamic single-machine scheduling problems. The International Journal of Advanced Manufacturing Technology 58 (5–8), 729–747.

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

Pfund, M.E., Mason, S.J., Fowler, J.W., 2006. Semiconductor manufacturing scheduling and dispatching. In: Handbook of Production Scheduling. International Series in Operations Research & Management Science, vol. 89, first ed. Springer, New York, pp. 213–241.

Pickardt, C., Branke, J., Hildebrandt, T., Heger, J., Scholz-Reiter, B., 2010. Generating dispatching rules for semiconductor manufacturing to minimize weighted tardiness. In: Proceedings of the 2010 Winter Simulation Conference, pp. 2504–2515.

Pierreval, H., 1992. Training a neural network by simulation for dispatching problems. In: Proceedings of the Third International Conference on Computer Integrated Manufacturing, pp. 332–336.

Pierreval, H., Ralambondrainy, H., 1990. A simulation and learning technique for generating knowledge about manufacturing systems behavior. Journal of the Operational Research Society 41 (6), 461–474.

Pinedo, M.L., 2008. Scheduling: Theory, Algorithms, and Systems, third ed. Springer, New York.

Poli, R., Langdon, W.B., McPhee, N.F., 2008. A Field Guide to Genetic Programming, first ed. Lulu.

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

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

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

Sheskin, D.J., 2007. Handbook of Parametric and Nonparametric Statistical Procedures, fourth ed. Chapman & Hall, CRC, Florida.

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

Uzsoy, R., Lee, C.Y., Martin-Vega, L.A., 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.

Vázquez-Rodríguez, J.A., Ochoa, G., 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., Morton, T.E., 1987. Priority rules for job shops with weighted tardiness costs. Management Science 33 (8), 1035–1047.

Wu, M.-C., Jiang, J.-H., Chang, W.-J., 2008. Scheduling a hybrid MTO/MTS semiconductor fab with machine-dedication features. International Journal of Production Economics 112 (1), 416–426.

Yang, T., Kuo, Y., Cho, C., 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., Liu, M., Wu, C., 2003. Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming. In: The 2003 Congress on Evolutionary Computation, 2003. CEC '03, vol. 2, pp. 1050–1055.