Optimising Machine Selection Rules for Sequence Dependent Setups with an Application to Cartoning [*]

Roman van der Krogt and James Little

Cork Constraint Computation Centre,
Department of Computer Science, University College Cork, Ireland
(email: {roman\j.little}@4c.ucc.ie)

Abstract: Conventional scheduling technology, which tries to optimise performance metrics such as utilisation and makespan, works well in environments where there is a high degree of stability, and hence certainty. However, in uncertain situations, schedules that try to achieve optimality have trouble achieving this target. An often used technique to circumvent the issues with uncertainty that optimal policies display, is the use of *rules* that are triggered when a job enters the system or a machine becomes available. Such rules are by definition reactive, and can thus deal very well with uncertainty.
The downside of these rules is that they make local decisions, which may result in non-optimal behaviour. This becomes especially apparent in light of significant sequence dependent setup times. These are, by nature, dependent upon the global sequence, something that rules cannot deal well with. In this paper, we investigate a method to generate custom machine selection rules that lead to improved setup times. We illustrate the advantages of this new type of rule by presenting an experimental analysis of using these rules at the cartoning department of a large manufacturing company.

Keywords: Dispatch Rules, Machine Selection Rules, Scheduling, Cartoning

1. INTRODUCTION

Manufacturers operate in a highly complex, distributed, and fragmented environment facing unprecedented pressures to cut operating costs, deliver on time, optimize use of available assets, and adhere to regulatory and compliance restrictions. At the same time, manufacturers are facing greater uncertainties than ever before, not only from exogenous sources, such as the economic climate, but also because they are part of a network of organisations, and thus more exposed to issues arising at other companies.

The trade-off between using the available production resources to their full extent, and being able to deal with unexpected changes, is therefore an important consideration for manufacturers. This is illustrated by considering a machine that is scheduled to run at full capacity. This resource is being used very efficiently, but even a small change to the machine will have immediate consequences for the remainder of the schedule. On the other hand, if there is a certain amount of slack between the jobs on this machine, then this could be considered a less efficient schedule, as the machine spends a certain amount of time idling. Minor issues can be quickly overcome, however, by exploiting the flexibility that the slack allows.

An often used technique to circumvent the issues with uncertainty that optimal policies can display, is the use of *scheduling rules*. Typically, these are triggered when a machine becomes available, i.e. when it finishes a job. A *dispatch rule* is then used to select the next job to process on the machine from a buffer of available jobs. One example rule is the so called Shortest Processing Time (SPT) rule, which loads the machine with the job that has shortest processing time on that machine. Dispatch rules can be augmented with *machine selection rules*. In such a setup, we can distinguish two phases. The first phase is triggered when a job is released: the machine selection rule now determines which machine the job is sent to. The second phase is triggered when a machine becomes available. Here, a dispatch rule is used to select a job from among the jobs that are waiting to be processed on it. Two surveys that together describe over a hundred different scheduling rules are presented by Panwalker and Iskander (1977) and Blackstone et al. (1982). However, these also show that none of the rules performs consistently better than the others and so we are left with the question of finding the best rule for a particular situation.

The situation we consider in this paper concerns the cartoning department of a large manufacturing company. This department faces a high degree of uncertainty with regards to what particular jobs will arrive and in which order, making optimal scheduling impossible. A critical aspect of this particular problem is formed by the sequence-dependent setup times that are involved, and so we are looking for scheduling rules that are able to effectively deal with those. Studying the problem, we found that existing rules display unsatisfactory behaviour, even though these rules often were designed with sequence dependent setups in mind. The reason for this, we found, is that batches of compatible types would end up on different machines, as the scheduling rules took too local a view in trying to keep machines occupied. To deal with this situation, we propose to generate rules that take a more global view.

In particular, we propose a machine selection rule that aims to concentrate compatible jobs on the same (set of) machines,

and route incompatible jobs to others. However, this presents us with the problem of selecting which jobs to route to which machines. To solve this, we present an optimisation model that computes the most effective assignments of jobs to machines. Given a particular order profile, we can use this model to compute a machine selection rule that assigns jobs to machines in such a way as to minimise the setup times that are incurred.

Experiments based on real-world data, support our claim. When we compare the combination of our machine selection rule with a First Come, First Served (FCFS) dispatch rule to a combination of the Shortest Queue First (SQF) machine selection rule, and the Similar Setup (SimSet) dispatch rule, we see a reduction in setup times of between 33% and 50%. As a direct result, we obtain a higher throughput rate for our rules.

The remainder of this paper is organised as follows. In the next section we formally introduce scheduling rules and show a method to compute an efficient set of rules. We demonstrate the effectiveness of such rules in Section 3, where we present a case study in which we apply our approach to the cartoning department mentioned before, using simulation to evaluate the rules. Then, we present related work, after which we conclude in Section 5.

2. SCHEDULING RULES

2.1 Definitions

Formally, we recognise a set of job types JT, and a set of resources R. For each job type $jt \in JT$, there is a set of actual jobs $J_{jt} = \{j_{jt,1}, \ldots, j_{jt,n_{jt}}\}$. We denote the type of a job j by $type(j)$, i.e. $j \in J_{type(j)}$. The set of all jobs is denoted by $J = \cup_{jt \in JT} J_{jt}$. Furthermore, we have a set C of constraints over these jobs and resources:

release times for each job $j \in J$ we have a release time $rel : J \to \mathbb{N}_0$;
precedence constraints for two jobs $j \neq j' \in J$, the relation $j \prec j'$ holds when j is to be processed before j'; and
resource restrictions for each job type $jt \in JT$ we have a set of allowed resources $res : JT \to 2^R$. This is straight-forwardly extended to the set of allowed resources for a particular job.

Now we can define machine selection and dispatch rules as follows.

Definition 1. A *dispatch rule* is a function $\delta : 2^J \to J$ that selects one job j from a given set, i.e. $\delta(J') \in J'$.

A dispatch rule is used to select which job to load next onto a resource that has just become idle. Let the set of jobs queuing for resource r be denoted by Q_r. When the resource r becomes idle, we load it with $\delta(Q_r)$. (Of course, it is possible that a machine is idle with no jobs waiting in its queue. In that case, the machine will start processing the first job that arrives in its queue as soon as it is released.) Often, we deal with a situation where all resources select jobs from a common queue Q, i.e. for all resources r, we have $Q_r = Q$. We, however, consider the case where each resource has its own pool of jobs to choose from. In this case, we have to decide which resource to send a job to that has just been released.

Definition 2. A *machine selection rule* is a function $\mu : JT \to R$, where $\forall jt \in JT, \mu(jt) \in res(jt)$. A machine selection rule assigns a resource to a job of a particular type.

The machine selection rule is used when a job is released: it decides to which of the possible resource a job is actually sent to. Usually, the rule is free to choose any of the possible resources $r \in res(jt)$ for a job of type jt. In our work, however, we want to consider only a specific subset of the resources, in order to get a better handle on the setup times involved. We therefore deal with *restricted machine selection rules*.

Definition 3. A *resource restriction function* is a function $\rho : JT \to 2^R$, where for all job types jt it holds that $\rho(jt) \subseteq res(jt)$. A *restricted machine selection rule* $\mu_\rho : JT \to R$ is a machine selection rule such that $\mu_\rho(jt) \in \rho(jt)$.

By definition, a machine selection rule is a restricted machine selection rule μ_{ρ_0} with ρ_0 defined as $\forall jt \in JT \cdot \rho_0(jt) = res(jt)$. However, as we show below, better results may be obtained by restricting the resources that each job type is allowed to go on.

Definition 4. A *scheduling rule* is a tuple $\langle \delta, \mu_\rho \rangle$ where δ is a dispatch rule, and μ_ρ is a (possibly restricted) machine selection function.

Example. To illustrate the different concepts introduced so far, consider the following small example. A small factory has three machines, $m_1, \ldots, m_3$, to produce its products. The products come in three types, $jt_1, \ldots, jt_3$. The processing time of a job j is determined by its type and the machine it runs on, and given by a function $p : JT \times R$. Jobs of type jt_1 have to be processed on m_1, whereas jt_2 and jt_3 can be run on any machine.

- A dispatch rule selects which job to run next. The popular Shortest Processing Time rule for machine m_i is defined as $\delta(Q_{m_i}) = \mathrm{argmin}_{j \in Q_{m_i}} p(j, m_i)$; it selects a job with minimum processing time from the queue Q_{m_i} associated with the machine.
- A machine selection rule is used to route jobs to a particular machine. Let $P_i = \sum_{j \in Q_i} p(j, m_i)$ be the total required processing time of all jobs waiting to be processed on machine i. Then, the Shortest Queue First rule can be stated as:

$$\mu(j) = \begin{cases} m_1 & \text{if } type(j) = jt_1 \\ \mathrm{argmin}_{\{m_1, \ldots, m_3\}} P_i & \text{otherwise} \end{cases}$$

- Imagine that products of type jt_1 are very different from the other types, and that switching between them may lead to large setup times. The company may then decide to dedicate machine m_1 to these products. This leads to a resource restriction function

$$\rho(jt) = \begin{cases} \{m_1\} & \text{if } type(j) = jt_1 \\ \{m_2, m_3\} & \text{otherwise} \end{cases}$$

The machine selection rule then becomes:

$$\mu(j) = \begin{cases} m_1 & \text{if } type(j) = jt_1 \\ \mathrm{argmin}_{m \in \rho(type(j))} P_i & \text{otherwise} \end{cases}$$

In this case, this equals:

$$\mu(j) = \begin{cases} m_1 & \text{if } type(j) = jt_1 \\ \mathrm{argmin}_{\{m_2, m_3\}} P_i & \text{otherwise} \end{cases}$$

Given that we have full knowledge of the future orders, we can construct a schedule that optimally assigns resources and start times to each job. Given such a schedule, the construction of a scheduling rule is straight-forward: each job j is represented

Given
R : the set of resources
JT : the set of job types
$res : JT \rightarrow 2^R$: resource restrictions for the job types
τ : duration of the period we consider
$n \in [1, \| R \|]$: number of resources to use in a rule
$\upsilon : J \rightarrow \mathbb{N}_0$: total volume of jobs of type j in units
$\sigma : R \rightarrow \mathbb{N}_0$: speed of a resource r in units/hour
$\pi : 2^J \rightarrow \mathbb{R}^*$: penalty function

Vars
$b_{jt,r} \in \{0,1\}$: type jt is assigned to resource r
$u_r \in \mathbb{R}^*$: utilisation of resource r
$p_r \in \mathbb{R}^*$: penalty for setups incurred on resource r

Constraints
$$\forall_{jt \in JT : \upsilon(jt) > 0} \sum_{r \in R} b_{jt,r} = n$$
$$\forall_{jt \in JT : \upsilon(jt) = 0} \sum_{r \in R} b_{jt,r} = 0$$
$$\forall_{jt \in JT, r \in R} \left[r \notin res(jt) \Rightarrow b_{jt,r} = 0 \right]$$
$$\forall_{r \in R} \left[u_r = \frac{1}{n} \times \frac{\sum_{jt \in JT} b_{jt,r} \times \upsilon(jt)}{\sigma(r)} \right]$$
$$\forall_{r \in R} \left[u_r \leq \tau \times \sigma(r) \right]$$
$$\forall_{r \in R} \left[p_r = \pi \left(\{ jt \mid b_{jt,r} = 1 \} \right) \right]$$

Minimise
$$\sum_{r \in R} \left| u_r + p_r - \frac{\sum_{r \in R} [u_r + p_r]}{\| R \|} \right|$$

Fig. 1. The optimisation model for the problem. $\| S \|$ represents the size of the set S.

by its own type $jt_j = type(j)$, and we assign $\mu_{\rho_0}(jt_j)$ to the machine that j is scheduled on. Furthermore, if we let $s(j)$ denote the start time that job j is assigned in the schedule, we let $\delta(J) = \operatorname{argmin}_{j \in J} s(j)$. It is clear that this combination of rules exactly mimics the schedule if no unexpected events happen. However, whereas the schedule might be invalidated if a job is not released at its predicted release time, the scheduling rules remains valid.

Notice that the scheduling rules in the example are brittle with respect to machine breakdowns, as μ always outputs the same resource, even when it is not available. It is straightforward to adapt the rule in order to make it more robust. Rather than restricting ourselves to the resource as used in the schedule, the rule could consider other options. However, in this case the schedule rule loses its one-to-one correspondence with schedule, and we have to resort to other methods to generate a scheduling rule.

2.2 Generating Rules

Given the definition of scheduling rules, the question that naturally arises is: what is a good set of rules, and how do

we compute such a set? We have seen that we can generate efficient rules from an optimised schedule, where "efficient" is defined in terms of the optimisation criterion that was used to compute the schedule. Unfortunately, this requires that we build a schedule to start with. In the uncertain environment that we are dealing with (the cartoning department mentioned before), this is impossible. Moreover, we want to generate *robust* rules, whereas the ones derived from a schedule are brittle.

To solve this problem, we have built an optimisation model that will generate resource restriction functions of a particular size. The idea is that we can then use this restriction function to augment an existing scheduling rule (such as the SQF+FCFS rule we used in our experiments). As setup times are an important aspect of our problem, the aim is to obtain a resource restriction function that will lessen the impact of the setups, i.e. lead to fewer change-overs.

To this end, the model features an abstract representation of a schedule, where we are only interested in the *load* that a particular resource is assigned. The load is an abstract value, representing the amount of work that is required, plus an additional amount representing the incurred setup times. The idea is that by minimising the load over the set of resources, we minimise, in effect, the setups that will be incurred when we use the rule at run-time. The resource restriction function can then be used with a standard machine selection rule, such as Shortest Queue First, to implement a full (restricted) machine selection rule.

In order to fully specify the model, we require some additional data:

- the time period τ that we consider;
- the size n of the rules we want to generate, i.e. the cardinality of $\rho(jt)$ for each job type jt;
- the total volume of jobs of each type, given by a function $\upsilon : J \rightarrow \mathbb{N}_0$ (in units to be processed);
- the speed of each machine, given by a function $\sigma : R \rightarrow \mathbb{N}_0$ (in units per hour); and finally
- a penalty function that estimates the setup times involved in running a set of job types on a particular machine, given by a function $\pi : R \times 2^J \rightarrow \mathbb{R}^*$.

The optimisation model that we propose is presented in Figure 1. In the model, we have one boolean decision variable $b_{jt,r}$ for each combination of job type jt, and resource r. We have $b_{jt,r} = 1$ iff (if and only if) resource r is involved in the rule for type jt. Furthermore, we have variables u_r to represent the utilisation of each resource r, as well as a setup penalty p_r, that together make up the load.

We have the following constraints in the model: firstly, we have a number of constraints that ensure we generate appropriate rules: of the required size, and respecting the resource restrictions of the job types. Then, we calculate the utilisation of a particular machine as

$$u_r = \frac{1}{n} \times \frac{\sum_{jt \in JT} b_{jt,r} \times \upsilon(jt)}{\sigma(r)}$$

i.e. we sum the total volume of the job types selected to run on this resource, and divide by the speed of the resource. Since we are generating rules of size n, we assume that only $1/n^{th}$ of the jobs is actually processed on this resource and hence we divide by n.

Thirdly, we also have that the utilisation for resource r cannot be more than $\tau \times \sigma(r)$, as this is the maximum number of jobs that can be processed within the given period. Finally, we calculate the setup penalty p_r for each resource as

$$p_r = \pi \left(\left\{ jt \mid b_{jt,r} = 1 \right\} \right)$$

The optimisation criterion that the model tries to minimise is the following:

$$\sum_{r \in R} \left| u_r + p_r - \frac{\sum_{r \in R} [u_r + p_r]}{\| R \|} \right|$$

That is, we aim to minimise and level the load across the resources, where the load is given by the utilisation u_r and the setup penalty incurred p_r. Since the total utilisation is primarily determined by the total volume of jobs, the effect of this optimisation criterion is to minimise and balance the setup times that are incurred on each resource.

A solution to this model is a set of assignments $b_{jt,r}$. Given these assignments, we can construct a resource restriction function ρ as follows:

$$\rho(jt) = \left\{ r \in R \mid b_{jt,r} = 1 \right\}$$

This can then be combined with a standard machine selection rule to obtain a restricted machine selection rule with the desired properties (in this case, that of limiting the setup times incurred). In the following section we evaluate the rules that this approach generates.

3. CASE STUDY: CARTONING

3.1 Background

The problem we deal with in the current work relates to the operation of a number of cartoning machines in a factory that produces lenses. There are some 15 different types of lenses produced, that are packaged in a total of 25 different packages. For example, lens type t_1 can be bought in a package that contains a single pair of lenses, or one that has 6 pairs of lenses; a certain other type t_2 is only sold in packs of 4. We will refer to the tuple $\langle t, p \rangle$ where t is a type of lens and p is a type of carton simply as the *type* of a batch. (Thus, lenses are produced in batches of the same type.) Each batch has to be packed in similar cartons. The batches range in size from as little as 100 cartons to over 10,000 cartons. There are two types of lenses that require a special cartoning machine (and those machines may only carton those types). For the other lenses, 6 cartoning machines are available. The slowest of these can process over 10,000 pairs of lenses per hour; the faster ones can process over twice that number. Notice that this speed is irrespective of the types of cartons, i.e. the slowest machine can output 10,000 single packs, or about 1,700 six packs. These speeds assume uninterrupted service. However, there are significant setup times involved: changing from one type of *pack* to another (but packing the same type of lens) takes around 15 minutes. However, if the type of *lens* changes, the setup can take up to one hour.

It is obvious that effective sequencing and assignment of the batches is required in order for the machines to be used efficiently. The cartoning department is, however, dependent upon how the upstream processes schedule their manufacturing. Since these processes are more constrained, and involve more expensive machines and tied-up capital in the form of work in progress, these processes are scheduled to ensure *their* efficiency, rather than that of the cartoning department. Add to this the fact that different units produce the different types of lenses with no common schedule across the whole factory and that equally the schedules which are produced are frequently changed. Consequently, it is clear that the cartoning department faces a huge uncertainty when it comes to what types and sizes of batches can be expected in the near future. For this reason, the department prefers to work with scheduling rules. It is imperative, however, that the scheduling rules deal effectively with sequence-dependent setup times, due to the significant factor these setups play in the efficient operation of the factory.

3.2 Experimental Results

As the demands vary throughout the year, we looked at two different data sets to evaluate our work. One covers a week with a work load that is relatively low, whereas the other set covers a work load very close to the capacity of the cartoning department. For both datasets, we had at our disposal distributions of arrival times and volumes of the different package types. From this, we calculated a set of rules per the model of Figure 1 using Ilog OPL Studio. This tool is able to generate initial solutions to our problem within seconds; the results reported here are based on the best solutions found within 5 minutes of CPU time. Having obtained a solution, we used the input data to generate 20 sets of jobs for a fixed period of time, and used Enterprise Dynamics to evaluate our rules on these particular sets of orders by simulation. As the variance in the size of the batches is quite high (as explained, there are batches as small as 100 packs, and some as large as 10,000), the number of jobs that are generated for our period varies greatly too, from one data set to the next. To make the results below easier to read, we have ordered them from smallest number of jobs in the period to the period with the highest number of jobs. The figures are the result of averaging three simulation runs. The variance between these runs is extremely low, as within each experiment there is very little uncertainty (we break ties between queues at random, but this seldom happens).

We compare four different scheduling rules in the results below:

(1) a Round Robin (RR) machine selection rule with the First Come, First Served dispatch function;
(2) a Shortest Queue First (SQF) machine selection rule with the Similar Setup (SimSet) dispatch function;
(3) SQF restricted by the resource restrictions of size $n = 2$, as computed by the model of Figure 1, with the FCFS dispatch function; and
(4) optimal scheduling based on an assumed full knowledge of the arrival times.

Figures 2.a and 2.b. describe the results for when the load is relatively low. As one can observe from the former figure, the SQF rules and our own are both very close to how the scheduling model performs in terms of the number of jobs that are executed within the period. The round-robin strategy achieves a much lower efficiency. This is to be expected, as although it distributes jobs evenly over the machines, it does not take into consideration the size of these jobs, nor any sequence dependent setups. Figure 2.b provides an analysis of the difference between SQF+SimSet and our results. In terms
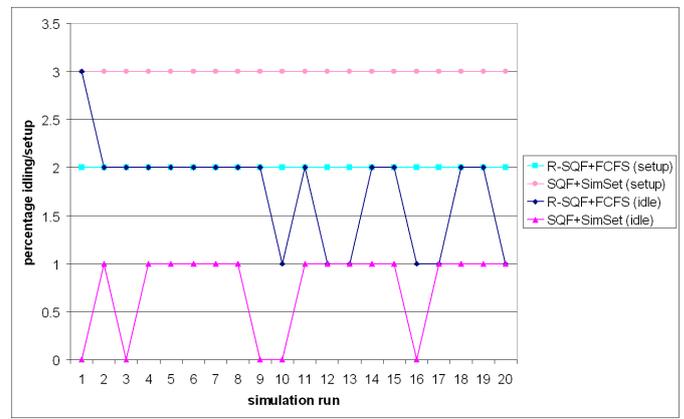
a. Results when load is low – number of batches finished within the period
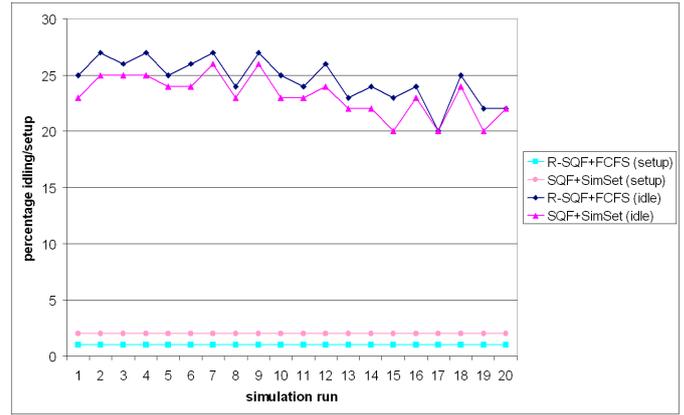
b. Results when load is low – percentages spent idling / setup
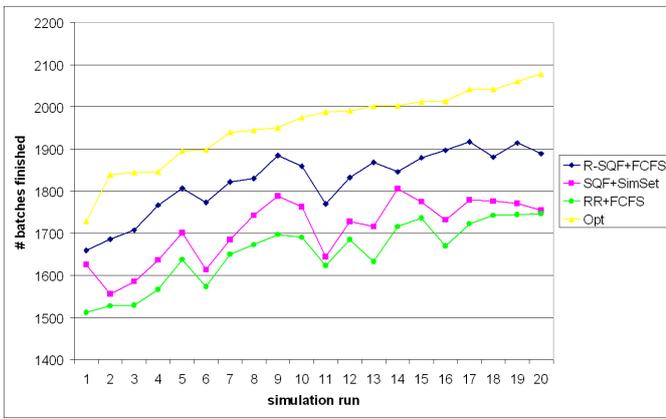
d. Results when load is high – number of batches finished within the period

d. Results when load is low – percentages spent idling/setup

Fig. 2. Results

of the time spent in setups, SQF+SimSet consistently spends around 2% of the total time in setups, whereas our rules lead to an expense of only 1%, *even though our method uses FCFS as a dispatch rule, and SQF uses SimSet.* This is, of course, a surprising result, as the SimSet rule specifically aims to reduce setups. The explanation is simple, however. By using unrestricted SQF, jobs of the same type are spread out over different machines, and there may well be no job of the same type at all in the queue. On the other hand, our restricted rule gathers jobs of similar types on the same machine, and routes jobs of most other types to different machines. The combined effect of this is that it is more likely that two subsequent jobs are of the same type when using our approach, than it is to have one matching job in the queue using SQF+SimSet. As a result, our rules lead to somewhat more idle time, indicating a more efficient use of the available resources. However, the low volume of jobs makes that the difference between the rules is very minor.

The differences become more pronounced when we consider a time period with a higher load, as shown in Figures 2.c and 2.d. Interestingly enough, the percentages of time spent idling and in setups is, again, almost level, at 2% and 3% for our method and SQF+SimSet, respectively. When we look at the number of jobs finished, we can observe the difference this makes. SQF+SimSet performs (on average) at around 87% compared to the scheduling results, whereas our rules obtain

slightly over 93% of the optimal lower bound (again, on average). These results were to be expected from the previous results: SQF+SimSet loses time by performing too much setups, whereas our rules are able to better deal with the changeovers. The scheduling model is able to deal with the setups even better, as it is free to delay jobs to align them with future jobs. However, we would like to stress again that the cartoning department faces a very uncertain situation. Scheduling as we have done here is simply not an option, as it is not known what type of jobs are coming down the line, and at what time they exactly arrive. Achieving a result of 93% is, therefore, very satisfactory.

4. RELATED WORK

Scheduling rules in environments with significant sequence dependent setup times have been previously studied by Kim and Bobrowski (1994). Their study compares four different dispatch rules (including SimSet) on a hypothetical job shop, and concludes that the "setup time must be given an explicit consideration in the scheduling decision when it is sequence-dependent". A more wider study was recently presented by Vinod and Sridharan (2008), who study 12 different rules using a simulated job shop. These papers re-iterate the conclusions of Panwalker and Iskander (1977) and Blackstone et al. (1982) that we mentioned earlier: no rule performs consistently better than any other. Hence, we are left with the question of finding the best rule for a particular situation and a particular optimi-

sation criterion. Below are some of the approaches to deal with this question.

El-Bouri and Shah (2006) present a hybridisation approach that selects different rules for different machines. They generate a number of random jobshop problems (for a single shop layout), and train a neural network to associate a rule for each machine with the particular job mix. Their work can be seen as an extension to that of Liu and Dong (1996), who use a neural network to select a single rule for all machines.

The main contrast between the works mentioned so far and ours, is that the other methods all use a portfolio-approach: from a given, fixed set of rules, the best rule is selected. In contrast, our work is aimed at deriving a particular rule for a given situation. This approach was previously studied by Lee and Pinedo (1997). They present a parameterised dispatch rule, and choose parameters by studying the properties of the problem instance at hand. The main difference with our work is the fact that their work considers a common pool of jobs for all resources, whereas our method is based on the idea of dividing the available jobs intelligently over the available machines. Chiang et al. (2008) propose a dispatch rule that shares aspects of both ours and Lee and Pinedo's approach. Their dispatch rules reason not only about which job is the best for the current machine, but also about the state of the other machines, in order to make a more global decision.

Tay and Ho (2008) also propose an approach that attempts to find a non-standard rule for all machines. They do this by combining several aspects of the jobs in the queue (such as their processing time, and the order in which they were added) in an algebraic expression. A genetic algorithm is used to find the expression that best matches the setup of the tools. Yoshida and Touzaki (1999) employ data mining rules for the same purpose.

5. CONCLUSIONS AND FUTURE WORK

The two main approaches to organising the day-to-day operations of a manufacturing plant are either *(i)* to compute (near-) optimal schedules in advance, or *(ii)* to use scheduling rules to create a schedule on-the-fly as orders arrive. These approaches are nearly opposites of each other: whereas optimisation can generate high-quality solutions, but cannot deal well with uncertainty, scheduling rules deal very well with uncertainty, but are often not close to the (possibly theoretical) optimal solution, as they make only local decisions. In our current work, we are particularly interested in an environment where *(i)* the uncertainty is high, and *(ii)* setup times are an important issue. Neither scheduling nor simulation rules are appropriate in this situation: scheduling is not possible due to the uncertainty involved, and the changeovers call for decisions made with a global view, rather than a local one, as is common in simulation.

To deal with this situation, we introduce *restricted machine selection rules*. These work like normal machine selection rules, selecting a resource to send a newly released job to, but only take into account a restricted set of possible resources when making the decision. We believe that these rules provide a valid alternative that combines the best of both worlds. On the one hand, these rules are reactive, and can thus deal well with uncertainty. On the other hand, we can enforce global properties by crafting them in a particular way. In this paper, we have shown how to optimise a set of rules for minimising setup times. Experimental results, using real-world data from the cartoning department of a large manufacturer, support our approach. The system is able to achieve a 93% efficiency on average (compared to a lower bound on the optimal solution as computed by a scheduling model of the cartoning department), whereas SQF+SimSet achieves only around 87% efficiency. This is achieved by reducing the setups involved by between 33% and 50% in comparison to a combination of the Shortest Queue First (SQF) machine selection rule with the Similar Setup (SimSet) dispatch rule.

Our main interest for the future is to develop a reactive system, which modifies the rules on-the-fly, as significant events happen. Examples of such situations are machines breaking down, or the job mix changing significantly. In these circumstances, the rules are still valid, but have likely become less efficient, and therefore require updating. We are also interested in extending our optimisation model to find resource restriction functions of different sizes for different job types, when appropriate.

Finally, we are planning to explore other global properties that one may want to enforce. For example, minimising buffer times requires a different optimisation model from the one we presented here.

REFERENCES

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

Chiang, T., Shen, Y., and Fu, L. (2008). A new paradigm for rule-based scheduling in the wafer probe centre. *International Journal of Production Research*, 46(15), 4111 – 4133.

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

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

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

Liu, H. and Dong, J. (1996). Dispatching rule selection using artificial neural networks for dynamic planning and scheduling. *Journal of Intelligent Manufacturing*, 7(3), 243–250.

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

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

Vinod, V. and Sridharan, R. (2008). Dynamic job-shop scheduling with sequence-dependent setup times: simulation modeling and analysis. *International Journal of Advanced Manufacturing Technology*, 36, 355–372.

Yoshida, T. and Touzaki, H. (1999). A study on association among dispatching rules in manufacturing scheduling problems. In *Proceedings of the 7th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'99)*, 1355–1360.