Finding Efficient Dispatching Rules using Optimisation and Simulation

Roman van der Krogt and James Little

Cork Constraint Computation Centre
Department of Computer Science
University College Cork, Cork, Ireland
{roman,jlittle}@4c.ucc.ie

Abstract. The COSAR project –Combination of Optimisation and Simulation in the Assessment of Risk– aims to investigate the combination of optimisation and simulation techniques to better deal with uncertainty in manufacturing environments. As a pilot study, we investigated the generation of robust dispatch rules for high-volume cartoning of contact lenses. It is often difficult to determine initially the best set of rules. Here we demonstrate how an initial good set of rules, generated from an optimization model can be improved (manually) through information from a simulation model.

Keywords: Reformulation, Rule Generation, Adaptive Models

1 Introduction

Recently, the Cork Constraint Computation Centre started a research project investigating the Combination of Optimisation and Simulation in the Assessment of Risk (COSAR).[1] The overall aim of this project is to assist manufacturing in anticipating change and being better prepared through improved scheduling. One strand of the research tries to harmonise different techniques surrounding scheduling and the measurement of risk through simulation. In this paper, we present an initial study into how optimisation and simulation can be combined in order to deal with uncertainty in scheduling through producing more robust dispatch rules.

The problem we deal with in the current work concerns the operation of a number of packaging machines in a factory that produces contact 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 packs to over 10,000 packs. There are two types of lenses that require a special packaging machine (and those machines may only package those types). For the other lenses, 6 packaging 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 packaging 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 packaging 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 packaging 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 packaging department prefers to work with simple *dispatching rules*, as any schedule prepared in advance would be quickly broken. Such rules are explained in more detail in the next section, but for now, we want to note that they are usually of the form "use the machine with the nearest completion time", or determined by some other aspect of the state of the machine. For various reasons that go beyond the scope of this paper, however, the packaging department is interested in fixed dispatching rules that do not rely on the states of the available machines. Instead, they would like to couple each particular batch size of each particular type of lens with a number of specific machines. Given a particular batch to process, the operators are to lookup which machines are allowed (based on size and type) and assign the batch to one of those machines. One of the benefits of this particular type of rule is that it possibly limits the setup times involved, as it allows limiting the number of machines that a particular lens type is processed on.

Due to the uncertain order in which the batches arrive, it is difficult to produce initially a good set of rules. Therefore, we have taken an iterative approach: we generate a set of rules using a simplified model of the operations. We then evaluate these rules using a simulation model of the packaging department, using real-life data taking into account the variability in batch sequences. The result of this evaluation is then used to inform and alter the optimisation model to come up with a better set of rules. In our current work, this feedback loop is performed by hand; identifying the undesirable features of the simulation results

and interpreting how the generation model could be improved through additional constraints, changes to the optimisation function, or other changes. We believe that the best model is also subject to change, when demand patterns change significantly. Therefore, the system we are developing is capable of intelligent model re-generation. However, this does not mean we can expect to generate new sets of rules every day as this would place too great a burdon on the packaging staff. We hope to use our experiences to automate this cycle in the future.

The remainder of this paper is organised as follows. First, we provide a brief overview of dispatching rules, followed by a discussion of the optimisation and simulation models in Section 3. We present then our experimental findings, and finish with a discussion.

2   Background

It is well known that scheduling is a notoriously hard problem. Even the common jobshop scheduling model is very difficult to solve, despite its simple description [1]. Dispatching rules form one class of heuristics that has been developed for this problem. Typically, such rules 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 popular 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. Other rules include longest processing time (LPT), first-come first-served (FCFS), and last-come first-served (LCFS). Two surveys that together describe over a hundred different dispatching rules are presented by Panwalker and Iskander [2] and Blackstone et al. [3]. However, they also show that none of these rules performs consistently better than the others and so we are left with the question of finding the best rule for a particular situation. A lot of research is aimed at either finding the right rule for specific conditions, or improving the performance of existing rules by using them in novel ways, e.g. in hybrid rules. While our research can be classified as being of the former type, our approach shares many of the characteristics of the latter type. For this reason we focus on this type of technique in this overview. In particular, we want to draw attention to work on automatically creating hybrid rules.

El-Bouri and Shah [4] present a hybridisation by selecting 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 [5], who use a neural network to select a single rule for all machines.

Tay and Ho [6] propose an approach that is closer to ours, in that it 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 [7] employ data mining rules for the same purpose.

Note that all of the approaches above use rules that take into account the current state of the machines. In contrast, the rules that we were asked to design are state-free. Very little research has been done in this area. The most similar is the work done by Feng et al. [8]. However, their work concerns dispatching jobs to a cluster of (database, www) servers which imply different conditions (e.g. small to negligible setup times).

There is some similarity to constraint generation research here which tries to detect implicit constraints or a reformulation of constraints [9,10,11]. Research in both of these areas is aimed at detecting additional constraints which improve the model. This improvement is usually in terms of solving performance, rather than solution quality.

3   Modelling the Problem

As outlined in the introduction, we have used an optimisation model to compute the rules and used a simulation model to evaluate them. We will first discuss the optimisation model, followed by the simulation one.

3.1   Optimisation Model

On the optimisation side, we decided to represent the above problem as a variation on the bin-packing problem [12]. Intuitively, each bin represents a machine. When we assign a certain type to a machine, we add to the bin an item of a size corresponding to the number of lenses that are involved in all batches of this particular type. Thus, if we expect two batches of 100 six packs of lens type t_1, and we assign $\langle t_1, 6 \rangle$ to machine m_1, we add to the bin associated with m_1 an item of size $2 \times 100 \times 6$. The bins have a fixed size, corresponding to their speed and the duration of the period under consideration. Unlike regular bin-packing, however, we are not interested in using a minimum number of bins; rather, we want to achieve as even a load as possible, i.e. fill the bins as evenly as possible. Given a solution to the bin-packing problem, i.e. a bin assignment, we can straight-forwardly extract the dispatching rules by observing which bin or bins (i.e. machines) are used for each type of package.

A formal description of the problem, cast as a Constraint Programming (CP) model, is given in Figure 1. It was implemented in Ilog OPL Studio 3.7.1. The following parameters are dictated by the actual circumstances. We have a set M of machines, a set T of lens types, a set P of carton types and we consider a period of length τ. Furthermore, we have a function $\#(t, p)$ that tell us how many batches of type $\langle t, p \rangle$ we can expect in the given period, a function $v(p)$ that gives the number of pairs of lenses required to fill a package of type p, and a function $\sigma(m)$ that gives the speed of machine m in pairs of lenses it can process per hour (assuming no setups). Furthermore, we have a function $\pi(n_t, n_{t,p})$ that estimates the cost of the setups required when we process n_t different lens types

$$
\begin{aligned}
\textbf{Given} \quad & M : \text{set of machines} \\
& T : \text{set of lens types} \\
& P : \text{set of carton types} \\
& \tau : \text{duration of the period we consider} \\
& n \in [1, \parallel M \parallel] : \text{number of machines to use in a rule} \\
& \# : T \times P \to [0, \infty) : \text{number of batches of type } \langle t, p \rangle \\
& \upsilon : P \to \langle 0, \infty \rangle : \text{pairs of lenses required for carton type } p \\
& \sigma : M \to \langle 0, \infty \rangle : \text{speed of machine } m \text{ in lenses/hour} \\
& \pi : [0, \parallel T \parallel] \times [0, \parallel T \parallel \times \parallel P \parallel] \to \mathbb{R}^* : \text{penalty function} \\
\textbf{Decision vars} \quad & b_{t,p,m} \in \{0, 1\} : \text{type } \langle t, p \rangle \text{ is assigned to machine } m \\
\textbf{Auxiliary vars} \quad & u_m \in \mathbb{R}^* : \text{utilisation of machine } m \\
& p_m \in \mathbb{R}^* : \text{penalty for setups incurred on machine } m \\
\textbf{Constraints} \quad & \forall_{t,p:\#_{t,p}>0} \sum_{m \in M} b_{t,p,m} = n \\
& \forall_{t,p:\#_{t,p}=0} \sum_{m \in M} b_{t,p,m} = 0 \\
& \forall_{m \in M} \left[u_m = \frac{1}{n} \times \frac{\sum_{t \in T, p \in P} b_{t,p,m} \times \#(t,p) \times \upsilon(p)}{\sigma(m)} \right] \\
& \forall_{m \in M} \left[u_m \leq \tau \times \sigma(m) \right] \\
& \forall_{m \in M} \left[p_m = \pi \left(\sum_{t \in T} [\exists p \in P \cdot b_{t,p,m} = 1], \sum_{t \in T} \sum_{p \in P} [b_{t,p,m} = 1] \right) \right] \\
\textbf{Minimise} \quad & \sum_{m \in M} \left| u_m - \frac{\sum_{m \in M} [u_m + p_m]}{\parallel M \parallel} \right|
\end{aligned}
$$

Fig. 1. The bin-packing model of the problem. $\parallel S \parallel$ represents the size of the set S.

on a machine, and $n_{t,p}$ different types of packages. Finally, we have a parameter n that describes how many machines should be involved in a rule, i.e. if $n = 2$, all rules are of the form "given a batch of type $\langle t, p \rangle$ process it either on machine m_1 or on machine m_2." We assume that each possible choice is actually chosen in roughly $1/n^{th}$ of the cases. Hence, when we consider a batch with a weight of w (reflecting both the size and the number of these batches), we add an item of size $\frac{s}{n}$ to each bin.

In the problem, we have one boolean decision variable $b_{t,p,m}$ for each combination of lens type t, carton type p and machine m. We have $b_{t,p,m} = 1$ iff machine m is involved in the rule for type $\langle t, p \rangle$. From the assignments of values to $b_{t,p,m}$ we calculate a utilisation u_m for each machine m, as well as a setup
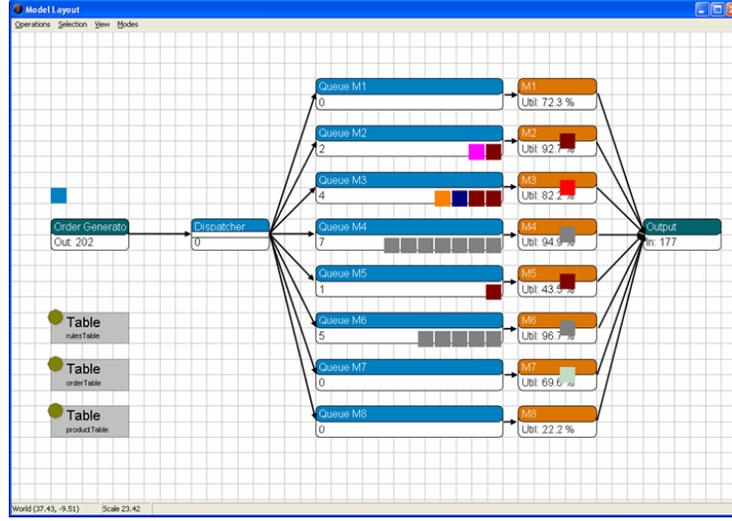
Fig. 2. The simulation model in action. M1 and M8 are machines dedicated to particular lens types. M2 through M7 are available to process the other lenses. Each batch in the system is represented by a cube. Each batch type is represented by a unique colour. The size of a particular batch (i.e. how many cartons are in the batch) is not graphically represented.

penalty p_m. We have the following constraints: firstly, we require n machines to be involved in the rule for a type $\langle t, p \rangle$ if there are any batches of that type, i.e. $\sum_{m \in M} b_{t,p,m} = n$. We calculate the utilisation of a particular machine as $u_m = \frac{1}{n} \times \frac{\sum_{t \in T, p \in P} b_{t,p,m} \times \#(t,p) \times \upsilon(p)}{\sigma(m)}$, i.e. we multiply the number of batches assigned to this machine by their size and divide by the speed of this machine. Since we divide the batches over n machines, we only take into account $1/n^{th}$ of this for machine m, as explained above. We also have that the utilisation cannot be more than $\tau \times \sigma(m)$, as this is the maximum number of lenses that can be processed within the given period. Finally, we calculate the setup penalty p_m for each machine as $\pi \left(\sum_{t \in T} \left[\exists p \in P \cdot b_{t,p,m} = 1 \right], \sum_{t \in T} \sum_{p \in P} \left[b_{t,p,m} = 1 \right] \right)$, where π is the penalty function, $\sum_{t \in T} \left[\exists p \in P \cdot b_{t,p,m} = 1 \right]$ gives us the number of lens types on a given machine, and $\sum_{t \in T} \sum_{p \in P} \left[b_{t,p,m} = 1 \right]$ gives us the total number of different types.

3.2 Simulation Model

The simulation model was built in Enterprise Dynamics 7.2. A screenshot of the model in action is shown in Figure 2. On the lefthand side is the order generator. This produces a random sequence of batches, according to given parameters. Each batch has a given size (in number of cartons) and a type of lens. The dis-

patcher then takes control over the batch and dispatches it to a queue associated with a particular machine, based on the current dispatching rules in effect. The machines process the batches in their queue on a first come, first served (FCFS) basis, incurring the associated setups when switching between types of cartons or types of lenses. After the batch has been packaged, it is sent to a store, labeled 'output' in the figure.

The model has been used in two ways. Firstly, when we generated a set of rules, we ran the simulation model in an interactive mode that allowed us to observe how the rules behave (cf. Figure 2). Any obvious problems with the rules (such as, e.g., a rule overloading a particular group of machines) could easily be observed this way. When a set of rules did not exhibit any obvious problems, we evaluated in in experimental mode. In this mode, the model is run multiple times and overall results are computed, without any feedback during the computation. After a specified number of runs, the specified key performance indicators (KPI) are presented, including their mean, standard deviation and minimum and maximum figures obtained. The KPI include the utilisation levels of the machines, times spent in buffers and overall output. We use these numbers to compare the different rules, as discussed in the next section.

4 Experimental Results

As explained before, we used the simulation model built in Enterprise Dynamics to explore different rule sets produced by the optimisation model of Figure 1. Some of the results we found are presented in Figures 3 through 5. Each of the figures shows some of the evaluation criteria of the best rule found using the particular model: the average amount of time a machine spends idling, busy (i.e. packaging), and being set up; and the throughput of the packaging department.[2] The former figures are averaged over the six machines that are available for general use, whereas the throughput includes all machines, i.e. both the specialised machines and the general purpose ones. The figures were obtained by averaging over 5 simulation runs, where each run observed a 168 hour period (i.e. a week). We note that we were primarily interested in rules mentioning of 2 or 3 machines during our search for a good model, as the company felt (and we verified) that this provides a good trade-off between the versatility of the rules and the potential setup costs incurred. For the purpose of this paper we also generated rules of greater length.

Figures 3 and 4 show a model that aims to even the load of each machine (i.e. the level to which each bin is filled). The model of the former figure does so by minimising the average deviation from the average load; the latter model minimises the maximum deviation. When we look at the rules, we see that, in general, the more machines a rule has, the more time is spent in setups. This is to be expected, when we realise that a machine is more likely to see more batches of different types, because the batches are more spread out, rather than
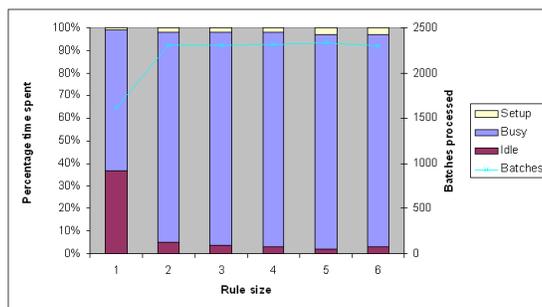
---

[2] We set a maximum of 120 CPU seconds to compute a set of rules. For some models, this was enough to prove optimality; however, for the vast majority it was not.
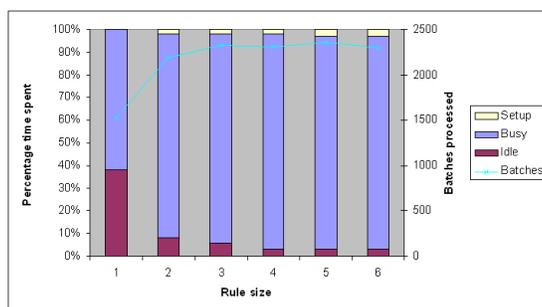
Fig. 3. Results for different numbers of machines involved in a rule. Minimising
$\sum_{m \in M} \left| u_m - \frac{\sum_{m \in M} [u_m + p_m]}{\|M\|} \right|$
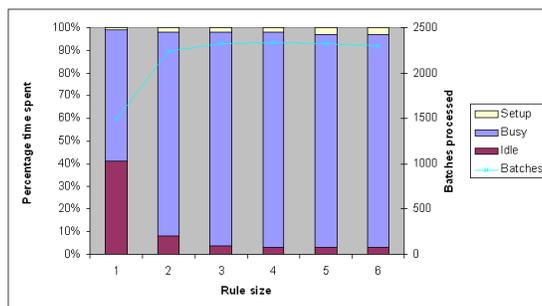
Fig. 4. Results for different numbers of machines involved in a rule. Minimising
$\max_{m \in M} \left| u_m - \frac{\sum_{m \in M} [u_m + p_m]}{\|M\|} \right|$

Fig. 5. Results for different numbers of machines involved in a rule. Minimising
$\max_{m \in M} \left| u_m - \frac{\sum_{m \in M} [u_m + p_m]}{\|M\|} \right| + 5 \times \max_{m \in M} \sum_{t \in T} [\exists p \in P \cdot b_{t,p,m} = 1]$

focused on a few machines. We also see that the rules that use only one or two machines package fewer batches than the rules with more machines (that differ minimally). The reason for this is that these rules have only limited choice when it comes to assigning a machine for a batch. Consider a one machine rule. Since the queues of the machines have only limited capacity, it is possible that the queue corresponding to the machine that the next batch should be dispatched to is full. In such a case, the batch has to wait at the dispatcher until the queue frees up. This, in turn, blocks the dispatching of subsequent batches.

This lead us to models that also take into account the number of different types and subtypes assigned to each machine. Through trial-and-error with different penalty functions, we found that

$$\max_{m \in M} \left| u_m - \frac{\sum_{m \in M} [u_m + p_m]}{\| M \|} \right| + 5 \times \max_{m \in M} \sum_{t \in T} [\exists p \in P \cdot b_{t,p,m} = 1]$$

as our minimisation criterion provided the best balance between penalty and evening the load. Notice that this does not take into account the fact that some types are less different than others (i.e. the package type differs for the same type of lens); it appears that the dominating factor is simply the number of different types. The effect of this change can be observed in Figure 5. Here, we still have a relatively low throughput for the set of rules involving a single machine, although it performs better than the corresponding rule sets in the other models. The true benefit of taking into account a penalty for the number of types on a machine is shown for the two machine rule set, however. This now performs on a par with the larger rule sets (i.e. involving more machines). In fact, we see that this rule set now neatly balances the aspect of spreading the workload over the different machines and that of using only few machines for each type of batch (to minimise the setups): it spends 1-2 percentage points more time idling, while processing roughly the same number of batches as the larger rule sets. This indicates a more efficient use of the available machines.

5   Discussion

This paper discussed a way to generate dispatching rules to govern the operation of a number of packaging machines in a factory that produces contact lenses. The manufacturer was interested in dispatching rules of a particular form: for each type of carton and batch size, we are to generate a set of machines that the particular batch may go on. On the factory floor, operators are to choose from the set of machines which machine to route the batch to.

We showed how one can use an optimisation model to generate the rules and evaluate them using a simulation model of the actual environment. The benefit of this setup is that we can use a more abstract optimisation model that is easier to solve than a model that takes into account the intricacies of the problem. Moreover, the optimisation model does not have to deal with uncertainty, as this is taken into account in the simulated evaluation of the rules. Our experimental

results validate our approach: we were able to generate efficient rules in a few iterations. We have manually identified features in the simulation results which prompted additions to the rules and consequently changes to the initial optimization model. This provides an opportunity for self-learning optimisation models in cases where conditions change significantly over time, so that one model is not enough or modelling experience is not comprehensive. It would allow, for example, seamlessly generating new rule sets when demand profiles change, or new machines are added to the department.

For the future, we believe that the features which led us to reformulate the optimization model can be measured and quantified for detection. In this way the process can be self-learning to develop the best set of rules – generated by a CP model, evaluated by simulation and refined by the COSAR system.

References

1. Nowicki, E., Smutnicki, C.: An advanced tabu search algorithm for the job shop problem. Journal of Scheduling **8** (2005) 145–159
2. Panwalker, S., Iskander, W.: A survey of scheduling rules. Operation Research **25** (1977) 45–61
3. Blackstone, J., Philips, D., Hogg, G.: A state-of-the-art survey of dispatching rules for manufacturing job shop operations. International Journal of Production Research **20** (1982) 27–45
4. El-Bouri, A., Shah, P.: A neural network for dispatching rule selection in a job shop. The International Journal of Advanced Manufacturing Technology **31** (2006) 342–349
5. Liu, H., Dong, J.: Dispatching rule selection using artificial neural networks for dynamic planning and scheduling. Journal of Intelligent Manufacturing **7** (1996) 243–250
6. Tay, J.C., Ho, N.B.: Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Computers and Industrial Engineering **54** (2008) 453–473
7. Yoshida, T., Touzaki, H.: 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). (1999) 1355–1360
8. Feng, H., Misra, V., Rubenstein, D.: Optimal state-free, size-aware dispatching for heterogeneous M/G/-type systems. Performance Evaluation **62** (2005) 475–492
9. Frisch, A., Miguel, I., Walsh, T.: Symmetry and implied constraints in the steel mill slab design problem. In: Proceedings of the CP'01 Workshop on Modelling and Problem Formulation. (2001) 8–15
10. Little, J., Gebruers, C., Bridge, D., Freuder, E.: Capturing constraint programming experience: A case-based approach. In: Proceedings of the CP'02 International Workshop on Reformulating Constraint Satisfaction Problems. (2002)
11. Law, Y.C., Lee, J.H., Smith, B.M.: Automatic generation of redundant models for permutation constraint satisfaction problems. Constraints **12** (2007) 469–505
12. Garey, M., Johnson, D.: Computers and intractability – a guide to the theory of NP-completeness. W.H. Freeman and company, New York, NY (1979)