Expert Systems With Applications 62 (2016) 131–147

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Correlation of job-shop scheduling problem features with scheduling efficiency

Sadegh Mirshekarian, Dušan N. Šormaz*

Department of Industrial and Systems Engineering, Ohio University, Athens, Ohio, USA

ARTICLE INFO

Article history:
Received 26 August 2015
Revised 8 June 2016
Accepted 8 June 2016
Available online 9 June 2016

Keywords:
Job-shop scheduling
Scheduling efficiency
Makespan prediction
Machine learning
Support vector machines

ABSTRACT

In this paper, we conduct a statistical study of the relationship between Job-Shop Scheduling Problem (JSSP) features and optimal makespan. To this end, a set of 380 mostly novel features, each representing a certain problem characteristic, are manually developed for the JSSP. We then establish the correlation of these features with optimal makespan through statistical analysis measures commonly used in machine learning, such as the Pearson Correlation Coefficient, and as a way to verify that the features capture most of the existing correlation, we further use them to develop machine learning models that attempt to predict the optimal makespan without actually solving a given instance. The prediction is done as classification of instances into coarse lower or higher-than-average classes. The results, which constitute cross-validation and test accuracy measures of around 80% on a set of 15,000 randomly generated problem instances, are reported and discussed. We argue that given the obtained correlation information, a human expert can earn insight into the JSSP structure, and consequently design better instances, design better heuristic or hyper-heuristics, design better benchmark instances, and in general make better decisions and perform better-informed trade-offs in various stages of the scheduling process. To support this idea, we also demonstrate how useful the obtained insight can be through a real-world application.

1. Introduction

The Job-Shop Scheduling Problem (JSSP) is one of the most difficult combinatorial optimization problems considered (Lawler, Lenstra, Rinnooy Kan, & Shmoys, 1993). Lenstra and Rinnooy Kan (1979) categorize it as NP-hard, while many of its variations have been proven to be NP-complete (Garey, Johnson, & Sethi, 1976; Gonzalez & Sahni, 1978; Lenstra, Rinnooy Kan, & Brucker, 1977). Different solution techniques including exact methods, heuristics, estimation methods and metaheuristics have been suggested for each JSSP variation (Błażewicz, Domschke, & Pesch, 1996; Hochbaum & Shmoys, 1987; Jain & Meeran, 1999; Pinedo, 2008). The variation on which we focus in this paper is the conventional JSSP with no sequence-dependent setup times, no due dates, no operation preemption, one operation per machine at a time, one machine per job at a time and one operation per machine for each job. However, most of the ideas and practices introduced in this paper can be used for other variations as well.

Analytical solution methods can quickly lose their applicability as problem size increases (Aytug, Bhattacharyya, Koehler, & Snowdon, 1994), and even very fast techniques for moderately-sized shops may not be useful in a dynamic real-life environment where changes in processes and machines are the order of the day. For this reason, Operation Research (OR) practitioners resort to *dispatching rules* or heuristics to solve practical-sized instances in reasonable time. Blackstone, Phillips, and Hogg (1982) provide a survey of such heuristics, while Aytug et al. (1994) noted that even though some dispatching rules give reasonable results for some problem instances, it is difficult to predict when or for what type of instances they give good results. On the other hand, Thesen and Lei (1986) observed that expert human intervention and adjustment of these heuristics can often improve their performance. Since being an expert implies having the necessary insight into problem structure and the pathways that exist between problem configuration and a desired output, we believe there is a justified need to research into the ways of improving that insight and discovering more pathways. In effect, in this paper, we do not attempt to design new heuristics or directly suggest improvements over the old ones. We instead investigate the use of inductive machine learning techniques in evaluating the relationship between various problem features and its optimal makespan, in order to provide more insight for the expert practitioners to use.

Aytug et al. (1994) present a preliminary review of the use of machine learning in scheduling. They mention such AI methods as expert systems, which were the earliest attempts made to incorporate intelligence in scheduling. Such systems however, relying

* Corresponding author.
E-mail addresses: sm774113@ohio.edu (S. Mirshekarian), sormaz@ohio.edu (D.N. Šormaz).

http://dx.doi.org/10.1016/j.eswa.2016.06.014
0957-4174/

Abbreviations and symbols

OPT	Operation processing time
JPT	Job processing time
MPT	Machine processing time
OSPT	Operation slot processing time
OSMM	Operation slot missing machines
OSRM	Operation slot repeated machines
OSRMA	OSRM amplified
OSCOMB	Operation slot repeated machines combined with processing times
OSCOMBA	OSCOMB amplified
MLDU	Machine load uniformity
MLDV	Machine load voids
MLDVA	MLDV amplified
STD	Standard deviation
JCT	Job completions time
MCT	Machine completion time
PCC	Pearson correlation coefficient
SNR	Signal-to-noise ratio
n	Number of jobs
m	Number of machines
P	Matrix of processing times
M	Matrix of machine allocations
S	Solution schedule
Φ	Feature vector
L	Label (for supervised machine learning)
F	Number of features
φ_f	Feature f in the feature vector
l_i	Total idle time of machine i
p_{jk}	Processing time of operation k of job j
m_{jk}	Machine allocated to operation k of job j
s_{it}	tth operation of machine i in a schedule
C	Makespan
C_{min}	Optimal makespan
C'	Normalized makespan
C'_{min}	Normalized optimal makespan
SPT	'Shortest processing time' heuristic
LPT	'Longest processing time' heuristic
MWRM	'Most work remaining' heuristic
LWRM	'Least work remaining' heuristic
FIFO_MWRM	'First-in-first-out with least work remaining conflict resolution' heuristic

heavily on the wit of the human expert, were criticized for not being effective for most dynamic environments. Other AI methods such as various "search techniques" followed to fill the gap, but they were not adaptive and were very slow. Machine learning was the latest technique to be used, and it overcame many of the early problems, by introducing adaptability and versatility without adding too much computation. Aytug et al. (1994) argue for the importance of automated knowledge acquisition and learning in scheduling, and cite researchers like Yih (1990) and Fox and Smith (1984) to support this premise.

Because most research on scheduling has been on designing and improving techniques that solve a given problem instance and find an optimal or near-optimal schedule in terms of a certain cost function such as makespan, most machine learning research has also followed the same lines and has been around automating and improving this process. Adaptive heuristics and adaptive hyper-heuristics are perhaps the two main categories of machine learning research on scheduling. In the first category, the aim is to use machine learning models (and models obtained through other AI methods like rule-based systems, evolutionary algorithms, etc.) 'as a heuristic' that is more adaptive than conventional dispatching rules and can incorporate more knowledge into the scheduling process. See for example Lee, Piramuthu, and Tsai (1997) who combined decision trees and genetic algorithms to develop adaptive schedulers, Zhang and Rose (2013) who used artificial intelligence techniques to develop an individual dispatcher for each machine, or Li, Zijin, Jiacheng, and Fei (2013) who developed an adaptive dispatching rule for a semiconductor manufacturing system. In the second category which is more active in recent literature, the focus is on designing models that can help 'design heuristics' or help choose the best dispatching rule for a given system state. See Branke, Nguyen, Pickardt, and Zhang (2015) and Burke et al. (2013) for a recent review, Nguyen, Zhang, Johnston, and Tan (2013) and Nguyen et al (2013b) who used genetic programming to discover new dispatching rules, Pickardt (2013) who used evolutionary algorithms to automate the design of dispatching rules for dynamic complex scheduling problems, Olaffson and Li (2010) and Li and Olaffson (2005) who used data mining and decision trees to learn new dispatching rules, and also Priore, de la Fuente, Gomez, and Puente (2001), Priore, de la Fuente, Puente, and Parreño (2006) and Burke, MacCarthy, Petrovic, and Qu (2003) for other applications.

Despite the existence of smarter scheduling algorithms, many practitioners still use their own expertise to make the final decision or make a decision in critical conditions. Their expertise usually comes from experience, and little theoretical work is done in the literature to support the understanding of scheduling problem structure in a meaningful and easy-to-relate way. Smith-Miles, James, Giffin, and Tu (2009) and Smith-Miles, van Hemert, and Lim (2010) investigated the use of data mining to understand the relationship between scheduling and the travelling salesman problem[1] structure and heuristic performance, while Ingimundardottir and Runarsson (2012) used machine learning to understand why and predict when a particular JSSP instance is easy or difficult for certain heuristics. We believe such work is valuable, because without a practical understanding of problem structure, moving towards a goal of better scheduling practice might just be a tedious campaign of trial-and-error. Practitioners also need supporting insight in a less cryptic and more palpable form. We try to achieve this in our paper, by developing a set of descriptive features that characterize a job-shop scheduling problem, and establishing their correlation with the optimal makespan.

Another aspect of scheduling that is not often addressed in the literature is at the shop design level, where the JSSP instance to be solved is actually defined. Researchers usually neglect the fact that JSSP instances are not always designed without any flexibility. Sometimes there is the option of choosing different machines for certain jobs (as for the example application given in Section 5), or even more commonly, the option of ordering job operations differently. Current research mostly supports the "solution" of a given JSSP instance, and so the shop design practitioners' only options are to trust their own insight and/or to run approximation algorithms repeatedly to find the best option. Since the practices of shop design and shop scheduling are inherently entwined, there is a justifiable need for work that can support both. The only work we are aware of that addresses these practices simultaneously is Mirshekarian and Šormaz (2015), and this paper is a continuation of that work.

The type of characterizing features that we define and evaluate in this paper, can support both practices (shop design and shop scheduling). For example, intuition can tell us that if one job has a much higher total processing time than other jobs, or more

[1] The travelling salesman problem is a special case of JSSP when the number of machines is 1 and there are sequence-dependent setup times.

generally, if the standard deviation of the total processing time of jobs is high, the optimal schedule may not use the resources as efficiently as possible (i.e., one job may dominate the makespan). We define a feature representing the standard deviation of job processing times, and assess it to see whether this intuition is actually correct. If so, the shop design practitioner can use this insight and design a more efficient shop, for example by avoiding uneven job processing times. The advantage of this approach compared to running approximation algorithms is the increased insight that can help avoid bad configurations without even trying them. This may save significant design time. The disadvantage is that the set of features might be incomplete, or may have complex correlation patterns that confuse the designer into making undesirable decisions, whereas a good approximator would give the end result regardless. We develop 380 features in this paper, which in our experiments capture a significant amount of the existing correlation. As other application examples, the features can also help hyper-heuristic design or help choose better heuristics. Another use would be for benchmark design. Benchmark JSSP instances are usually generated randomly (see Demirkol, Mehta, & Uzsoy, 1998 or Taillard, 1993), but by using the features we develop in this paper, we would be able to use our knowledge of high-impact features to design benchmark cases with a special distribution of certain characteristics like optimal makespan.

In summary, the main contributions of this work are the developed set of features, plus the proposed machine learning approach to enable better design decisions. Also, the features and the approach can work together as a heuristic that approximates the optimal makespan. The paper is written in six sections. First we explain the methodology we used, clarifying the definitions and assumptions, especially for the label that we use for supervised machine learning. Then we define the features and describe them in detail. Section 4 covers the experiments and results using those features. We did both single-feature analysis and feature set classification, and the results are presented in order with discussion on their possible significance and meaning. Section 5 gives an example application of the feature correlation insight presented by the paper. The last section summarizes the conclusions and suggests routes for improvements. To save space, supplementary explanations for concepts like job-shop scheduling, supervised machine learning and support vector machines are only given minimally. There are three appendices at the end in which detailed results and the source code for our optimization software is presented.

2. Methodology

In this section we explain the format used to represent problem input, the format used for feature representation of a problem instance, the approach used to define and evaluate features, and the specific choice of label for supervised machine learning.

2.1. Problem representation

We employ the format used by Taillard (1993), which is in accordance with the disjunctive graph model introduced by Błażewicz, Pesch, and Sterna (2000). In this format, a JSSP instance x is given in the form of two n-by-m matrices, the processing time matrix P and the machine allocation matrix M, where n is the number of jobs and m the number of machines. Every element p_{jk} in P represents the processing time of the k^{th} operation of job j, while the corresponding element m_{jk} in M represents the machine responsible for that operation. We assume that the number of operations of each machine is exactly the same as the number of machines m, and that if a job does not required a certain machine; its corresponding processing time is set to zero. Note that the order of operations for each job is encoded in the order of appearance of its elements in both P and M. A schedule, which is a solution to the JSSP problem instance, is given by an m-by-n matrix S, in which every element s_{it} represents the operation of job $j = s_{it}$ for which machine i is responsible. The processing time of this operation is given by p_{jk} such that $m_{\text{jk}} = i$.

Having the data for a JSSP instance x, we can represent it as a pair (Φ, L), in which Φ is the set of F features $\{\varphi_1, \varphi_2, \ldots, \varphi_F\}$ that incorporate the most important characteristics of x, and L is a special target feature called the label. A supervised machine learning problem is then defined as using the (Φ, L) values of a set of training instances to build a model that can predict the value of L for some test instance y, knowing only the values of its F features. It is a classification problem if the label is discrete and a regression problem otherwise. In any case, one central and often challenging task in designing a machine learning framework for a label of interest is defining a good set of features. A good set of features is one that has a high label prediction power or a high *correlation* with the label. The most reliable way of measuring this correlation is through building the actual model using a machine learning algorithms such as SVM, but to measure the correlation of 'individual' features with the label; several other methods have been suggested in the literature. Overall, methods of measuring the correlation of individual features with the label are grouped into two categories: *filter methods* that measure a statistical correlation without regard to a specific machine learning algorithm, and *wrapper methods* that measure the prediction power using a specific machine learning algorithm. We employed both of these methods in this research, and for the latter we used SVM as the algorithm.

2.2. Feature design and evaluation

The common approach for defining (or designing) features involves a considerable amount of trial and error. A set of preliminary features are first defined, and then ranking methods (filter or wrapper methods mentioned above) are used to measure the prediction power of those features. The problem with this approach is that some features may have high prediction power only when considered together with other features, not individually. This means that a certain feature cannot be dismissed as not useful unless all of its combinations with other features are tested. However, this sweeping of all combinations is computationally expensive and intractable for most important problems, and therefore several alternative selection schemes have been suggested to overcome this problem (see McCallum, 2003 or Zhang, 2009 for example). In this paper, we do not sweep combinations or use such schemes. Instead, we consider only individual features and a selected set of feature groups.

When considering individual features, we used the Pearson Correlation Coefficient (PCC), Signal-to-Noise Ratio (SNR) and a T-test evaluation as filter methods to assess their correlation with the label. PCC is a measure of the strength of linear dependence (correlation) between two continuous variables and has values in the range $[-1, 1]$. A PCC of zero means there is no linear correlation between the two variables, while a PCC of 1 or -1 means there is full direct or opposite linear correlation. If X and Y are two sets of samples taken from the two variables, the PCC of those variables can be calculated by Eq. (1). For our problem, X is an individual feature $\varphi_f \in \mathbb{R}$ and Y is the label $L \in \mathbb{R}$, both sampled from a set of JSSP instances (the choice made for the labael is discussed in Section 2.3).

$$PCC(X,Y) = \frac{\sum_i (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_i (X_i - \bar{X})^2}\sqrt{(\sum_i (Y_i - \bar{Y})^2)}} \tag{1}$$

Table 1
Sample JSSP instance and its optimal solution.

j	P							M			*i*			S		
0	96	70	6	58	86	2	0	1	4	3	*0*	0	1	2	3	4
1	37	26	23	14	34	2	1	0	3	4	*1*	2	4	1	0	3
2	21	83	29	66	25	2	1	4	3	0	*2*	2	0	1	4	3
3	18	73	28	29	89	4	1	2	0	3	*3*	2	4	1	0	3
4	87	41	32	77	77	4	1	2	3	0	*4*	4	2	3	0	1

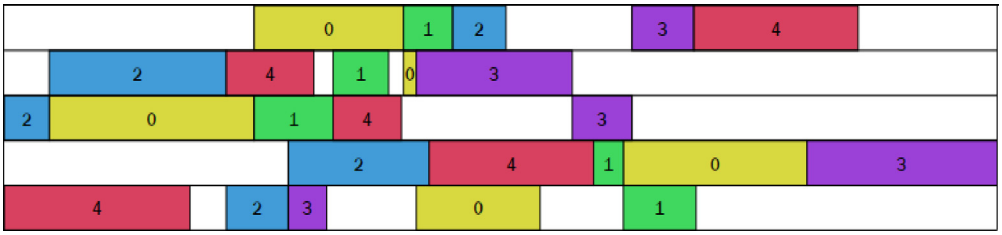

Fig. 1. Optimal schedule for the sample instance (each row of blocks is for a machine).

In case the label is binary, the SNR and T-test measures can be calculated by Eqs. (2) and (3), in which the subscripts + and – correspond to variables associated with instances with positive and negative labels respectively. As such, μ_+, σ_+ and n_+ are the mean, standard deviation (STD) and the number of instances that have a positive label.

$$SNR(X,\ Y) = \frac{|\mu_+ - \mu_-|}{\sigma_+ + \sigma_-} \tag{2}$$

$$T(X,\ Y) = \frac{\mu_+ - \mu_-}{\sqrt{\frac{\sigma_+}{n_+} + \frac{\sigma_-}{n_-}}} \tag{3}$$

These filter methods are good for measuring correlation, but they are limited to only one feature at a time, and they may not provide the best prediction power we can get from a specific machine learning algorithm. We can overcome these problems by using wrapper methods, at the cost of more computation. In this paper, we trained a SVM as the wrapper algorithm, using a quadratic kernel when considering individual features alone, and a Gaussian kernel when considering a combination of features. These kernels are chosen for their relative performance when different numbers of features are used. In our experiments, a quadratic kernel outperformed a fine-tuned Gaussian kernel when individual feature were considered, and it was the opposite when multiple features were considered.

2.3. The label

As mentioned before, the label is a special target characteristic of the JSSP instance which we are interested in predicting. The optimal makespan C_{min} (defined as the least time it takes for all operations to be processed) is one of the most important of JSSP characteristics. Consider the sample JSSP instance provided in Table 1, along with its optimal schedule (and $C_{min} = 465$) which was obtained using a conventional constraint programming software[2]. In the provided data, job and machine indices start from zero. A visualization of the optimal schedule including is shown in Fig. 1.[3]

If we take C_{min} itself as the label, the machine learning problem becomes rather trivial, because problems with higher total processing time are very likely to have a higher C_{min} as well. This issue will be more serious when problem size (values of n and m) changes. Between two instances with the same total processing time, the one which has a higher ratio of n/m will have a higher C_{min}. Such trivial correlations can overshadow the much fainter correlations that we wish to uncover. To deal with this problem, Mirshekarian and Šormaz (2015) propose the scheduling efficiency metric given in Eq. (4) as the label. This metric successfully considers the effect of total processing time and number of machines on the relative optimal makespan of JSSP instances, while maintaining the direct effect of makespan.

$$C' = 1 + \frac{\sum_i l_i}{\sum_{j,k} p_{jk}} \tag{4}$$

In Eq. (4), $\sum_{j,k} p_{jk}$ is the total processing time of all jobs and l_i is the total idle time of machine i. Note that for a given schedule, the value of C' is 1 if it has no idle time (makes perfect use of all of the machines), and that C' increases as the amount of idle time increases. Now consider Eq. (5):

$$\sum_{j,k} p_{jk} + \sum_i l_i = C.m \tag{5}$$

This equation is easy to verify by inspection from the sample schedule given in Fig. 1. $\sum_i l_i$ is the sum of all the white spaces (machine idle times), and by adding this to the sum of all processing times, we get the area of the rectangle that has height m, width C and area $C.m$ (assuming unit height for each block). Dividing both sides of Eq. (5) by $\sum_{j,k} p_{jk}$ we get:

$$1 + \frac{\sum_i l_i}{\sum_{j,k} p_{jk}} = \frac{C.m}{\sum_{j,k} p_{jk}} \tag{6}$$

The LHS of this equation is C' as given by Eq. (4), which gives us the following result:

$$C' = \frac{C.m}{\sum_{j,k} p_{jk}} \tag{7}$$

This equation shows that C' is in fact C normalized w.r.t the total processing time and number of machines. This normalized makespan metric is ideal for comparing problem instances with different size, which in turn makes it ideal as the label for the purposes of this paper. To make the label binary for classification, we take as reference the average of the class of problems we are interested, and label an instance positive if it has an optimal C' higher than this average and negative otherwise. Note that a higher C' is equivalent to a lower scheduling efficiency and vice versa. Since C' is comparable across instances of different size, we can have a

[2] We used a modified constraint programming code written in IBM ILOG OPL. See Appendix A for the code.

[3] The visualization is produced using an online tool (Mirshekarian, 2015).

fully heterogeneous class. Additionally, note that for a given problem instance, having C'_{min} gives us the makespan of the optimal solution C_{min}. This is also valuable, because if we can design a machine learning model to predict C'_{min}, we will be able to predict the optimal makespan itself without actually calculating the optimal schedule. As Mirshekarian and Šormaz (2015) argue, this can help design better heuristics or have a better insight into the correlation of problem features with optimal makespan.

3. JSSP features

Defining the right set of features is a central task in supervised machine learning. The process of feature engineering is iterative and usually involves a significant amount of trial and error. It also requires at least a basic level of knowledge about the problem at hand. To start the process, we developed a preliminary set of 380 features, as will be explained in this section, and to see which one of them has a higher prediction power, we ranked them based on their correlation with the label C'_{min} using the filter and wrapper methods explained in Section 2.2. The features proposed in this paper cover two general aspects of a JSSP instance, one being the general problem configuration (taken directly from the P and M matrices), and the other being the temporal aspects (taken from the output of a dispatching rule or heuristic applied to the problem). A total of 90 of the features presented here have already been introduced in Mirshekarian and Šormaz (2015), but they are more thoroughly explained in this paper.

3.1. Configuration features

Configurations features are taken directly from the P and M matrices. When deriving this group of features, the values inside those matrices are not associated with processing times and machines; they are merely numbers. This group constitutes around 75% of the features presented in this paper (288 out of a total of 380) and extracting its features is computationally inexpensive. There are several different domains or sub-categories that are covered by this category, as listed in Table 5 and explained below. The value of each of the 380 features is calculated for our sample problem and is given in Appendix B.

3.1.1. Overall problem instance

There are only three features in this domain: number of jobs (n), number of machines (m) and total processing time $\sum_{j,k} p_{jk}$.

3.1.2. Individual operations

Looking at the matrix of processing times P, we can derive five useful features from individual operation processing times ($\mathrm{OPT_{jk}} \equiv p_{jk}$) by taking the mean, median, STD, minimum and maximum of OPT values. This is a common trend that we use for extraction of useful data characteristics whenever there are multiple numbers in the same category. Note again that we are treating the values in P as mere numbers without any meaning.

3.1.3. Jobs

Each job j corresponds to a row in P with m elements. If we add up these elements, we get what we call a job processing time for job j ($\mathrm{JPT_j} \equiv \sum_k p_{jk}$). Following along the same lines as for OPT, we take the mean, median, STD, minimum and maximum of the JPT values of all jobs and include them as five new features. However, because we are potentially comparing problem instances of different sizes in our classification problem, we should note that the values of these five features depend highly on the mean JPT itself. For a problem instance with higher JPT, they are likely to be higher as well. To overcome this issue of dominant correlation, we normalized the obtained five values w.r.t mean JPT (and discarded the first one which is always 1) and included the result as four new features. We also normalized them w.r.t mean OPT and considered the result as five separate features. This made the total number of features for this domain equal to 14. Note that if the number of machines stays the same across the class of interest, these two normalized feature sets will be the same and one of them can be discarded.

3.1.4. Machines

Each machine i has to perform n operations corresponding to n jobs. If we add up the processing time of these operations, we get what we call a machine processing time for machine i ($\mathrm{MPT_i} \equiv \sum_{j,k|m_{jk}=i} p_{jk}$). Having m different values of MPT, we extract the exact same type of features as explained for the 'Jobs' domain, a total of 14 features.

3.1.5. Operation slots

Each column k in the P and M matrices corresponds to an operation slot. If we add up the elements of column k, we get what we call an operation slot processing time ($\mathrm{OSPT_k} \equiv \sum_j p_{jk}$). However, unlike JPT and MPT, we consider only the standard deviation of OSPT and its normalized values w.r.t mean OSPT and w.r.t mean OPT as three features (note that mean OSPT is the same as mean MPT, because in our version of JSSP the number of operations is always the same as the number of machines). Moving on, we can argue that since OSPT values denote the amount of processing need at each operation slot, their *positional* values can have significance in terms of the effect on C'. For example, it may be important whether the first operation slot has a high processing need while the next one does not. Similarly, the operation slots close to the middle or those at the end may carry their own special influence. In order to capture these potential effects, we define 11 positional features for OSPT. We first put aside the first value $\mathrm{OSPT_1}$ as a separate feature, and then divide the rest of the values equally into 10 different zones. Averaging across overlaps is done to propagate OSPT values evenly into positional zones. Consider Fig. 2 for example.

In Fig 2, the first operation slot has its own zone and the others are divided into 10 separate zones. The value of zone 2 is calculated by taking a weighted average of the second and third operation slots, while the value of zone 3 is calculated the same way from the third and fourth operation slots. For simplicity, we could also take a plain average of the overlapping slots. The important thing is that these zones, each corresponding to a feature, hold information regarding the positional values of OSPTs. So if OSPTs have a special pattern that changes from left to right (as time progresses), these features will hopefully be able to capture at least part of it. Clearly, when the number of machines is much higher than 10, we can consider more zones for adequate positional resolution, but for m less than 30, the current number should be fine. It will give us a total of 11 new features, and after normalization w.r.t mean OSPT and w.r.t mean OPT, we will have 33 positional features for OSPT.

Now let us consider columns in the machine allocation matrix M. One important aspect of the configuration of this matrix is the arrangement of machine allocations for different jobs. For our sample problem given in Table 1, jobs 0, 1 and 2 all need machine 2 at the same time. This conflict means that there will be waiting time for the jobs, which in turn can translate into a higher makespan (and potentially a higher C'_{min}). Also note that machines 0, 1 and 3 are not required by any jobs at the first operation slot. This will directly affect machine idle time, which in turn can increase C'_{min}. These effects will be less clear as we move to the following operation slots, but there is clearly a pattern to be captured by new

OS (*k*)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
zone	1	2		3	4		5	6		7	8		9	10		11

Fig. 2. Construction of positional features.

Table 2
Operation slot features taken from *M* for the sample problem.

OS (*k*)	1	2	3	4	5
OSMM	3	3	1	2	2
OSRM	3	3	1	2	2
OSRMA	4	6	1	3	2
OSCOMB	155.2	167.2	30	104.7	138.5
OSCOMBA	206.5	334.5	30	157	138.5

Table 3
Sample problem machine loads and their features.

						MLDU	MLDV	MLDVA
M0	0	1	1	1	2	2	1	1
M1	0	4	1	0	0	8	3	4
M2	3	0	2	0	0	7	3	4
M3	0	0	0	3	2	4	3	6
M4	2	0	1	1	1	3	1	1

features. We define two sets of features to capture these effects. One is called the operation slot missing machines (OSMM) and the other operation slot repeated machines (OSRM). For OSMM, we simply count the number of machines that are not present in an operation slot and produce a list of *m* values (see Table 2 for sample calculations). We then compute the mean, median, STD, maximum and minimum of these values, normalize them w.r.t the number of machines and add them to our list of features. The reason for normalization is that the number of machines directly affects these five values. We then add another four features corresponding to the average of these values w.r.t their mean. This will be a total of nine features for OSMM. The same nine features can be computed for OSRM, only this time the list of values are the total number of machine repetitions in each operation slot. The first two rows of Table 2 show the OSMM and OSRM values for our sample problem. Note that they are the same for this instance because the number of jobs and the number of machines are the same. In other words, for every repeated machine, there is exactly one missing machine and vice versa.

Even though OSRM captures the conflicts in machine allocation, it still does not capture the intensity properly because we simply add up the repetitions without regard to what actually happens when such conflicts exist. Suppose there are one repetition of machine 0 and one repetition of machine 1 in the first operation slot of an instance. For each of these machines, one of the jobs has to wait exactly once to have the chance of being processed. Now consider another instance in which there are only two repetitions of machine 0. In our definition of OSRM this will also be counted as two repetitions like the previous instance, but what happens is that one of the jobs has to wait once and the other twice. So in fact 3 waiting periods potentially exist. For this reason, we define another set of features that we call the operation slot repeated machine amplified (OSRMA), in which the value counted for each repetition increases linearly as the number of repetitions increases. So for one repetition of a machine we count 1, for two we count 3, for three we count 6 and for r we count $(1..r) = r(r+1)/2$. For our sample instance this will give the values in row three of Table 2. The same nine features that we computed for OSRM and OSMM will also be computed from these values.

If we think about the values obtained for the three categories above, the positional values of those parameters can also be important. For example, conflicts in machine allocation may have different significance when they happen at the beginning of the schedule or near the end. To capture this potential effect, we defined positional values for OSMM, OSRM and OSRMA exactly as explained for OSPT, only this time we include just the normalized versions. This will give us 22 features per parameter, which is a total of 66 features for the operation slot values taken from *M*.

3.1.6. Operations slots combined with processing times

This is another domain of features we considered in parallel to the operations slots domain, and it is consisted of two groups. One is called the operation slot repeated machines combined with processing times (OSCOMB), and the other its amplified counterpart. The idea behind OSCOMB is that even though OSRM captures the conflicts between machine allocations to jobs, the impact of these conflicts is highly dependent on the amount of processing time involved in the conflicting operations. It is not hard to see that if this amount increases, the amount of waiting and the effect on C' increase as well. OSCOMB captures this effect by multiplying the OSRM values with the mean of the corresponding operation processing times. As an example, consider the first operation slot of our sample problem, where machine 2 is occurs 3 times (repeated twice), and the corresponding operation processing times are 96, 37 and 21. This counts as (3-1)∗mean(96, 37, 21) = 102.67 towards the OSCOMB of the first operation slot. Adding the value for the repetitions of machine 4, we get a total OSCOMB value of 155.17 for the first operation slot. The other values are also computed and given in Table 2. Similar to OSRM, we compute 9 non-positional and 22 positional features for OSCOMB, only this time we add another 5 non-positional and another 11 positional ones by normalizing w.r.t the product of the number of machines and mean OPT. This is because OSCOMB depends proportionally on both of these values. Finally, we also consider an amplified version of OSCOMB called OSCOMBA, which is based on OSRMA values instead of OSRM, and compute 47 features for it as well.

3.1.7. Machine load

The last configuration feature domain considered is taken again from *M*. A machine load value for machine i is defined as an array containing the number of occurrences of that machine in each operation slot of the matrix *M*. For example the machine load of machine 0 for our sample instance is (0, 1, 1, 1, 2), because it has not been used in the first slot and it has been used twice in the last slot. Table 3 shows the machine loads for all five machines for our sample instance.

Now we define two groups of features based on these machine load values. One is called machine load uniformity (MLDU) and the other machine load voids (MLDV). MLDU is calculated for each machine by adding up the net change in machine loads for that machine. For example consider *M0* in Table 3. The net change in machine load is 1 from slot 0 to 1, zeros for the next two slot changes, and 1 for the last slot change. This makes MLDU for machine 0 equal to 2. For machine 1 a net change of 4 is seen for the first slot change, 3 for the next, 1 after that and then zero in the end, adding up to a total of 8. The rest of the MLDU values are also computed and given in Table 3. The other group is MLDV, for which we simply count the number of occurrences of

‘0′ in the machine load array of each machine. We also considered an amplified version of MLDV, called MLDVA, which counts increasingly higher values for *consecutive* occurrences of 0′s, similar to the scheme we used for OSRMA. Note that here instead of repetitions, we are interested in consecutive occurrences. For example, machine 0 has one occurrence of ‘0′ so its MLDVA is simply 1. Machine 1 has one single occurrence which counts as 1 and two consecutive occurrences which count as $1+2=3$; a total of 4. Machine 3 has three consecutive occurrences which are counted as $1+2+3=6$. The MLDVA values stress consecutive occurrences of 0′s because we believe when the load on a machine has lower uniformity, that machine is more likely to be idle.

Having the MDLU, MLDV and MLDVA values, we can compute their mean, median, STD, maximum and minimum, and then normalize them w.r.t their mean and the number of machines to be included as 9 new features. This is a total of 27 features for this domain. We also add two separate features to stress the consecutive occurrences of ‘0′ in machine loads even more. One feature is computed by counting the number of consecutive occurrences longer than 30% of the number of machines m, and the other for those more than 40% of m, counting in an increasingly higher number as the sequence gets bigger. For our problem instance with $m=5$, machine 0 has no consecutive occurrence so it is counted as zero for both features. Machines 1 and 2 have a ‘0′ sequence of length 2, so since $2 \geq 0.3(5)$ each machine is counted as 1 towards the first features, and since $2 \geq 0.4(5)$ each machine is also counted as 1 for the second features. For machine 3 there is a ‘0′ sequence of length 3, so it is counted as 2 for both of the features. Such case does not exist in our sample problem, but if we had a ‘0′ sequence of length 4, it would be counted as 3 for both features and so on. Note that the difference between these two features will be more clear when m is greater than 10.

3.2. Temporal features

Configuration features neglect what the values in the P and M matrices really mean, but here we use the fact that P contains time values and M contains machines that operate in time. One way to do this is by simulating the allocation of operations to machines and how conflicts are resolved. Conflict resolution is done using simple dispatching rules and the final result is an actual schedule that may or may not be optimal. We can take the resulting schedule characteristics, like its makespan, as one set of features, and follow the simulation and use measurements of characteristics of what happens through time as another set of features. We do both of these here using five simple dispatching rules. They are chosen to be simple in order to lower computational costs and also to have easier-to-understand performance.

3.2.1. Shortest processing time (SPT)

Ready-to-work machines start processing operations as soon as they are ready, and if more than one operation is ready, the one with the minimum processing time is chosen. After that operation is performed, new jobs may have joined the queue and the process is repeated with those included. We solve every problem instance with this simple heuristic and record the resulting makespan and C' as two features. We also count the number of times the heuristic has resolved conflicts and use that as another feature. If at one point there are two conflicting operations, a value of 1 is counted towards this feature, and if there are four conflicting operations, a value of 3. We then look at the schedule produced by the heuristic, and record the completion times of all jobs and all machines as two sets of values, and then compute the mean, median, STD, minimum and maximum of those values as 10 other features. The only difference between these sets of five with the ones we had previously is that the mean is not normalized, while the others are by dividing by the mean. The completion time values for our sample problem are given in Table 4. Notice that the maximum of either of these sets of values is the makespan of the schedule obtained by SPT, which is 532.

Table 4
Sample problem completion times of 5 jobs and 5 machines for SPT.

	0	1	2	3	4
JCT	443	188	393	532	368
MCT	443	443	443	532	443

The last set of features produced from SPT is called the schedule operation slot repeated jobs amplified (SOSRJA). To compute these features we look at every operation slot in the obtained schedule and count the number of repeated jobs, increasing the value linearly for repetitions more than two (just as we did for the amplified version of OSRM). This will give us a list of n values, from which we compute the regular five features mean, median, STD, minimum and maximum, after normalizing them w.r.t the number of jobs. This will make it a total of 18 features for SPT (and for each dispatching rule that we used).

3.2.2. Longest processing time (LPT)

LPT is similar to SPT, except that when resolving a conflict, the operation with the highest processing time is given priority. The same set of 18 features is computed for LPT as well.

3.2.3. Most work remaining (MWRM)

When there are conflicting operations, we look at their corresponding jobs and choose the one for which the corresponding job has the most total processing time remaining. The motivation behind MWRM is that jobs with overall higher processing time needs might make the makespan even longer than they potentially can if they are not given priority. In fact, MWRM obtained the best performance in our experiments, perhaps because this was the most important consideration for the type of problem we considered. The same 18 features were computed for MWRM as well.

3.2.4. Least work remaining (LWRM)

Even though the argument behind MWRM seems convincing, there are still cases where even a completely opposite direction can yield better results (and that is why one simple heuristic cannot solve all types of problems with a stable level of performance). For this reason, we considered LWRM as well, in which conflicts are resolved by choosing the operation with the corresponding job that has the least total processing time remaining, and computed the 18 features for it.

3.2.5. First-in-first-out MWRM (FIFO-MWRM)

The priority is given here to the job that has become ready the earliest. If more than one job have this property simultaneously, the conflict is resolved according to the most work remaining measure. For our randomly generated problem instances, such conflicts usually happen and the beginning of the schedule, and therefore this rule can give completely different results than the pure MWRM rule. Again the same 18 features were computed for this heuristic.

After computing all the individual heuristic features, we take the minimum C' of all heuristics and count it as one new feature. This feature represents the best we can get in terms of estimation of the true C'. We also normalized this minimum heuristic C' w.r.t the maximum of the largest of JPT and MPT values and counted it as another extra feature, which concluded our list of 92 temporal features, and also our list of total 380 features. We evaluate the features and experiment with them in the next section. See

Table 5
Full list of 380 features and their descriptions.

Domain	ID	Description
Overall	1–3	Number of jobs (n), number of machines (m) and total processing time of all operations $\sum_{j,k} p_{jk}$
Operations	4–8	Mean, median, STD, min and max of Operation Processing Times (OPT $\equiv p_{jk}$)
Jobs	9–13	Mean, median, STD, min and max of Job Processing Times (JPT $\equiv \sum_{k} p_{jk}$)
	14–17	Median, STD, min and max of JPT, divided by the mean of JPT
	18–22	Mean, median, STD, min and max of JPT, divided by the mean of OPT
Machines	23–27	Mean, median, STD, min and max of Machine Processing Times (MPT $\equiv \sum_{j,k\|M_{jk}=i} p_{jk}$)
	28–31	Median, STD, min and max of MPT, divided by the mean of MPT
	32–36	Mean, median, STD, min and max of MPT, divided by the mean of OPT
Operation Slots	37	STD of the Operation Slot Processing Times (OSPT $\equiv \sum_{j} p_{jk}$)
	38	STD of OSPT divided by the mean of OSPT
	39	STD of OSPT divided by the mean of OPT
	40–50	Positional values of OSPT
	51–61	Positional values of OSPT divided by the mean of OSPT
	62–72	Positional values of OSPT divided by the mean of OPT
	73–76	Median, STD, min and max of OSMM, divided by the mean of OSMM
	77–81	Mean, median, STD, min and max of OSMM, divided by the number of machines (m)
	82–85	Median, STD, min and max of OSRM, divided by the mean of OSRM
	86–90	Mean, median, STD, min and max of OSRM, divided by the number of machines (m)
	91–94	Median, STD, min and max of OSRMA, divided by the mean of OSRMA
	95–99	Mean, median, STD, min and max of OSRMA, divided by the number of machines (m)
	100–110	Positional values of OSMM, divided by mean OSMM
	111–121	Positional values of OSMM, divided by the number of machines (m)
	122-132	Positional values of OSRM, divided by mean OSRM
	133––143	Positional values of OSRM, divided by the number of machines (m)
	144–154	Positional values of OSRMA, divided by mean OSRMA
	155–165	Positional values of OSRMA, divided by the number of machines (m)
Operation Slots Combined with Processing Times	166–170	Mean, median, STD, min and max of OSCOMB, divided by the number of machines (m)
	171–174	Median, STD, min and max of OSCOMB, divided by the mean of OSCOMB
	175–179	Mean, median, STD, min and max of OSCOMB, divided by the product of m and mean OPT
	180–190	Positional values of OSCOMB, divided by the number of machines (m)
	191–201	Positional values of OSCOMB, divided by the mean of OSCOMB
	202–212	Positional values of OSCOMB, divided by the product of m and mean OPT
	213–217	Mean, median, STD, min and max of OSCOMBA, divided by the number of machines (m)
	218–221	Median, STD, min and max of OSCOMBA, divided by the mean of OSCOMBA
	222–226	Mean, median, STD, min and max of OSCOMBA, divided by the product of m and mean OPT
	227–237	Positional values of OSCOMBA, divided by the number of machines (m)
	238–248	Positional values of OSCOMBA, divided by the mean of OSCOMBA
	249–259	Positional values of OSCOMBA, divided by the product of m and mean OPT
Machine Load	260–263	Median, STD, min and max of MLDU, divided by the mean of MLDU
	264–268	Mean, median, STD, min and max of MLDU, divided by the number of machines (m)
	269–272	Median, STD, min and max of MLDV, divided by the mean of MLDV
	273–277	Mean, median, STD, min and max of MLDV, divided by the number of machines (m)
	278–281	Median, STD, min and max of MLDVA, divided by the mean of MLDVA
	282–286	Mean, median, STD, min and max of MLDVA, divided by the number of machines (m)
	287	Number of consecutive zero loads longer than 30% of m, divided by the number of machines (m)
	288	Number of consecutive zero loads longer than 40% of m, divided by the number of machines (m)
Heuristics	289–290	Makespan and C' obtained by the SPT heuristic
	291	Conflict resolutions made by the SPT heuristic, divided by the number of operations ($n.m$)
	292	Mean of the SPT heuristic job completion times
	293–296	Median, STD, min and max of the SPT heuristic job completion times, divided by their mean
	297	Mean of the SPT heuristic machine completion times
	298–301	Median, STD, min and max of the SPT heuristic machine completion times, divided by their mean
	302–306	Mean, median, STD, min and max of the SPT heuristic SOSRJA
	307–308	Makespan and C' obtained by the LPT heuristic
	309	Conflict resolutions made by the LPT heuristic, divided by the number of operations ($n.m$)
	310	Mean of the LPT heuristic job completion times
	311–314	Median, STD, min and max of the LPT heuristic job completion times, divided by their mean
	315	Mean of the LPT heuristic machine completion times
	316–319	Median, STD, min and max of the LPT heuristic machine completion times, divided by their mean
	320–324	Mean, median, STD, min and max of the LPT heuristic SOSRJA
	325–326	Makespan and C' obtained by the MWRM heuristic
	327	Conflict resolutions made by the MWRM heuristic, divided by the number of operations ($n.m$)
	328	Mean of the MWRM heuristic job completion times
	329–332	Median, STD, min and max of the MWRM heuristic job completion times, divided by their mean
	333	Mean of the MWRM heuristic machine completion times
	334–337	Median, STD, min and max of the MWRM heuristic machine completion times, div. by their mean
	338–342	Mean, median, STD, min and max of the MWRM heuristic SOSRJA
	343–344	Makespan and C' obtained by the LWRM heuristic

(*continued on next page*)

Table 5 (*continued*)

Domain	ID	Description
	345	Conflict resolutions made by the LWRM heuristic, divided by the number of operations ($n.m$)
	346	Mean of the LWRM heuristic job completion times
	347–350	Median, STD, min and max of the LWRM heuristic job completion times, divided by their mean
	351	Mean of the LWRM heuristic machine completion times
	352–355	Median, STD, min and max of the LWRM heuristic machine completion times, div. by their mean
	356–360	Mean, median, STD, min and max of the LWRM heuristic SOSRJA
	361–362	Makespan and C' obtained by the FIFO_MWRM heuristic
	363	Conflict resolutions made by the FIFO_MWRM heuristic, div. by the number of operations ($n.m$)
	364	Mean of the FIFO_MWRM heuristic job completion times
	365–368	Median, STD, min and max of the FIFO_MWRM heuristic job compl. times, div. by their mean
	369	Mean of the FIFO_MWRM heuristic machine completion times
	370–373	Median, STD, min and max of the FIFO_MWRM heuristic mach. compl. times, div. by their mean
	374–378	Mean, median, STD, min and max of the FIFO_MWRM heuristic SOSRJA
	379	Minimum of heuristic C' values
	380	Minimum of heuristic C'values, divided by the max of (max MPT, max JPT)

Table 6
Top and bottom ten overall features plus top ten configuration features for the 10J10M batch.

Ave. Rank	ID	PCC	SNR	T-value	SVM	Description
1	379	0.0311	0.7570	53.1956	77.63%	Minimum of heuristic C' values
2	326	0.0278	0.6266	43.9898	73.65%	C'obtained by MWRM
3	362	0.0241	0.5310	37.3280	70.25%	C'obtained by FIFO_MWRM
4	290	0.0219	0.4786	33.7181	68.33%	C'obtained by SPT
5	327	0.0200	0.4319	30.3643	66.65%	Conflict resolutions made by MWRM, div. by num. of operations
6	294	0.0189	0.4111	28.9503	65.25%	STD of SPT job completion times, divided by their mean
7	333	0.0172	0.3548	24.9296	64.22%	Mean of MWRM machine completion times
9	308	0.0172	0.3508	24.7682	64.03%	C'obtained by LPT
9	348	0.0166	0.3516	24.8198	62.92%	STD of LWRM job completion times, divided by their mean
10	325	0.0169	0.3463	24.3483	63.52%	Makespan obtained by MWRM
20	222	0.014	0.2757	19.3716	59.68%	Mean of OSCOMBA, divided by (m $*$ mean OPT)
25	175	0.0127	0.2423	17.0976	58.75%	Mean of OSCOMB, divided by (m $*$ mean OPT)
29	282	0.0124	0.2305	16.2097	58.47%	Mean of MLDVA, divided by the number of machines
34	20	0.0107	0.2352	16.5216	58.63%	STD of JPT, divided by mean OPT
35	15	0.0107	0.2352	16.5216	58.63%	STD of JPT, divided by mean JPT
37	95	0.0112	0.2181	15.3415	58.42%	Mean of OSRMA, divided by the number of machines
39	36	0.0113	0.2092	14.6659	57.00%	Maximum of MPT, divided by mean OPT
39	31	0.0113	0.2092	14.6659	57.00%	Maximum of MPT, divided by mean MPT
43	34	0.0108	0.207	14.5969	57.47%	STD of MPT, divided by mean OPT
44	29	0.0108	0.207	14.5969	57.47%	STD of MPT, divided by mean MPT
367	60	0.0002	0.0023	0.1633	52.35%	Positional values of OSPT, divided by mean OSPT (No. 10)
368	61	0.0002	0.0023	0.1633	52.35%	Positional values of OSPT, divided by mean OSPT (No. 11)
369	172	0.0004	0.0002	0.0122	52.20%	STD of OSCOMB, divided by mean OSCOMB
369	71	0.0002	0.0023	0.1633	52.35%	Positional values of OSPT, divided by mean OPT (No. 10)
370	72	0.0002	0.0023	0.1633	52.35%	Positional values of OSPT, divided by mean OPT (No. 11)
374	267	0.0001	0.0008	0.0589	52.35%	Minimum of MLDU, divided by the number of machines
374	355	0.0002	0.0002	0.0156	52.35%	Max of LWRM mach. completion times, div. by their mean
376	185	0.0002	0.0000	0.0007	52.35%	Positional OSCOMB, divided by num. of machines (No. 6)
378	1	0.0000	0.0000	0.0000	52.35%	Number of jobs (n)
379	2	0.0000	0.0000	0.0000	52.35%	Number of machines (m)

Table 5 for the full listing of the 380 features and their description, and also Appendix B for the values of these features for our sample problem.

4. Results and discussion

We generated a total of 15,000 JSSP instances and divided them equally into three groups of different sizes: the first group for instances with 8 jobs and 8 machines, called the 8J8M group, the second for those with 10 jobs and 10 machines, 10J10M, and the third for those with 11 jobs and 11 machines, 11J11M. Both the processing times and machine allocations were generated using a uniform random distribution, with processing times in the range [1, 100]. The random numbers were produced using MATLAB's default random number generator with a shuffled seed[4]. The problem instances were then solved to optimality using constraint programming on IBM ILOG OPL, and the resulting makespan values were recorded. These values were then used to compute C'_{min} for each instance to be used as the label. MATLAB was then used to read problem data, extract the 380 features and run the experiments. In order to evaluate the effect of problem size on feature performance, we ran the experiments on two different batches of instances; one batch containing only the 10J10M instances, and the other containing all the 15,000 randomly mixed up. Each batch was further divided into cross-validation and test instances, such that from each group in each of the batches, 1000 instances were set aside for final testing and the rest were used for cross-validation. When calculating the label for mixed batch instances, C'_{min} of each instance was compared to the mean C'_{min} of all the 15,000 instances together (which was 1.6361).

The first experiment was a single-feature correlation analysis. We first used the three correlation measures of Section 2.2 and computed an average ranking using the ranks obtained by the

[4] All of the data used for this research plus the source codes are available online or upon request.

Table 7
Top and bottom ten overall features plus top ten configuration features for the mixed batch.

Ave. Rank	ID	PCC	SNR	T-value	SVM	Description
1	379	0.0315	0.6828	82.9712	75.11%	Minimum of heuristic C' values
2	326	0.0285	0.5875	71.2743	72.03%	C'obtained by MWRM
3	362	0.0246	0.4922	59.8435	68.97%	C'obtained by FIFO_MWRM
4	290	0.0227	0.4463	54.3222	67.40%	C'obtained by SPT
5	294	0.0208	0.4064	49.3581	65.33%	STD of SPT job completion times, divided by their mean
6	327	0.0191	0.3758	45.6529	64.67%	Conflict resolutions made by MWRM, div. by num. of operations
7	348	0.0180	0.3439	41.8983	62.54%	STD of LWRM job completion times, divided by their mean
8	308	0.0169	0.3207	39.1719	62.84%	C'obtained by LPT
9	291	0.0160	0.3032	36.8558	61.96%	Conflict resolutions made by SPT, div. by num. of operations
10	296	0.0159	0.3008	36.5734	62.39%	Maximum of SPT job completion times, divided by their mean
18	222	0.0132	0.2423	29.3915	59.32%	Mean of OSCOMBA, divided by (m * mean OPT)
20	15	0.0122	0.2426	29.3322	59.62%	STD of JPT, divided by mean JPT
20	29	0.0132	0.2317	28.0869	59.03%	STD of MPT, divided by mean MPT
24	12	0.0117	0.2311	28.1811	59.03%	Minimum of JPT
28	175	0.0121	0.2193	26.7212	59.06%	Mean of OSCOMB, divided by (m * mean OPT)
29	31	0.0127	0.2081	25.1554	57.89%	Maximum of MPT, divided by mean MPT
32	26	0.0112	0.2106	25.6935	58.45%	Minimum of MPT
34	224	0.0112	0.1967	23.6805	57.00%	STD of OSCOMBA, divided by (m * mean OPT)
37	282	0.0110	0.1940	23.5685	57.97%	Mean of MLDVA, divided by the number of machines
37	21	0.0100	0.2019	24.6151	58.17%	Minimum of JPT, divided by mean OPT
371	269	0.0001	0.0036	0.4380	53.74%	Median of MLDV, divided by mean MLDV
371	59	0.0002	0.0022	0.2733	53.74%	Positional values of OSPT, divided by mean OSPT (No. 9)
372	39	0.0001	0.0032	0.3964	53.74%	STD of OSPT, divided by mean OPT
373	61	0.0003	0.0008	0.0980	53.74%	Positional values of OSPT, divided by mean OSPT (No. 11)
373	185	0.0003	0.0015	0.1884	53.74%	Positional OSCOMB, divided by the number of machines (No. 6)
374	52	0.0000	0.0020	0.2487	53.74%	Positional values of OSPT, divided by mean OSPT (No. 2)
374	56	0.0001	0.0016	0.1952	53.74%	Positional values of OSPT, divided by mean OSPT (No. 6)
376	184	0.0001	0.0012	0.1501	53.74%	Positional OSCOMB, divided by the number of machines (No. 5)
377	54	0.0002	0.0008	0.0926	53.74%	Positional values of OSPT, divided by mean OSPT (No. 4)
380	262	0.0000	0.0002	0.0244	53.74%	Minimum of MLDU, divided by mean MLDU

Table 8
Performance of feature groups according to the individual performance of their features.

Feature Group	Num. of Features	*10J10M*			*Mixed*		
		Group Rank	Mean of Feature Ranks	STD of Feature Ranks	Group Rank	Mean of Feature Ranks	STD of Feature Ranks
Heuristic makespan and C'	12	1	16	16.7	4	108	126.5
Heuristic JCT	25	3	95	87.6	6	120	119.2
Heuristic MCT	25	13	171	107.6	25	268	104.5
Heuristic SOSRJA	25	6	103	47.2	3	106	52.9
SPT Heuristic	18	4	95	94.8	8	137	132.4
LPT Heuristic	18	7	120	85.5	14	183	121.4
MWRM Heuristic	18	5	102	76.4	10	158	104.9
LWRM Heuristic	18	11	129	115.9	11	160	133.4
FIFO_MWRM Heuristic	18	2	84	74.3	7	126	115.1
OPT	5	16	202	106.4	15	188	109.6
JPT	14	8	120	85.5	1	96	85.1
MPT	14	9	120	73.8	2	99	88.1
OSPT	3	26	314	7.9	26	270	75.4
OSPT Positional	33	25	305	55.9	17	217	106.5
OSMM	9	19	218	96.2	20	228	110.4
OSMM Positional	22	22	234	72.9	22	239	66.4
OSRM	9	20	219	96.2	21	229	110.4
OSRM Positional	22	23	235	72.7	23	240	66.4
OSRMA	9	14	189	107.2	13	177	100.4
OSRMA Positional	22	21	223	74.6	19	227	69.6
OSCOMB	14	15	190	111.3	12	169	105.3
OSCOMB Positional	33	24	235	80.4	24	240	80.9
OSCOMBA	14	10	125	81.1	5	116	79.1
OSCOMBA Positional	33	17	210	78.6	16	212	76.6
MLDU	9	27	349	16.2	27	329	31.5
MLDV	9	18	214	89.4	18	219	103.2
MLDVA	11	12	142	85.0	9	146	94.3

three methods, and we then used SVM with a quadratic kernel to train and evaluate the performance of each individual feature through a 5-fold cross-validation classification[5]. We used accuracy (defined as the ratio of the number of correct classifications over the number of classified instances) as the performance measure.

[5] The results reported in this paper are achieved using MATLAB's `fitcsvm` tool with automatically tuned parameters. We also experimented with SVMLight (Joachims, 1999) and could achieve slightly better results with parameter fine-tuning.

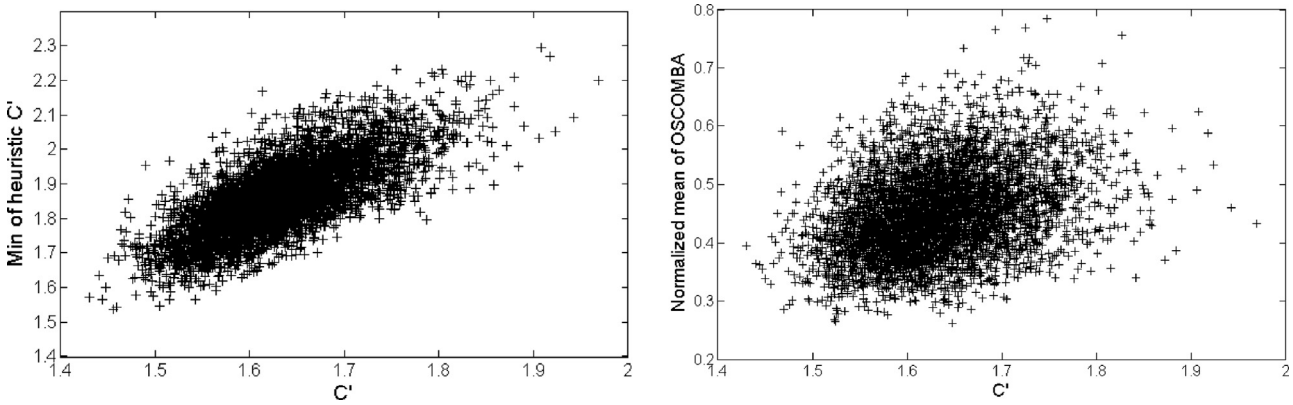

Fig. 3. Correlation of the top overall (left) and top configuration (right) features with true C'_{min} for 10J10M.

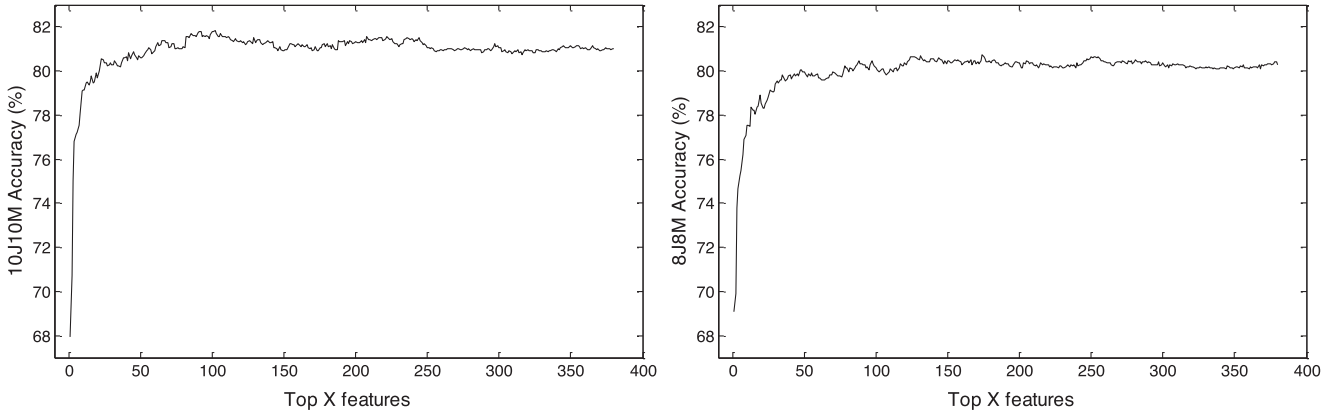

Fig. 4. Classification performance for 10J10M (left) and 8J8M (right).

Tables 6 and 7 show the top and bottom ten overall features, along with ten of the top configuration features according to the average ranking for the 10J10M batch and the mixed batch respectively (See Appendix C for the full list). Inspecting the results, the following observations can be made:

- The top feature in both batches is the one corresponding to the minimum of heuristic C' values. This is not very surprising because this feature represents the closest we can get to the label using our five heuristics. Note that heuristics are themselves predictors for the label, and this is clear from the high individual prediction performance for heuristic C' features. Fig. 3 shows the correlation of this feature and also the top configuration feature (normalized mean of OSCOMBA) with the true C'_{min} for the 10J10M batch. Note that a linear correlation is apparent for both features, but stronger for the top overall feature.
- One interesting feature in the top 10 for both batches is the normalized number of conflict resolutions made by the MWRM heuristic. The reason is not obvious, but this indicates that trial-and-error is an important part of feature engineering. Some features that are seemingly not useful, or their impact is not obvious, can turn out to be important and vice versa. Another curious point about the conflict resolution features is that only the one associated with MWRM is this good. For example the conflict resolution feature for FIFO_MWRM ranks 110 and 74 for the two batches which is nowhere near as good.
- Since n and m are invariant in the first batch, their corresponding features are at the bottom of the list for this batch (meaning that they are completely independent of the label). However, these two features achieved ranks 94 and 95 for the mixed batch which is obviously better. This highlights the effect of problem size on feature performance.
- Some of the features from the normalized OSCOMBA group performed better than all other configuration features. This is particularly interesting if we note that some other "obvious" configuration features are ranked slightly lower. For example, it is clear that the higher the standard deviation of job processing times, the more likely it is for C'_{min} to be high (lower scheduling efficiency). However, the corresponding feature is not as correlated to C'_{min} as is the normalized mean of OSCOMBA.
- No feature group can be said to have dominated the ranking list. We can see features from the OSCOMB group at the bottom of the mixed batch rankings, or even heuristic-related features at the bottom of the 10J10M batch. This further highlights the importance of trial-and-error in identifying the significance of impact of certain features. Having said that however, some feature groups are more-or-less performing better than others on average. See Table 8 which shows the mean and standard deviation of the rankings obtained by the features in each group on each batch. The makespan and C' group is in the top five for both batches, while FIFO_MWRM is the best-performing heuristic overall. Among configuration features, JPT, MPT, OSCOMBA and MLDVA are at the top, while MLDU is unanimously the worst-performing group for both batches.
- The performance of positional features was mixed. On the one hand, there is some good performance by the first feature in the normalized OSCOMBA positionals, which was ranked 75st and 72th for the 10J10M and mixed batches respectively, and also the 10th and 11th positions which follow it closely. On the other hand, most of the positional features did poorly and were ranked even among the bottom 10. A trend that was visible was

Table 9
Categorical classification performance on the two batches.

Feature Set	Num. of Features	*10J10M*		*Mixed*	
		10-fold Cross-Validation	Test	10-fold Cross-Validation	Test
All	380	81.17%	80.50%	80.59%	80.43%
Configuration Only	288	69.52%	71.60%	70.13%	69.23%
Temporal Only	92	79.52%	81.50%	79.02%	79.63%

Table 10
Possible JSSP instances obtained for the Prism manufacturing example.

Job	P (four options per job)								M (fixed for each job)							
AES94	16	28	0	41	0	0	0	0	7	8	1	2	3	4	5	6
	8	15	29	17	0	14	0	16								
	0	11	14	30	25	0	0	32								
	0	0	16	29	0	25	32	0								
Bracket	12	156	0	84	0	0	0	0	7	8	1	2	4	5	3	6
	5	60	39	52	27	71	45	0								
	37	63	55	39	14	64	20	0								
	16	12	39	52	14	64	0	97								
NetEx	9	17	0	28	0	0	0	0	7	8	1	2	5	6	3	4
	3	6	23	10	22	0	0	33								
	0	7	23	10	3	3	4	25								
	0	0	34	0	29	6	4	25								
Plate	0	6	1	0	0	0	0	0	1	2	7	8	4	5	3	6
	4	3	1	0	0	0	1	0								
	3	3	0	3	1	0	0	0								
	3	4	0	0	1	1	0	0								
Slider	0	1	0	0	7	0	0	16	4	5	3	6	7	8	1	2
	0	0	1	4	0	4	7	11								
	0	0	0	4	0	6	8	10								
	7	0	4	0	0	1	6	11								
USC	0	0	0	0	0	52	46	13	4	5	3	6	1	2	7	8
	38	0	0	0	37	21	0	62								
	15	0	10	16	37	21	0	23								
	0	49	18	0	62	0	31	0								

Table 11
Results of using highly correlated features to identify best instances.

Selection Criterion	Ave. C'_{min} for top 10	Ave. C'_{min} for bottom 10	Ave. C'_{min} of all	Min. C'_{min} of all	Max. C'_{min} of all
Normalized OSCOMBA	3.456	4.312	3.658	3.062	4.448
Normalized OSCOMBA + STD of JPT divided by mean JPT	3.153	4.010			
Normalized OSCOMBA + STD of JPT divided by mean JPT + Mean of MLDVA divided m + Mean of OSRMA divided by m + STD of JPT divided by mean OPT	3.152	4.072			

that features corresponding to positions near the two ends (like the first and 11th positions) performed considerably better than those for positions near the middle (like the 5th and 6th positions). This is true for both batched, although more evident for the mixed batch, and may have certain clues as to how the arrangement of values in P and M can affect scheduling efficiency.

After individual correlation analysis, the next experiment was classification using SVM on a *set* of features taken from the 10J10M batch. We used the following simple algorithm to do classification F times, every time with a different set of features. The resulting classification accuracy is shown in Fig. 4 for two different size classes. In the algorithm below, Φ is the current set of features to be used and R is the ranked list of all features ($R(1)$ denotes the highest ranking feature and $R[1..X]$ denotes the top X features). We used a Gaussian kernel with automatically tuned parameters for SVM.

```
for X from 1 to F do
    set Φ to R[1..X]
    do 5-fold cross-validation with SVM, using Φ
    record the average classification accuracy
```

It is observed from the resulting diagrams in Fig. 4 that the accuracy reaches its highest level when the first 100–150 features are included, and after that the remaining features are actually being detrimental to performance. This can mean that they are less than useful, although we cannot assert that with reasonable confidence until a more advanced feature selection method like greedy forward/backward selection is employed (see Zhang, 2009). Even the level of performance drop seems statistically insignificant. We did not do greedy/backward forward selection in this paper. Instead, we used a simpler method to try and estimate the real impact of the features at the bottom of the ranking. Note that as mentioned before, being at the bottom of a rank based on individual performance does not automatically disqualify a feature, because some features only play a role when combined with others. Unfortunately, testing all the possible combinations of features is intractable.

The last experiment was for assessing category performance in classification. To do that, we chose three different sets of features and classified both the cross-validation and test instances of each batch using SVM with an automatically tuned Gaussian kernel. One set consisted of all the features, the next only

configuration features and the last only temporal features. Table 9 shows the resulting classification accuracy for each set. Note that all sets seem to have similar performance when applied to 10J10M or the mixed batches. The important point here is that configuration features can reach the accuracy figure of 70%, while adding the temporal features increases this by about 10%. Temporal features can perform just as well when used alone, so it can mean that configuration features are redundant to them. In terms of practicality, obtaining configuration features may be easier and less time-consuming than temporal features. Note that the main objective of this paper was to identify features that have a high correlation with optimal makespan, and this is independent of whether these features are redundant to each other or not. Knowing that a feature like mean OSCOMBA can determine optimal makespan and/or scheduling efficiency, can be a great help to the problem designer in the quest for designing more efficient shops.

5. Prismatic part manufacturing: an example application

Prismatic part manufacturing is usually done using flexible manufacturing systems (FMS) in a manner described in Šormaz, Arumugam, Harihara, Patel, and Neerukonda (2010). In this context, the scheduling of jobs is applied to a set of six jobs that can be manufactured on eight different machines. Those machines can perform similar tasks, but with a different efficiency (speed). Possible alternative routings can be generated by the procedure described in Šormaz and Khoshnevis (2003). For each job, four different routings have been generated as shown in Table 10. This leads to set of possible configurations for the FMS processing, i.e., a set of possible JSSP instances. The order of machines is fixed for each job (and therefore so is the machine allocation matrix). The problem here is to choose the right set of machines for each job (i.e., the right processing time matrix), such that the optimal scheduling efficiency of the resulting JSSP instance is maximized. Note that having the least machine idle time is more important for this problem than having the least optimal makespan. One way to tackle the problem is to solve all of the 4096 possible instances, and choose the one with minimum C'_{min}. This is indeed feasible here, as the size of each instance is very small (we use a small example here to be able to validate our approach). However, when problem size increases, as it does for some other similar cases we had to deal with, solving all possible instances will not be a realizable option. This is where the methodology and insight introduced in this paper will be helpful.

When we cannot solve all the possible instances, one way to choose a good instance is by looking at the features that have high correlation with C'_{min}. These features can be taken from Table 6. As an example, since normalized OSCOMBA has the highest correlation among configuration features, one can simply choose the instance that has the highest normalized OSCOMBA. However, it is perhaps more reliable to compute a weighted average of a set of top features (after normalizing each feature to take away the effect of value range) and use that as the selection criterion instead. To evaluate the usefulness of this approach, we compared the average C'_{min} of the top 10 and bottom 10 of its resulting instances with the average C'_{min} of all instances (obtained by solving the whole 4096 set through OPL) for various selection criterions. The results, shown in Table 11, indicate that the approach can indeed help us filter out close-to-ideal instances by looking only at a few configuration features. Note that the closer the top and bottom 10 results get to minimum and maximum C'_{min}, the better.

Another way of dealing with our problem is to extract all features and do classification as we did in Section 4, to determine which instances have less than average C'_{min}. This is indeed a coarse measure and will not give us the top choices as we achieved using the previous approach, but we can use the features in a regression setting and obtain actual predictions of C'_{min} for the instances. We have not done regression in this paper, but it should be straightforward having the features and the data. For comparison however, we did a 10-fold cross-validation for classification, similar to what was done in Section 4, and obtained a 83.50% classification accuracy. This is better than the results reported in Table 9, probably because of the smaller 6 by 8 size. In any case, we used the first approach for all the problems that we had in our work, while using the classification results just as a coarse guideline.

6. Conclusion and future work

In this paper, we elaborated on 90 features already introduced by Mirshekarian and Šormaz (2015) and introduced 290 new ones. The features were divided into two categories, configuration features for which we only use 'values' in the processing time matrix P and machine allocation matrix M without regard to their meaning, and temporal features for which we take advantage of the meaning of those values and run simulations with heuristics to extract features that denote temporal characteristics. We randomly generated and solved 15,000 JSSP instances of different sizes, divided them into two batches, one including a mix of all instances and the other including only instances with 10 jobs and 10 machines, both to be used for evaluation of the features and help extract relationships that may exist between them and the optimal makespan. As such, we ran three experiments with these instances, using a normalized measure of optimal makespan called C'_{min} introduced by Mirshekarian and Šormaz (2015) to be comparable for instances of different size.

In the first experiment, we used three different methods to assess the correlation of individual features with C' and then rank them accordingly. Temporal features, expectedly, ranked among the highest for both batches, whereas some configuration features performed well too. Among configuration features, one those representing the repetitions of machines in the columns of the matrix M performed the best, especially when combined with the corresponding processing times. Such individual rankings can greatly help researchers understand or verify correlation pathways between the structure of problem input, the P and M matrices, and optimal makespan. Understanding those pathways can in turn be helpful in designing better shops. For example, since our experiments showed that machine repetitions in the columns near the two ends of the matrix M increase C'_{min} (decrease optimal scheduling efficiency) more than those in the middle columns, the shop designer should try to reduce those repetitions in the critical zones of the matrix (or decrease the corresponding processing times if possible), while not being worried as much about repetitions in the middle columns. Such insight is hard to come by through experience.

The second and third experiments were an effort to see which feature combinations can have a higher prediction power in a machine learning problem to classify instances as having a C'_{min} higher or lower than class average. The results showed that only having temporal features we can get accuracies above 80%, whereas only having configuration features we could only get up to 70%. We concluded that most configuration features may be redundant to temporal ones, even though we cannot be certain that temporal features contain all the information in the configuration ones. The results also showed that for the 10J10M batch, the highest accuracy is achieved when the top 102 features are used, and performance drops slightly if we include more. However, experimentation with other data sets (8J8M) showed that this observation is not a trend and may be a result of randomness.

Lastly, we showed that the insight obtained through the correlation information presented in the paper can successfully be applied in practice, by using the highly correlated configuration

features to rank a set of possible JSSP instances for a prismatic part manufacturing problem. The results showed that the top/bottom 10 candidates have a scheduling efficiency near the max/min of the actual values, indicating that the features can be used to make a quick but close-to-optimal pick from the thousands of possibilities.

Since the development of an effective set of features is the main contribution of our work, the clearest way to improve upon our results would be to discover or design better features. It is also possible to use feature combination approaches, such as using deep neural networks, to find hard-to-design feature combinations. As we showed in this paper, any new set of features can be tested by implementing it in an inductive learning process. We believe that this paper can promote the idea of using machine learning in scheduling at large, where having better features is almost always helpful. Another way to improve what we did is to employ more advanced feature selection methods, either ones like greedy forward/backward selection which are already present in the literature, or completely new ones suited for this problem. This can help the other aspect of this research, which is being practical by helping predict the optimal makespan of a JSSP instance without having to solve it exactly. Selecting the best set of features can help achieve the best estimation of the true optimal makespan. This practical aspect is also what we will tackle in more detail in another paper.

Appendix A

The following constraint programming code written in IBM ILOG OPL was used to solve the JSSP instances and get the optimal schedule. It is taken from IBM ILOG OPL examples and modified for our purposes. It reads the number of jobs, the number of machines and a matrix of pairs containing the values $p_{jk,}$ and m_{jk} (instead of two separate matrices P and M), and outputs the variable `itvs` which holds the information for the schedule. Some post-processing on this output will result in the schedule as used in this paper.

```
using CP;

int nbJobs = ...;
int nbMchs = ...;

range Jobs = 0..nbJobs-1;
range Mchs = 0..nbMchs-1;

tuple Operation {
    int mch;
    int pt;
};

Operation Ops[j in Jobs][m in Mchs] = ...;

dvar interval itvs[j in Jobs][o in Mchs]
 size Ops[j][o].pt;

dvar sequence mchs[m in Mchs] in all(j in Jobs,
  o in Mchs : Ops[j][o].mch == m) itvs[j][o];

minimize max(j in Jobs) endOf(itvs[j][nbMchs-1]);
subject to {
    forall (m in Mchs)
        noOverlap(mchs[m]);
    forall (j in Jobs, o in 0..nbMchs-2)
        endBeforeStart(itvs[j][o], itvs[j][o+1]);
}
```

Appendix B

Table B.1
Values for all the 380 features, derived from the sample problem given in Table 1.

ID	Value	ID	Value	ID	Value	ID	Value	ID	Value	ID	Value	ID	Value	ID	Value
1	5	*49*	311	*97*	0.3441	*145*	1.294	*193*	1.384	*241*	1.966	*289*	532	*337*	1.052
2	5	*50*	311	*98*	0.2	*146*	1.941	*194*	1.384	*242*	0.1763	*290*	2.171	*338*	0.76
3	1225	*51*	1.057	*99*	1.2	*147*	1.941	*195*	0.2482	*243*	0.1763	*291*	0.32	*339*	0.6
4	49	*52*	1.057	*100*	1.375	*148*	0.3235	*196*	0.2482	*244*	0.9228	*292*	384.8	*340*	0.2608
5	37	*53*	1.196	*101*	1.375	*149*	0.3235	*197*	0.8659	*245*	0.9228	*293*	1.021	*341*	0.6
6	27.78	*54*	1.196	*102*	1.375	*150*	0.9706	*198*	0.8659	*246*	0.8141	*294*	0.3291	*342*	1.2
7	6	*55*	0.4816	*103*	1.375	*151*	0.9706	*199*	1.146	*247*	0.8141	*295*	0.4886	*343*	624
8	96	*56*	0.4816	*104*	0.4583	*152*	0.6471	*200*	1.146	*248*	0.8141	*296*	1.383	*344*	2.547
9	245	*57*	0.9959	*105*	0.4583	*153*	0.6471	*201*	1.146	*249*	0.8429	*297*	460.8	*345*	0.32
10	237	*58*	0.9959	*106*	0.9167	*154*	0.6471	*202*	0.6333	*250*	0.8429	*298*	0.9614	*346*	397.2
11	67.27	*59*	1.269	*107*	0.9167	*155*	0.8	*203*	0.6333	*251*	1.365	*299*	0.0863	*347*	0.9668
12	134	*60*	1.269	*108*	0.9167	*156*	0.8	*204*	0.6827	*252*	1.365	*300*	0.9614	*348*	0.4111
13	316	*61*	1.269	*109*	0.9167	*157*	1.2	*205*	0.6827	*253*	0.1224	*301*	1.155	*349*	0.4733
14	0.9673	*62*	5.286	*110*	0.9167	*158*	1.2	*206*	0.1224	*254*	0.1224	*302*	0.24	*350*	1.571
15	0.2746	*63*	5.286	*111*	0.6	*159*	0.2	*207*	0.1224	*255*	0.6408	*303*	0.2	*351*	562.4
16	0.5469	*64*	5.98	*112*	0.6	*160*	0.2	*208*	0.4272	*256*	0.6408	*304*	0.1673	*352*	0.9726
17	1.29	*65*	5.98	*113*	0.6	*161*	0.6	*209*	0.4272	*257*	0.5653	*305*	0	*353*	0.0612
18	5	*66*	2.408	*114*	0.6	*162*	0.6	*210*	0.5653	*258*	0.5653	*306*	0.4	*354*	0.9726
19	4.837	*67*	2.408	*115*	0.2	*163*	0.4	*211*	0.5653	*259*	0.5653	*307*	547	*355*	1.11
20	1.373	*68*	4.98	*116*	0.2	*164*	0.4	*212*	0.5653	*260*	0.8333	*308*	2.233	*356*	0.44
21	2.735	*69*	4.98	*117*	0.4	*165*	0.4	*213*	34.66	*261*	0.4823	*309*	0.36	*357*	0.2
22	6.449	*70*	6.347	*118*	0.4	*166*	23.82	*214*	31.4	*262*	0.4167	*310*	433.2	*358*	0.3286
23	245	*71*	6.347	*119*	0.4	*167*	27.7	*215*	19.82	*263*	1.667	*311*	1.011	*359*	0.2
24	226	*72*	6.347	*120*	0.4	*168*	9.856	*216*	6	*264*	0.96	*312*	0.214	*360*	0.8
25	43.79	*73*	0.9091	*121*	0.4	*169*	6	*217*	66.9	*265*	0.8	*313*	0.7733	*361*	518
26	214	*74*	0.3402	*122*	1.375	*170*	33.45	*218*	0.9059	*266*	0.463	*314*	1.263	*362*	2.114
27	332	*75*	0.4545	*123*	1.375	*171*	1.163	*219*	0.5718	*267*	0.4	*315*	519.8	*363*	0.16
28	0.9224	*76*	1.364	*124*	1.375	*172*	0.4137	*220*	0.1731	*268*	1.6	*316*	0.9869	*364*	423.8
29	0.1787	*77*	0.44	*125*	1.375	*173*	0.2519	*221*	1.93	*269*	1.364	*317*	0.0292	*365*	1.045
30	0.8735	*78*	0.4	*126*	0.4583	*174*	1.404	*222*	0.7073	*270*	0.4454	*318*	0.9869	*366*	0.1797
31	1.355	*79*	0.1497	*127*	0.4583	*175*	0.4862	*223*	0.6408	*271*	0.4545	*319*	1.052	*367*	0.8023
32	5	*80*	0.2	*128*	0.9167	*176*	0.5653	*224*	0.4045	*272*	1.364	*320*	0.68	*368*	1.222
33	4.612	*81*	0.6	*129*	0.9167	*177*	0.2011	*225*	0.1224	*273*	0.44	*321*	0.8	*369*	394.4
34	0.8937	*82*	0.9091	*130*	0.9167	*178*	0.1224	*226*	1.365	*274*	0.6	*322*	0.3899	*370*	1.123
35	4.367	*83*	0.3402	*131*	0.9167	*179*	0.6827	*227*	41.3	*275*	0.196	*323*	0.2	*371*	0.3007
36	6.776	*84*	0.4545	*132*	0.9167	*180*	31.03	*228*	41.3	*276*	0.2	*324*	1.2	*372*	0.5806
37	67.8	*85*	1.364	*133*	0.6	*181*	31.03	*229*	66.9	*277*	0.6	*325*	552	*373*	1.313
38	0.2767	*86*	0.44	*134*	0.6	*182*	33.45	*230*	66.9	*278*	1.25	*326*	2.253	*374*	0.68
39	1.384	*87*	0.4	*135*	0.6	*183*	33.45	*231*	6	*279*	0.606	*327*	0.36	*375*	0.6
40	259	*88*	0.1497	*136*	0.6	*184*	6	*232*	6	*280*	0.3125	*328*	438.2	*376*	0.3633
41	259	*89*	0.2	*137*	0.2	*185*	6	*233*	31.4	*281*	1.875	*329*	1.011	*377*	0.2
42	293	*90*	0.6	*138*	0.2	*186*	20.93	*234*	31.4	*282*	0.64	*330*	0.2116	*378*	1.2
43	293	*91*	0.9375	*139*	0.4	*187*	20.93	*235*	27.7	*283*	0.8	*331*	0.7759	*379*	2.114
44	118	*92*	0.5376	*140*	0.4	*188*	27.7	*236*	27.7	*284*	0.3878	*332*	1.26	*380*	0.0063
45	118	*93*	0.3125	*141*	0.4	*189*	27.7	*237*	27.7	*285*	0.2	*333*	524.8		
46	244	*94*	1.875	*142*	0.4	*190*	27.7	*238*	1.214	*286*	1.2	*334*	0.987		
47	244	*95*	0.64	*143*	0.4	*191*	1.284	*239*	1.214	*287*	0.8	*335*	0.0289		
48	311	*96*	0.6	*144*	1.294	*192*	1.284	*240*	1.966	*288*	0.8	*336*	0.987		

Appendix C

Table C.1
Full average ranking obtained through the three ranking methods for the two batches.

10J10M								Mixed							
379	374	155	257	122	48	75	263	379	9	18	305	89	104	230	300
326	364	284	98	41	74	79	266	326	303	288	342	38	196	265	352
362	287	288	292	47	140	84	7	362	23	32	293	225	126	139	199
290	303	164	156	162	83	88	186	290	213	121	346	297	147	207	173
327	312	165	322	318	233	92	206	294	223	47	323	181	198	271	246
294	17	202	336	221	149	220	353	327	287	166	69	251	37	264	365
333	12	99	228	316	157	193	51	348	215	143	257	79	161	159	108
308	11	214	225	300	254	54	184	308	176	274	203	88	216	272	299
348	22	211	203	181	102	65	62	291	313	214	189	333	244	146	130
325	224	236	293	195	197	107	115	296	20	339	322	325	105	91	152
309	320	212	163	45	319	76	151	371	77	375	228	100	127	239	317
291	343	237	285	298	124	129	246	366	34	78	13	279	149	22	36
380	306	120	334	216	146	187	137	380	86	87	156	122	334	315	267
344	77	363	347	148	40	365	152	309	320	314	285	178	329	192	351
361	176	121	340	251	229	85	199	344	273	170	68	341	278	206	354
296	86	142	109	255	14	207	264	302	16	46	324	370	106	353	171
289	332	274	210	337	19	311	261	295	331	44	66	377	245	240	260
297	351	143	110	104	91	232	173	222	217	376	347	218	128	116	73
302	273	5	131	126	241	253	174	15	3	45	67	361	254	102	268
222	321	378	132	279	299	182	66	29	374	49	71	204	114	138	82
371	304	111	80	281	18	280	55	330	321	111	63	275	136	124	369
366	313	78	178	44	230	39	108	345	4	164	210	188	107	307	7
345	310	133	112	188	32	38	68	12	249	133	163	247	129	319	51
328	179	87	89	205	117	271	70	349	259	41	65	336	169	311	301
175	25	166	134	33	139	239	130	372	338	81	238	153	158	318	57
356	249	359	238	218	208	301	57	368	363	378	276	28	150	364	74
295	283	97	235	28	103	245	59	373	286	43	70	109	209	266	83
282	367	180	377	161	125	116	262	175	96	90	112	131	103	93	187
349	286	376	153	169	272	138	231	356	283	50	72	256	253	115	53
357	217	305	154	252	42	268	265	31	177	236	221	92	335	137	298
20	331	314	247	234	244	159	93	312	30	48	134	242	125	232	55
369	338	189	248	13	160	194	73	26	284	211	248	157	337	316	328
15	96	81	276	196	354	145	82	350	33	359	292	200	8	183	186
315	258	190	144	346	106	58	260	224	227	358	98	229	194	193	60
95	259	360	191	46	128	69	60	332	25	42	110	255	208	101	269
330	215	90	200	317	352	8	61	282	165	277	64	172	14	123	59
368	24	170	201	209	114	37	172	21	11	40	132	148	118	58	39
36	4	324	119	243	158	52	71	226	99	168	154	252	174	75	61
31	9	358	335	147	240	63	72	357	202	190	62	162	234	145	185
307	3	250	141	204	136	171	267	95	19	180	27	243	182	84	52
350	23	177	256	49	113	101	355	304	155	120	235	241	310	76	56
372	10	167	219	329	135	123	185	17	212	142	119	197	140	85	184
34	21	342	370	50	183	53	1	306	237	340	201	281	94	270	54
29	26	35	242	278	198	64	2	24	97	219	141	195	160	231	262
223	227	30	27	270	150	67		367	258	250	191	113	151	263	
226	16	277	100	105	43	56		35	1	360	80	220	233	343	
213	339	323	341	118	275	269		179	2	167	144	135	261	355	
373	375	6	168	127	94	192		10	5	6	289	205	117	280	

References

Aytug, H., Bhattacharyya, S., Koehler, G. J., & Snowdon, J. L. (1994). Review of machine learning in scheduling. *IEEE Transactions on Engineering Management, 41*, 165–171.

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*, 27–45.

Błażewicz, J., Pesch, E., & Sterna, M. (2000). The disjunctive graph machine representation of the job shop scheduling problem. *European Journal of Operational Research, 127*(2), 317–331.

Błażewicz, J., Domschke, W., & Pesch, E. (1996). The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research, 93*(1), 1–33.

Branke, J., Nguyen, S., Pickardt, C., & Zhang, M. (2015). Automated design of production scheduling heuristics: A review. *Evolutionary Computation, IEEE Transactions on*.

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Qu, R. (2013). Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society, 64*, 1695–1724.

Burke, E. K., MacCarthy, B. L., Petrovic, S., & Qu, R. (2003). Knowledge discovery in a hyper-heuristic for course timetabling using case-based reasoning. *Practice and Theory of Automated Timetabling IV, Lecture Notes in Computer Science, 2740*, 276–287.

Demirkol, E., Mehta, S., & Uzsoy, R. (1998). Benchmarks for shop scheduling problems. *European Journal of Operational Research, 109*, 137–141.

Fox, M. S., & Smith, S. (1984). The Role of Intelligent reactive processing in production management. In *Proceedings of 13th Annual CAMI Technical Conference*.

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

Gonzalez, T., & Sahni, S. (1978). Flowshop and jobshop schedules: Complexity and approximation. *Operations Research, 26*(1), 36–52.

Hochbaum, D. S., & Shmoys, D. (1987). Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM, 34*(1), 144–162.

Ingimundardottir, H., & Runarsson, T. P. (2012). Determining the characteristic of difficult job shop scheduling instances for a heuristic solution method. In *Learning and Intelligent Optimization: 6th International Conference, LION 6, Paris, France, January 16–20, 2012, Revised Selected Papers* (pp. 408–412). Berlin, Heidelberg: Springer Berlin Heidelberg.

Jain, A. S., & Meeran, S. (1999). Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research, 113*, 390–434.

Joachims, T. (1999). Making large-scale svm learning practical. *Advances in Kernel Methods - Support Vector Learning*. MIT-Press.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., & Shmoys, D. B. (1993). Sequencing and scheduling: Algorithms and complexity. In *Handbooks in Operations Research and Management Science: Vol. 4* (pp. 445–552). Amsterdam: North-Holland: Logistics of Production and Inventory.

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

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

Lenstra, J. K., & Rinnooy Kan, A. H. G. (1979). Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics, 4*, 121–140.

Li, L., Zijin, S., Jiacheng, N., & Fei, Q. (2013). Data-based scheduling framework and adaptive dispatching rule of complex manufacturing systems. *The International Journal of Advanced Manufacturing Technology, 66*(9-12), 1891–1905.

Li, X., & Olafsson, S. (2005). Discovering Dispatching rules using data mining. *Journal of Scheduling, 8*(6), 515–527.

McCallum, A. (2002). Efficiently inducing features of conditional random fields. In *Proceedings of the 19th conference on Uncertainty in Artificial Intelligence (UAI'03)* (pp. 403–410).

Mirshekarian, S., & Šormaz, D. N. (2015). Understanding the job-shop scheduling problem structure using supervised machine learning. In *Proceedings of the 2015 INFORMS Workshop on Data Mining and Analytics*.

Mirshekarian, S.. *Online job-shop scheduling visualization* accessed http://mirshekarian.me .

Nguyen, S., Zhang, M., Johnston, M., & Tan, K. C. (2013). A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *Evolutionary Computation, IEEE Transactions on, 17*(5), 621–639.

Nguyen, S., Zhang, M., Johnston, M., & Tan, K. C. (2013). Learning iterative dispatching rules for job shop scheduling with genetic programming. *The International Journal of Advanced Manufacturing Technology, 67*(1-4), 85–100.

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.

Pickardt, C. W. (2013). *Evolutionary methods for the design of dispatching rules for complex and dynamic scheduling problems* PhD thesis. Coventry, England: University of Warwick.

Pinedo, M. L. (2008). *Scheduling: Theory, Algorithms, and Systems*. American Society of Mechanical Engineers University of Warwick is in Coventry, England.

Priore, P., de la Fuente, D., Puente, J., & Parreño, J. (2006). A comparison of machine-learning algorithms for dynamic scheduling of flexible manufacturing systems. *Engineering Applications of Artificial Intelligence, 19*(3), 247–255.

Priore, P., de la Fuente, D., Gomez, A., & Puente, J. (2001). A review of machine learning in dynamic scheduling of flexible manufacturing systems. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 15*(3), 251–263.

Smith-Miles, K. A., James, R. J. W., Giffin, J. W., & Tu, Y. (2009). A knowledge discovery approach to understanding relationships between scheduling problem structure and heuristic performance. *Learning and Intelligent Optimization, Lecture Notes in Computer Science, 5851*, 89–103.

Smith-Miles, K. A., van Hemert, J., & Lim, X. Y. (2010). Understanding TSP difficulty by learning from evolved instances. In *Learning and Intelligent Optimization: 4th International Conference, LION 4, Venice, Italy, January 18–22, 2010. Selected Papers: 6073* (pp. 266–280). Berlin, Heidelberg: Springer Berlin Heidelberg.

Šormaz, D. N., & Khoshnevis, B. (2003). Generation of alternative process plans in integrated manufacturing systems. *Journal of Intelligent Manufacturing, 14*(6), 509–526.

Šormaz, D. N., Arumugam, J., Harihara, R. S., Patel, C., & Neerukonda, N. (2010). Integration of product design, process planning, scheduling, and FMS control using XML data representation. *Robotics and Computer-Integrated Manufacturing, 26*(6), 583–595.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research, 64*, 278–285.

Thesen, A., & Lei, L. (1986). An expert system for scheduling robot in a flexible electroplating system with dynamically changing work loads. In *Flexible Manufacturing Systems: Operation Research Models and Applications* (pp. 555–566).

Yih, Y. (1990). Trace-driven knowledge acquisition (TDKA) for rule-based real time scheduling systems. *Journal of Intelligent Manufacturing, 1*(4), 217–229.

Zhang, T. (2009). Adaptive forward-backward greedy algorithm for sparse learning with linear models. *Advances in Neural Information Processing Systems, 21*, 1921–1928.

Zhang, T., & Rose, O. (2013). Intelligent dispatching in dynamic stochastic job shops. In *Proceedings of Winter Simulation Conference (WSC)* (pp. 2622–2632).