Int. J. Production Economics 128 (2010) 118–126

Contents lists available at ScienceDirect

Int. J. Production Economics

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

Learning effective new single machine dispatching rules from optimal scheduling data

Sigurdur Olafsson *, Xiaonan Li

Department of Industrial and Manufacturing Systems Engineering, 3004 Black Engineering, Iowa State University, Ames, IA 50011, USA

ARTICLE INFO

Article history:
Received 13 November 2009
Accepted 8 June 2010
Available online 15 June 2010

Keywords:
Scheduling
Dispatching rules
Data mining
Classification
Decision trees

ABSTRACT

The expertise of the scheduler plays an important role in creating production schedules, and the schedules created in the past thus provide important information about how they should be done in the future. Motivated by this observation, we learn new scheduling rules from existing schedules using data mining techniques. However, direct data mining of scheduling data can at best mimic existing scheduling practices. We therefore propose a novel two-phase approach for learning, where we first learn which part of the data correspond to best scheduling practices and then use this data and decision tree induction to learn new and previously unknown dispatching rules. Our numerical results indicate that the newly learned rules can be a significant improvement upon the underlying scheduling rules, thus going beyond mimicking existing practice.

1. Introduction

Scheduling is a key economic activity in many production systems; and over the past several decades a myriad of operations research models have been proposed to solve scheduling problems for specific production environments (Pinedo, 1995). However, in a real production system it is often difficult to account for all of the constraints and competing objectives using such a modeling approach (Hestermann and Wolber, 1997; Wiers, 1997). In practice scheduling is often more ad-hoc, and does not rely solely on well-defined rules or principles but also on the intuition and experience of the scheduler. For example, in an empirical study, Berglund and Karitun (2007) found that schedules in real production systems are influenced by non-automated human capabilities of the scheduler, problem-solving when the technical systems fail, and negation between groups with competing objectives. In recent years, models have been developed for production scheduling that account for the human and organizational factors often ignored (Jackson et al., 2004; McKay, 1999).

We do not propose a new model for scheduling, but our work is motivated by the importance of the human scheduler. Indeed, the premise of this paper is that as a complement to traditional model-based methods the expertise and experience of the scheduler provides valuable data on how scheduling should be done. Following this premise, scheduling rules for a given system could be based on how the system has been successfully scheduled in the past, regardless of whether the underlying principles are well understood. To that end, we propose a novel two-phased data mining approach to learning how to dispatch jobs that is based on formulating a classification problem for dispatching. Namely, given two jobs we want to learn from existing schedules how to predict which job should be dispatched first. Using this classification problem, we first learn which instances in the training data (based on past schedules) provide the best data for such learning, that is, correspond to the best scheduling practices, and then use this data to learn new dispatching rules.

The remainder of the paper is organized as follows. In Section 2 we provide background to the relevant scheduling and data mining issues. In Section 3 we show how to learn dispatching rules from data, and in Section 4 we show how to improve this learning by selecting the appropriate data before the actual learning starts. In Section 5 we evaluate the new approach on a simulated single-machine system, and Section 6 contains some concluding remarks.

2. Background

2.1. Production scheduling notation and terminology

In this section we review the notations and terminology associated with the scheduling task of sequencing a set of jobs in order to optimize some scheduling performance measure (Pinedo, 1995). Given a set of n jobs, each job j is assumed to have certain known data, such as its processing time p_j, the release time r_j

* Corresponding author. Tel.: +1 515 294 8908.
E-mail address: olafsson@iastate.edu (S. Olafsson).

0925-5273/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.ijpe.2010.06.004

when the job is available for processing, the due date d_j of the job, and the weight w_j of the job, which measures its importance. Based on this data a schedule is created, which results in the completion time C_j of each job becoming known.

The performance of the schedule is usually represented in terms of the given data and the completion time. Common performance measures include the makespan of the schedule, that is the completion time of the last job $C_{max}=\max_j C_j$, the total tardiness $\sum_{j=1}^{n}\max\{C_j-d_j,0\}$ of the schedule, or its maximum lateness $L_{max}=\max_j(C_j-d_j)$. Many performance measures could also be weighted, for example, in this paper we will study systems where the weighted maximum lateness $L_{max}^{weight}=\max_j w_j(C_j-d_j)$ will be the measure of interest.

Optimal and heuristic solution algorithms have been extensively studied for these and other objective functions for many production environments. Classic scheduling environments include single machine systems, parallel machine system, job shops, and flow shops. Even though such environments are somewhat abstract and do not contain the complexities of real production systems, studying them may reveal insights into how real systems should be scheduled.

2.2. Data mining and classification

Our work falls within the broad category of data mining for manufacturing, which has been the subject of several recent surveys (Choudhary et al., 2009; Harding et al., 2006; Wang et al., 2007). Several basic data mining concepts will therefore be needed and those are defined in this section. Data mining is the automated learning of patterns from data, and may include integration of data from heterogeneous databases, preprocessing of the data, and induction of a model with a learning algorithm (Han and Kamber, 2006). Regardless of the learning algorithm all data mining starts with a set of data called the training set, which consist of instances describing the observed values of certain variables. These instances are then used to learn a given target concept or pattern. This includes classification, where the training data is labeled, that is, each instance is identified as belonging to one of the two or more classes; and an inductive learning algorithm is used to create a model that discriminates between those class values. The model can then be used to classify any new instance according to this class variable. The primary objective is usually for the classification to be as accurate as possible, but this is not the only relevant quality measure. The interpretability of the results is also important and this can be directly related to the complexity of the resulting classification algorithm. This distinction – accuracy versus complexity – becomes important as we consider different techniques for classification.

Decision trees are a popular and relatively simple technique for classification. Decision tree induction algorithms construct a tree in a top-down manner by selecting variables one at a time and splitting the data according to the values of those variables. The most important variable is selected as the top split node, the next most important variable is considered at the next level, and so forth. For example, in the popular C4.5 algorithm (Quinlan, 1993), variables are chosen to maximize what is called the information gain ratio of the split. For our purpose in this paper the interpretability of the trees is important, which leads us to a focus on the size of the tree. Large trees are often difficult to interpret, so in addition to measuring the quality of decision trees by their accuracy, we will also account for the number of nodes in the tree, with preference for smaller trees with fewer nodes.

2.3. Data mining in production scheduling

Data mining, and before that expert systems and machine learning, is not new in scheduling; and such methods have received considerable attention over the past several decades. The earliest work focused primarily on expert system for scheduling (Kanet and Adelsberger, 1987; Kusiak and Chen, 1988). Other early examples include neural networks (Jain and Meeran, 1998), induction (Shaw et al., 1992), case-based reasoning (Schmidt, 1998), and unsupervised learning (Bowden and Bullington, 1996). Of these various methods, the works that are the most related to the present paper are inductive learning methods that have for example been used to select between several dispatching rules (Nakasuka and Yoshida, 1992; Shaw et al., 1992; Piramuthu et al., 1994). Surveys of this literature can be found in Aytug et al. (1994) and Priore et al. (2001).

Inductive learning in production scheduling has primarily been devoted to issues such as selecting the best dispatching rule by learning from simulated data. This assumes that all the dispatching rules are known in advance and that the performance of these rules can be accurately simulated. A notable exception to this is the recent work of Geiger et al. (2006), where a genetic algorithm is used to discover new dispatching rules in a flow shop environment, and Li and Olafsson (2005) where data mining is applied directly to production data in order to discover new and interesting dispatching rules. More recently, several authors have used similar approaches to learn rules for specific scheduling environments. For example, Malik et al. (2008) and Russell et al. (2009) use inductive learning to generate new heuristics for block instruction scheduling, and Baykasoglu et al. (2008) use inductive learning to generate new rules for job shop scheduling.

The present paper builds on the work of Li and Olafsson (2005). The limitation of that work is that learning directly from existing schedules simply mimics the principles used to schedule in the past and then applies those to schedule future jobs. This means that both good and bad scheduling practices will be mimicked. In this paper we address this limitation and show how very effective dispatching rules can be induced from scheduling data by combining decision tree induction with instance selection. The basic idea is to select the instances that result in the decision trees with the best scheduling performance, which can thus be interpreted as corresponding to the best scheduling practices. Before continuing with this idea we first review the basics of the prior work.

3. Learning dispatching rules

In a previous paper we show how data mining on production data can be used to learn dispatching rules and capture both implicit and explicit knowledge (Li and Olafsson, 2005). The novelty of this approach is to apply data mining directly to production data in order to discover previously unknown meaningful patterns, such as new and interesting dispatching rules. The potential benefits are that (a) the implicit knowledge of expert schedulers is discovered and can be used to generate future schedules with little or no direct involvement of such experts; and (b) existing scheduling practices are generalized into explicit scheduling rules. These rules can then be applied to both situations that have occurred before and to new scenarios.

As noted above, data mining is concerned with learning some target concept; and the first step of applying data mining to planning and scheduling is therefore to identify what concepts are of interest. These concepts are determined by what decision we want to learn how to make, and in scheduling, such decisions include routing of jobs, line balancing, batching, allocating

resources, and sequencing. A common scheduling decision is to sequence jobs. Reducing the decisions involved in constructing a sequence to a simple concept, we note that if for any two jobs we can determine a priority, that is, which job should come first a complete sequence could be constructed. Thus, the target concept can be taken as if a job comes ahead of another job. Given this target concept a classification problem can be defined through a pair-wise comparison of jobs. That is, given Job 1 and Job 2, and instance in the training set for the classification problem includes all of the production data for those two jobs along with a class attribute that indicates whether Job 1 or Job 2 is scheduled first. (See Section 4.2 below for an example.) For a set of n jobs, the first job is therefore compared to the other $n-1$ jobs, the second job to the remaining $n-2$ jobs, and so forth; and a dispatching list for n jobs is converted into a training dataset with $\sum_{j=1}^{n-1}(n-j)$ instances.

Given this classification problem, a learning algorithm is applied to induce a predictive model based on the training data. This model discriminates between the class values (that is, whether a job should be scheduled ahead of another job or not) based on the independent variables (that is, the system data relevant to the schedule, such as processing time, due dates, and release times of the jobs). Many methods have been found effective for such classification (Han and Kamber, 2006), but when learning how to dispatch jobs the desired outcome is a specific type of pattern, namely decision rules that can be easily interpreted and applied by the scheduler.

Based on the need for a set of transparent dispatching rules, the primary choice for classification algorithm is between either methods that learn decision rules directly or decision tree induction methods such as C4.5 (Quinlan, 1993), which allow decision rules to be read from the tree that is learned. Here we choose to use C4.5, which is one of the widely used decision tree methods, but any other rule or decision tree induction method could be used instead.

4. Improving the learning by identifying best practices

As was pointed out above, a limitation to learning directly from production scheduling data is that the learning algorithm will learn from poor scheduling practices as well as good scheduling practices, which may simply lead to making poor practices explicit. It would therefore be beneficial if the inductive learning phase would be preceded by the identification of best practices and the corresponding data instances. Such identification of best practices is, of course, non-trivial and important in its own right. Our premise is that such best practices can be implicitly identified by selecting the data instances that result in the best classification models.

4.1. Selecting the best scheduling data

It is indeed intuitively appealing that we should only learn from best scheduling practices, but what is not straightforward is how to determine those best practices. As the ultimate goal here is to learn the best set of decision rules for dispatching, we propose a corresponding indirect approach, that is, to select the data from which we learn based on how good the rules are that result when the selected classification algorithm is applied.

Thus, rather than defining a measure of the quality of each scheduling decision, that is, a measure that would evaluate how well each job was scheduled, we define a quality measure for a subset A of the training data. Specifically, say that based on using only the data in A, a decision tree $Tree(A)$ is induced using the selected learning algorithm. (In our numerical results we always use the C4.5 algorithm, but in general any learning algorithm can be used.) This tree can then be converted into decision rules and those decision rules can be used to dispatch any new set J of jobs, resulting in a dispatching list $Tree(A)(J)$. For a specific system there will be some scheduling measure, or multiple measures of interest (e.g., makespan, total weighted tardiness, and maximum lateness). We let f denote this scheduling performance, so $f(Tree(A)(J))$ is the performance of dispatching list when jobs J are scheduled according to the decision rules in $Tree(A)$. Thus, $f(Tree(A)(J))$ is a measure of the quality of the subset A.

However, simply finding the subset A that optimizes the scheduling performance $f(Tree(A)(J))$ may not result in an easily implementable dispatching rule. Decision trees often grow excessively large, which would make them difficult to implement, and lack the transparency that is desirable when it is deployed in practice. Specifically, we note that the number of leaf nodes in the tree corresponds to the number of decision rules in the overall dispatching policy, and the total number of nodes is the total length of the rules needed to schedule arbitrary jobs. Thus, we let $size(Tree(A))$ denote the number of nodes in the tree and observe that this is another, complementary, measure of the quality of the subset A.

Based on the above observations, we suggest that the best subset A of training instances, which therefore should be used to learn a dispatching rule, is found by solving the following optimization problem, where the weights $w_1^{(obj)}$ and $w_2^{(obj)}$ scale the two different components of the objective function and provide an ability to weigh their relative importance:

$$\min_A w_1^{(obj)} size(Tree(A)) + w_2^{(obj)} f(Tree(A)(J)).$$

The number of possible subsets is $2^{\sum_{j=1}^{n-1}(n-j)}$, where n is the total number of jobs, as the dispatching list for n jobs is converted into a training set with $\sum_{j=1}^{n-1}(n-j)$ instances. It is therefore clear that even for moderately many jobs the search space will be huge; and it is necessary to employ some type of heuristic for finding a good solution.

4.2. Genetic algorithm solution

To solve the instance selection optimization problem formulated in the previous section, we use the genetic algorithm (GA) approach of Wu (2007), who proposes similar instance selection to improve decision trees where the objective is simple trees with high classification accuracy. Thus, the same algorithm can be used but with our proposed performance measure of schedule quality replacing the accuracy of the decision tree. As the focus of this paper is not on appropriate solution method to the optimization problem, but rather the value of solving the problem, we refer the interested reader to Wu (2007) for more complete details and experimental results for the algorithm; as well as more discussion of instance selection for improving decision trees.

The first step in applying the GA is to define the solution space in terms of what is usually referred to as the chromosomes. In most GA applications the chromosomes are binary strings, but here we take a slightly different approach and let y_i denote the position of the ith training instance in the training dataset T. The chromosomal unit of each subset is thus defined as a vector $C=[y_1,y_2,\ldots,y_N]$ of integers, where N is the number of training instances in a subset and represents the length of the chromosomal unit.

Given this representation, we start by obtaining an initial population $P_0=\{C_1^{(0)},C_2^{(0)},\ldots,C_M^{(0)}\}$ of chromosomes as follows. Given a total of $n=|T|$ instances, we divide these instances into a fixed number of M subsets of size $N=n/M$ by sampling the training set N times with replacement to generate $C_1^{(0)}$; and then

repeating the sampling process a total of M times to generate the remaining chromosomes. Now, starting with this initial population, the usual GA operations of selection, crossover, and mutation are applied to improve the population. These operations are defined as follows. In the kth step of the GA search, that is, the kth generation, the chromosomes of the current population P_k is ranked according to the fitness, that is

$$f(S(C_{[1]}^{(k)})) \le f(S(C_{[2]}^{(k)})) \le \ldots \le f(S(C_{[M]}^{(k)}))$$

where $S(C_{[j]}^{(k)})$ denotes the set of all instances in the subset chromosome $C_{[j]}^{(k)}, j=1,2,\ldots,M$. The $(1-c)M$ fittest ones are selected into the next generation, where c is the crossover rate. The crossover operator probabilistically selects $cM/2$ pairs from P_k, chooses two random points and then swaps the parts between crossover points among parent chromosomes. Meanwhile the instances corresponding to the position numbers will be exchanged between these two subsets. In mutation, if the element g_i in the chromosomal unit is chosen to be mutated (the probability of mutation is determined by the mutation rate m), a new random number g'_i is to be generated uniformly from $\{1,2,\ldots,|T|\}$ and then used to replace g_i. Furthermore, the instance in position g_i will be replaced by the new instance in position g'_i.

The GA operations are repeated for a given number of g generations, resulting in a final population $P_0=\{C_1^{(g)},C_2^{(g)},\ldots,C_m^{(g)}\}$, which is ranked as before. The solution A is then found from this final population by selecting all of the instances that are contained in the best subset of the final generation, that is, $A=S(C_{[1]}^{(g)})$. Finally, we note that the key parameters of the algorithm are the number of subsets m, number of generations g, crossover rate c, and mutation rate m_u.

4.3. An illustrative example

We illustrate the new approach by applying it to schedule a simulated single-machine system with ten jobs and a maximum lateness objective (Hall and Shmoys, 1992); that is, each of the ten jobs has a due date and the objective is to find the sequence that results in the minimum weighted maximum lateness among all the jobs. We choose this objective function as a relatively large number of job parameters are potentially relative, including the release time, due date, processing time, and weight of the job; and this make for potentially more complex and interesting decision trees being learned from the data. A myriad of studies has addressed similar single machine scheduling problems where a due date should be met for each job. In such studies the objective function is either assumed to be to minimize lateness, tardiness, earliness, or a combination of measures (e.g., Chen and Sheen, 2007; Schaller, 2007). Furthermore, many such studies assume that the jobs have different release dates as well as due dates (e.g. Chang et al., 2006). For each of the ten jobs we thus generate a release time, due date, processing time, and weight (Table 1).

Table 1
An example of ten jobs dispatched to be processed on a single machine.

Job ID	Release time	Due date	Processing time	Weight	Sequence
1	2	24	7	5	2
2	0	7	3	3	1
3	15	10	4	1	8
4	5	36	18	3	9
5	3	25	6	3	7
6	7	20	2	1	10
7	8	11	12	3	3
8	12	15	5	3	4
9	9	29	9	5	6
10	6	39	17	7	5

There is a simulated scheduler who follows a kind of weighted Earliest Due Date (EDD) for all released job rules when dispatching the jobs. In other words, the simulated scheduler looks at all of the available jobs and selects the job with the smallest value of the ratio w_j/d_j. This results in the sequence shown in the last column of Table 1. This underlying scheduling principle is unknown by the data mining algorithms; but when the decision tree algorithm is employed to mimic the existing scheduling practice, we expect a successful result to be a decision tree involving a combination of release time, weight, and due dates. We choose this rule as the underlying principle of the simulated scheduler since it implies that the scheduler is following an intelligent rule, but yet it is not guaranteed to be optimal so there is room for improvement to be found by our data mining approach. In fact, the EDD rule is known to be optimal for the single-machine maximum lateness problem when the release times and weights are constant; but the general problem is NP-complete and there is thus no simple optimal dispatching rule (Hall and Shmoys, 1992). However, the method in which the simulated scheduler incorporates both release time and weight consideration is intuitively appealing.

The key issue to be addressed by this simulated single-machine example is whether it is potentially beneficial to use instance selection to identify the instances corresponding to best practices. We are thus interested in evaluating the value of instance selection, that is, to learn only from best practices, versus to mimic existing scheduling practice based on all of the scheduling data; as well as to compare the scheduling performance of the dispatching rules that result from decision tree induction (with or without instance selection) to the performance of the underlying dispatching rule use by the simulated scheduler.

This dataset is transformed into a training dataset as described in Section 3 and the results are partially shown in Table 2. In this approach, every training instance consists of a comparison between two jobs and a class label indicating which job is dispatched first in the actual schedule. Thus, each row in the training data includes the release time (r_j), due date (d_j), processing time (p_j), and weight (w_j) for each of the two jobs, as well as the class attribute "Job 1 Scheduled First," which is a binary variable taking values "yes" or "no." Note that even for just a ten job problem, the number of instances in the training set is $\sum_{j=1}^{n-1}(n-j)=45$, and the number of possible instance subsets is 2^{45}; that is, the search space for finding the optimal instance subset is huge even for a very small problem.

In addition to these original attributes, we create four derivative indicator attributes that indicate if the first job in the row is released earlier, due earlier, has lower processing time and/or higher weight. While it does not add any new data, Li and Olafsson (2005) found the inclusion of such derivative attributes to be very helpful in subsequent learning and suggested an automated method for their generation. The ideal derivative attribute would undoubtedly be the ratio of weight and due date, which is what the simulated scheduler uses as the primary discriminator, but this is unknown by the learning algorithm.

The dataset in Table 2 is taken as the training dataset, and an independent test dataset is generated with same distributions and scheduling rule (EDD) as for the training data. The test data can therefore be viewed as new data collected from the same system and used to independently evaluate the performance of the decision tree model. The dispatching list for the test data is shown in Table 3.

Now any classification algorithm can be applied directly to the training data (Table 2) and evaluated on the test data similarly derived from Table 3. We employ a standard C4.5 decision tree induction algorithm using the Weka data mining software (Hall et al., 2009); and the default parameters of the algorithm were not changed. This results in the decision tree of Fig. 1. This decision

Table 2
A partial training dataset generated from the ten-job single-machine example.

Job1	r_1	d_1	p_1	w_1	Job2	r_2	d_2	p_2	w_2	Job1 release earlier	Job1 due earlier	Job1 PT lower	Job1 weight higher	Job1 sched. first
1	2	24	7	5	2	0	7	3	3	no	no	no	yes	no
1	2	24	7	5	3	15	10	4	1	yes	no	no	yes	yes
1	2	24	7	5	4	5	36	18	3	yes	yes	yes	yes	yes
1	2	24	7	5	5	3	25	6	3	yes	yes	no	yes	yes
1	2	24	7	5	6	7	20	2	1	yes	no	no	yes	yes
⋮														⋮
9	9	29	9	5	10	6	39	17	7	no	yes	yes	no	no

The training data includes all pair-wise comparisons between jobs, and the complete training data therefore have 45 instances.

Table 3
An independent test schedule for the ten-job single-machine problem.

Job_ID	Release	Due date	Processing	Weight	Sequence
1	11	20	7	3	8
2	3	24	6	5	6
3	10	27	8	7	4
4	4	57	11	3	9
5	8	9	7	3	3
6	0	49	13	3	1
7	14	29	10	5	7
8	2	7	12	3	2
9	9	21	15	5	5
10	5	44	4	1	10

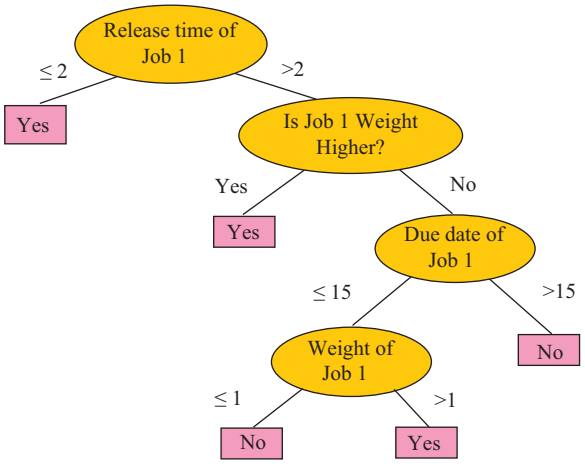

Fig. 1. Decision tree for the 10-job single-machine example where the learning algorithm uses all of the data. As expected this tree uses a combination of release time, weight, and due date, which is all the data used by the simulated scheduler.

tree can then be transformed into rules (e.g., if the release time of Job 1 is less than or equal to two, then Job 1 will be dispatched first), and these rules can then be applied to schedule any new data. In particular, by applying these rules to schedule the test dataset shown in Table 4 we can obtain a sequence as follows. When comparing Job 1 and Job 2 we note that the release time of Job 1 is greater than 2 ($r_1 > 2$), the weight of Job 1 is not higher ($w_1 < w_2$), and the due date of Job 1 is greater than 15 ($d_1 > 15$), so the classification is "No" according to the right-most leaf node in the tree. This implies that Job 1 does not come ahead of Job 2 in the sequence. Similarly, comparing Job 1 and Job 10, we note that the release time of Job 1 is greater than 2 ($r_1 > 2$) and the weight of Job 1 is higher ($w_1 > w_{10}$) so the classification is "Yes" according to the second leaf node from the left. This implies that Job 1 does come ahead of Job 10 in the sequence. Thus we know

Table 4
The instances selected by the genetic algorithm heuristic.

Job 1	r_1	d_1	p_1	w_1	Job2	r_2	d_2	p_2	w_2	Job1 first?
$\lvert w_1 - w_2 \rvert = 0$										
2	0	7	3	3	8	12	15	5	3	yes
4	5	36	18	3	5	3	25	6	3	no
4	5	36	18	3	8	12	15	5	3	no
7	8	11	12	3	8	12	15	5	3	yes
$\lvert w_1 - w_2 \rvert > 0$										
1	2	24	7	5	5	3	25	6	3	yes
3	15	10	4	1	4	5	36	18	3	yes
3	15	10	4	1	7	8	11	12	3	no
4	5	36	18	3	6	7	20	2	1	yes
9	9	29	9	5	10	6	39	17	7	no
6	7	20	2	1	10	6	39	17	7	no

For the first four instances the weight of the two jobs is equal and the sequence depends only on the due date. For the next six instances the weights are different and the weight is the most important factor in sequencing the jobs.

that these three jobs are ordered as follows: Job 2 – Job 1 – Job 10. Continuing in the same manner, we can compare as many jobs as necessary to obtain a complete sequence of Job 3 – Job 2 – Job 5 – Job 6 – Job 7 – Job 8 – Job 9 – Job 4 – Job 1 – Job 10. It is important to note that this does not required comparing all jobs to all other jobs, as many efficient sorting algorithms utilize pair-wise comparisons, which is precisely what is given by the decision tree. The final sequence obtained has weighted maximum lateness of

$$L_{\max}^{weight} = \max_{j \in \{1,2,\ldots,10\}} w_j \ \max\{0, C_j - d_j\} = w_9(C_9 - d_9) = 300.$$

That is, Job 9 has the highest weighted lateness. This does not compare favorably to the sequence obtained by the original EDD rule, where $L_{max}^{weight} = 210$. Indeed, we do not expect the new rule to be better than EDD as the decision tree induction algorithm is only attempting to mimic the weighted EDD for all released jobs rule, not improve upon it.

We now use the GA-based instance selection to heuristically find a subset of instances before we apply decision tree induction to the selected instance subset. The parameters of the GA algorithm were set as follows. The number of subsets is set to $m=6$, number of generations $g=30$, crossover rate $c=0.6$, and mutation rate $m_u=0.05$. The weights in the objective function are set to one, that is, $w_1^{(obj)} = w_2^{(obj)} = 1$, which due to the scale of the two measures thus places more importance on the scheduling performance than the size of the tree. After again applying the standard C4.5 algorithm using Weka, this results in the decision tree shown in Fig. 2. Applying the rules read from this decision tree to dispatch the jobs in the test dataset, that is, by sorting the jobs while performing pair-wise comparison using the new decision tree, we obtain the following sequence: Job 9 – Job 7 – Job 3 – Job 2 – Job 8 – Job 5 – Job 1 – Job 6 – Job 4 – Job 10. The

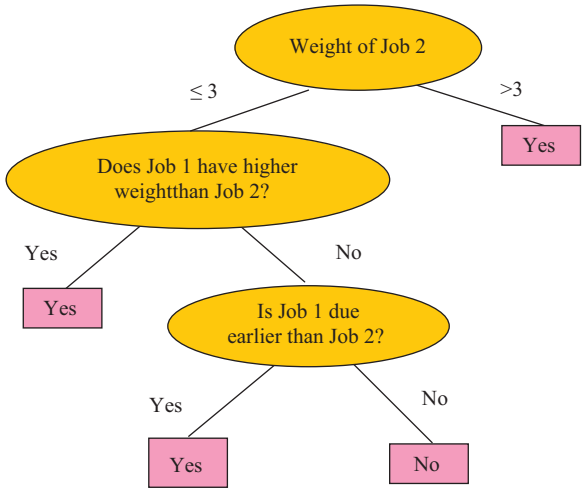

Fig. 2. Decision tree for the 10-job single-machine example when instance selection is used to identify which data should be used for the learning. Unlike the tree in Fig. 1, this tree does not use release time information to sequence the jobs.

performance (weighted maximum lateness) of this sequence is

$$L_{max}^{weight} = \max_{j \in \{1,2,\ldots,10\}} w_j \max\{0, C_j - d_j\} = w_5(C_5 - d_5) = 174.$$

That is, Job 5 has the highest weighted lateness, which is 42% smaller than the $L_{max}^{weight} = 300$ obtained by the previous decision tree; and 17% better than the previous best value of $L_{max}^{weight} = 210$ obtained by the weighted EDD for released jobs rule. Thus, by first learning which data corresponds to the best scheduling practices, and then learning to dispatch only from this data, rather than all of the scheduling data, we are able to significantly improve on the practice of the current (simulated) scheduler.

It is also instructive to consider the selected set of instances, which is believed to contain the best practices, as well as to compare the sequence obtained by the new dispatching rules to the sequence obtained by the simulated scheduler. First consider the selected instances (see Table 4). For interpretation purposes, the selected instances have been separated into two groups depending on the weight difference $|w_1 - w_2|$ of the two jobs being compared. For the first four jobs, the weight is equal and we note that the sequence follows the EDD rule. However, for the remaining six jobs, where the weights of the jobs are different, the sequence is strictly EDD only for the first two jobs. However, for five of these six jobs, the job with the higher weight is scheduled first. The one exception is the sixth instance (comparing Job 3 with Job 4), where EDD rule holds. The use of due date and weight comparisons is noteworthy, but this set of instances is also noteworthy for what is not there. In particular, there is no instance where release time plays a role even though this is a consideration for the simulated scheduler. Thus, the instance selection is implicitly claiming that release times should not be considered in best scheduling practices, but rather that due dates should be considered when the weights of the jobs are equal, and that when weights are not equal the weights should be the dominant consideration. This is reflected by the decision tree in Fig. 2, where there is no release time consideration, whereas release time is a consideration in the decision tree in Fig. 1, which is based on learning from all of the data.

Finally, we compare the sequence Job 9 – Job 7 – Job 3 – Job 2 – Job 8 – Job 5 – Job 1 – Job 6 – Job 4 – Job 10, obtained by applying the improved decision tree to the test data, to the sequence Job 6 – Job 8 – Job 5 – Job 3 – Job 9 – Job 2 – Job 7 – Job 1 – Job 4 – Job 10, obtained by applying the simulated scheduler's rules. When comparing the two sequences, we note that half of the jobs move more than two positions in the sequences. For example, Job 6 is the only job released at time zero, so the simulated scheduler starts that job first (first position), whereas the improved decision tree would not sequence this job until the eight positions and this job is therefore delayed by 7 positions. On the other hand, Job 7 is not sequenced until the seventh position by the simulated scheduler, but the improved decision tree would sequence this job in the second position. To explore the reason for those changes we consider the weight and release time data for the five jobs with the greatest position change:

Job (j)	6	8	5	9	7
Position change	+7	+3	+3	−4	−5
Weight (w_j)	3	3	3	5	5
Release time (r_j)	0	2	8	9	14

It is clear that all of the jobs that are delayed by the improved decision tree (that is, Job 6, Job 8, and Job 5) have lower weight and earlier release time than the others; and vice versa for the jobs that are expedited by the improved decision tree. To make this comparison of sequence position more rigorous, we calculate the correlation between the position change and the job data. The correlation between position change and release date is $\rho_{\Delta,r} = 0.72$, position change and due date is $\rho_{\Delta,d} = -0.13$, position change and processing time is $\rho_{\Delta,p} = -0.10$, and position change and weight is $\rho_{\Delta,w} = 0.48$. Thus, the release date and weight of a job have the highest correlation with position change of that job in the sequence, which further supports the above observations. We conclude that the instance selection indicates that release time is too big of a consideration in current scheduling practice, whereas weight should be more important.

Even though this simple simulated example contains none of the human and organizational complexities of a real production system, it illustrates that by using the two-phased data mining approach it is possible to learn scheduling rules that match and exceed the performance of the simulated scheduler; and that useful insights about the nature of best scheduling practices can be obtained by analyzing which instances were selected because they result in the best dispatching rules. In this context it is important to observe that the methodology does not depend on any assumptions about the system so it can be applied to arbitrary production systems, including real systems where human behavior has a significant impact.

5. Performance evaluation on a simulated single machine system

We now validate the results from the ten-job example in Section 4 above with respect to improvement in scheduling performance. We thus continue to focus on a single machine system with a weighted maximum lateness objective.

5.1. Generating simulated data

To evaluate if the results are robust with respect to the data generation (that is, to show that the results reported in Section 4.2 are not overly dependent on how the data for the ten jobs in generated), we generate data to reflect various levels of variability for the input data, as well as the due date tightness. Thus, the variability the release time, due date and processing times is set to different levels by fixing the coefficient of variation of the underlying distribution. (Note that the coefficient of variation is defined as $cv = s/x$, where s is the standard deviation and x is the mean of the distribution.) Since the objective function is due date

Table 5
Experimental design of production data for evaluating a single-machine system.

	Release time (cv and $\bar{r}$)	**Due date** (cv and $\bar{d}$)	**Processing time** (cv and p)	**Tightness** ($\bar{d}-\bar{r}+\bar{p}$)
Set 1	$cv=1$	$cv=0.5$	$cv=2$	
1A	5	8	2	5
1B	10	15	5	10
1C	15	25	5	15
Set 2	$cv=2$	$cv=0.5$	$cv=1$	
2A	5	8	2	5
2B	10	15	5	10
2C	15	25	5	15
Set 3	$cv=2$	$cv=1$	$cv=0.5$	
3A	5	8	2	5
3B	10	15	5	10
3C	15	25	5	15
Set 4	$cv=0.5$	$cv=1$	$cv=2$	
4A	5	8	2	5
4B	10	15	5	10
4C	15	25	5	15
Set 5	$cv=0.5$	$cv=2$	$cv=1$	
5A	5	8	2	5
5B	10	15	5	10
5C	15	25	5	15
Set 6	$cv=1$	$cv=2$	$cv=0.5$	
6A	5	8	2	5
6B	10	15	5	10
6C	15	25	5	15

related, it is reasonable to suspect that the method might perform differently if the due dates are difficult to meet (tight) versus relatively easy (loose), so we evaluate various levels of the due date tightness $\bar{d}-\bar{r}+\bar{p}$, where $\bar{d}$,$\bar{r}$, and $\bar{p}$ represent the average due date, release time and processing time, respectively.

We use three distributions to create release time, due date and processing times: Erlang distribution with $cv=0.5$, exponential distribution with $cv=1$, and hyper-exponential distribution with $cv=2$. Based on those we design six sets of simulation experiments with different combinations of these three distributions assigned to the three attributes. Furthermore, within in each set there are three different datasets with different tightness values of 5, 10, and 15. Therefore, there are total of eighteen simulated production datasets, as shown in Table 5. For the instance selection, four independent replications were run of the GA search with its parameters set as before.

5.2. Evaluating the simplicity of new rules

One part of the instance selection objective function to be minimized is the size of the tree. It is well known than decision trees often grow excessively large and if there are too many nodes in the tree the set of rules will be large. Table 6 compares the size of the decision trees induced on all of the data (original) and the smaller subset of best instances (improved). The original trees have an average number of nodes ranging from 237 to 333, for the 18 sets of production data used for evaluation, resulting in very complex rules based on those trees. On the other hand, by optimizing the instances used for learning, the average tree size is reduced to 7–14 nodes, which would result in only moderately complex rules. The average reduction in the number of nodes is 97% and it is very stable across the datasets with the standard deviation being less than one percentage point. Thus, the instance selection is robust with respect to improving the size of the decision trees and hence the simplicity of the new dispatching rules.

Table 6
Number of nodes in the decision tree that are induced on all of the scheduling data (original) versus only using the best data (improved).

Dataset	**Number of nodes in tree**		
	Original	Improved	Percentage improvement (%)
1A	237	14	94
1B	285	11	96
1C	250	10	96
2A	251	11	96
2B	265	7	97
2C	295	10	97
3A	329	7	98
3B	308	10	97
3C	305	8	97
4A	333	7	98
4B	305	8	97
4C	351	8	98
5A	301	10	97
5B	313	12	96
5C	293	9	97
6A	319	8	97
6B	301	9	97
6C	316	9	97

5.3. Evaluating the scheduling performance of new rules

In addition to smaller tree, the goal is to induce decision trees that improve scheduling performance when applied as dispatching rules. The third and fourth columns of Table 7 compare the weighted maximum lateness of the decision trees induced on all the data (original) versus only the best data (improved).

It is evident that the improved trees found by employing instance selection are significantly better. The overall average weighted maximum lateness is reduced by 46% from 4402 to 2377. The variation between datasets is somewhat larger than when looking at the number of nodes, as the standard deviation of the percentage reduction is 5.3 percentage points. The range of reduction is 38–55%, which is significant for all of the datasets. Combining the results of Tables 6 and 7, it is clear that the use of instance selection results in vastly improved trees that are both much smaller, and hence more interpretable, and perform better when applied to dispatch new data.

Finally, we are interested in comparing the new dispatching rules with the original data, that is, the underlying scheduling policy used by the simulated scheduler. Table 7 also shows this comparison (compare the second and fourth column). Again, there is a significant improvement for every dataset, ranging from 27% to 50% reduction in the weighted maximum lateness. While still robust, and lacking any general pattern as to where the new method works best, the standard deviation of the percentage improvement is slightly larger than before at 6.4 percentage points. Thus, the use of instance selection is not only a significant improvement on the trees induced without instance selection, but this approach actually results in dispatching rules that are significantly better than the underlying rule, which produces the data used for learning.

6. Conclusions

Motivated by the importance of the human schedulers, and their ability to account for complex constraints, we have proposed a novel data mining approach for learning directly from prior schedules. To the best of our knowledge, Li and Olafsson (2005) were the first to propose decision tree learning directly from scheduling data, and the key contribution of this paper is the

Table 7
Comparison of the scheduling performance of the decision trees induced on all data (original) versus best data (improved) when they are applied to schedule new jobs.

Dataset	Weighted maximum lateness			Percentage improvement	
	Simulated scheduler	Original tree	Improved tree	Compared to sim. scheduler (%)	Compared to original tree (%)
1A	1890	1790	1094	42	39
1B	3553	3589	2215	38	38
1C	4583	4788	2555	44	47
2A	2035	2280	1017	50	55
1B	4529	5103	2528	44	50
1C	3857	4725	2494	35	47
1A	1688	2189	1226	27	44
1B	4764	5548	3250	32	41
1C	4965	5706	2912	41	49
1A	1672	2245	1108	34	51
1B	4814	6597	3356	30	49
1C	4381	4887	3138	28	36
1A	2212	2520	1263	43	50
1B	5683	6189	3097	46	50
1C	5152	5607	3389	34	40
1A	2369	2768	1410	40	49
1B	4957	6250	3284	34	47
1C	5515	6453	3458	37	46

novel idea of using instance selection to first identify which part of the training data (existing schedules) should be used for learning. We achieve this by indirectly measuring the quality of a subset of instances by the quality of the dispatching rules that result if decision tree induction is applied to learn from those instances. Our results based on simulated single-machine systems show that by using this approach the learning can go beyond mimicking existing practices, and results in dispatching rules that considerably improve the underlying rules for the system. Furthermore, significant insights about best scheduling practices can be obtained by analyzing the selected subset of instances and by comparing the outcome of the new set of dispatching rules with existing practices.

Several future research directions are apparent. First and perhaps most importantly, the new approach needs to be tested on a real production system. Second, the creation of the decision trees and their use as improved dispatching rules for scheduling new jobs may be completely automated using the approach described in this paper, but our analysis of the selected instance subset to extract insight about best practices is more ad hoc. It would therefore be valuable to develop additional techniques and procedures for this process. Third, the solution approach for the instance selection problem could undoubtedly be improved over the genetic algorithm search used in this paper. Thus, further research is needed to investigate other heuristic approached for its solution. Fourth, it would be valuable to validate the new methods by applying them to other more complex simulated systems, such flow shops or job shops. From the data mining perspective there is nothing inherently different between learning from data generate from a single machine system and that of any other more complex system – the data from which we learn will simply have more variables (attributes) and examples (instances). However, testing the approach on larger systems with more parameters and interactions between its various components would validate the results and provide further insights into its scalability to large systems.

Acknowledgements

The authors would like to thank Dr. Shuning Wu for providing the original code for the genetic algorithm adapted for the instance selection.

References

Aytug, H., Bhattacharyya, S., Koehler, G.J., Snowdon, J.L., 1994. A review of machine learning in scheduling. IEEE Transactions on Engineering Management 41 (2), 165–171.

Baykasoglu, A., Gocken, M., Ozbakir, L., Kulluk, S., 2008. Composite dispatching rule generation through data mining in a simulated job shop. Communications in Computer and Information Science 14, 389–398.

Berglund, M., Karitun, J., 2007. Human, technological and organizational aspects influencing the production scheduling process. International Journal of Production Economics 110 (1–2), 160–174.

Bowden, R., Bullington, S.F., 1996. Development of manufacturing control strategies using unsupervised machine learning. IIE Transactions 28, 319–331.

Chang, P.-C.J.-C., Hsieh, Liu, C.H., 2006. A case-injected genetic algorithm for single machine scheduling problems with release time. International Journal of Production Economics 103 (2), 551–564.

Chen, W.Y., Sheen, G.J., 2007. Single-machine scheduling with multiple performance measures: minimizing job-dependent earliness and tardiness subject to the number of tardy jobs. International Journal of Production Economics 109 (1–2), 214–229.

Choudhary, A.K., Harding, J.A., Tiwari, M.K., 2009. Data mining in manufacturing: a review based on the kind of knowledge. Journal of Intelligent Manufacturing 20 (5), 501–521.

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

Hall, L.A., Shmoys, D.B., 1992. Jackson's rule for single-machine scheduling: making a good heuristic better. Mathematics of Operations Research 17 (1), 22–35.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H., 2009. The WEKA data mining software: an update. SIGKDD Explorations 11 (1).

Han, J., Kamber, M., 2006. Data Mining: Concepts and Techniques 2nd ed. Morgan Kaufmann, San Francisco, CA.

Harding, J.A., Shahbax, M., Srinivas, S., Kusiak, A., 2006. Data mining in manufacturing: a review. Journal of Manufacturing Science and Engineering 128, 969–976.

Hestermann C., M. Wolber, 1997. A comparison between operations research-models and real world scheduling problems. In: Proceedings of the European Conference on Intelligent Management Systems in Operations, University of Salford, UK, 25–26 March 1997.

Jackson, S., Wilson, J.R., MacCarthy, B.L., 2004. A new model of scheduling in manufacturing: tasks roles, and monitoring. Human Factors 46 (3), 533–550.

Jain, A.S., Meeran, S., 1998. Job-shop scheduling using neural networks. International Journal of Production Research 36 (5), 1249–1272.

Kanet, J.J., Adelsberger, H.H., 1987. Expert systems in production scheduling. European Journal of Operational Research 29, 51–59.

Kusiak, A., Chen, M., 1988. Expert systems for planning and scheduling manufacturing systems. European Journal of Operational Research 34, 113–130.

Li, X., Olafsson, S., 2005. Discovering dispatching rules using data mining. Journal of Scheduling 8, 515–527.

Malik, A.M., Russell, T., Chase, M., van Beek, P., 2008. Learning heuristics for basic block instruction scheduling. Journal of Heuristics 14 (6), 549–569.

McKay, K.N., 1999. Unifying the theory and practice of production scheduling. Journal of Manufacturing Systems 18 (4), 241–255.

Nakasuka, S., Yoshida, T., 1992. Dynamic scheduling system utilizing machine learning as a knowledge acquisition tool. International Journal of Production Research 30 (2), 411–431.
Pinedo, M., 1995. Scheduling: Theory, Algorithms and Systems. Prentice Hall.
Piramuthu, S., Raman, N., Shaw, M.J., 1994. Learning-based scheduling in a flexible manufacturing flow line. IEEE Transactions on Engineering Management 41 (2), 172–182.
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, 251–264.
Quinlan, J.R., 1993. C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA.
Russell, T., Malik, A.M., Chase, M., van Beek, P., 2009. Learning heuristics for the superblock instruction scheduling problem. IEEE Transactions on Knowledge and Data Engineering 21 (10), 1489–1502.
Schaller, J., 2007. Scheduling on a single machine with family setups to minimize total tardiness. International Journal of Production Economics 105 (2), 329–344.
Schmidt, G., 1998. Case-based reasoning for production scheduling. International Journal of Production Economics 56, 537–546.
Shaw, M.J., Park, S., Raman, N., 1992. Intelligent scheduling with machine learning capabilities: the induction of scheduling knowledge. IIE Transactions 24 (2), 156–168.
Wang, K., Tong, S., Eynard, B., Roucoules, L., Matta, N., 2007. Review on application of data mining in product design and manufacturing. In: Proceedings of the Fourth International Conference on Fuzzy Systems and Knowledge Discovery, vol. 4, pp. 613–618.
Wiers, V.C.S., 1997. A review of the applicability of OR and AI scheduling techniques in practice. OMEGA—The International Journal of Management Science 25 (2), 145–153.
Wu, S., 2007. Optimal Instance Selection for Improved Decision Tree (Dissertation). Iowa State University, Ames, IA.