2006

Application of data mining in scheduling of single machine system

Xiaonan Li
Iowa State University

Application of data mining in scheduling of single machine system

by

Xiaonan Li

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

Major: Industrial Engineering

Iowa State University

Ames, Iowa

2006

Graduate College
Iowa State University

This is to certify that the doctoral dissertation of

Xiaonan Li

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Major Professor

Signature was redacted for privacy.

For the Major Program

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I am grateful to a lot of people for this dissertation, which could not have been written without their support, help, patience and love of my family, friends and my advisor, Dr. Sigurdur Olafsson. Whenever I was in troubles and difficulties, he always presented a solution to me with his smile and then I could go further in my research journey. I deeply appreciate his generosity, encouragement, mentoring, and research support throughout my doctoral studies.

I would like to express special thanks to my committee members -Dr. John Jackman, Dr. Sarah Ryan, Dr. Vasant Honavar and Dr. Dianne Cook. Their thoughtful concern, encouragement, and valuable comments were truly helpful and improved my dissertation.

Especially, I would like to thank my husband Lei for his encouragement, support and love during the past few years. I am also deeply grateful to my parents, brother Hong, and sister Fang for their encouragement, love, dedication and the many years of support during my studies that provided the foundation for this work. I love and thank you with all my heart.

Thank you all!

ABSTRACT

The rapidly growing field of data mining has the potential of improving performance of existing scheduling systems. Such systems generate large amounts of data, which is often not utilized to its potential. The problem is whether it is possible to discover the implicit knowledge behind scheduling practice and then, with this knowledge, we could improve current scheduling practice.

In this dissertation, we propose a novel methodology for generating scheduling rules using a data-driven approach. We show how to use data mining to discover previously unknown dispatching rules by applying the learning algorithms directly to production data. We also consider how by using this new approach unexpected knowledge and insights can be obtained, in a manner that would not be possible if an explicit model of the system or the basic scheduling rules had to be obtained beforehand. This approach involves preprocessing of historic scheduling data into an appropriate data file, discovery of key scheduling concepts, and representation of the data mining results in a way that enables its use for job scheduling.

However, direct data mining of production data can at least mimic scheduling practices. The problem is whether scheduling practice could be improved with the knowledge discovered by data mining. We present an optimization based instance selection approach for scheduling to address this problem. In this approach, we use a genetic algorithm to find a heuristic solution to the optimal instances selection problem, and then induce a decision tree from this subset of instances. The optimal instance selection can be viewed as determining the best practices from what has been done in the past, and the data mining can then learn new dispatching rules from those best practices. Furthermore, data mining is also employed to analyze the selected best instances.

1 INTRODUCTION

Production scheduling provides an important role in most manufacturing facilities and has received a great amount of attention from both the operations research and artificial intelligence communities. There is, however, some discontinuity between scheduling practice and the academic research of scheduling models and algorithms. In particular, for complex systems it may be difficult in practice to account for all relevant aspects in an optimization model or to elicit all relevant scheduling rules directly from an expert. In practice, on the other hand, scheduling is often done on an ad-hoc basis that does not rely solely on well-defined rules or principles, but also on the intuition and experience of the scheduler. When such knowledge is not explicitly captured, any model-based approach may fail to account for important considerations.

Data mining and knowledge discovery is an emerging area of research and applications that draw on machine learning and statistical methods to learn previously unknown and useful knowledge from examples in large databases. All data mining starts with a set of data or a training set, which consists of instances describing the values of certain attributes. Data mining has made a significant impact on numerous industries, but its function in manufacturing and production scheduling in particular, has only received moderate attention to date. Its applicability in this area is evident, however, as the strengths of data mining lie where it is difficult or impossible to capture all aspects of a system a priori in a model, either because of its complexity or because of incomplete existing knowledge, and large volumes of data are generated by the system. Both situations commonly exist in production environments, which are often too complex for simple mathematical models to adequately capture all essential elements of the system, and much of the knowledge of the operation of the system may be implicit and would thus not be captured by the mathematical models.

In this dissertation, we show how data mining on production data can be used to capture both explicit and implicit knowledge that is used to create production schedules. Integrated with optimization, new optimized instances selection methodology for scheduling is proposed to identify the best scheduling practice. Through learning from the best data, the induced model can work better as a new dispatching rule, which indicate that the current scheduling practice could be improved. Finally we propose to apply data mining to analyze the best data selected by optimization-based instance selection methodology.

1.1 Data Mining for Scheduling

Scheduling is a decision-making process that plays an important role in most manufacturing and service industries. It is used in procurement and production, in transportation and distribution, and in information processing and communication. The scheduling function usually uses mathematical techniques or heuristic methods to allocate limited resources to the processing of tasks. A proper allocation of resources enables the company to optimize its objectives and achieve its goals. Resources may be machines in a workshop, runways at an airport, or crews at a construction set. Tasks may be operations in a workshop, takeoff and landings at an airport, or stages in a construction project. Each task may have a priority level, an earliest possible starting time, and a due date. The objectives may also take many forms, such as minimizing the time to complete all tasks or minimizing the worst performance of the schedule.

Many approaches from operations research and artificial intelligence are employed to deal with scheduling problem. But usually, the requirements to know the rules for doing scheduling tasks should be known in advance highly limit the successful applications of these approaches. Due to the complexities of manufacturing systems, it is not always possible to find out those rules about scheduling practice.

Data mining can be applied successfully where it is difficult or impossible to capture all aspects of a system in a model, either because of its complexity or because of incomplete existing knowledge, and where large volumes of data are generated by the system. Motivated by the strength of data mining, we propose a new framework for applying data mining in scheduling to discover new dispatching rules, which can then be applied to automate the scheduling function. Furthermore, hidden patterns discovered in the schedule generation may add insights not realized by the schedulers themselves, suggesting ways in which current scheduling practices can be improved.

1.2 Optimal Instance Selection for Scheduling

The idea of this data mining approach to production scheduling is to complement more traditional operations research approaches. The data mining approach is particularly applicable for large, complex production environments, where the complexity makes it difficult to model the system explicitly. However, an implicit assumption is that it is worthwhile to capture the current practices from historical data. Furthermore, this approach does not seek to directly improve any scheduling performance measure. Thus, combining the data mining with some type of optimization approach that improves the discovered dispatching rules would be a natural extension.

There is another assumption that not all historical scheduling data represent good scheduling practice. If we apply the decision tree learning algorithm directly to these data, then the tree model after learning could not perform well enough as a new dispatching rule, because direct data mining of production data can mainly mimic scheduling practices. Thus, we propose a hypothesis that if we could identify the subset of good instances that represent best scheduling practice, then the induced decision tree model will perform well as an experienced scheduler.

Motivated by this hypothesis, we propose a novel optimization-based instance selection methodology for scheduling, by combining data mining with optimization for effective production scheduling. In this approach, we use a genetic algorithm to find a heuristic solution to the optimal instances selection problem, and then induce a decision tree from this subset of best instances. The optimal instance selection can be viewed as determining the best practices from what has been done in the past, and the data mining can then learn new dispatching rules from those best practices.

1.3 Best Instances Analysis

When the best data are selected by optimization-based instance selection method, it is interesting to analyze these best data to see why they are good and how to identify them. Therefore, we propose an approach to analyze how the best data are selected through instance selection procedure. The basic idea of this approach is to apply data mining directly to learn the knowledge about best data identification. Therefore, a new target concept is defined and accordingly, a new class attribute is introduced. After applying data mining algorithm, the performances of the models are analyzed extensively. In addition, attributes selection is also employed to identify those factors that are most important for instance selection decision process.

The remainder of this dissertation is organized as follows:

- Chapter 2

  This chapter reviews the literature related to the dissertation. Methods that have been proposed to apply data mining in scheduling, instances selection is discussed with some notion on relationship between those works and this dissertation. We briefly review some of the closely related research, and in particular, how machine learning has been used for scheduling in prior work.

- Chapter 3

  We propose a new research framework to apply data mining in scheduling. We show how to use data mining to discover new dispatching rules, which can then be applied to automate the scheduling function. Furthermore, hidden patterns discovered in the schedule generation may add insights not realized by the schedulers themselves, suggesting ways in which current scheduling practices can be improved.

- Chapter 4

In this chapter, we present a novel genetic algorithm based optimal instance selection methodology for scheduling. Through numerical example and experiments, we show that this approach is an effective and efficient to identify the best scheduling practice and the model can work as a new dispatching rule with much better scheduling performance than optimal heuristic dispatching rules.

- Chapter 5

  In this chapter, we propose to apply data mining to analyze the best instances selected by genetic algorithm based instance selection method for scheduling. We formulate the task as a classification problem and report our numerical results to evaluate the performance of this approach.

- Chapter 6

  We conclude with a summary of the contribution of this dissertation and address some interesting directions for future research.

2 LITERATURE REVIEW

In this section, we will review the literatures related to our research. We first review artificial intelligence methods for scheduling, while in the second selection, we will discuss instance selection methods and applications. Unbalanced class problems, attribute construction and selection and preference ordering learning are also reviewed because they are related to our research.

2.1 Artificial Intelligence Methods for Scheduling

Artificial intelligence methods for scheduling have received considerable attention over the past two decades (see e.g., Aytug et al., 1994; Kanet and Adelsberger, 1987; Kusiak and Chen, 1988; Noronha and Sarma, 1991; Priore et al., 2001). Such methods were developed in part to address the limitation imposed by the need for an explicit model of the system. Examples of such methods include neural networks (Jain and Meeran, 1998; Min et al., 1998), induction (Shaw et al., 1992), hybrid approaches (Kim et al., 1998; Lee et al., 1997), and unsupervised learning (Bowden and Bullington, 1996; Li and She, 1994). Of these various methods, the work that is the most related to the dissertation are inductive learning methods that have for example been used to select between several dispatching rules. Some of this work is described further below and more comprehensive surveys can be found in Aytug et al. (1994) and Priore et al. (2001).

In an early work, Nakasuka and Yoshida (1992) used empirical data to generate a binary decision automatically. The practical data were obtained through iterative production line simulations and afterwards the binary decision tree decides which rule to be used at decision points during the actual production operations. Later, Wang et al. (1995) proposed a scheduling system with inductive learning called the System Attribute Oriented Knowledge Based Scheduling System (SAOSS). A simulation model is again used to generate training examples. A continuous ID3 algorithm is then employed to induce decision rules for scheduling by converting corresponding decision trees into the hidden layers of a self-generated neural network. The connection weights between hidden units of the self-generated neural network imply the scheduling heuristics, which are then formulated into scheduling rules.

Shaw et al. (1992) also present an inductive learning approach integrated with simulation. Similarly to the work mentioned above, simulation model is used to generate training examples for a given state of manufacturing process. The simulation runs are replicated for each scheduling rule considered. The rule that has the best performance is the class value for this set of attributes' values. As for the criteria for best performance, the scheduling objectives should be designed in advance. An ID3 algorithm is employed to describe the system pattern for selecting each scheduling rule. A decision tree is formulated and then it is translated into a set of patterns directed heuristics for scheduling.

In later extension of this work, Piramuthu et al. (1993, 1994) employ the C4.5 algorithm, a refinement of the ID3 algorithm with pruning capability. Improvement has been done in two aspects: (a) a different decision tree is constructed, for choosing chattering threshold values under different patterns. As a result, overreaction to filter noise arising from transient patterns will be avoided efficiently; (b) a critic module is employed to refine the decision tree. The performance of the system is compared with that of individual scheduling rule periodically. If the performance of system degraded, the decision tree needs to be refined by generating new training examples to certain over generalized concepts. In this way, the system can learn new knowledge incrementally.

Considering multi-objective flexible manufacturing system (FMS) scheduling problem, Kim et al. (1998) propose an integrated approach of inductive learning and competitive neural networks. Simulation is applied to generate unclassified training data, which then are classified by competitive neural network approach. After classified data generation, C4.5 is employed to discover scheduling rules. From the results of experiments, they showed that this integrated approach is effective to address complex scheduling problem in FMS, but they also mention that system status variables selection is necessary.

The integration of Inductive learning and genetic algorithms is also applied in scheduling. Lee et al. (1997) propose the approach to use C4.5 to select the best rule for releasing jobs to the system, and then employ genetic algorithms to select the most suitable dispatching rule to schedule jobs for each machine in the system. Chiu and Yih (1995) give a methodology to use genetic algorithms to search for a set of good training examples, and decision tree is applied to learn from selected training data. They also use genetic algorithm to do modification on the decision tree model when new examples included.

Taking into account of the benefits of proper learning biases to improve the knowledge base, Chen et al. (1999) propose an auto-bias selection approach which can determine good feature set and a suitable learning algorithm. They use hybrid approach called FSSNCA (Feature Subset Selection based on Nonlinear Correlation Analysis), which includes a filter stage and a search stage. In filter stage, a measure of nonlinear correlation called dispersion function is applied to do feature selection. The search stage is similar to the wrapper method, but the dispersion measure is used to prune the search space by feature selection. Through case study, they conclude that the knowledge base with the feature subset selected can yield higher accuracy.

As this sampling of prior work indicates, inductive learning in production scheduling has primarily been devoted to issues such as selecting the best dispatching rule by learning from simulated data. This, of course, 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. (2003), where genetic algorithm is used to discover new dispatching rules in a flow shop environment. However, to the best of our knowledge data mining applied directly to production data to discover new and interesting dispatching rules has not been considered before.

2.2 Instance Selection Methods and Applications

Data mining processes include data selection, preprocessing, applying learning algorithm, interpretation and evaluation. The first two processes play a critical role in successful data mining. Instance selection is recently getting more and more attention from researchers and practitioners. There are mainly two perspectives for instance selection methods: one is to address the need to reduce storage requirements and computational loads [Kuncheva, 1995]; the other perspective is to achieve enhanced performance from the learning algorithm through instance selection, as pointed in [Dasarathy, 1990].

Facing the challenges of enormous amounts of data, Liu and Motoda (1998, 2002) worked on scaling down the data to select the relevant data and then present it to a data mining algorithm. The authors pointed out that instance selection is an alternative to scale up the algorithm, and the combination of the two is a "two edged sword" in data mining to deal with massive data sets. According to their research work, the ideal outcome of instance selection is a model independent, minimum sample of data that can accomplish tasks with little or no performance deterioration.

As an important part of instance selection, sampling is a procedure that draws a sample by random process in which each sample has an appropriate probability distribution. There are two basic sampling methods: simple random sampling and stratified random sampling. The former is to select a sample of instances such that every instance has an equal probability of being chosen. In the latter method, the whole data set is first divided into a certain number of subsets, which are non-overlapping and called strata. Then simple random sampling is performed in each strata independently and respectively. A more advanced sampling method is adaptive sampling, where selecting instances in a sample depends on the results obtained from the sample. That is sampling and mining are integrated together to take the advantage of better sampling for further mining and vice versa. The objective of adaptive sampling is to take the advantage of data characteristics in order to get more good results. The idea of adaptive sampling is very similar to our optimization-based instance selection for scheduling presented in Chapter 4.

Another category of instance selection methods relies on data mining algorithms. Hart and Cover (1967) made one of the first attempts to develop an instance selection rule and they proposed Nearest Neighbor (NN) learning algorithm. The ideal of NN is to select a subset such that every instance of the original training set is closer to an instance of the selected subset with the same class than one with different class. Later, more extended versions of NN were proposed. Ritter, Woodruff, Lowry and Isenhour (1975) presented the Selective Nerarest Nerighbor algorithm, while Gates (1972) proposed the Reduced Nearest Neighbor algorithm. In addition, Wilson (1972) introduced the Edited Nearest Neighbor algorithm and Tomek (1976) proposed the all k-NN method.

When there are too much data, NN algorithm could be very slow and very sensitive to noise especially in later phases, thus instance-based algorithm is introduced to by Aha (1991) to keep only the most important instances and remove the redundant data points. Instance-based learning algorithms are considered as a method of knowledge refinement and it maintains the instance-case. Smith (1998) proposed an approach to remove harmful and useless cases. McSherry (2000) introduced a method, called discover, integrating an evaluation strategy of the coverage contributions of a candidate instance in the process of build the case set.

Cano (2003) proposed to an instance selection method based on evolutionary algorithm, which is an adaptive method that stems from natural evolution and very useful for search and optimization. Through different empirical study, it is shown that with evolutionary algorithm, better instance reduction and higher classification accuracy can be successfully achieved. Furthermore, the authors pointed out that evolutionary algorithm would be prominent and effective tool in research field of the instance selection method. This is quite consistent with our conclusion that the genetic algorithm based instance selection proposed in this dissertation is an effective and universal instance selection method.

Kim (2006) presented a genetic algorithm based instance selection method in Artificial Neural Networks (ANN) for financial forecasting. The evolutionary instance selection is employed to reduce the dimensionality of data and may also remove the harmful and redundant instances. In addition, evolutionary search strategy is also used to find the ideal connection weights between layers in ANN.

2.3 Unbalanced Class Problems

Although the majority of learning systems previously designed usually assume that training sets are well-balanced, this assumption is not necessarily correct. Indeed, there exist many domains for which one class is represented by a large number of examples while the other is represented by only a few. If 99% of the data are from one class, for most realistic problems a learning algorithm will be hard pressed to do better than the 99% accuracy achievable by the trivial classifier that labels everything with the majority class.

Japkowicz (2000) discussed the effect of imbalance in a dataset. The author evaluated two resampling methods. Random resampling consisted of resampling the smaller class at random until it consisted of as many samples as the majority class, which is also called oversampling. Random under-sampling was also considered, which involved under-sampling the majority class samples at random until their numbers matched the number of minority class samples. She noted that both the sampling approaches were effective, and she also observed that using the sophisticated sampling techniques did not give any clear advantage in the domain considered.

Ling and Li (1998) combined over-sampling of the minority class with under-sampling of the majority class. They used lift analysis instead of accuracy to measure a classifier's performance. Lift is a measure of the effectiveness of a predictive model calculated as the ratio between the results obtained with and without the predictive model. In one experiment, they under-sampled the majority class and noted that the best lift index is obtained when the classes are equally represented. In another experiment, they over-sampled the minority examples with replacement to match the number of negative majority examples to the number of positive examples. The over-sampling and under-sampling combination did not provide significant improvement in the lift index.

Lewis and Catlett (1994) examined heterogeneous uncertainty sampling for supervised learning. This method is useful for training samples with uncertain classes. The training samples are labeled incrementally in two phases and the uncertain instances are passed on to the next phase. They modified C4.5 to include a loss ratio for determining the class values at the leaves. The class values were determined by comparison with a probability threshold of LR/(LR+1), where LR is the loss ratio.

In imbalanced class problems, only one measure, like accuracy is usually not enough to analyze the performance of the model. Confusion Matrix (Kohavi and Provost, 1998) contains information about the actual and predicted classifications done by a classification algorithm. Performance of such is commonly evaluated using the data in the matrix. The following table shows the confusion matrix for a two class classifier.

Table 2.1 Confusion Matrix

		Predicted	
		Negative	Positive
Actual	Negative	a	b
	Positive	c	d

The entries in the confusion matrix have the following meaning in the context of the study:

- *a* is the number of correct predictions that an instance is negative,
- *b* is the number of incorrect predictions that an instance is positive,
- *c* is the number of incorrect predictions that an instance is negative,
- *d* is the number of correct predictions that an instance is positive.

The accuracy (AC) is defined as the proportion of the total number of predictions that were correct. It is determined using the following equation:

$$accuracy(AC) = \frac{a+d}{a+b+c+d} \tag{2.1}$$

The recall or true positive rate (TP) is the proportion of positive cases that were correctly identified, as calculated using the following equation:

$$recall(TP) = \frac{d}{c+d} \tag{2.2}$$

The precision (P) is the proportion of the predicted positive cases that were correct, as calculated using the following formula:

$$precision(P) = \frac{d}{b+d} \tag{2.3}$$

The false positive rate (FP) is the proportion of negative cases that were incorrectly classified as positive, as calculated by the following formula:

$$FP = \frac{b}{a+b} \tag{2.4}$$

The true negative rate (TN) is defined as the proportion of negative cases that were classified correctly, as calculated by the equation:

$$TN = \frac{a}{a+b} \tag{2.5}$$

While the false negative rate (FN) is the proportion of positive cases that were incorrectly classified as negative, as calculated using the equation:

$$FN = \frac{c}{c+d} \tag{2.6}$$

When there is imbalance class, the accuracy determined using equation (2.1) may not be an adequate measure. Other performance measures account for this by including recall (TP), precision (P) or a product of these two measures: for example, F-Measure defined by Lewis and Gale (1994) as in the following equation.

$$F = \frac{(\beta^2 + 1) * P * TP}{\beta^2 * P + TP} \tag{2.7}$$

Using precision, recall, and F-Measure, we analyze the performance of the models after applying decision tree algorithms to the reprocessed data in the next sections.

2.4 Attribute Construction and Selection

Data engineering plays a critical role in constructing more accurate models, building more interpretable models, and in providing insights into which factors are most important in making the scheduling decisions. Specifically, the attributes that are recorded as part of the raw production data may not be the attributes that are the most useful for the data mining itself. Thus, new attributes creation must be considered (Chen and Yih, 1996). In some cases this could be done manually using intuitive processes, but it can also be discovered automatically.

Attribute selection using frequent itemset generation was introduced in the seminal paper by Agrawal et al. (1993) and this approach has been found to be useful in many areas of knowledge discovery. In this context, an item is an attribute value pair, that is an attribute along with one of its possible values, and an itemset is a simply a set of items. An item set is called frequent if it occurs some minimum number of times in the database, that is, meets the minimum support. It is possible to use such frequent itemset generation to construct new attributes that are helpful for classification learning (Lesh et al., 1999; Deshpande and Karypis, 2002). With this approach, new composite attributes are added based on attributes that occur frequently together, which allows the subsequent learning algorithm to use conjunction of attribute values. For example, if itemsets involving processing time of job 1, p_1, and processing time of job 2, p_2, occur frequently together then we attempt to construct new composite attribute using this pair with four basic arithmetic operations: $p_1 - p_2$, $p_1 + p_2$, ${}^{p_1}\!/_{p_2}$, and $p_1 \cdot p_2$.

A problem of estimating the quality of attributes (features) is an important issue in data mining. Normally, attribute selection is employed to address the problems that there are many irrelevant, but noisy attributes which provide very little information. Accordingly, the objective of attribute selection is to choose a small subset of attributes that ideally are necessary and sufficient to describe the target concept. However, there is another benefit of attribute selection is that we can sometimes generate some interesting insights from the attributes selected. One example of such insights is that which kind of factors are the most important regarding to the target concept of interests. Therefore, attributes selection is a useful and necessary procedure to be employed to analyze the best data selection.

Taking into account whether or not feature selection process embeds a learning algorithm, we can classify them into one of both filter and wrapper approaches [John, Kohavi and Pfleger, 1994]. The filter approach methods use a fitness function for evaluating feature subsets rather than using a learning algorithm. The most widely known existing algorithms that fall into the filter approach are FOCUS [Almuallim and Dietterich, 1994], Relief [Kira and Rendell, 1992] and its variants [Kononenko, 1994], CFS (Correlation Based Feature Selection) [Hall, 1998], and cross-entropy filter [Koller and Sahami, 1996]. Other algorithms except FOCUS stated in the previous subsection will be dealt with in main chapters later in detail.

On the other hand, the wrapper approaches use a learning algorithm in the feature selection process. The wrappers usually provide better accuracy but are computationally more expensive than the filters [Raman and Ioerger, 2002]. Kohavi and John (1997) used the best first search for a wrapper approach. Many algorithms stated previously such as genetic algorithm [Yang and Honavar, 1998] and LVW [Liu and Setiono, 1996] incorporate with a learning algorithm for evaluating feature subsets.

The ReliefF algorithm is an extended version of Relief algorithm, which is developed by Kira and Rendell (1992) for estimating the quality of attributes that consider interdependency between attributes. The basic idea of Relief algorithms is, as shown in Figure 2.1, to estimate the quality of attributes according to how well their values distinguish neighbor instances from different classes by having different values and having the same values for neighbor instances from the same class.

Given a randomly selected instance r_i, the algorithm searches for its two nearest neighbors, one from the same class (nearest hit h) and the other from a different class (nearest miss m). The quality measure $Q[A_i]$ is updated for all attributes A depending on their values for r_i, h, and m. If instances r_i and h have different values in attribute

A, then the attribute A separates two instances with the same class, which is not desirable, and then $Q[A_i]$ is decreased. On the other hand, if instances r_i and m have different values in attribute A, then the attribute A separates two instances with the different class, which is desirable, and then $Q[A_i]$ is increased.

```
for $i = 1$ to $a$
  set $Q[A_i]$=0.0
for $i = 1$ to $num$
  randomly select an instance $r_i$
  find nearest hit $h$ and nearest miss $m$
  for $j = 1$ to $a$
    $Q[A_j] = Q[A_j] - diff(A_j, r, h)/num + diff(A_j, r, m)/num$
```

Figure 2.1 Pseudo code of the Relief algorithm.

The Relief algorithm randomly selects *num* instances, where *num* is a user-defined parameter and usually set to 10 that has been believed to perform well in many situations. In the pseudo code of the Relief algorithm, for a discrete attribute, *diff* (*Attribute, Instance1, Instance2*) is 0 if the values are equal, otherwise it is 1, while for continuous attribute the difference is the actual difference normalized to the interval [0, 1].

The ReliefF algorithm, developed by Kononenko (1994) is not limited to two class problems, is more robust and can deal with incomplete and noisy data. Similarly to Relief, ReliefF randomly selects an instance r_i, but then searches for k of its nearest neighbors from the same class, and also k nearest neighbors from each the different classes. $Q[A_i]$ is the quality measure and will be updated as the follow:

$$Q[A_j] = Q[A_j] - diff(A_j, r, h)/num + \sum_{C \neq class(r)} [P(C) * diff(A_j, r, t(C))]/num \tag{2.8}$$

Thus, it can estimate the capability of attributes to distinguish each pair of classes regardless of which two classes are closer to each other.

2.5 Preference Orderings Learning

Due to the increasing trend to personalization or products and services e-commerce, recommender systems, in various fields, preference learning has been received considerable attention in machine learning community (Goldberg et al, 1992; Herbrich et al., 1998; Kautz 1998; Chajewska et al. 2001; Aiolli and Sperduti, 2004). A key point in these applications is to discover and capture the individual or customer's preferences. Chu and Ghahramani summarized that there are cases for preference learning: label preference learning and instance preference learning.

In label preference learning, the preference relations are over a predefined set of labels for each instance. Label preference learning problems can viewed as multi-class problems. Furnkranz and Hullermeier (2002) discuss a technique, called round robin classification, for handling multi-class problems with binary classifiers by learning one classifier for each pair of classes. The authors later (2003) propose to use a set of pairwise preferences between labels (classes) to predict a total order, a ranking, of all possible labels for a new training example. They seek to induce a ranking function that maps instances to rankings over a fixed set of decision labels, similarly to a classification function that maps instances to single labels. The authors (2003) also investigate different ranking procedures through empirical results. Har-Peled et al. (2002) introduce constraint classification to address multi-classification and ranking problems based on binary classifiers. They propose to label each instance with a set of constraints relating multiple classes. Each such constraint specifies the relative order of two classes for this instance. The goal is to learn a classifier consistent with these constraints. Learning is achieved by a simple transformation mapping each example into a set of examples (one for each constraint) and the application of any binary classifier on the mapped examples. Chu and Ghahramani (2005) develop a Gaussian process algorithm with a new likelihood function for preference learning problems.

In instance preference learning, more related to our research work, a collection of instances are associated with a complete or partial order relation. The training data are a set of pairwise preference between instances, rather than each instance assigned a single class in standard supervised learning. The learning task is to identify the underlying ordering of the instances involved from these pairwise preferences. Haddawy et al. (2003) apply neural network to learn from training data which are pairwise comparisons of instances. Cohen et al. (1999) propose a two-stage approach to learn how to order instances. In the first stage, a preference function is learned and this function returns a numerical measure of how certain that one instance should be ranked before another instance. In the second stage, the learned preference function is evaluated by applying it to order a set of new instances.

In our research context, we consider job scheduling problems, where the original production data are about jobs accompany with their sequences in the schedule. Our problem is a type of instance preference learning problem. The goal of learning here is to learn how jobs are scheduled, which is a kind of preference learning problem. Instead of learning the ranking function directly, we first transform the original jobs dispatched list into pairwise comparison between any two jobs a with a class variable that reveal whether the first job is scheduled earlier than the second one. Then decision tree algorithm is employed to learn the pairwise preference data.

2.6 Summary and Discussion

In this chapter, we addressed that many researchers proposed to apply artificial intelligence approaches in complex scheduling context. In general, these methods focus more on how to select an already known dispatching rule at a given time point, and thus, those dispatching rules would have to be known beforehand. The issue of getting the insights of how direct schedules decisions are made has not been explored thoroughly.

Furthermore, we discussed research work on instance selection by many researchers. The previous methods are very unique in themselves as each has different requirements and underlying principles. But a universal and effective model or method of instance selection has not been explored. Therefore, it is expected that more research in this field would be proposed.

In addition, we reviewed unbalanced data problems, which occur normally in the applications of inductive learning. Related resampling strategies and suitable performance measures are addressed in terms of tackling unbalance data problems. The prior related research work to attribute selection, one of the important data mining issues is also reviewed. Due to the particular characteristics of scheduling data, we reviewed related research work on preference learning. In preference learning, we review literature on label preference learning and instance preference learning respectively.

3 DISCOVERING DISPATCHING RULES

3.1 Introduction

As indicated by the literature in Chapter 2, significant amount of work has been carried out related to the use of artificial intelligence in scheduling. Most of these approaches and applications focus on how to select the best dispatching rule from a certain candidate set. That is to say that there is an assumption that those candidate dispatching rules have to been known in advance. In reality, however, it is usually not possible to get background knowledge about the production scheduling due to the complexity of the system or lack of explicit scheduling knowledge. Thus, the problem of how to discover knowledge about scheduling practice from production data when there is little background information available about the system need to be addressed and researched. In this chapter we propose a data driven approach to apply data mining directly to production data to discover previously unknown meaningful patterns, such as new and interesting dispatching rules and identification of priority jobs. We show how to use this approach to discover key concept and useful knowledge from production data without background information about the system.

The potential benefits of this approach include:

- 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.
- 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.
- In addition to the predictive scheduling rules that allow for dispatching of jobs, structural knowledge may be gained that leads to new rules that improve scheduling performance.

The remainder of this chapter is organized as follows. In Section 3.2, we show how data mining algorithm can be used to learn dispatching rules and give a general framework for a knowledge discovery in production data process. In Section 3.3, using a simple numerical example, we show how this approach works and how knowledge is discovered. In Section 3.4 we evaluate how well such decision trees perform as dispatching rules, and consider the data engineering issue of attribute construction and selection. In Section 3.5, we show how this both improves the performance of the subsequent data mining models and provides important insights into how scheduling decisions are made. Finally, Section 3.6 contains some concluding remarks and future research directions.

3.2 Framework for Inductive Learning on Scheduling

In this section, we propose a general framework for knowledge discovery in production data. Thus, from the data mining perspective, we specify the target concept to be learned to be priority jobs, machine allocation patterns, and a dispatching rule, according to different types of scheduling decisions. In particular, for issue of learning priority jobs, we want to determine where the job considered is a priority job or not, based on its characteristics. If the target concept is to learn machine allocation patterns, we compare two machines and decide if one of them should process the current released job or not, considering the relevant characteristics of the two machines and also the features of the released job. When learning a dispatching rule, given two jobs we want to learn to determine which job should be dispatched first. This knowledge would allow us to dispatch the next job at any given time and create dispatching lists for any set of jobs. Note that we do not need to predict the starting time of each job but simply when we compare two jobs, which job should be processed first. Given this target concept the framework has two main phases: a) data preparation including aggregation, instance selection, attribute construction, and attribute selection, and b) model induction and interpretation (see Figure 3.1).

The initial preparation of data is highly significant since to mine any useful knowledge from the raw production data it must typically be transformed considerably.

For data mining, the database should be represented as a flat data file where the columns represent the attributes of the data and each row of the file represents one piece of information: an example from which we can learn. We refer to each such an example or a row in the file as an instance. The raw production data that is available is likely to be in various formats. These data could for example include job dispatch list, work schedules, bill of materials, and so forth. All of these data sources must be combined into a single flat file.

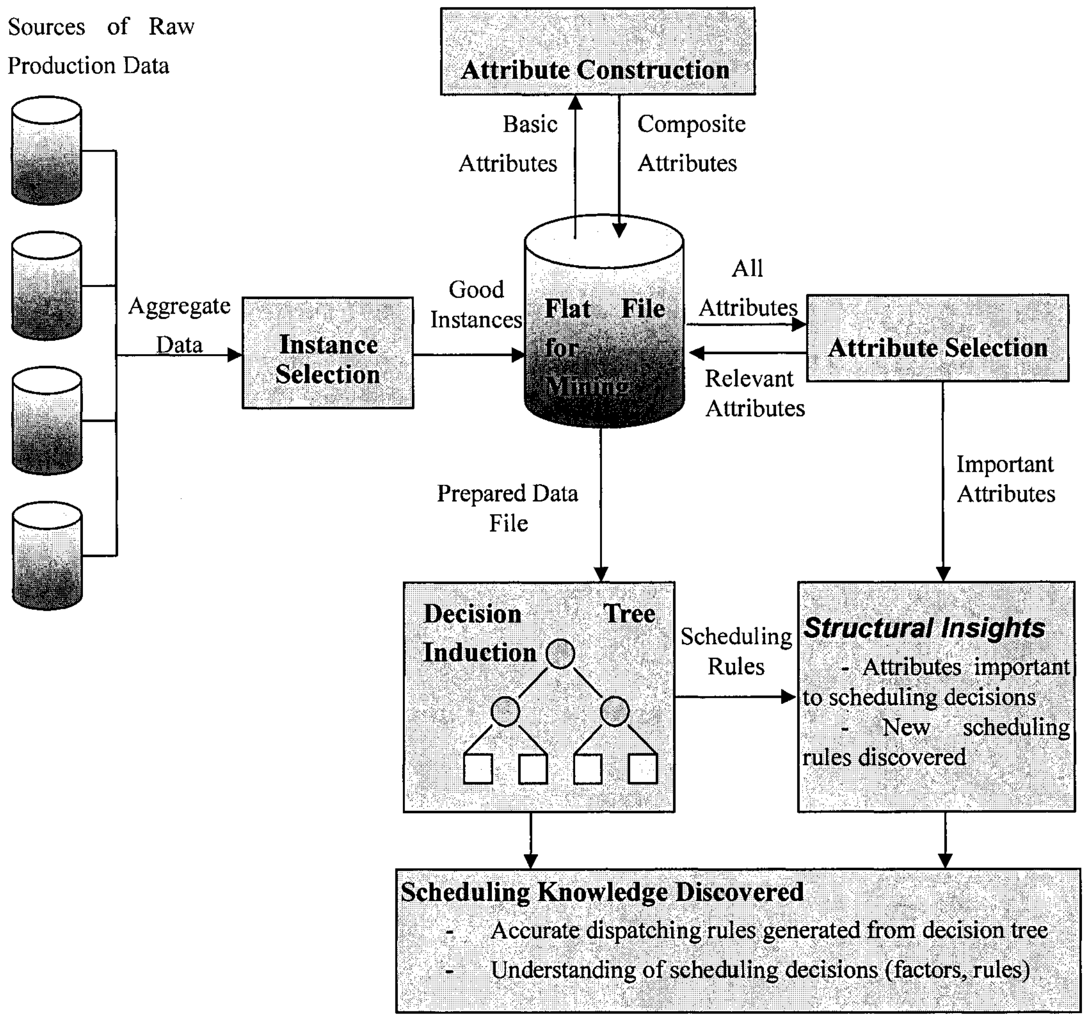

Figure 3.1 High-level framework for discovering scheduling knowledge

There are various data sources that could include released jobs' information, data with respect to machines layout and configuration, data sources of MRP systems and demand details from customers. Beyond the initial combination of these various data sources into a single database, the engineering of this database plays a critical role in the usefulness of the knowledge discovered. The fact is that not all the instances represent good scheduling practice. The existence of those instances generated from bad scheduling decisions will prevent us gaining key knowledge of good scheduling practice after learning. Therefore, effective instance selection approach is necessary to be applied on aggregate data and only those good examples are selected for learning. It is indeed unlikely that the attributes that are recorded as part of the raw production data are the attributes that are the most useful for data mining. Thus, new attributes creation must be considered using both intuitive processes and automated learning. On the other hand, using attribute selection to eliminate certain redundant and irrelevant attributes is also critical to the effectiveness of the subsequent model induction. Attribute selection also has an inherent value in that insightful structural knowledge may be obtained by selecting which attributes are important, that is, which factors most influence the scheduling decision.

After the data file has been constructed a learning algorithm is applied to induce a predictive model for the target concept. There are many ways in which such learning can be achieved, including using decision tree induction, statistical learning, neural networks, and support vector machines. For our framework, we focus on transparent methods, as we believe that black-box models that are difficult or impossible for the user to comprehend are not likely to be used effectively in an actual production environment. In particular, we focus on using decision trees and decision rules derived from such trees.

The decision tree induced using the learning algorithm can be applied directly as a predictive model to predict the target concept. Firstly, if the target is a priority job, the tree will predict the given job is a priority job or not, and also which kind of factors are influential for the decision will be revealed by the tree. Secondly if the target concept is to learn machine allocation scheme, the tree will, according to the characteristics of the current released job, compare any two machines and decide whether the job should be processed by one of them or not. On the other hand, if the target concept to learn is a dispatching rule, the tree will, given any two jobs, predict which job should be dispatched first and can be thought of as a new, previously unknown, dispatching rule. In addition to the prediction, decision trees and decision rules often also reveal insightful structural knowledge that can be used to further enhance the scheduling decision.

3.3 Numerical example

To illustrate the framework described in the last subsection we now consider a simple numerical example. Here, the target concept is to learn a dispatching rule. We assume that the dispatching list shown in Table 3.1 has been used for processing five jobs on a certain single machine system.

Table 3.1 Dispatching list for simple example

Job ID	Release Time	Start Time	Processing Time	Completion Time
J5	0	0	17	17
J1	10	17	15	32
J3	18	32	20	52
J4	0	52	7	59
J2	30	59	5	64

Now assume that we do not know how jobs were scheduled but still wish to construct an automatic system to dispatch new jobs according to the same logic. We thus need to induce from the data above some rule that can be used to dispatch jobs and therefore our target concept to be learned will be a simple dispatching rule. (Note that the actual rule used to create the data in the example is to dispatch the jobs is longest processing time first for all jobs that have been released, but this is considered unknown.)

The first step of the framework is data file construction. Given the target concept, any such data file should have a class attribute called *Job1First*, which can take two values: yes or no. Thus, when all the attributes are specified for two jobs, job 1 and job 2, the objective is to predict if job 1 comes first (class value 'yes') or not (class value 'no'). In addition to make the prediction, the model should ideally reveal some structural knowledge about the production system that reveals why one job is dispatched ahead of another. In this example, that knowledge should be that jobs are dispatched if it is the longest job of those that have been released. With this in mind, we construct the initial data set as shown in Table 3.2.

Table 3.2 Data set constructed for data mining

Job1	ProcessingTime1	Release1	Job2	ProcessingTime2	Release2	Job1ScheduledFirst
J1	15	10	J2	5	30	Yes
J1	15	10	J3	20	18	Yes
J1	15	10	J4	7	0	Yes
J1	15	10	J5	17	0	No
J2	5	30	J3	20	18	No
J2	5	30	J4	7	0	No
J2	5	30	J5	17	0	No
J3	20	18	J4	7	0	Yes
J3	20	18	J5	17	0	No
J4	7	0	J5	17	0	No

As usually for knowledge discovery data files, due to the requirement that each line, or instance, be an example of the concept to be learned that is independent of all other instances, this file contains some inefficiency. However, it is clear that once it has been designed, it is not difficult to construct this data automatically from the dispatching list in Table 3.1. This straightforward transformation will result in dependency of instances in the training set shown in Table 3.2. The assumption of independence is among the most enduring and deeply buried assumptions for machine learning methods. But this assumption is often contradicted in many relational data sets in reality, like in our case. Neville at el. (2003) address the challenge of learning probabilistic models in relational data, where the traditional assumption of instance is violated. However, in this paper, the authors focus on the dependency among attributes, while in our problems, dependency is among instances.

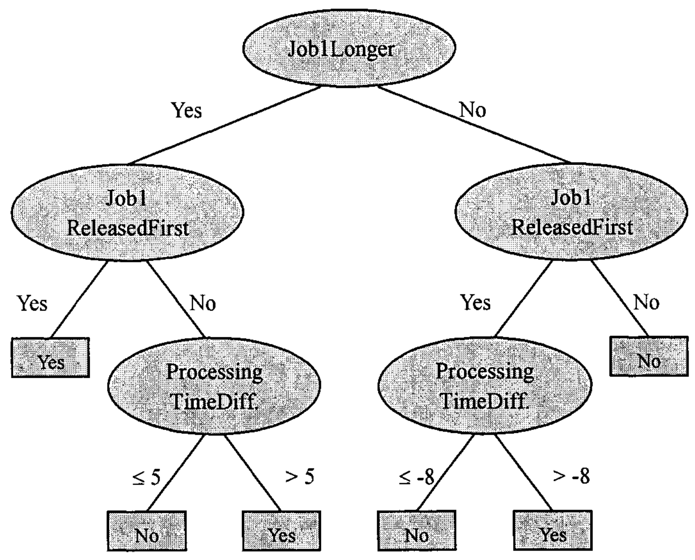

Figure 3.2 Decision tree for dispatching jobs

As discussed above, attribute construction and selection is an essential element of being able to successfully mine scheduling patterns from the data. However, for illustration purposes we first apply the well-known C4.5 decision tree algorithm of Quinlan (1993) to the data in Table 2 directly. For brevity, we omit the actual tree, but it corresponds to the following classification rules, which in this case is a set of dispatching rules:

If *ProcessingTime1* ≤ 7 **then** dispatch Job 2 first.

If *ProcessingTime1* > 7 **and** *ProcessingTime2* ≤ 7 **then** dispatch Job 1 first.

If *ProcessingTime1* > 7 **and** *ProcessingTime2* > 7 **then** dispatch Job 1 first

These scheduling rules dispatch all but one of the training instances correctly, that is, it would replicate the dispatching list in Table 3.1 almost exactly (the order of Job 1 and Job 3 is reversed) for a dispatch order of J5-J3-J1-J4-J2. Since there are ten instances, this corresponds to the model having 90% accuracy. However, it is also quite limited in the amount of structural knowledge or insights that can be obtained.

In order to obtain more meaningful decision tree, and hence dispatching rules, a better data file must be constructed before the decision tree induction. Hence, we add two new attributes, *ProcessingTimeDifference* and *ReleaseDifference*, which are defined as follows: *ProcessingTimeDifference* = *ProcessingTime1* – *ProcessingTime2* and *ReleaseDifference* = *Release1* – *Release2*. We also add two attributes that indicate which job is longer and which is released first: *Job1Longer* and *Job1ReleasedFirst*, that both take values yes or no. We then apply the same decision tree algorithm again, resulting in the decision tree shown in Figure 3.2. Note that a 'Yes' in a leaf node implies that Job 1 should be dispatched first, and vice versa. The decision tree, or alternatively the corresponding decision rules, can be used to directly dispatch new coming jobs. Given any two jobs a selection can be made which job should be dispatched first and hence, a dispatch list could be generated for any given set of jobs that has been released into the system

In addition to being used for prediction, in our framework we also look at what previously unknown structural information can be discovered (see Figure 3.1). Looking at the decision tree we note that there are two easy classification cases. If Job 1 is both longer and is released first, then it is dispatched first (leaf node furthest to the left). Vice versa if neither holds then Job 2 is dispatched first (leaf node furthest to the right). The ambiguous cases occur in between and in those cases the decision as to which job is dispatched first are determined by the difference in processing time between the two jobs. These rules clearly reveal more structural knowledge and get to the point by focusing on the processing time difference. In particular, note that the first rule says that if the processing time of Job 1 is much smaller than that of Job 2, then Job 2 should be dispatched first even if Job 1 is released first. What counts as being 'much smaller' is determined by the data, and in particular how much delay a late release date can possibly cause for this particular system. Thus we have discovered the following nuggets of information:

- If a job is both longer and released ahead of another job it should be dispatched first.

- A job that is released first into the system should wait for an anticipated longer job to be available if that job is much longer, specifically if its processing time is 5 to 8 units longer.

According to the second rule, the machine may be idle for a period of time, which is allowed in our context. It may be informed that there is a job with higher priority will come and the machine needs to wait to perform this more important job.

Such observations can then be assessed to see if scheduling practices should be changed. Should longer jobs be favored? Is it justified to idle machines to wait for those longer jobs? How should a 'much longer' job be defined?

We content that in an actual production situation, such observations could result in significant improvements. Indeed, more may be discovered than could be found out even if the dispatcher could accurately describe the formal process. Here the dispatcher would simply state that jobs are dispatched according to LPT for all jobs that have been released. The induction algorithm is learning from examples that use this rule and the processing time and release time data. Thus, the induced model can take into account the possible range of processing times, and the largest possible delay that can be caused by a job not being released, and thus discovering new structural patterns that may not be explicitly known by the dispatcher.

Despite its simplicity, this example lends itself to numerous observations. It is clear that data mining algorithms, and in particular decision trees, can be used to extract knowledge concerning scheduling concepts from production data such as dispatch lists. Such knowledge can be both predictive, as in determining which of two jobs should be dispatched first, and descriptive, for example by revealing why a job should be dispatched first. However, discovering descriptive knowledge that may be further used to improve schedule performance is not a trivial matter, and as illustrated by the example depends on the representation of the data. Thus, we infer that to mine for information in scheduling data, the data should be represented in ways that are significantly different from traditional scheduling data outputs. Furthermore, a careful construction of new attributes to represent the scheduling data can greatly enhance the value of the models obtained, in particular with respect to the structural knowledge that is gained.

3.4 Decision Trees as Dispatching Rules

The example in section 3.2 motivates that decision tree induction can be used to learn novel dispatching rules. In this section, we evaluate the quality of such dispatching rules more extensively through a series of simulation experiments, which we did in our previous paper (Li and Olafsson, 2003).

3.4.1 Experimental Setup

This evaluation is based on data created using four simple and well known dispatching rules for a single machine problem, namely, weighted earliest due date dispatched first (EDD), weighted shortest processing time dispatched first (WSPT), minimum slackness dispatched first (MS), and earliest release date dispatched first(ERD). Corresponding to each of these rules, there is a priority index $I_j^{RULE}\left(p_j, r_j, d_j, w_j\right)$ that is a function of the basic characteristics of the job, that is, the processing time (p_j), release time (r_j), due date (d_j), and weight (w_j). One thing needs to mention: when apply EDD dispatching rule, weight is considered to dispatch jobs. That is to say, the job with earlier due date and higher weight will be dispatched earlier. This is not only limited to this experiment, but is employed in any experiment related to EDD in our research context.

As in Section 3.2, the basic data p_j, r_j, d_j and w_j are first generated for a set of jobs j=1,2,...,m. We use simulation to generate data for p_j, r_j, d_j and w_j. Law and Kelton (2000) explained how to generate random variants. We employ exponential distributions with different parameters to generate release time r_j and due date d_j, keeping the mean of d_j greater than the mean of r_j. First, random numbers are generated and then use the exponential distribution inverse transformation formula to generate random variants for r_j and d_j. We employ Weibu distribution to generate processing time p_j. The process is similar as before, random numbers are generated and transformed into Weibu variants through Weibu inverse transformation formula. As for weight w_j, first generate random numbers, and if the number is less than 0.3 then assign w_j to 1; if the number is between 0.3 and 0.5 then assign w_j to 3; if the number is between 0.5 to 0.8 then assign w_j to 5; otherwise assign w_j to 7.

In these experiments the total number of jobs used ranges from *m=120* to *m=200.* Given the basic data, the jobs are ordered according to the appropriate priority index $I_j^{RULE}(p_j, r_j, d_j, w_j)$ determined by different dispatching rules, with the job with the lowest index being first, and so forth. This results in a dispatching list similar to Table 3.1. This dispatching list is then transformed into a flat file similar to the one in Table 3.2, except that due dates and weights of each job are also included. As before the concept to be learned is which of two jobs is dispatched first. All experiments are replicated four times with four different data sets with same distribution for each variable, which was found to be sufficient due to relatively low variability in the knowledge discovery process.

We start by addressing the question of whether the decision tree algorithm can accurately replicate the dispatching list generated by the rule. Table 3.3 shows the results when the C4.5 decision tree algorithm is applied to each of the four flat files.

Table 3.3 Accuracy of decision trees in replicating dispatching lists

Rule	**Average Accuracy (%)**	**Standard Deviation**	**Minimum Accuracy (%)**	**Maximum Accuracy (%)**
EDD	96.18	0.28	95.9	96.5
WSPT	96.30	0.36	95.8	96.3
MS	98.20	1.05	97.2	99.3
ERD	99.20	0.24	98.9	99.1

It is clear that in each case the decision tree algorithm accurately discovers how the jobs were dispatched by the particular rule, although the EDD and WSPT rules appear to slightly more difficult to discover. This supports the claim made based on the example in Section 3.3, but as in that example, further improvements can be made by engineering the data.

3.4.2 Attribute Construction and Selection

Data engineering plays a critical role in constructing more accurate models, building more interpretable models, and in providing insights into which factors are most important in making the scheduling decisions. Specifically, the attributes that are recorded as part of the raw production data may not be the attributes that are the most useful for the data mining itself. Thus, new attributes creation must be considered. In some cases this could be done manually using intuitive processes such as those illustrated in the example in Section 3.2 above, but now we show how such new attributes can be discovered automatically.

We use frequent itemset generation (Agrawal et al., 1993) approach to create new attributes. In this context, an item is an attribute value pair, that is an attribute along with one of its possible values, and an itemset is a simply a set of items. An item set is called frequent if it occurs some minimum number of times in the database, that is, meets the minimum support. With this approach, new composite attributes are added based on attributes that occur frequently together, which allows the subsequent learning algorithm to use conjunction of attribute values. For example, if itemsets involving processing time of job 1, p_1, and processing time of job 2, p_2, occur frequently together then we attempt to construct new composite attribute using this pair with four basic arithmetic operations: $p_1 - p_2$, $p_1 + p_2$, ${p_1}/{p_2}$, and $p_1 \cdot p_2$.

In addition to creating new attributes, using attribute selection to eliminate certain attributes is also critical to the effectiveness of the subsequent model induction. Attribute selection in general is an important part of the knowledge discovery for numerous reasons.

It can be used to eliminate redundant and irrelevant attributes from a data set, resulting in a dimensionality reduction that reduces the learning time needed for induction algorithms that are applied to the data set, and in many cases also results in better (that is, more accurate) predictive models. Careful attribute selection can improve the scalability of a data mining system as the induction is usually much faster with fewer attributes, and finally, attribute selection also has an inherent value in that insightful structural knowledge may be obtained by selecting which attributes are important.

To illustrate how the engineering of the data improves the performance of the decision tree models, we compare and analyze the difference between decision trees before and after data engineering (attribute construction and selection) in accuracy and size. Table 3.4 shows the accuracy of the decision trees after attribute construction and selection for the same data sets as those reported in Table 3.3.

Table 3.4 Accuracy comparison of decision trees with and without data engineering

Rule	Data Set	Accuracy (Original Tree)	Accuracy (Attribute construction and attribute selection)	Difference	Average of Difference	Standard Deviation. of Difference	Confidence Interval (95%) of Difference
EDD	1	96.00%	99.00%	3.00%	2.93%	0.22%	(2.71%, 3.14%)
	2	96.50%	99.20%	2.70%			
	3	96.30%	99.10%	2.80%			
	4	95.90%	99.10%	3.20%			
WSPT	1	96.30%	98.70%	2.40%	2.55%	0.50%	(2.06%, 3.04%)
	2	96.50%	99.10%	2.60%			
	3	96.60%	98.60%	2.00%			
	4	95.80%	99.00%	3.20%			
MS	1	97.80%	97.80%	0.00%	0.23%	0.40%	(-0.17%, 0.62%)
	2	99.30%	99.20%	-0.10%			
	3	98.10%	98.30%	0.20%			
	4	97.20%	98%	0.80%			
ERD	1	98.90%	99.90%	1.00%	0.70%	0.35%	(0.35%, 1.05%)
	2	99.10%	99.90%	0.80%			
	3	99.40%	99.90%	0.50%			
	4	99.40%	99.90%	0.50%			

From Table 3.4, we can see that there is a significant difference in accuracy after performing data engineering, apart from MS experiments. Positive upper and lower 95% confidence levels indicate improvements in accuracy brought by attribute construction and selection, except for MS. The reason why there is no significant improvement on accuracy for MS cases is that the new derivative attributes created and selected do not contribute as much as other new derivative attributes do in other experiments.

However, as noted before the primary benefit of attribute creation and selection may not be the improved accuracy, but rather simpler models and structural insights. Thus, Table 3.5 compares the size of the decision tree both before and after the data engineering.

Table 3.5 Reduction in size of decision trees after data engineering

Rule	Data Set	Size (Original)	Size (Attribute construction and attribute selection)	Size Reduction	Average of Size Reduction	Standard Deviation of Size Reduction	Confidence Interval (95%) of Size Reduction
EDD	1	196	61	135	113	18	(95, 130)
	2	154	63	91			
	3	174	59	115			
	4	178	68	110			
WSPT	1	171	55	116	117	7	(110, 124)
	2	167	54	113			
	3	171	59	112			
	4	190	62	128			
MS	1	50	38	12	16	8	(8, 24)
	2	58	39	19			
	3	67	59	8			
	4	136	110	26			
ERD	1	40	2	38	57	14	(43, 71)
	2	56	2	54			
	3	72	2	70			
	4	68	2	66			

From Table 3.5, we can see that there is significant difference in the size of decision trees before and after data engineering. It is clear that the decision trees generated after the data engineering are much smaller, and hence typically simpler to interpret, than the trees generated using the original data. The attribute creation and selection is the least useful for the data generated using the MS rule, where there is only 27% reduction in tree size. On the other hand, for the data generated by the ERD rule there is a very dramatic reduction down to only two attributes, which is a reduction of 97% in size.

3.5 Obtaining Structural Insights from Decision Trees

Data mining not only leads to generate predictive models applicable for future data, but also makes the models descriptive and reveals useful structural insights. Some of unsupervised methods, such as clustering and association rule discovery, can be used as descriptive tools. Clustering assumes data is not labeled with class information. The goal is to create structure for data by objectively partitioning data into homogeneous groups where the within group object similarity and the between group object dissimilarity are optimized. This technique has been used extensively and successfully in discovering structure and insights from data where domain knowledge is not available or incomplete. Association rule discovery aims to discover interesting correlation or other relationships in large database. In section 3.3 we show how to use frequent item set generation to construct new composite attributes. Although transparent classification methods, such as decision tree, mainly focus on generating a set of grouping rules which can be used to classify future data, interesting and useful structural insights and knowledge can also be revealed by the attributes used in the model.

In section 3.4, we show that improvements in accuracy are achieved by first creating derived attributes and then selecting a subset of attributes that is the most important. To obtain some insights into what is important it is interesting to consider what composite attributes are created for each data set. Table 3.6 shows the attribute pairs used as well as the number of derived attributes used by the model (after attribute selection).

Table3. 6 Attributes discovered by the attribute creation and selection process

Rule	**Priority Index**	**Attribute Pairs Used**	**Number of Derived Attributes Used**
EDD	$I_j^{EDD} = \frac{d_j}{w_j}$	(w_1,d_1) *, (w_2,d_2) *	2
WSPT	$I_j^{WSPT} = \frac{p_j}{w_j}$	$(w_1,p_2),(w_2,p_1)$, (w_1,p_1), (w_2,p_2)	2
MS	$I_j^{MS} = \max\left\{\frac{d_j - p_j - t}{w_j}, 0\right\}$	(d_1,r_1), $(d_2,r_2),(d_1,d_2)$, (w_2,d_1), (w_2,p_2)	2-3
ERD	$I_j^{ERD} = r_j$	(r_1,r_2) *	1

* *Used by every data set*

Some attribute pairs were used in every replication, whereas others were only used by some. For convenience, the table also shows the priority index for each rule. These indices are of course unknown by the learning algorithm, but ideally one would expect the attribute construction and selection to discover the components of the relevant index. We note that for the EDD rule, a composite of the weight and due date of each attribute was discovered by the attribute construction in every replication. This is quite intuitive since the priority index is $I_j^{EDD} = \frac{d_j}{w_j}$, and two composite attributes $\frac{d_1}{w_1}$ and $\frac{d_2}{w_2}$ are thus sufficient to determine which job is dispatched first, Job 1 or Job 2. We emphasize again that the EDD rule is assumed unknown a priori, but the data mining has discovered that $\frac{d_1}{w_1}$ and $\frac{d_2}{w_2}$ are most important factors in the scheduling decision, which is the essential structural insight for this particular system.

Similarly, for ERD, Job 1 goes ahead of Job 2 if and only if $r_1 < r_2$, so a composite attribute of $r_1 - r_2$ is sufficient. The attribute construction discovers this, and creates this composite attribute for every replication. For the WSPT and MS datasets, the process does not always discover the same attribute pairs. For the WSPT two composites of processing time and weight were used every time, but not always the same combination. Finally, for the dataset generated by the more complex MS rule, two or three composite attributes were used each replication, involving composites of due dates, release times, weights, and processing time. Overall, these results show that useful composite attributes are constructed and these attributes provide significant insights into what factors are important in each scheduling situation.

3.6 Summary and Discussion

In this chapter we address the problem that whether unknown knowledge behind historical scheduling data could be identified through direct data mining without background knowledge about the system. We have introduced a new framework for applying data mining directly to discover unknown dispatching rules from production data. We transform original dispatched list into a flat file of comparisons of any pair of jobs. A new target concept is specified to reveal whether the first job is dispatched earlier than the second job. This new target and flat file transformation enable us to apply data mining to learn dispatching rules directly without any background knowledge about the scheduling practice in advance. We show that by using decision tree algorithm to learn flat files of comparisons between each pair of jobs, we can learn knowledge about how jobs are dispatched. The induced decision tree model not only is a predictive model that can be used as a dispatching rule, but from the model previously unknown structural knowledge can be obtained that provides new insights and may be used to improve scheduling performance. Furthermore, we develop methods for using frequent item set generation to construct composite attributes that improve the performance of the predictive models, and in combination with attribute selection method reveal what factors are the most influential in making the scheduling decisions.

4 OPTIMAL INSTANCE SELECTION

4.1 Introduction

In Chapter 3, we developed a methodology for applying data mining to learn directly from scheduling data. The resulting decision tree model can be applied as a predictive model, in scheduling context, as a new dispatching rule. As a result, the scheduling decision process can be automated by applying the tree model directly to future data. In this chapter we address another important problem how to employ knowledge to improve scheduling practice.

When we apply data mining to production data, a new interesting question emerges: do the historical scheduling data instances all represent good scheduling practice for learning? If there is a new worker coming to do the scheduling job, he probably could not do the job very well because of lacking experiences or expertise. Therefore, the data generated by the system when he was working may include some data not so suitable for learning. In this situation, if we apply the decision tree learning algorithm directly to this data, then the tree model after learning could not perform well enough as a new dispatching rule. Direct data mining of production data can mainly mimic scheduling practices. Thus, we propose the hypothesis that if we could identify the subset of good instances that represent best scheduling practice, then the induced decision tree model will perform ideally as an experienced scheduler, which means the scheduling practice could be improved.

Motivated by this hypothesis, we first investigate whether tree models, induced from different subsets of scheduling data, perform differently as a new dispatching rule. Then we propose a novel instance selection methodology for scheduling, by combining data mining with optimization. In this approach, we use a genetic algorithm to find a heuristic solution to the optimal instances selection problem, and then induce a decision tree from this subset of best instances. The optimal instance selection can be viewed as determining the best practices from what has been done in the past, and the data mining can then learn new dispatching rules from those best practices. Through scheduling performance analysis, it is shown that induced model with optimal instance selection performs better than the original heuristic dispatching rule, which indicate scheduling practice could be improved through optimal instances selection.

The remainder of this chapter is organized as follows. First, investigation into instance selection for scheduling is addressed. Then the genetic algorithm based instance selection methodology is presented to show how to identify best scheduling practices from scheduling data. Numerical results are also presented to show the scheduling performance of the induced decision tree after employing instance selection phase.

4.2 Instance Selection for Scheduling

As indicated in Chapter 3, the instances in the aggregated data are not usually all good for learning. Some examples may represent bad scheduling practice, for example, when a new scheduler without much expertise did the scheduling task. Such instances are not good for learning, because they will restrict us to discover important and useful knowledge that we can get if we only learn from good examples. Therefore, instance selection becomes a critical data engineering issues and the exploration of effective instance selection methodologies is very meaningful for knowledge discovery in production scheduling.

Instance selection is firstly related to which kind of performance measure (i.e. objective function) is concerned, objective function value. Different objective function will lead to different set of data selected for learning.

To investigate the value of instance selection, a set of simulation experiments is performed. Here we also consider the job scheduling problem. Basic data p_j, r_j, d_j, and w_j are first generated for a set of jobs j=1,2,...,m in the same way as shown in Section 3.4. In these experiments the total number of jobs used is $m=200$. We split the whole set of jobs into four subsets according to the value of release time r_j. There are fifty jobs in each subset after splitting. Four simple and well known dispatching rules for a single machine problem, namely, weighted earliest due date (EDD), weighted longest processing time first (LPT), weighted shortest processing time first (WSPT), and earliest release date (ERD) are applied to these four subsets respectively. Different objective function values are evaluated as the performance measures of the dispatching rules. We do experiments in this way just to simulate a kind of practical scheduling environment: there are four schedulers with different scheduling knowledge and they schedule jobs during four sequentially different time slots respectively. Those objective functions of interest include weighted maximum lateness ($WL_{\max}$), total weighted tardiness ($\sum w_j T_j$), total weighted completion times ($\sum w_j C_j$), and makespan ($C_{\max}$). The calculation formula for these four measures are shown below, where C_j is the completion time for job j.

Maximum Lateness:

$$WL_{\max} = \underset{j}{Max} \quad w_j * (\max\{0, C_j - d_j\}) \tag{4.1}$$

Makespan:

$$C_{\max} = \max_j \{C_j\} \text{ Completion time of the final job.} \tag{4.2}$$

Total Weighted Tardiness:

$$\sum_{j=1}^{n} w_j T_j = \sum_{j=1}^{n} w_j * \max(C_j - d_j, 0) \tag{4.3}$$

Total Completion Times:

$$\sum_{j=1}^{n} w_j * C_j \tag{4.4}$$

Table 4.1 Objective function values by different dispatching rules applied on subsets

Subset #	Dispatching Rule Applied	$WL_{\max}$	$\sum w_j T_j$	$\sum w_j C_j$	$C_{\max}$
Set 1	EDD	** **379**	3972	76866	1009
Set 2	LPT	1690	13316	75029	777
Set 3	WSPT	3084	15613	** 74869	1037
Set 4	ERD	782	7205	92827	989

Note that weighted earliest due date (EDD) is known to result in smaller weighted maximum lateness ($WL_{\max}$) than do other three dispatching rules, which is revealed in the above table ($WL_{\max}$=379 is the minimum with EDD rule). Similarly, weighted shortest processing time first (WSPT) will result in smaller weighted completion times ($\sum w_j C_j$) than do other three dispatching rules, which is revealed in the above table ($\sum w_j C_j$=74869 is the minimum with WSPT rule).

We choose weighted maximum lateness ($WL_{\max}$) as the performance measure for selecting good instances. Since the first subset of jobs dispatched with EDD rule lead to minimum $WL_{\max}$, we select this set of scheduled jobs' data and transform them into flat-file form for learning. In order to get more general results, we normalized the release time and due date of these selected jobs according to the minimum release time and maximum due date in this sample data set. New derivative attributes are constructed from frequent itemset (section 3.1) and attributes selection is also employed during learning, as was described in section 3.3.

After inductive learning, we get a decision tree model and apply it to the whole data set as a new dispatching rule. Then a new dispatching list is generated and the objective function value $WL_{\max}$ is calculated, which is 6988. If EDD rule is applied directly to the whole data set, the value of $WL_{\max}$ equals to 4459. Therefore, there is a large gap between the performance of the new dispatching rule and the performance of the EDD rule when the new dispatching rule is induced from all the data.

Although there are not many positive results from the above experiments, we can still get some insights. The first insight is that, the subset of jobs scheduled by EDD may include many good instances but probably some instances are not good enough to learn from, although this subset is with the minimum value of $WL_{\max}$. Good instances means that learning from those instances can lead to good decision tree models, which can perform well enough as dispatching rules. While in other subsets, jobs dispatched by other dispatched rules, there may be some good instances worthy of learning. From this point of view, we are interested in finding out how to select good instances from these four subsets of scheduled jobs data for learning, when the performance measure considered is $WL_{\max}$.

We choose tardiness as the scheduling performance measure, and select all the "un-late" jobs from all the four subsets. "Un-late" jobs are those jobs finished before their due dates. In this way, total of fifty eight jobs are selected, twenty eight jobs from subset 1, fifteen from subset 2, ten from subset 3, and five from subset 4. After replicating the experiment in the same way as before, we find a decision tree after learning in only eleven leaves and with accuracy 94.5%. After applying this new dispatching rule to the whole data set, the objective function value of $WL_{\max}$ is 6717, which is somewhat improved a little compared with the previous decision tree before sampling ($WL_{\max}$=6988). In this experiment, tardiness is considered. However, probably there are some other important factors should be taken into account. Thus, more research should be done for this part.

The second insight is that from the decision tree model, can we get some knowledge that can be used to do improvements on the current model? The answer is yes. From the decision tree after instance selection, we prune the tree into a new one only with six leaves. The following set of dispatching rules is generated from the pruned tree:

If $d_1 < d_2$, then dispatch job1 first

Except if $r_2 = 0$ or $\frac{w_1}{w_2} \le 0.6$

Else: dispatch job2 first

Except if $w_1 > w_2$, $w_2 \le 3$ and $r_2 > 0.000305$

Note that r_2 is the normalized released time according to earliest release time and longest due date.

After applying this set of dispatching rule to the whole data set, we get a new dispatching list for these two hundred jobs and calculate the objective function value of $WL_{\max}$, which is 5516. Compared to the objective function values of $WL_{\max}$ before instance selection (6988) and after instance selection but before aggressive pruning (6717), we can see this set of dispatching rules performs much better with regards to maximum lateness' performance measure. Therefore, this kind of analysis and aggressive pruning is useful and effective.

We performed more experiments to explore direct optimization on the schedule. We first find the latest job in the schedule generated by the decision tree and construct its neighborhood set, in which each neighbor is a new schedule. We obtain each new schedule by interchanging the positions of the latest job and of any one of other jobs. Then, we calculate the objective function value of each new schedule in the neighborhood. Arbitrarily select one from the schedules with the minimum objective function value. The selected schedule is the new object schedule. Then replicate the above steps.

In this way, after 20 iterations, we find a good schedule ($WL_{\max} = 4388$) from the original schedule generated by applying decision tree after instance selection. This schedule performs even better than EDD rule ($WL_{\max} = 4459$) does. From this fact that only with twenty iterations, a good enough schedule can be found, it is clear that this approach can be effective.

4.3 Genetic Algorithm Based Instance Selection Methodology

In this section we present a new instance selection methodology for scheduling based on genetic algorithm for scheduling. In particular, we first discuss the genetic algorithm and its strength and applications in various areas, then investigate how to employ genetic algorithm in our instance selection context, finally, we evaluate the scheduling performance of the decision tree model with this methodology through a series of simulation experiments.

4.3.1 Genetic Algorithm

First pioneered by John Halland at University of Michigan in the 60s, genetic algorithms(GA) has been widely studied, experimented and applied in many fields. Not only do GAs provide alternative methods to solving problems; it consistently outperforms other traditional methods in many of the problems. Many of the real world problems involved finding optimal parameters, which might prove difficult for traditional methods but ideal for GAs. GAs have been applied successfully to a variety of learning tasks and to other optimization problems. For example, they have been used to learn collections of rules for robot control and to optimize the topology and learning parameters for artificial neural networks.

GAs are a class of optimization algorithms inspired by population genetics and the Darwinian principle of natural selection. Given an objective function, the typical GA begins with a random population (generation) of solutions (chromosomes). Each solution is represented by a sequence of characters (genes) each having certain values (alleles). By crossover and mutation the best solutions (as measured by some fitness value), the GA produces a new population of improved solutions (offspring). The average fitness of the population, as well as the fitness of the best solutions, improves at each generation. This process continues until the GA has determined an acceptable solution to the problem (as determined by the developer).

The crossover operator produces two new successors from two parent solutions, by copying selected bits from each parent. The bit at position i in each offspring is copied from the bit at position i in one of its parent. The choice of which parent contributes the bit for position i is determined by an additional string called the crossover mask (Mitchell, 1997). There are basically three types of crossover: single-point crossover, two-point crossover, and uniform crossover.

In single-point crossover, one offspring takes the first n bits from the first parent and its remaining bits from the second parent. The second offspring uses the same crossover mask, but switches the roles of the two parents. It uses the first n bits from the second parent and its remaining from the first parent. The crossover mask contains n contiguous

1s, and the following necessary number of 0s. This results in offspring in which the first n bits are contributed by one parent and the remaining bits by the second parent. Each time the simple-point crossover operator is applied, the crossover point n is chosen at random, and the crossover mask is then created and applied.

In two-point crossover, offspring are created by replacing intermediate segments of one parent by the middle part of the second parent string. The crossover mask contains the fist n_0 0s and a following n_1 1s, then the necessary number of 0s. Each time the two-point crossover operator is applied, a mask is generated by randomly choosing the integers n_0 and n_1.

In uniform crossover, bits of offspring are generated by uniformly sampling from the two parents. Accordingly, the crossover mask is generated as a random bit string with each bit chosen at random and independent of each others.

In addition to recombination operators that produce offspring by combining segments of two parents, there is another type of operator, called mutation, produces offspring from a single parent. The mutation operator only produces some small changes to the bit string by choosing a single bit at random, then change its value. Mutation is often performed after crossover.

4.3.2 Genetic Algorithm Based Instance Selection Methodology

In this section we discuss a novel instance selection methodology that combines data mining with optimization for effective production scheduling. In this approach, we use a genetic algorithm to find a heuristic solution to the optimal instances selection problem (Wu and Olafsson, 2005), and then induce a decision tree from this subset of instances. Figure 4.1 shows the process and procedure of this approach.

GA Based Instance Selection Algorithm

Notation:

T_r: Training data set,

T_e : Test data set,

$S_i = [e_1^i, e_2^i, e_3^i, \ldots, e_{N_i-2}^i, e_{N_i-1}^i, e_{N_i}^i]$, where S_i is ith subset, $i = 1, \ldots, m$, e_k^i is the kth instance in subset S_i, N_i is the total number of instances in S_i,

m : Total number of subsets of training data set,

g : The number of generations performed,

c : Crossover rate, $c \in (0,1)$,

m_u : Mutation rate, $m_u \in (0,1)$.

Algorithm:

Step 1: Partition the training set T_r into m subsets through random sampling;

Step 2: Apply decision tree algorithm to each subset S_1, S_2,.., S_m;

Step 3: Apply each induced tree from step2 $(Tree_1, Tree_2, \ldots, Tree_m)$ to the test set T_e;

Step 4: Use fitness function to evaluate the performance of all the trees, and rank the trees with their related subsets according to the trees' performance;

Step 5: Perform GA operations:

a. Selection: select the top $(1-c)m$ subsets and keep them intact into the next generation;
b. Crossover: for the remaining $cm/2$ pairs, perform two points crossover;
c. Mutation: randomly select m_u subsets to perform mutation operation. Randomly replace one instance in the selected subset by one instance randomly selected from the original training data set.

Step 6: New subsets are created from Step 5 as the next new generation, then replicate Step 2 to Step 6, until identify a subset and a related tree with ideal performance.

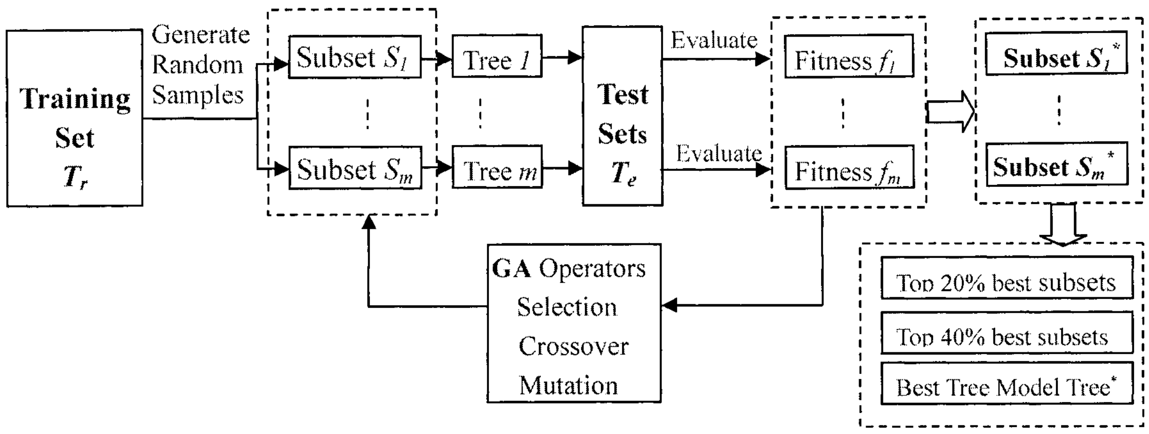

Figure 4.1. GA based Instance Selection Approach

In Step 1, the training data set is divided into a certain number (m) of subsets, by randomly sampling. Then, for each subset, C4.5 decision tree algorithm is applied respectively (see Step 2). As a result, m different decision trees are generated by learning all the subsets of the training data. In the next step, all these decision trees are applied to the test data set as a predictive model to predict which job will be scheduled earlier when compare any two jobs. Then the sequence of the whole set of jobs can be derived. That is, each decision tree will work as a new dispatching rule to dispatch a new set of jobs, represented in test data set.

In the following Step 4, a defined fitness function will be used to evaluate each decision tree's performance. The exact definition and format of the fitness function will vary according to different applications. In our research context, the decision tree will perform as a new dispatching rule; therefore, what we concern is the scheduling performance of the decision tree. Here, we choose maximum lateness ($WL_{\max}$), over all the scheduled jobs as the performance measure. That is, the objective of the optimal instance selection is to identify the best data set which will generate an ideal decision tree with best scheduling performance. Thus, the fitness function is defined as the following formula:

$$f(S_i) = Max \quad w_j * (\max\{0,(C_j - d_j)\}) \qquad i = 1,...,m; \; j = 1,...,N_i \qquad (4.2)$$

Where, similarly to before, S_i stands for the ith subset of training data; w_j is the jth job's weight; C_j represents the completion time of the jth job; d_j represents the due date of the jth job; N_i stands for the total number of jobs in subset S_i.

The lateness of the jth job is represented by $\max\{0,(C_j - d_j)\}$. If the job is finished processing before its due date, which means $C_j - d_j < 0$, the lateness is 0. On the other hand, if it is finished later than its due date ($C_j > d_j$), the lateness equals $C_j - d_j$. The product of lateness and weight w_j represents the weighted lateness.

After evaluation by fitness function, all the trees are ranked according to their fitness. Select the top $\lfloor (1-c)m \rfloor$ subsets and keep them intact into the next generation. As for the remaining $\lceil cm \rceil$ subsets, a two-point crossover operator is performed for these $\lceil cm/2 \rceil$ pair of subsets. Both two-point and single point crossovers are the most common types of crossover in Genetic Algorithm. I chose two-point crossover in the GA based instance selection method. I also performed experiments to compare the scheduling performance of GA based instance selection method with these two crossovers on same data sets respectively. The results show that the scheduling performance is very similar between these two crossovers, which indicate that whether to employ single point crossover or two-point crossover does not make big difference on the scheduling performance of decision trees. Because there is no ordering to the instances, it is not surprising that single point vs. two-point does not make much difference. The two-point crossover operator is shown below.

Parents **Offspring**

$$\begin{tikzcd}
S_i = [e_1^i, e_2^i, \underline{e_3^i, \ldots, e_{N_i-2}^i}, e_{N_i-1}^i, e_{N_i}^i] \arrow[r] \arrow[dr] & S_i^{'} = [e_1^i, e_2^i, \underline{e_3^j, \ldots, e_{N_i-2}^j}, e_{N_i-1}^i, e_{N_i}^i] \\
S_j = [e_1^j, e_2^j, \underline{e_3^j, \ldots e_{N_j-2}^j}, e_{N_j-1}^j, e_{N_j}^j] \arrow[r] \arrow[ur] & S_j^{'} = [e_1^j, e_2^j, \underline{e_3^i, \ldots e_{N_j-2}^i}, e_{N_j-1}^j, e_{N_j}^j]
\end{tikzcd}$$

Figure 4.2 Two point crossover operator in GA Based Instance Selection

Where, S_i and S_j represent two parents respectively, while $S_i^{'}$ and $S_j^{'}$ represent two offspring after crossover operation respectively. $S_i^{'}$ is created by substituting the intermediate segment of parent S_i (instances from e_3^i to $e_{N_i}^i$) into the middle of the parent S_j (instances from e_3^j to $e_{N_j}^j$). While $S_j^{'}$ is created by substituting the intermediate segment of parent S_j (instances from e_3^j to $e_{N_j}^j$) into the middle of the parent S_i (instances from e_3^i to $e_{N_i}^i$). In this figure, the intermediate segment starts from the third instance in the subsets, but actually, the starting position of the intermediate segment is determined randomly.

In the next mutation operation, m_u subsets are selected randomly. For each selected subset, randomly choose one instance, and replace it by another instance randomly selected from the original training data set.

When all these GA operations performed, a new generation begins. The best decision trees with their related data sets in each generation are kept in the record through the whole process. After a certain number of generations, a subset of training data and the related decision tree with the best performance can be identified.

We contend this optimal instance selection methodology is a broadly applicable approach. As long as users define their own fitness function based on different applications, all the other steps remain the same. In the next section, a numerical example is given out to show how to apply this optimal instances selection approach to identify the best scheduling practice.

4.3.3 Numerical Example

In this section, we will use one numerical example to illustrate how to apply the GA based instance selection methodology in scheduling context. A small data set with 10 jobs is created as the training data with release time, due date, processing and weights. The data set creation process is similar to data sets creation in simulation experiments explained Section 3.4. Also similar to the numerical example in Chapter 3, single machine system is concerned here, and ten jobs are scheduled according to EDD (earliest due date dispatching rule). The original variable set is the same as before: attribute *Release* stands for the job's released time, specifically, when the job is available for processing; attribute *DueDate* stands for the due date of the job; *Processing* stands for the processing time needed to finish the job by the machine; *Weight* reveals the priority of the job; *Sequence* represents the order in which to process the job within this whole set of jobs after scheduling. The original data set is shown below in Table 4.2.

Table 4.2 Training data dispatch list in numerical example for GA based instance selection methodology

Job_ID	Release	DueDate	Processing	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

This original data set is transformed into a flat file (shown in Table 4.3) by the same strategy explained in the research framework of applying data mining in scheduling (see Chapter 3). Each instance represents a comparison between two jobs. *R1*, *D1*, *P1*, and *W1* represent the released time, due date, processing time, and weight of the first job respectively. Similarly, *R2*, *D2*, *P2*, and *W2* represent the released time, due date, processing time, and weight of the second job respectively. The last attribute, *Job1 Scheduled First* is the class attribute and will be "yes", if Job1 is scheduled earlier than Job2 according to the original dispatch list, and vice versa.

Apart from these original attributes, we create four derivative categorical attributes: *Job1_Released_Earlier*, *Job1_Due_Earlier*, *Job1_PT_Lower*, and *Job1_Weight_Higher*. These four attributes are interaction terms related a pair of original attributes. In particular, the value for *Job1_Released_Earlier* will be "yes" if Job1 is released earlier than Job2, and vice versa; the value of *Job1_Due_Earlier* will be "yes" if Job1 dues earlier than Job2, and vice versa; the value of *Job1_PT_Lower* will be "yes" if Job1 needs shorter processing time than does Job2, and vice versa; the value for *Job1_Weight_Higher* will be "yes" if Job1 is with higher weight than is Job2, and vice versa..

Table 4.3 Training data flat File in numerical example for GA based Instance selection methodology

Job 1	R1	D1	P1	W1	Job 2	R2	D2	P2	W2	Job1 Released Earlier	Job1 Due Earlier	Job1 PT Lower	Job1 Weight Higher	Job1 Scheduled 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

As explained in Section 3.3, transformation from dispatch list into flat file will result in dependency of instances in the training set shown in Table 4.3. The assumption of independence for normal machine learning methods is violated. Exploring the effects of dependent data on induced model would be an important issue for future research.

Similarly, a test data set is generated with same distributions as in above training data. In this way, test data can be seen as the new data collected from the same system and used to evaluate the performance of the decision tree model without overfitting. Table 4.4 shows the test data set dispatch list by EDD dispatching rule and Table 4.5 shows the test data flat file.

Table 4.4 Test data dispatch list in numerical example for GA based instance selection methodology

Job_ID	Release	Due Date	Processing	Weight	Sequence(EDD)
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

Table 4.5 Test data flat File in numerical example for GA based Instance selection methodology

Job1	R1	D1	P1	W1	Job2	R2	D2	P2	W2	Job1 Released Earlier	Job1 Due Earlier	Job1 PT Lower	Job1 Weight Higher	Job1 Scheduled First
1	11	20	7	3	2	3	24	6	5	no	yes	no	no	no
1	11	20	7	3	3	10	27	8	7	no	yes	yes	no	no
1	11	20	7	3	4	4	57	11	3	no	yes	yes	no	yes
1	11	20	7	3	5	8	9	7	3	no	no	yes	no	no
1	11	20	7	3	6	0	49	13	3	no	yes	yes	no	no
⋮														⋮
9	9	21	15	5	10	5	44	4	1	no	yes	no	yes	yes

If we apply C4.5 decision tree algorithm directly to learn from the training data flat without instance selection, we can get the following tree model (Figure. 4.3) with five leaves.

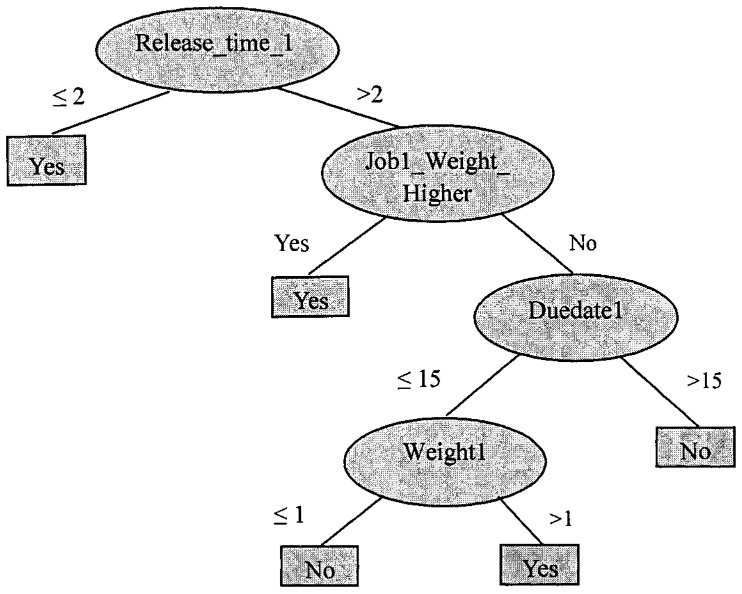

Figure 4.3 Decision tree without instance selection in the numerical example

From this tree model we can derive the following set of dispatching rules:

If Release_Time_1≤ 2 **or** Job1 weight is higher, **then** schedule Job1 first;

If Release_Time_1>2 **and** Job2 Weight is higher **and** Duedate1>15, **then** schedule Job2 first;

If Release_Time_1>2 **and** Job2 Weight is higher **and** Duedate1≤ 15 **and** wehght1≤ 1, **then** schedule Job2 first;

If Release_Time_1>2 **and** Job2 Weight is higher **and** Duedate1<=15 **and** wehght1>1, **then** schedule Job1 first;

After applying this decision model, that is, applying the above set of dispatching rule to the test data set, we can get the following job sequence: J3-J2-J5-J6-J7-J8-J9-J4-J1-J10. Figure 4.4 shows the Gannt chart for this schedule and the weighted lateness for each job in this schedule.

In Figure 4.4, a series of horizontal bars represent the sequence of job for processing. The first job, J3, starts after it is released at 10. Before J3 is finished, J2 has arrived to the machine, thus, J2 immediately starts after J3 is finished. In the similar way, J10 is last job to be processed. Another series of vertical bars represent each job's relevant weighted lateness ($w_j * \max\{0,(C_j - d_j)\}$). We can see that there are four jobs (J3, J2, J6, and J10) finished with no lateness, but the $WL_{\max}$ =300 (J9's weighted lateness), which shows the scheduling performance of the above decision tree without instance selection. Based on the dispatched sequence in the original test data, the relevant maximum lateness can be calculated, $WL_{\max}$ =210. The decision tree algorithm is only trying to learn and mimic the EDD rule, thus, it is not expected to perform better than EDD rule.

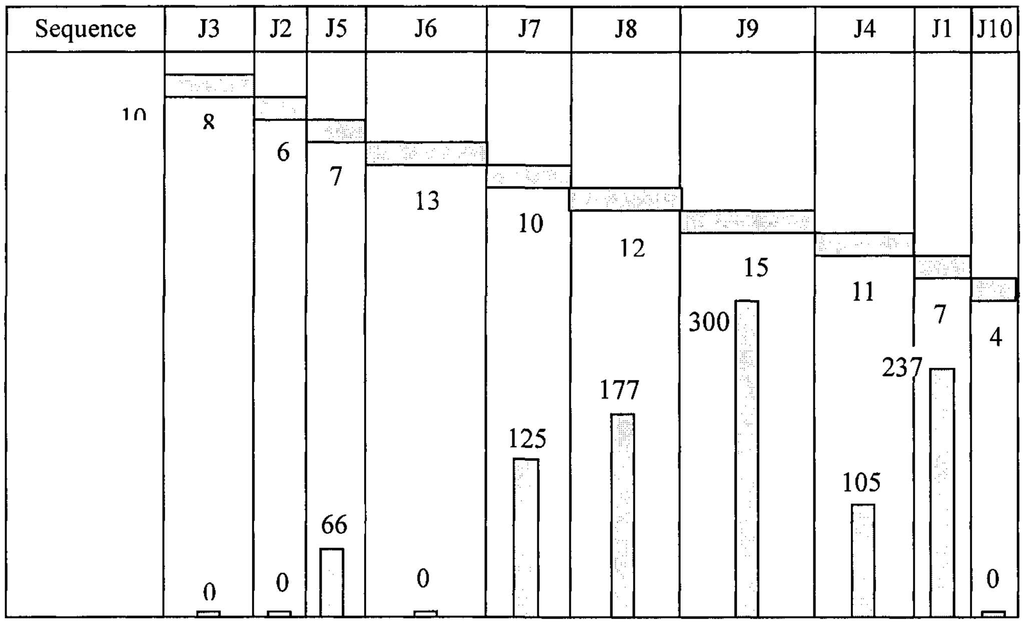

Figure 4.4 Gannt Chart for job dispatch list by tree model without instance selection.

Now, we apply the GA based instance selection approach presented in the previous section (4.3.2). The training data set is divided into six subsets (m=6). There are total forty-five instances in the training data set, and accordingly in each subset, there are seven or eight instances (N_i=7 or 8). We set the number of generations to be thirty (g=30), crossover rate to be 0.6 (c=0.6), mutation rate to be 0.05 (m_u =0.05).

In the first generation, the training data are divided into six subsets. C4.5 decision tree algorithm is applied to each subset, and the relevant fitness ($WL_{\max}$), is calculated. According to $WL_{\max}$, we rank the six subsets and sort them into an ascending sequence. Then we select the top 1, 2, 3, 4 subsets respectively, and record the relevant $WL_{\max}$ and set of instances.

The next step is to perform GA operations: select the top two (which is $\lfloor (1-c)m \rfloor = \lfloor (1-0.6)*6 \rfloor = 2$) subsets and keep them intact into the next generation; two point crossover operation is performed to the remaining two pairs ($\lceil cm/2 \rceil = \lceil 0.6*6/2 \rceil = 2$); randomly select one subset ($\lceil m*m_u \rceil = \lceil 6*0.05/2 \rceil = 1$), and replace one instance though random selection by one instance randomly selected from the whole training data set.

After finishing GA operations, a new generation is created and the above procedures are repeated. In this way, we find an improved decision tree model (shown in Figure 4.5) after thirty generations.

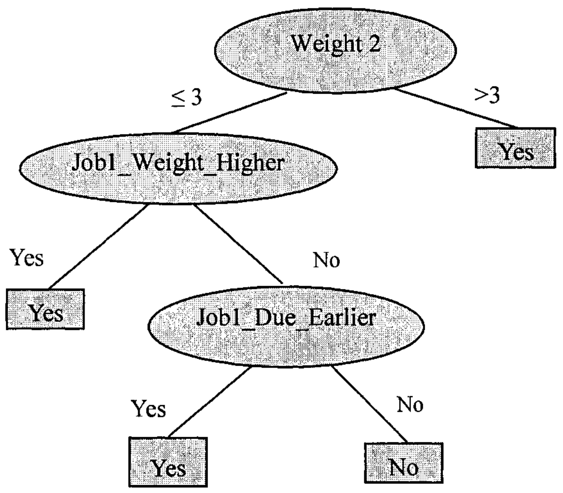

Figure 4.5 Decision tree with instance selection in the numerical example

Compared to the decision tree model in Figure 4.5, there are four leaves nodes. The derived dispatching rule is quite simple:

If Weight 2 ≤ 3 **and** Job2 Weight Higher and Job2 Due Earlier, **then** dispatch Job2 first,

else: dispatch Job1 first.

When apply this new dispatching rule to test data flat file, we get a new dispatch list: J9-J7-J3-J2-J8-J5-J1-J6-J4-J10. The related Gannt chart is shown in Figure 4.6.

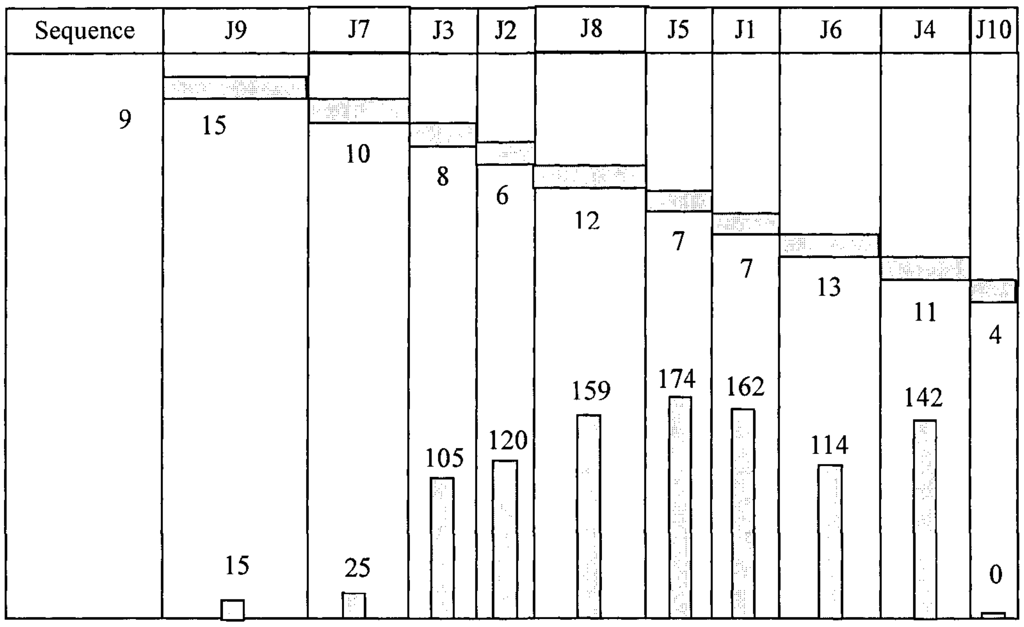

Figure 4.6 Gannt Chart for job dispatch list by tree model with instance selection

Similarly to Figure 4.5, a series of horizontal bars represent job dispatched for processing, while another series of vertical bars represent job's related weighted lateness ($w_j * \max\{0,(C_j - d_j)\}$). Compared to Figure 4.5, there is only one job (J10) is finished before its due date, but the maximum lateness of the whole set of jobs is only $WL_{\max}$ =174, which is much lower than that of previous dispatch list ($WL_{\max}$ =300) found using the tree model without instance selection. With instance selection, the scheduling performance of the tree model has been improved 42.0%. Furthermore, it performs even better than the EDD ($WL_{\max}$ =210), which is the best heuristic dispatching rule for single machine scheduling problems with respect to $WL_{\max}$. The relevant best set of instances is shown in Table 4.6.

Therefore, from this numerical example, we can conclude that with instance selection the scheduling performance of decision tree model could be improved greatly, that is the GA based instance selection methodology can be an effective approach to identify best scheduling practice from historical scheduling data.

Table 4.6 Best instances set selected by GA based instance selection approach

J1	R1	D1	P1	W1	J2	R2	D2	P2	W2	Job1 Released First	Job1 Dues First	Job1 PT Shorter	Job1 Weight Higher	Job1 Scheduled First
1	2	24	7	5	5	3	25	6	3	yes	yes	no	yes	yes
2	0	7	3	3	8	12	15	5	3	yes	yes	yes	no	yes
3	15	10	4	1	4	5	36	18	3	no	yes	yes	no	yes
3	15	10	4	1	7	8	11	12	3	no	yes	yes	no	no
4	5	36	18	3	5	3	25	6	3	no	no	no	no	no
4	5	36	18	3	6	7	20	2	1	yes	no	no	yes	yes
4	5	36	18	3	8	12	15	5	3	yes	no	no	no	no
6	7	20	2	1	10	6	39	17	7	no	yes	yes	no	no
7	8	11	12	3	8	12	15	5	3	yes	yes	no	no	yes
9	9	29	9	5	10	6	39	17	7	no	yes	yes	no	no

4.4 Scheduling Performance Analysis

The example in section 4.3.3 indicated that GA based instance selection methodology is an effective approach to improve decision tree model's scheduling performance. In this section, we evaluate the implementation of GA based instance selection methodology in scheduling more extensively through a series of simulation experiments.

4.4.1 Experimental Setup

This evaluation is based on data created considering different levels of variability and tightness. The data's variability is revealed by coefficient of variation, which provides a relative measure of data dispersion compared to the mean. The coefficient of variation is represented by: $Cv = s/\bar{x}$, where s stands for the standard deviation, and $\bar{x}$ stands for the mean. Different coefficient of variability will be employed to create the data for release time (*Release*), due date *(DueDate)*, processing time (*Processing*) respectively. Tightness is another factor considered when we create data. Its definition is:

$$Ti = \overline{D} - \overline{R} + \overline{P}$$

Where, Ti stands for tightness, $\overline{D}$ stands for the mean of due date, $\overline{R}$ stands for the mean of release time, $\overline{P}$ stands for the mean of processing time.

Table 4.7 Simulation experiments design for scheduling analysis

	Release (Cv & $\overline{R}$)	DueDate (Cv & $\overline{D}$)	Processing (Cv & $\overline{P}$)	Tightness
Set 1	$Cv = 1$	$Cv = 0.5$	$Cv = 2$	Ti
Set 1A	5	8	2	5
Set 1B	10	15	5	10
Set 1C	15	25	5	15
Set 2	$Cv = 2$	$Cv = 0.5$	$Cv = 1$	
Set 2A	5	8	2	5
Set 2B	10	15	5	10
Set 2C	15	25	5	15
Set 3	$Cv = 2$	$Cv = 1$	$Cv = 0.5$	
Set 3A	5	8	2	5
Set 3B	10	15	5	10
Set 3C	15	25	5	15
Set 4	$Cv = 0.5$	$Cv = 1$	$Cv = 2$	
Set 4A	5	8	2	5
Set 4B	10	15	5	10
Set 4C	15	25	5	15
Set 5	$Cv = 0.5$	$Cv = 2$	$Cv = 1$	
Set 5A	5	8	2	5
Set 5B	10	15	5	10
Set 5C	15	25	5	15
Set 6	$Cv = 1$	$Cv = 2$	$Cv = 0.5$	
Set 6A	5	8	2	5
Set 6B	10	15	5	10
Set 6C	15	25	5	15

There are three distributions are employed to create release time, due date and processing time: Erlang distribution with $Cv = 0.5$, Exponential distribution with $Cv = 1$, and Hyper exponential distribution with $Cv = 2$. We design six sets of simulation experiments with different combinations of these three distributions assigned to the three attributes: release time (*Release*), due date *(DueDate*), processing time (*Processing*) respectively. We use the same approach (in Section 3.4.1) to generate simulation data for each attribute based on the determined distribution. Furthermore, in each set, there are three different data with respect to different tightness: $Ti = 5$, $Ti = 10$, and $Ti = 15$. Therefore, there are total of eighteen simulation data sets, and the related simulation experimental design is shown below in Table 4.7.

4.4.2 Tree Models Performance Analysis

Table 4.8 shows the comparisons between decision trees without and with instance selection for each simulation experiment. From this table, we can see that the size of decision trees with instance selection decreases greatly: from almost 300 leaves down to only 7 or 14 leaves. This indicates that, with instance selection, decision tree models are capable to discover the most important information and knowledge of the scheduling data, and they are much easier to be interpreted. There is no case that zero is within the 95% confidence interval which indicate the size of decision tree model after instance selection is significantly different from the size of the original tree models. Therefore, we can conclude that after instance selection, the size of induced tree is reduced greatly.

Apart from size evaluation, the scheduling performance of the decision trees with instance selection is our ultimate concern. Table 4.9 shows the scheduling performance ($WL_{\max}$) comparison between decision tree models without and with instance selection for each simulation experiment.

Table 4.8 Size of decision trees without and with instance selection

Release	Due Date	Process	Tightness	*WLmax*		Difference of *WLmax* ($\Delta WL_{\max}$)		
$Cv = 0.5$	$Cv = 1$	$Cv = 2$		Original	Instance Selection	Average of Difference	Standard deviation of Difference	Confidence Interval (95%) of Difference
5	8	2	5	237	14	-223	4	(-227, -219)
10	15	5	10	285	11	-274	2	(-276, -272)
25	15	5	15	250	10	-240	2	(-242, -238)
$Cv = 0.5$	$Cv = 2$	$Cv = 1$						
5	8	2	5	251	11	-240	4	(-244, -236)
10	15	5	10	265	7	-258	1	(-259, -257)
15	25	5	15	295	10	-285	1	(-286, -284)
$Cv = 1$	$Cv = 0.5$	$Cv = 2$						
5	8	2	5	329	7	-322	1	(-323, -321)
10	15	5	10	308	10	-299	2	(-301, -296)
15	25	5	15	305	8	-297	2	(-299, -296)
$Cv = 1$	$Cv = 2$	$Cv = 0.5$						
5	8	2	5	333	7	-327	1	(-327, -326)
10	15	5	10	305	8	-297	0	(-297, -297)
15	25	5	15	351	8	-343	1	(-344, -342)
$Cv = 2$	$Cv = 0.5$	$Cv = 1$						
5	8	2	5	301	10	-291	3	(-294, -288)
10	15	5	10	313	12	-301	8	(-309, -294)
15	25	5	15	293	9	-285	1	(-285, -284)
$Cv = 2$	$Cv = 1$	$Cv = 0.5$						
5	8	2	5	319	8	-311	2	(-312, -309)
10	15	5	10	301	9	-292	1	(-293, -291)
15	25	5	15	316	9	-308	2	(-309, -306)

Four each data set, we replicate four time with GA based instance selection method and record the $WL_{\max}$ values. Then calculate the difference between $WL_{\max}$ of original decision tree model without instance selection and the $WL_{\max}$ value of the new decision tree model after performing instance selection. The last three columns give out the average, standard deviation, and 95% confidence levels of the difference in $WL_{\max}$ based on four replications respectively. From the confidence levels results, we can see that there is a significant difference between $WL_{\max}$ values of decision tree models with and without instance selection. Because both upper and lower limits are negative, which indicates that with instance selection, $WL_{\max}$ could be improve significantly though instance selection.

Table 4.9 Scheduling performance comparison of decision trees without and with instance selection

Release	Due Date	Process	Tightness	*WLmax*		Difference of *WLmax* ($\Delta WL_{\max}$)		
				Original	Instance Selection	Average of Difference	Standard deviation of Difference	Confidence Interval (95%) of difference
$Cv = 0.5$	$Cv = 1$	$Cv = 2$						
5	8	2	5	1790	1094	-710	11	(-714, -598)
10	15	5	10	3589	2215	-1360	14	(-1374, -1347)
25	15	5	15	4788	2555	-2204	26	(-2260, -2237)
$Cv = 0.5$	$Cv = 2$	$Cv = 1$						
5	8	2	5	2280	1017	-1259	3	(-1283, -1189)
10	15	5	10	5103	2528	-2571	13	(-2592, -2530)
15	25	5	15	4725	2494	-2211	16	(-2228, -2178)
$Cv = 1$	$Cv = 0.5$	$Cv = 2$						
5	8	2	5	2189	1226	-956	7	(-1172, -1125)
10	15	5	10	5548	3250	-2280	13	(-3197, -2934)
15	25	5	15	5706	2912	-3503	45	(-1732, -1582)
$Cv = 1$	$Cv = 2$	$Cv = 0.5$						
5	8	2	5	2245	1108	-1122	11	(-931, -813)
10	15	5	10	6597	3356	-3226	14	(-2631, -2164)
15	25	5	15	4887	3138	-1731	19	(-2882, -2569)
$Cv = 2$	$Cv = 0.5$	$Cv = 1$						
5	8	2	5	2520	1263	-1241	14	(-1316, -1239)
10	15	5	10	6189	3097	-3057	26	(-3318, -2953)
15	25	5	15	5607	3389	-2204	15	(-2924, -2312)
$Cv = 2$	$Cv = 1$	$Cv = 0.5$						
5	8	2	5	2768	1410	-1347	9	(-1391, -1221)
10	15	5	10	6250	3284	-2939	22	(-3074, -2949)
15	25	5	15	6453	3458	-2981	14	(-2830, -2122)

Apart from analyzing the difference between induced models with and without instance selection, we also compared the scheduling performance of the induced model after instance selection and the EDD which is employed to create the original training data sets. Table 4.10 shows the numerical results about the scheduling performance comparison between induced decision tree model and EDD.

Table 4.10 Scheduling performance comparison between EDD and tree model with instance selection

Release	Due Date	Process	Tightness	*WLmax*		Difference of *WLmax* ($\Delta WL_{\max}$)		
$Cv = 0.5$	$Cv = 1$	$Cv = 2$		EDD	Instance Selection	Average of Difference	Standard deviation of Difference	Confidence Interval (95%) of difference
5	8	2	5	1890	1094	-756	59	(-814, -698)
10	15	5	10	3553	2215	-1324	14	(-1338, -1311)
25	15	5	15	4583	2555	-2044	12	(-2055, -2032)
$Cv = 0.5$	$Cv = 2$	$Cv = 1$						
5	8	2	5	2035	1017	-991	48	(-1038, -944)
10	15	5	10	4529	2528	-1987	32	(-2018, -1956)
15	25	5	15	3857	2494	-1335	26	(-1360, -1310)
$Cv = 1$	$Cv = 0.5$	$Cv = 2$						
5	8	2	5	1688	1226	-592	24	(-615, -568)
10	15	5	10	4764	3250	3	134	(-1364, -1101)
15	25	5	15	4965	2912	-1735	77	(-1810, -1660)
$Cv = 1$	$Cv = 2$	$Cv = 0.5$						
5	8	2	5	1672	1108	-355	60	(-414, -296)
10	15	5	10	4814	3356	-1664	238	(-1897, -1430)
15	25	5	15	4381	3138	-1401	160	(-1557, -1244)
$Cv = 2$	$Cv = 0.5$	$Cv = 1$						
5	8	2	5	2212	1263	-970	39	(-1008, -931)
10	15	5	10	5683	3097	-2629	186	(-2812, -2447)
15	25	5	15	5152	3389	-2163	312	(-2469, -1857)
$Cv = 2$	$Cv = 1$	$Cv = 0.5$						
5	8	2	5	2369	1410	-907	87	(-992, -822)
10	15	5	10	4957	3284	-1719	64	(-1781, -1656)
15	25	5	15	5515	3458	-1538	361	(-1892, -1184)

From 95% confidence intervals in Table 4.10, we can see that there is a significant difference between the tree model after instance selection and EDD. Similarly to the analysis result about Table 4.9, the induced tree models could perform significantly better than EDD in terms of $WL_{\max}$, due to all negative upper and lower limits.

From the above numerical results, the $WL_{\max}$ by decision trees with instance selection is much lower than that of the decision trees without instance selection (Original), the improvements are from 43% to 50%. Moreover, decision trees with instance selection perform even better than EDD dispatching rule: the improvements are from 30% to 42%. That fact that much higher scheduling performance of decision trees with instance selection than the other two holds for each simulation experiment, thus, we can conclude that this GA based instance selection is an effective methodology to improve scheduling performance when applied in scheduling context.

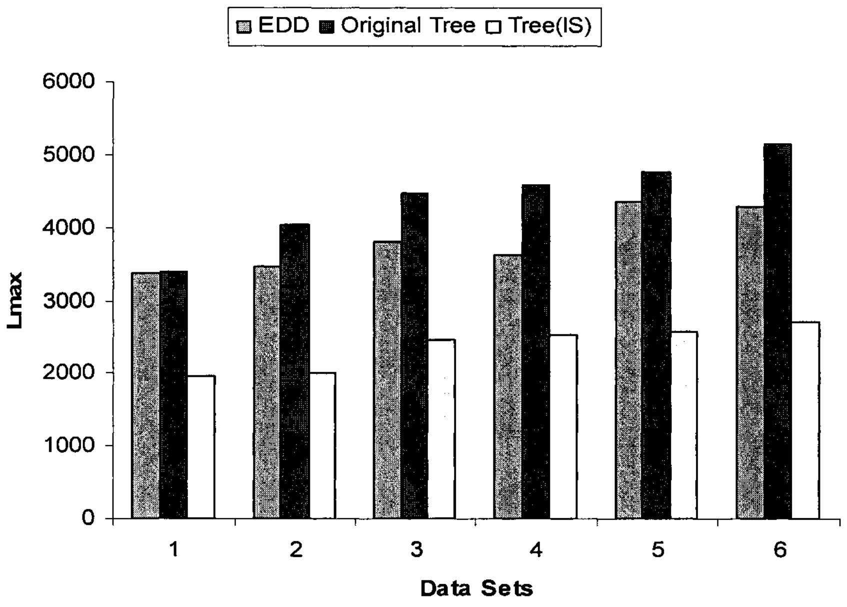

Figure 4.7 Average $WL_{\max}$ comparisons between EDD, original tree and the tree with instance selection

4.4.3 Minimum Splitting Size's impact analysis

In Section 4.4.2, we analyze the scheduling performance of trees after instance selection by comparing models with and without instance selection, and comparing the models after instance selection and EDD. To avoid confounding, we keep the parameters for C4.5 as unchanged for all experiments. But, it is necessary to take into account the effects by adjusting parameters when we apply decision tree algorithm to training data. Minimum splitting size is one of most important parameters for applying C4.5 algorithm, which indicate the minimum number of instances allowed in each leaf of the decision tree. Usually, the greater the minimum splitting size, the smaller induces decision trees. From the above numerical experiments results, it is shown that instance selection can reduce the tree size greatly, thus we perform more experiments to compare the scheduling performances of similar size of trees by changing minimum splitting size parameter and by instance selection.

We use the same eighteen data sets as used before. For each data set, we first generate decision tree with same and similar size of tree with instance selection by resetting minimum splitting size (minNumObj in WEKA). Then new trees will be applied to the relevant test data to calculate $WL_{\max}$. In the simulation experiments in Section 4.4.2, for any data set of the total eighteen data sets, we perform four replications to account for the randomness from GA based instance selection. Therefore, there would be at most four experiments need to be performed for each data set when the size of tree models are all different after instance selection. Table 4.11 shows the experiments results.

In Table 4.11, both average values of $WL_{\max}$ by trees with resetting minimum splitting size and with instance selection are listed, but we need to mention that for each data set, there are four replications, and thus, we can calculate the standard deviation and 95% confidence intervals for the difference between $WL_{\max}$ by each pair of trees. From Table 4.11, we can see that there is no case with zero located within the 95% confidence intervals, which indicate that $WL_{\max}$ values are significantly different between trees by changing minimum splitting size and by instance selection. In addition, all the upper and lower limits are negative, which shows that the $WL_{\max}$ value by trees with instance selection are significantly smaller than that of trees by adjusting minimum splitting size. In conclusion, by adjusting the minimum splitting size of trees cannot achieve comparable improvements on scheduling performance as by instance selection, especially when the scheduling performance measure is weighted maximum lateness $WL_{\max}$.

Table 4.11 Scheduling performance comparison between same size of trees with adjusting minimum splitting size and with instance selection

Release	Due Date	Process	Tightness	*WLmax*		Difference of *WLmax* ($\Delta WL_{\max}$)		
				Average by adjusting minimum splitting size	Average by Instance Selection	Average of Difference	Standard deviation of Difference	Confidence Interval (95%) of difference
$Cv = 0.5$	$Cv = 1$	$Cv = 2$						
5	8	2	5	1946	1094	-756	59	(-814, -698)
10	15	5	10	4773	2215	-2805	14	(-2819, -2792)
25	15	5	15	5296	2555	-703	12	(-714, -691)
$Cv = 0.5$	$Cv = 2$	$Cv = 1$						
5	8	2	5	2377	1017	-1493	48	(-1540, -1446)
10	15	5	10	5379	2528	-4151	32	(-4182, -4120)
15	25	5	15	5309	2494	-2756	26	(-2781, -2731)
$Cv = 1$	$Cv = 0.5$	$Cv = 2$						
5	8	2	5	5296	1226	-1498	24	(-1521, -1474)
10	15	5	10	7384	3250	-5701	134	(-5832, -5569)
15	25	5	15	4794	2912	-1691	77	(-1766, -1616)
$Cv = 1$	$Cv = 2$	$Cv = 0.5$						
5	8	2	5	2278	1108	-961	60	(-1020, -902)
10	15	5	10	6058	3356	-3322	238	(-3555, -3088)
15	25	5	15	5493	3138	-2243	160	(-2399, -2086)
$Cv = 2$	$Cv = 0.5$	$Cv = 1$						
5	8	2	5	2744	1263	-1049	39	(-1087, -1010)
10	15	5	10	6587	3097	-3418	186	(-3601, -3236)
15	25	5	15	5971	3389	-3558	312	(-3864, -3252)
$Cv = 2$	$Cv = 1$	$Cv = 0.5$						
5	8	2	5	2784	1410	-1327	87	(-1412, -1242)
10	15	5	10	5750	3284	-2360	64	(-2422, -2297)
15	25	5	15	6550	3458	-2442	361	(-2796, -2088)

4.5 Summary and Discussion

In this chapter, we investigate in the problem whether we could improve current scheduling performance through applying data mining in scheduling directly. We presented an optimization based instance selection approach to select good instances from production data to identify the best scheduling practice. As a result, the induced decision tree models can work well as new dispatching rules with improved scheduling performance.

We first investigated whether different tree models, induced from different subsets of scheduling data, perform differently as a new dispatching rule. Then we proposed a optimization based instance selection methodology for scheduling, by combining data mining with optimization for effective production scheduling. In this approach, we use a genetic algorithm to find a heuristic solution to the optimal instances selection problem, and then induce a decision tree from this subset of best instances. A simple numerical example is employed to illustrate how to perform GA based instance selection approach in scheduling. Finally, through extensive numerical results, we conclude that this GA based instance selection methodology is an effective approach to identify the best scheduling practice and induced decision tree's scheduling performance can be improved effectively compared to the models without instance selection. Moreover, it is illustrated by numerical results that the comparable improvements by instance selection cannot be achieved by adjusting minimum splitting size, one of the most important parameters for decision tree algorithm.

5 BEST INSTANCES ANALYSIS

5.1 Introduction

In Chapter 4, the GA based instance selection methodology is illustrated and its application in scheduling is analyzed. Through a series of simulation experiments, the numerical results show that this approach is an effective methodology to identify the best data from large database, leading to construct robust decision tree models with improved scheduling performances. Meanwhile, an interesting question comes up: why and how these instances selected by the algorithm are "good" and represent "good scheduling practice".

In this chapter, we propose a new approach to analyze how the best data are selected through instance selection procedure. The basic idea of this approach is to apply data mining directly to learn about best data identification. This is a classification problem: a new target concept about how data are selected is defined and accordingly, classification algorithms are employed to learn to determine whether the data instances should be selected or not.

The remainder of this chapter is as the following. First, we briefly reviewed some related issues and the basic idea of this analytical approach; then we use a simple example to illustrate this new approach: how to apply data mining to learn how best instances are selected. More extensive experiments are conducted using same data sets involved in Chapter 4 and the numerical results are discussed further. After applying data mining algorithm, the performances of the models are analyzed extensively. In addition, attributes selection is also employed to identify those factors that are most important for instance selection decision process.

5.2 Analytical Approach

It is already shown that GA based instance selection methodology can identify the best instances from the original data set efficiently. Now more research needs to be done to find out why or how these best data can be characterized. Our strategy is to reprocess the data and apply data mining directly to learn how those instances are selected. We defined a new target concept to be learned as to determine whether to select a certain instance or not. Motivated by this strategy, a new class attribute is introduced to the original training data set, called "*Selected*", which reveals whether a certain instance is selected by GA based Instance Selection method or not. If the record is selected, then the value for this class attribute is "yes", otherwise it is "no". After this new class attribute is created, we can apply any classification algorithm to the data directly. Same as before, due to its good transparency, decision tree algorithm is employed to learn how best instances are identified.

From the extensive numerical experiments in the Chapter 4, we noticed that the best data selected by GA based Instance Selection method only accounts for around 1% of the original training data sets, while in the small numerical example, the selected instances account for a little more than 20%. The intuitive reason for this difference appears to be that, in the small example, the total number of instances in the original training data is very small, only 45 records. The small number of selected instances will cause severe imbalance in the class attribute *Selected.* Traditional machine learning algorithms may be biased towards the majority class (*Selected*="no") and thus, may predict the minority class examples (*Selected*="yes") poorly. Therefore, oversampling is necessary to be performed to deal with this problem.

Random oversampling resamples at random the minority class examples until their number is equal to the number of instances in the majority class. The supervised resample filter in Weka is employed here to perform this oversampling procedure. There is one parameter called biasToUniformClass, which represents the setting value for the bias towards a uniform class. A value of 0 leaves the class distribution as-is, a value of 1 ensures the class distribution is uniform in the output data. Another parameter is the sampleSizePercentage, which represents the percentage rate of sample size versus the original data set. Through setting values for these two parameters, we can choose the bias degree to uniform class and sample size.

After data reprocessing, oversampling, and applying decision tree algorithm, the predicting capability and performance of this tree model are analyzed. The predicting capability is that the capability to distinguish selected instances from those instances that are selected by Instance Selection method. Due to oversampling, many examples falling in minority class *(Selected*=yes) are sampled more than once. As a result, the accuracy is biased measure to analyze the performance of the tree model. Instead, Precision, Recall and Confusion Matrix are more suitable and meaningful measures for analysis in these situations.

Using ReliefF as an attribute evaluator and Ranker method provided in Weka, we generated the rank list of attributes and analyze which attributes are more important related to the decision process about whether a certain instance should be selected or not. The results are reported in the next sections.

5.3 Numeric Example

In Section 4.3.3, a numeric example with a small data set is employed to illustrate the application of the GA based instance selection methodology. There are forty-five instances totally in the original data set, shown in Table 4.3. After applying instance selection procedure, only ten instances are selected as shown in Table 5.1 (same as Table 4.6).

The training data set is reprocessed by adding the *Distance* attribute and new class attribute *Selected. Distance* attribute is numeric, which gives out the distance information between the two jobs involved in each instance in terms of their positions in the original scheduling sequence by EDD. The addition of this numeric attribute is motivated by the assumption that distances between jobs have some impact on instance selection. We will check this assumption through numerical experiments analysis. The new complete attributes list is shown below in Table 5.2.

Table 5.1 Best instances selected by GA based Instance Selection approach

Job1	R1	D1	P1	W1	Job2	R2	D2	P2	W2	Job1 Released First	Job1 Due First	Job1 PT Shorter	Job1 Weight Higher	Job1 Scheduled First
1	2	24	7	5	5	3	25	6	3	yes	yes	no	yes	yes
2	0	7	3	3	8	12	15	5	3	yes	yes	yes	no	yes
3	15	10	4	1	4	5	36	18	3	no	yes	yes	no	yes
3	15	10	4	1	7	8	11	12	3	no	yes	yes	no	no
4	5	36	18	3	5	3	25	6	3	no	no	no	no	no
4	5	36	18	3	6	7	20	2	1	yes	no	no	yes	yes
4	5	36	18	3	8	12	15	5	3	yes	no	no	no	no
6	7	20	2	1	10	6	39	17	7	no	yes	yes	no	no
7	8	11	12	3	8	12	15	5	3	yes	yes	no	no	yes
9	9	29	9	5	10	6	39	17	7	no	yes	yes	no	no

Table 5.2 Complete attributes list in the reprocessed data set

Attributes Name	Type	Definition
J1	Numeric	First Job's ID
ReleaseTime1	Numeric	First Job's release time
DueDate1	Numeric	First Job's due date
ProcessingTime1	Numeric	First Job's processing time
Weight1	Numeric	First Job's weight
J2	Numeric	Second Job's ID
ReleaseTime2	Numeric	Second Job's release time
DueDate2	Numeric	Second Job's due date
ProcessingTime2	Numeric	Second Job's processing time
Weight2	Numeric	Second Job's weight
J1ReleasedFirst	Categorical {yes, no}	If the first job is released earlier than the second job=> yes, otherwise =>no
J1DuesFirst	Categorical {yes, no}	If the first job dues earlier than the second job=> yes, otherwise =>no
J1ProcessShorter	Categorical {yes, no}	If the first job's processing time is shorter than the second job=> yes, otherwise =>no
J1WeightHigher	Categorical {yes, no}	If the first job's weight is higher than the second job=> yes, otherwise =>no
J1ScheduledFIrst	Categorical {yes, no}	If the first job is scheduled earlier than the second job=> yes, otherwise =>no
Distance	Categorical {yes, no}	The distance between these two jobs according to their EDD scheduling sequence
Selected (Class)	Categorical {yes, no}	If the first job is released earlier than the second job=> yes, otherwise =>no

The new reprocessed training data set is shown below in Table 5.3. There are ten instances with "yes" in *Selected* class attribute, namely the instances with comparisons among the following job pairs: J1-J5, J2-J8, J3-J4, J3-J7, J4-J5, J4-J6, J4-J8, J6-J10, J7-J8, and J9-J10. Other instances are with "no" in *Selected* class attribute. Both second column and forth column are abbreviations of basic attributes about the two jobs: release time, due dates, processing time, and weights.

Table 5.3 Reprocessed training data in small numeric example

J1	Rl-W1	J2	R2-W2	J1 Released First	J1 Dues First	J1 Process Shorter	J1 Weight Higher	J1 Scheduled First	Distance	**Selected**
1	......	2	......	no	no	no	yes	no	1	no
1	......	3	......	yes	no	no	yes	yes	6	no
1	......	4	......	yes	yes	yes	yes	yes	7	no
1	......	5	......	yes	yes	no	yes	yes	5	yes
1	......	6	......	yes	no	no	yes	yes	8	no
.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.
9		10	......	no	yes	yes	no	no	1	yes

The resample filter provided in Weka is employed to perform the oversampling procedure, which resamples at random the minority class (*Selected* ="yes") examples until the number is equal to the number of instances in the majority class (*Selected* ="no"). We choose to resample as the same number of instances as in the original data set, namely 45. Before applying decision tree algorithm, we hold one third of these data is out for testing. The tree model after learning is shown below in Figure 5.1.

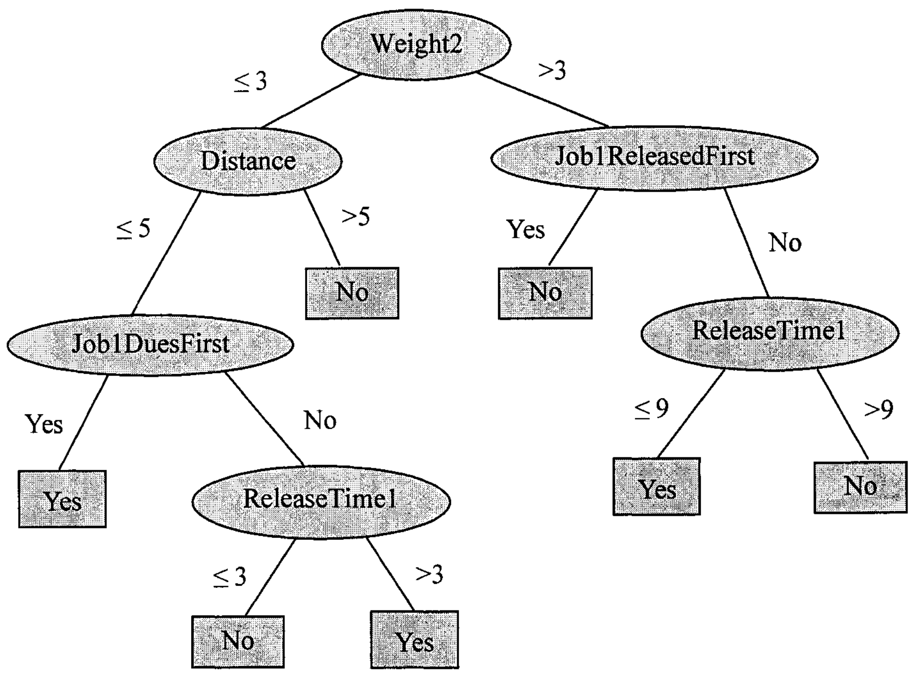

Figure 5.1 Decision tree to learn class attribute *Selected* in the numerical example

From the above tree model, we can generate the following rules:

If *Weight2*≤3 **and** *Distance*≤5 **and** *Job1DuesFirst*, **Then** the instance **is selected;**

If *Weight2*≤3 **and** *Distance*≤5 **and** *Job2DuesFirst* **and** ReleaseTime1>3, **Then** the instance **is selected;**

If *Weight* >3 **and** *Job2ReleasedFirst* **and** *ReleaseTime1*≤9, **Then** the instance **is selected;**

These rules give out the pattern how the instances are selected by the GA based Instance Selection method. The importance of *Weight*, *Distance* and *DueDate* or relevant comparison attribute is revealed by the tree model. This is consistent with the scheduling practice of EDD dispatching rule, in which weights and due dates are the most important attributes for making scheduling decisions. We will examine the importance of attributes through a different procedure (attribute selection methods using ReliefF attribute evaluator) later in this section.

Now, the predicting capability and performance of this tree model needs to be addressed. The predicting capability is the capability to distinguish the instances selected by the instance selection method from other normal instances. Due to oversampling, many examples falling minority class *(Selected* ="yes") are sampled more than once. As a result, the accuracy is a biased measure to analyze the performance of the tree model. Therefore, Precision, Recall and Confusion Matrix are more meaningful and suitable measures for analysis instead.

Table 5.4 Detailed accuracy by class in small numeric example

TP Rate	FP Rate	Precision	Recall	F-Measure	Class (Selected)
0.875	0.125	0.875	0.875	0.875	yes
0.875	0.125	0.875	0.875	0.875	no

Table 5.5 Confusion matrix in small numeric example

a	b	Class (Selected)
7	1	a = yes
1	7	b = no

From the above tables about the detailed accuracy and confusion matrix, we can see that the precision and recall for both classes are the same, which indicate that, through oversampling, the tree model can predict minority class (*Selected*="yes") as well as majority class (*Selected*="no"). Thus, we can conclude that this tree model is able to distinguish minority class (*Selected*="yes") well.

Apart from applying decision tree algorithm to the data to learn how the instances are selected by GA based instance selection method; we also use ReliefF evaluator and Ranker method to analyze the attributes' relationship to the target concept. Similarly, oversampling is also a preceding process for attributes selection and analysis. In order to get more general results, ten-fold cross validation is employed to do attribute selection and the results are shown below in Table 5.6. We can see that *Job1ReleasedFirst*, *Weight2*, *ReleaseTime1*, and *Distance* are in the top five of the ranking list of all the attributes. Similar to the tree model after learning from the data, these four attributes are also the main attributes in the splitting nodes.

Table 5.6 Attributes ranks by ReliefF evaluator in small numeric example

Rank	Attribute
1	Job1ReleasedFirst
2	Weight2
3	ReleaseTime1
4	Weight1
5	Distance
6	ReleaseTime2
7	ProcessingTime2
8	DueDate2
9	DueDate1
10	Job1WeightHigher
11	ProcessingTime1
12	Job1DuesFirst
13	Job1ProcessTimeShorter
14	Job1ScheduleFirst

Although *Job1DuesFirst*, which appears in the tree model, is not with higher rank in the above list, reveals the comparison between the two jobs in each instance. In fact, *Duedate1* and *Duedate2* are ranked higher in the list, thus, the consistency between the attribute selection results and tree model still holds. In conclusion, jobs' weights, release time, distance, and due dates are most highly related to how to select instances by GA based instance selection method. Furthermore, we can conclude that it is an effective strategy to define the new target concept to determine whether the instance should be selected or not, to apply data mining directly to learn this concept, and then construct a robust predictive model which also with good descriptive capability.

5.4 Numerical Experiment Results

The example in section 5.3 illustrates the effective approach by defining new target concept, reprocessing data, applying data mining algorithm and analyzing results, to learn how GA based instance selection method select the best instances. In this section, we evaluate this approach more extensively using the data from the simulation experiments in Chapter 4.

In each previous simulation experiment, one training data set and one test data set both with 200 jobs are created according to a certain set of parameter values of coefficient (Cv), tightness(Ti), and means of release time, due date and processing time respectively. As a result, there are totally 18 experiments. In both training data and test data creation, the 200 jobs are dispatched according to EDD dispatching rule, and the scheduling performance measure considered is also Weighted Maximum Lateness ($WL_{\max}$).

After performed GA based instance selection procedure, best instances are selected in each simulation experiment. We keep the record of these best instances selected and reprocess the training data set by adding new class attribute (Selected). For any instance in the training data set, if it is selected by the Instance Selection procedure, then the value for the class attribute Selected is "yes", otherwise is "no". Furthermore, the new attribute *Distance* is added in to the training data set, which represents the distance of two jobs in any instance according to their EDD scheduling sequence. This data reprocessing procedure is the same as illustrated in the numeric example explained in the previous section.

The next step is to resample the training data to balance the classes. We first make the class distribution falls into exact uniform distribution, and keep the resample size as 50% of the original data set. Then decision tree algorithm is applied to the reprocessed data. This same experiment procedure is performed to all the 18 data sets, and the results of learning are shown in Table 5.8 below. The second column "Size (IS)" represents the size of best instances selected by GA based instance selection method. Tree size is measured by the total number of leaves in the decision tree model like before in Chapter 3 and Chapter 4.

To avoid confounding of the experiment, when only apply decision tree C4.5 algorithm to a sample of the original data, we need to reset the minimum split size to be the same percentage of the original value. For example, if the original value for minimum split size is 2, then the new value would be 1 if the resampling rate is 50%.

Table 5.7 Size and detailed accuracy of decision tree models

Data Set	Size (Instance Selection)	Tree Size	Accuracy	TP Rate	FP Rate	Precision	Recall	F-Measure	Class (Selected)
1	199	304	95.9%	1	0.083	0.924	1	0.96	yes
				0.917	0	1	0.917	0.957	no
2	98	177	97.7%	1	0.046	0.956	1	0.978	yes
				0.954	0	1	0.954	0.976	no
3	158	257	96.5%	1	0.071	0.934	1	0.966	yes
				0.929	0	1	0.929	0.963	no
4	79	141	98.2%	1	0.037	0.965	1	0.982	yes
				0.963	0	1	0.963	0.981	no
5	79	156	98.2%	1	0.036	0.965	1	0.982	yes
				0.964	0	1	0.964	0.982	no
6	79	145	98.3%	1	0.034	0.967	1	0.983	yes
				0.966	0	1	0.966	0.982	no
7	78	156	97.7%	1	0.045	0.957	1	0.978	yes
				0.955	0	1	0.955	0.977	no
8	157	249	96.8%	1	0.064	0.94	1	0.969	yes
				0.936	0	1	0.936	0.967	no
9	79	135	98.2%	1	0.037	0.964	1	0.982	yes
				0.963	0	1	0.963	0.981	no
10	79	147	98.3%	1	0.033	0.968	1	0.984	yes
				0.967	0	1	0.967	0.983	no
11	79	142	98.5%	1	0.031	0.9745	1	0.985	yes
				0.969	0	1	0.969	0.984	no
12	78	144	98.2%	1	0.037	0.964	1	0.982	yes
				0.963	0	1	0.963	0.981	no
13	79	151	98.3%	1	0.034	0.967	1	0.983	yes
				0.966	0	1	0.966	0.983	no
14	79	153	98.1%	1	0.038	0.963	1	0.981	yes
				0.962	0	1	0.962	0.98	no
15	79	141	98.2%	1	0.037	0.964	1	0.982	yes
				0.963	0	1	0.953	0.981	no
16	79	150	98.4%	1	0.032	0.969	1	0.984	yes
				0.968	0	1	0.968	0.984	no
17	79	151	98.2%	1	0.035	0.966	1	0.983	yes
				0.965	0	1	0.965	0.982	no
18	79	146	98.2%	1	0.037	0.964	1	0.982	yes
				0.963	0	1	0.963	0.981	no

From Table 5.7, we can see that in most experiments, 79 instances are selected from the original 19900 instances, only one case with 98 instances, two cases with 157 instances, and another case with 199 instances. The relationship between the size of the decision tree model and the size of the best instances set selected by the GA based instance selection method is shown below in Figure 5.2. For those data sets with 78 or 79 best instances selected, the size of the tree model is between 129 and 151. But as the size best instances set increases from 79, to 98, 157, 158 and 199, the size of the tree model also increases proportionally. This result is quite intuitive and reasonable, since when the best data set is larger, more information needs to be learned by the decision tree algorithm. As a result, the tree model after learning will be more complex comparable to situations when the best data set is small.

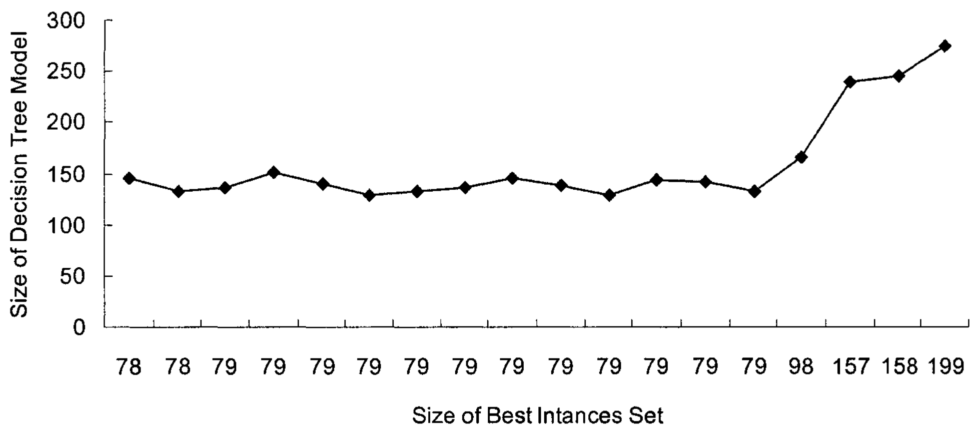

Figure 5.2 Size of the decision tree model as the function of the size of best instances set

Figure 5.3 shows the relationship between accuracy of the decision tree model and the size of best data set. Generally, the accuracy of the decision tree models is not low: the average accuracy is 97.0% and the standard deviation is 1%. But we can see there is clear trend that as the best data set becomes larger, the accuracy of the tree models decreases. For the cases where best data size is about 79, the accuracy of the decision tree is pretty high, from 96.7% to 98.0%. But when the size best data increases, the accuracy of the tree model decreases: from 96.4% when the size of best data set is 98, to 93.6%

when the size of the best data set is 199.

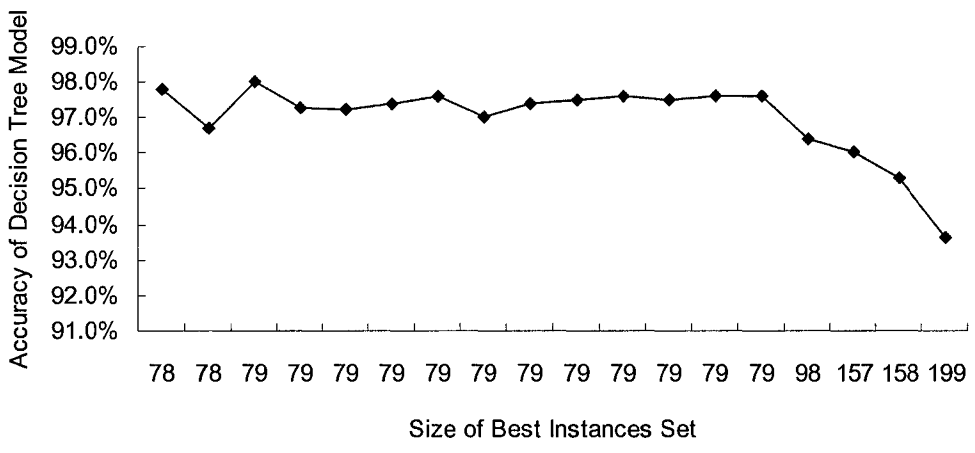

Figure 5.3 Accuracy of the decision tree model as the function of the size of best instances set

Apart from the accuracy analysis, the relationship between the F-Measure of decision tree models and the size of best data set is shown in Figure 5.4. The trends for both classes are similar: as the size of best data set increases, the F-Measure decreases for both classes, which are also very similar to the accuracy trends in Figure 5.3. The reason for this is that the resample size is kept same (50% of the original data set, namely 9950 records), regardless to the change of the best data size. Those instances with minority class (Selected="yes") will be randomly oversampled a little over 100 times when the size of best data set is around 79. However, the instances with minority class (Selected="yes") will be randomly oversampled about 50 times when the size of best data set is around 199. As a result, the decision tree algorithm will be more biased to predict minority class more accurately when the minority class is resmapled with higher rate, at the expense of predicting majority class less accurately, since the final accuracy of the model will be lower in this way. This can be testified by FP rate (false positive rate) information in the fifth column of Table 5.7. Best data set with smaller size (79) is with lower FP rate, which means smaller number of majority class incorrectly classified into minority class, comparable to the cases when best data set is with larger size (199).

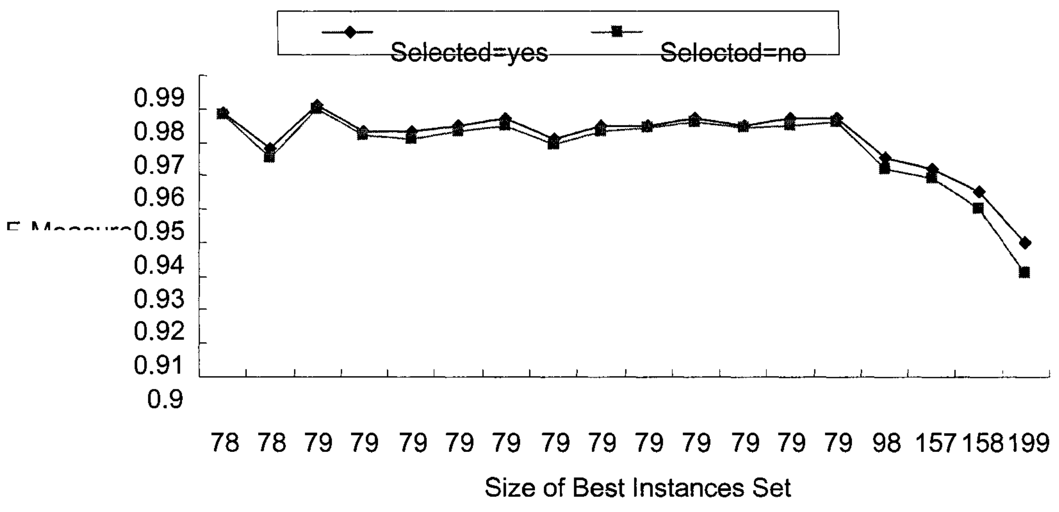

Figure 5.4 F-Measure of decision tree model as the function of the size of best instances set

In the previous section, the experiments are performed when we resample data at 50% of the original data. In this section, we change the resample size and analyze the impact of resample rate on the decision tree model performance. The numerical results are shown below in Table 5.8. Similarly to the previous experiments, we also reset the minimum split size with same percentage as resampling rate to avoid confounding of experiments.

From the Table 5.8, it is clear that both the size and accuracy of the decision trees increases as the resample rate increases. This can be illustrated in a more clear way in Figure 5.5, Figure 5.6 and Figure 5.7.

Table 5.8 Detailed accuracy of tree models at different resample rates.

Percentage (%)	Tree Size	Accuracy	TP Rate	FP Rate	Precision	Recall	F-Measure	Class (Selected)
10	148	83.1%	0.921	0.253	0.773	0.921	0.841	yes
			0.747	0.079	0.91	0.747	0.82	no
20	231	90.7%	0.985	0.17	0.849	0.985	0.912	yes
			0.83	0.015	0.983	0.83	0.9	no
30	267	93.2%	0.999	0.135	0.881	0.999	0.936	yes
			0.865	0.001	0.998	0.865	0.927	no
40	276	95.4%	1	0.092	0.916	1	0.956	yes
			0.908	0	1	0.908	0.952	no
50	306	95.9%	1	0.083	0.924	1	0.96	yes
			0.917	0	1	0.917	0.957	no
60	327	96.5%	1	0.069	0.935	1	0.967	yes
			0.931	0	1	0.931	0.964	no
70	327	97.2%	1	0.055	0.947	1	0.973	yes
			0.945	0	1	0.945	0.972	no
80	346	97.3%	1	0.054	0.949	1	0.974	yes
			0.946	0	1	0.946	0.972	no
90	338	97.7%	1	0.045	0.957	1	0.978	yes
			0.955	0	1	0.955	0.977	no
100	345	97.8%	1	0.044	0.958	1	0.979	yes
			0.956	0	1	0.956	0.978	no

Figure 5.5 shows a positive relationship between the size of decision tree models and resample rate. Similarly, both Figure 5.6 and Figure 5.7 show positive relationship between the accuracy and the resample rate, and between F-Measure and resample rate. The reason for positive relationship between any measure of size, accuracy, and F-Measure and resample rate is quite intuitive: when the resample size is larger, more instances in both majority class and minority class are available for learning.

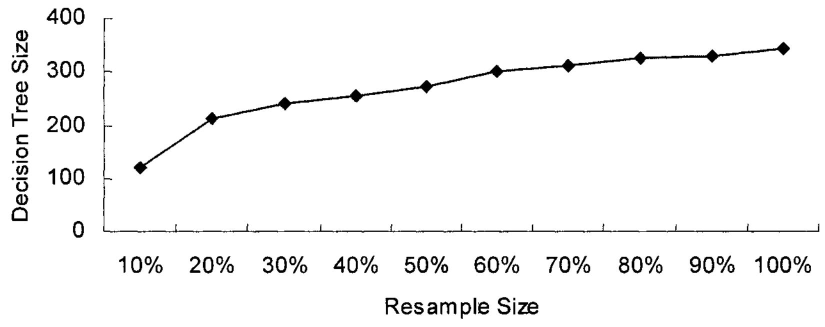

Figure 5.5 Decision tree size as the function of the Resample size (Percentage of original data set)

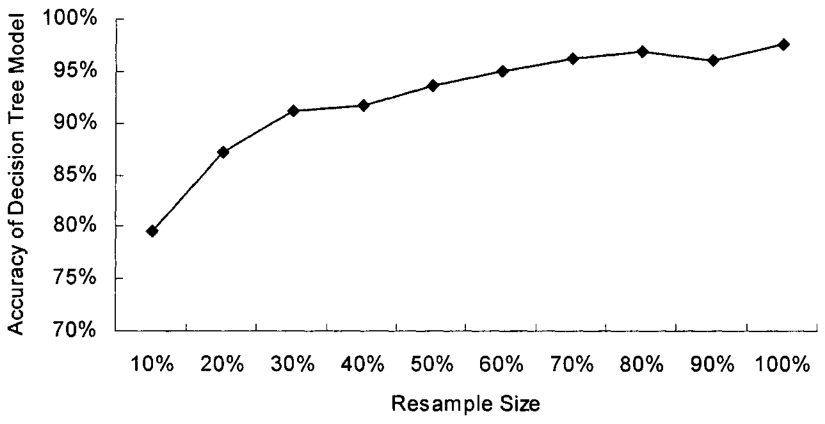

Figure 5.6 Decision tree accuracy as the function of the Resample size (Percentage of original data set)

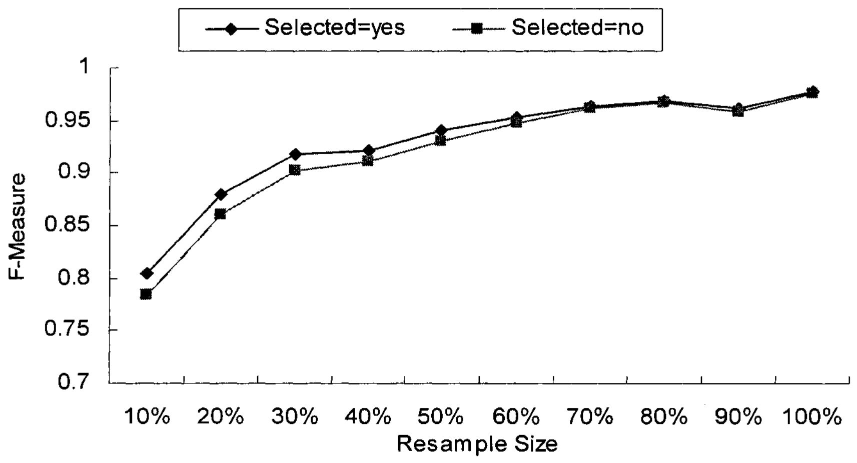

Figure 5.7 F-Measures as the function of the Resample size (Percentage of original data set)

Through the above numerical experiments, we analyze size and accuracy of the models after apply decision tree algorithm, and the numerical results show that the models can distinguish those best data selected by instance selection procedure. In this section, we further perform attribute selection by using the ReliefF attribute evaluator and Ranker method, in order to find out the most important attributes related to best data selection. The same eighteen data sets, used in previous experiments, are employed to conduct attribute evaluation, and the results are shown below in Table 5.9.

Table 5.9 Top seven attributes selected by ReliefF evaluator

Data Set	Top 7 Attributes						
1	DueDate2	ReleaseTime2	distance	ReleaseTime1	DueDate1	Weight1	Weight2
2	Distance	DueDate1	Weight1	DueDate2	Weight2	ReleaseTime1	
3	Distance	ReleaseTime2	ReleaseTime1	DueDate2	DueDate1	Weight1	
4	Weight1	distance	ProcessingTime2	Weight2	Processingtime1	DueDate2	
5	Weight2	distance	Weight1	Processingtime1	DueDate2	DueDate1	
6	Distance	Processingtime2	Weight2	DueDate2	Weight1	DueDate1	
7	Distance	Weight2	Weight1	ReleaseTime2	ReleaseTime1	DueDate1	
8	Distance	ReleaseTime1	ReleaseTime2	Weight1	DueDate2	Weight2	
9	Distance	Weight2	DueDate2	ReleaseTime2	DueDate1	Weight1	
10	Distance	Weight2	Processingtime2	Weight1	Processingtime1	DueDate1	
11	Distance	Weight1	Weight2	Processingtime2	Processingtime1	DueDate1	
12	Distance	Weight2	Processingtime2	Processingtime1	Weight1	DueDate2	
13	Distance	Weight2	Weight1	ReleaseTime2	Processingtime2	ReleaseTime1	
14	Distance	Weight1	Processingtime1	ReleaseTime1	Weight2	ReleaseTime2	
15	Distance	ReleaseTime2	ReleaseTime1	Weight1	Weight2	Processingtime2	
16	Distance	Processingtime1	Weight2	Weight1	Processingtime2	ReleaseTime2	
17	Distance	Weight2	Processingtime2	Processingtime1	ReleaseTime2		
18	Distance	Processingtime2	Weight2	Processingtime1	Weight1	ReleaseTime2	

From the attributes evaluation results, we can see that Distance ranks top one in most experiments. This is consistent with the result from tree models: Distance is always located in the very upper splitting nodes. Therefore, we can conclude that Distance is indeed highly related to best data selection. Apart from Distance, Weight ranks within top three nearly all the time, which indicate it is another important factor in terms of best data selection decision.

5.5 Summary and Discussion

In this chapter, we propose a new approach to learn the knowledge about how best data are selected by GA based instance selection method. We use a simple example to illustrate this new approach: how to apply data mining to learn how best instances are selected. New objective target is defined as to learn to determine whether a certain instance is selected or not. New attribute Distance is created which reveal the distance between the two jobs in each instance based on their original position in the dispatched list by EDD dispatching rule. Before applying decision tree algorithm, resampling is performed to balance the classes. Based on the recall, precision and F-measure, we conclude that the decision tree models created perform well to distinguish those instances selected by GA based instance selection method from other normal instances. Attribute selection results show that weight, distance, due dates and release time are the most important factors in respect to whether a certain instance is selected or not, which is consistent with the results from decision model.

More experiments are conducted to the data set used in Chapter 4. The extensive numerical results show that decision tree models are with good performances to learn the knowledge about the best data selection. Furthermore, we conclude that the larger the best data set, the larger size of the decision tree models, while the accuracy and F-Measures decrease if we keep the resample rate as the same. On the other hand, if we only change the resample rate, both size and accuracy of the decision models increase as we enlarge the resample size. Attributes selection results also reveal that Weight, Distance, Duedate are the most related factors in respect to best data selection, which is almost consistent to decision tree model where the attributes are in the most upper splitting nodes.

6 CONCLUSION

In this dissertation, we address the problem that whether it is possible to discover behind knowledge about implicit scheduling practices and then with this knowledge, scheduling practice could be improved. We proposed a new research framework for applying data mining in scheduling. Through numerical experiments, it is shown that this method allows us to apply data mining directly to scheduling data to learn how scheduling decision are made and new dispatching rules. Moreover, we investigate a further important problem whether scheduling practice could be improved using the knowledge discovered by data mining. A novel optimization-based instance selection method for scheduling is presented and is shown that the scheduling performance could be improved effectively through learning from the best data after instance selection. The major contributions of this dissertation are as follows:

- Data mining for scheduling framework

  It is shown that the new framework for applying data mining in scheduling provide effective approach to build a predictive model, which can work as a new dispatching rule, and obtain previously unknown structural knowledge and insights about scheduling decision making process. The target concept and flat file format and creation are novel in scheduling context and universally applicable for different scheduling problems. It is this special and appropriate flat file format that enable to apply data mining to learn dispatching rules and scheduling functions, and thus, any background knowledge about the scheduling practice is not necessary to be known before we apply this approach. Moreover this approach is scalable to complex scheduling context and the main difference is that the flat file would be more complicated with more attributes. However, there is another issue that needs further research. Both in Chapter 3 and Chapter 4 we mention the dependency of instances of our flat files transformed from original dispatch lists. We did not analyze the effects of data dependency on the performance of induced models in our research context, but this is an important issue that needs more attentions and research in future.

- Optimization based instance selection for scheduling

  The motivation of this approach is to only select those good quality instances for further learning, and such good instances represent optimal scheduling practice. We showed that the genetic algorithm based instance selection method is very effective to greatly improve the scheduling performance of model after learning from historical scheduling data. This method makes outstanding contribution to scheduling context, where it is very complex and almost impossible to capture all the aspects of the system. This is a universal method to build robust models which can work well as new dispatching rules. Furthermore, this instance selection method is universally applicable to other problems as long as the fitness function is changed according to the objectives of specific applications. In addition, only a very small number of instances are selected by this method, which indicate this method not only can improve the performance of the model through selecting the most critical data points, but also it can reduce the data size greatly for interpretation.

- Apply data mining to best instances analysis

  We proposed to employ data mining back to identify the selected best instances by defining a new target concept, new attribute creation, and oversampling. It is shown that this approach is effective and also applicable to other problems and contexts.

All of the research problems considered in this dissertation address important elements of applying data mining in scheduling and benefits of optimal instance selection methods. Some of the future research directions include: use real production data to research on effective methods to find out the insights behind why the selected instances are representing good or optimal scheduling practice. Because these insights are representation of implicit scheduling knowledge, and accordingly, this method will lead us a way to make critical implicit scheduling knowledge from implicit to explicit. As a result, good scheduling practice knowledge can be generalized as rules to improve scheduling decision making process.

REFERENCES

Aiolli, F. and Sperduti, A. (2004). Multiclass Classification with Multi-Prototype Support Vector Machines. Journal of Machine Learning Research. Vol(6), p817-850.

Almuallim, H., and Dietterich, T., 1994, "Learning Boolean concepts in the presence of many irrelevant features", Artificial Intelligence, 69 (1-2), pp 279-305.

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

Bowden, R. and Bullington, S. (1996). Development of manufacturing control strategies using unsupervised machine learning. IIE Transactions 28, 319-331.

Chajewska, U., Getoor, L., Norman, J., and Shahar, Y. (1998). Utility elicitation as a classi- fication problem. In G. F. Cooper and S. Moral, editors, Proceedings of the 14th Conference on Uncertainty in AI (UAI-98), p 79–88.

Chen, C.C. and Yih, Y. (1996). Identifying attributes for knowledge-based development in dynamic scheduling environments. International Journal of Production Research 34(6), 1739-1755.

Chu, Wei, and Ghahramani, Zoubin (2005). Preference learning with Gaussian processes. ICML 2005 - Proceedings of the 22nd International Conference on Machine Learning, 2005, p137-144.

Cohen, W., Schapire, R., and Y. Singer (1999). Learning to order things. Journal of Artificial Intelligence Research, vol(10), p243–270.

Deshpande, M. and G. Karypis (2002). Using conjunction of attribute values for classification. CIKM'02, Nov. 4-9, McLean, VA, 356-364.

Furnkranz, J. (2002). Round robin classification. Journal of Machine Learning Research, Vol (2), p721–747.

Furnkranz, Johannes, and Hullermeier, Eyke (2003). Pairwise preference learning and ranking. Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in

Computer Science), v 2837, Machine Learning: ECML 2003, p 145-156.

Geiger, C.D., R. Uzsoy, and H. Aytug (2003). Autonomous Learning of Effective Dispatch Policies for Flowshop Scheduling Problems. In Proceedings of the Industrial Engineering Research Conference (IERC'03).

Gervasio, Melinda, Moffitt, Michael, Pollack, Martha, Taylor, Joseph, and Uribe, Tomas (2005). Active Preference Learning for Personalized Calendar Scheduling Assistance. Proceedings of the 10th international conference on Intelligent user interfaces, ISBN:1-58113-894-6, Page: 90 – 97.

Goldberg, D., Nichols, D., Oki, B., and Terry, D. (1998). Using collaborative filtering to weave and information tapestry. Communications of the ACM, vol(35) p61–70.

Haddawy, V., Restificar, A., Geisler, B., and Miyamoto, J. (2003). Preference elicitation via theory refinement. Journal of Machine Learning Research. vol(4), p317-337.

Hall, M.A., 1998, "Correlation-based feature selection for discrete and numeric class machine learning", in Proceedings of the Seventeenth International Conference on Machine Learning, Stanford University, CA. Morgan Kaufmann.

Har-Peled, S., Roth, D., and Zimak, D. (2002). Constraint classification: A new approach to multiclass classifi-cation and ranking. Advances in Neural Information Processing Systems 15.

Hestermann C. and M. Wolber. (1997). A comparison between Operations Research-models and real world scheduling problems. The European Conference on Intelligent Management Systems in Operations. University of Salford, U.K., 25-26 March 1997.

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

Japkowicz, N., (2000). The Class Imbalance Problem: Significance and Strategies. Proceedings of the 2000 International Conference on Artificial Intelligence (IC-AI'2000): Special Track on Inductive Learning, (Las Vegas, Nevada).

John, G., Kohavi, R., and Pfleger, K., 1994, "Irrelevant features and the subset selection problem", in Proceedings of the Eleventh International Conference on Machine Learning, pp 121-129, New Brunswick, NJ. Morgan Kaufmann.

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

Kautz, H. (1998). Recommender Systems: Papers from the AAAI Workshop, Menlo Park, CA, AAAI Press. Technical Report WS-98-08.

Kim, C.-O., Min, H.-S., and Yih, Y. (1998). Integration of inductive learning and neural networks for multi-objective FMS scheduling. International Journal of Production Research 36(9), 2497-2509.

Kira, K. and Rendell, L. (1992). A Practical Approach to Feature Selection. Proceedings. Ninth International Conference of Machine Learning, pp. 249-256.

Kohavi, R. and John, G. H., 1997, "Wrappers for feature subset selection", Artificial Intelligence, Vol. 97, No. 1-2, pp 273-324.

Kohavi, R. and Provost, F. (1998). Robust classification systems for imprecise environments. In Proceedings of the Fifteenth National Conference on Artificial Intelligence, 706-713.

Koller, D. and Sahami, M., 1997, "Hierarchically classifying documents using very few words", in Proceedings of the International Conference on Machine Learning, pp 170-178.

Kononenko, I. (1994). Estimating Attributes: Analysis and Extensions of RELIEF. Proceedings of 1994 European Conference of Machine Learning, 1994.

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

Law, A. and Kelton, W. (2000). Simulation Modeling and Analysis. McGraw-Hill Education – Europe. ISBN: 0071165371.

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

Lesh, N. M.J. Zaki, and M Ogihara (1999). Mining features for sequence classification. In 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

Lewis, D. and Gale, W. (1994). A sequential algorithm for training text classifiers. Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval, p.3-12, July 03-06, Dublin, Ireland.

Lewis, D. and Catlett, J. (1994). Heterogeneous uncertainty sampling for supervised Learning. Proceedings of the Eleventh International Conference of Machine Learning, (San Francisco, CA), pp. 148-156, Morgan Kaufmann.

Li, D.-C. and She, I.-S. (1994). Using unsupervised learning technologies to induce scheduling knowledge for FMSs. International Journal of Production Research 32(9), 2187-2199.

Li, X. and Olafsson, S. (2005). Discovering dispatching rules using data mining. Journal of Scheduling. Vol. 8, Page. 515-527.

Ling, C. and Li, C. (1998). Data Mining for Direct Marketing Problems and Solutions. Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-*98)*, (New York, NY), AAAI Press.

Liu, R., and Setiono, R., 1996, solution", in Proceedings of the Thirteenth International Conference on Machine Learning, Morgan Kaufmann.

Min, H.-S., Yih, Y., and Kim, C.-O. (1998). A competitive neural network approach to multi-objective FMS scheduling. International Journal of Production Research 36(7), 1749-1765.

Mitchell, T. (1997) Machine Learning.McGraw-Hill International Editions. ISBN 0-07-042807-7.

Nakasuka, S. and Yoshida, T. (1992). Dynamic scheduling system utilizing machine learning as a knowledge acquisition tool. International Journal of Production Research 30(2), 411-431.

Neville, J., Jensen, D., Friedland, L., and M. Hay. Learning relational probability trees. In Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD03), 2003.

Noronha, S. and Sarma.,V. (1991). Knowledge-based approaches for scheduling problems: a survey. IEEE Transactions on Knowledge and Data Engineering 3(2), 160-171.

Pinedo, M. (1995). *Scheduling: Theory, Algorithms and Systems*. Prentice Hall.

Piramuthu, S., Raman, N., and Shaw, M.J. (1994). Learning-based scheduling in a flexible manufacturing flow line. IEEE Transactions on Engineering Management 41(2), 172-182.

Piramuthu, S., Raman, N., Shaw, and M.J., Park, S. (1993). Integration of simulation modeling and inductive learning in an adaptive decision support system. Decision Support Systems 9, 127-142.

Priore, P., Fuente, D., Gomez, A., and 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.

Raman, B. and Ioerger, T., 2002, "Instance based filter for feature selection", Journal of Machine Learning Research 1-23.

Shaw, M.J., Park, S., and Raman, N. (1992). Intelligent scheduling with machine learning capabilities: the induction of scheduling knowledge. IIE Transactions 24(2), 156-168.

Shaw, M.J. and Whinston, A.B. (1989). An artificial intelligence approach to the scheduling of flexible manufacturing systems. IIE Transactions 21(2), 170-183.

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. and Olafsson, S. (2005). Optimal Instance Selection for Improved Decision Tree Induction.

Yang, J. and Honavar, V. (1998). Feature subset selection using a genetic algorithm. In H. Motada and H. Liu (eds), Feature Selection, Construction, and Subset Selection: A Data Mining Perspective, Kluwer, New York.