2007

Optimal instance selection for improved decision tree

Shuning Wu
Iowa State University

Optimal instance selection for improved decision tree

by

Shuning Wu

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Industrial Engineering

Program of Study Committee:
Sigurdur Olafsson, Major Professor
John Jackman
Sarah Ryan
Di Cook
Dan Zhu

Iowa State University

Ames, Iowa

2007

UMI Number: 3274836

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Instance selection plays an important role in improving scalability of data mining algorithms, but it can also be used to improve the quality of the data mining results. In this dissertation we present a new optimization-based approach for instance selection that uses a genetic algorithm (GA) to select a subset of instances to produce a simpler decision tree with acceptable accuracy. The resultant trees are likely to be easier to comprehend and interpret by the decision maker and hence more useful in practice. We present numerical results for several difficult test datasets that indicate that GA-based instance selection can often reduce the size of the decision tree by an order of magnitude while still maintaining good prediction accuracy. The results suggest that GA-based instance selection works best for low entropy datasets. With higher entropy, there will be less benefit from instance selection. A comparison between GA and other heuristic approaches such as *Rmhc* (Random Mutation Hill Climbing) and simple construction heuristic, indicates that GA is able to obtain a good solution with low computation cost even for some large datasets. One advantage of instance selection is that it is able to increase the average instances associated with the leaves of the decision trees to avoid overfitting, thus instance selection can be used as an effective alternative to prune decision trees. Finally, the analysis on the selected instances reveals that instance selection helps to reduce outliers, reduce missing values, and select the most useful instances for separating classes.

CHAPTER 1. INTRODUCTION

1.1 Literature Review and Related Research

In recent years, the field of data mining has seen an explosion of interest from both academia and industry. Driving this interest is the fact that data collection and storage has become easier and less expensive, so databases in modern enterprises are now often massive. These massive databases often contain a wealth of important data that traditional methods of analysis fail to transform into relevant knowledge. Specifically, meaningful knowledge is often hidden and unexpected, and hypothesis driven methods, such as on-line analytical processing and most statistical methods, will generally fail to uncover such knowledge. Inductive methods, which learn directly from the data without an a priori hypothesis, can uncover hidden patterns and knowledge (Olafsson et al. 2004).

In this study, the term data mining is used to refer to all aspects of an automated or semi-automated process for extracting previously unknown and potentially useful knowledge and patterns from large databases. This field has become very popular and found various applications. The process of data mining involves numerous steps, including data integration and preprocessing, inductive learning from the instances in the prepared database, and evaluation and interpretation of the resulting patterns. Inductive learning, which may be considered as the core of the process, typically involves one or more of three learning tasks:

classification, data clustering, or association rule discovery. This study focuses on classification, which is one of the most common learning tasks. In classification, there is a specific attribute called the class attribute that can take a given number of values, and the goal is to induce a model that can be used to discriminate new data into classes according to those values. The induction is based on a labelled training set, where each instance is labelled according to the value of the class attribute. The objective of the classification is to first analyse the training data and develop an accurate description or a model for each class using the attributes available in the data. Such class descriptions are then used to classify future independent test data or to develop a better description for each class. The accuracy of the model is usually measured by the proportion of the number of correctly classified instances over the number of total instances in the test data. Many methods have been studied for classification (Fayad et al., 1996; Weiss and Kulikowski, 1991), including decision tree induction (Quinlan, 1993), support vector machines (Boser et al., 1992), neural networks (Ripley, 1996), and Bayesian networks (Jensen, 2001).

Data preparation is one of the most important and time consuming phases in knowledge discovery (Reinartz, 2001). Preparation tasks (such as data selection, data cleaning, data construction, data integration, and data formatting) often determine the success of data mining engagements. In this research, the importance of instance selection is the primary focus because the size of current and future databases often exceeds the amount of data which current data mining algorithms can handle properly. Hence, we argue that by using

instance selection to reduce the data before data mining, existing data mining algorithms can be used to analyze the data.

1.1.1 Instance Selection

Instance selection has only recently been investigated from the perspective of selecting the best instances to improve model performance. Originally there were several reasons for instance selection. The first of them is to reduce the noise in original dataset because some learning algorithms may be noise-fragile (for example, linear discrimination methods (Duda et al. 1997)). The second reason to shrink the training set is to reduce the amount of computation, especially for instance-based learning algorithms (Ada et al. 1991) such as the k-nearest neighbours (Cover and Hart, 1967), or for very large training sets.

Historically, instance selection to improve model performance has been focused on improving the efficiency of the nearest neighbour classifier. This was a clear motivation since storage requirements and computational costs make the nearest neighbour approach unsuitable for dealing with very large datasets. Therefore, quite a few of methods have therefore been proposed to select instances for the nearest neighbour approach. Several methods have been proposed involving sampling techniques.

For the sampling approaches, perhaps the most important but very difficult issue is determining the appropriate sample size to maintain acceptable accuracy. This includes simple random sampling, stratified sampling by selecting the minor classes more frequently

in order to make the class values uniformly distributed, adaptive sampling in which sequential sampling is used to scale up knowledge discovery algorithms (Domingo et al. 2001), progressive sampling by using progressively larger samples as long as model accuracy improves to obtain appropriate training set size (Provost et al. 2001), and the *Rmhc* method of Skalak (1994), which is one kind of random mutation hill climbing algorithm: First, subset *S* is formed by randomly selecting from training set *T*, containing a fixed number of instances. In each iteration, the algorithm interchanges an instance from *S* with another from *T*. The change is maintained if it offers better accuracy. A similar approach was used by Wilson (1997), except that instead of interchanging instances one at a time all of the instances in *S* are exchanged and the new candidate instances were kept if the accuracy improved. Additional methods based on nearest neighbour (NN) rules are listed in Table 1 (Cano et al. 2003; Jankowski and GrochowskiL, 2004):

Table 1 Different methods for instance selection based on NN rules

Method	Reference	Brief description of the method
CNN	Hart (1968)	Condensed Nearest Neighbor Rule: CNN tries to find a consistent subset, which correctly classifies all of the remaining points in the sample set.
ENN	Wilson (1972)	Edited Nearest Neighbor: ENN removes a given instance if its class does not agree with the majority class of its neighbors. This removes noisy instances, as well as close border cases, leaving smoother decision boundaries.
RENN	Wilson (1972)	Repeated Edited Nearest Neighbor: RENN applies ENN repeatedly as long as any changes are observed in the selected set.
RNN	Gates (1972)	Reduced Nearest Neighbor: RNN starts from original training set and rejects only those instances that do not decrease accuracy.
VSM	Lowe (1995)	It removes an instance if most of its nearest neighbors classify it correctly or incorrectly.
Multiedit	Devijver and Kittler (1982)	It is a modification over ENN algorithm that guarantees the statistical independence in the prototype selected.

Table 1 Different methods for instance selection based on NN rules (continued)

Method	Reference	Brief description of the method
Shrink	Kibbler and Aha (1987)	Similar to the RNN, it retains border points, but unlike RNN, this algorithm is sensitive to noise.
IB2	Kibbler and Aha (1987)	It is similar to CNN but selecting only those instances that cannot be correctly classified. IB2 is sensitive to noise.
IB3	Aha and Kibbler (1991)	IB3 reduces the noise sensitivity of IB2 by only retaining acceptable misclassified instances. IB3 achieves greater data reductions and higher accuracy than IB2 on unseen instances.
ICF	Brighton and Mellish (2002)	Iterative Case Filtering: ICF tries to select the instances which classify more prototypes correctly. *Reachability* and *coverage* are used in the selection.
Drop1	Wilson and Martinez (1997)	Drop1 removes instance x from the training set if it does not change classification of instances from $A(x)$ (only those instances depend on x).
Drop2	Wilson and Martinez (1997)	Drop2 sorts instances according to their distances from the nearest opposite class instance.
Drop3	Wilson and Martinez (1997)	Drop3 additionally runs the ENN algorithm before starting the Drop2 algorithm.

Another approach to instance selection is an evolutionary algorithm (EA). The success of evolutionary algorithms is largely due to their ability to exploit the information accumulated about an initially unknown search space. This is their key feature, particularly in large, complex, and poorly understood search spaces, where classical optimization methods do not work well. In such cases, they offer an alternative approach to problems requiring efficient and effective search techniques. Examples of such work includes the work of Reeves (2001), in which GAs were used for instance selection in order to improve the generalization of Radial Basis Function (RBF) networks. RBF networks have traditionally been associated with radial functions in a single-layer linear neural network. Ishibuchi et al. (2001) presented a genetic-algorithm-based instance and feature selection in a nearest neighbour classifier. The

results show that the generalization ability of nearest neighbour classifiers improves for datasets with large overlaps between different classes.

Cano et al. (2003) described four models of EAs that were evolutionary instance selection algorithms, namely, two classical GA models including generational genetic algorithm and steady-state genetic algorithm, heterogeneous recombination and cataclysmic mutation (CHC) adaptive search algorithm, which is a classical model that introduces different features to obtain a trade-off between exploration and exploitation, and Population-Based Incremental Learning (PBIL), which is a specific EA designed for binary search spaces and attempts to explicitly maintain statistics about the search space to decide where to sample next. Their results show that EAs outperform the classical algorithms based on nearest neighbour rules and random sampling, simultaneously offering two main advantages: better data reduction percentages and higher classification accuracy.

There are two key issues in using evolutionary algorithms for instance selection: the representation of the solutions and the definition of the fitness function. Prior research primarily used binary representation of the solution (Ramon, 2003; Cano 2004). A chromosome consists of genes (one for each instance in *T*) with two possible states: 0 and 1. If the gene is 1, then its associated instance is included in *S* (the subset of *T*) which represents the chromosome. If it is 0, then it is not included. A widely used fitness function is the combination of two values: the classification accuracy associated with the selected subset *S*

of instances and the percent reduction in instances of *S* with regards to *T* (Ramon, 2003; Cano 2004).

Some unconventional instance selection methods have been used in specific applications. Lam et al. (2001) integrated instance-filtering and instance-averaging techniques for instance-based learning algorithms with good performance in data reduction and classification accuracy. Wang (2001) generated a set of representative instances based on a model of data built on hypertuples for nearest neighbour classifier and in some cases the selected instances outperformed the C5 decision tree classifier. Wright and Hodges (2001) incorporated domain knowledge (i.e. missing attributes and the relative importance of different attributes) into a multi-criteria decision-making technique to guide instance selection.

1.1.2 Decision Trees

Decision trees are a popular technique for classification. The main reason behind their popularity is their relative advantage in terms of interpretability. Their popularity is also aided by available implementations such as CART (Breiman et al., 1984) and C4.5 (Quinlan, 1993). Several advantages of the decision tree as a classification tool have been pointed out in the literature (Maimon and Rokach, 2005):

- Decision tree is self-explanatory and easy to follow. Furthermore, they can be converted to a set of decision rules by forming a rule corresponding to each path from the root of the tree to each of its leaves. Thus, this representation is considered comprehensible.
- Decision tree can handle both nominal and numerical attributes.
- Decision tree is capable of handling datasets that may have errors or missing values.
- Decision tree is considered to be a nonparametric method, that is, there are no assumptions about the space distribution and the classifier structure.

A decision tree is expressed as a recursive partition of the instance space. Most decision tree induction algorithms construct a tree in a top-down manner by selecting attributes one at a time and splitting the data according to the values of those attributes. The most important attribute is selected as the top split node, and so forth. Some common splitting criteria include impurity-based criteria (Rokach and Maimon, 2005), information gain (Quinlan, 1987), gain ratio (Quinlan, 1993), gini index (Breiman et al. 1984), likelihood-ratio Chi-squared statistics (Attneave, 1959), DKM criterion (Dietterich et al. 1996), distance measure (Lopez de Mantras, 1991), towing criterion (Breiman et al. 1984), orthogonal criterion (Fayyad and Irani, 1992), Kolmogorov-Smirnov criterion (Friedman, 1977), and AUC-splitting criteria (Ferri, et al. 2002). Comparative studies of the splitting criteria have been conducted by several researchers during the last thirty years, such as Baker and Jain (1976), Fayyad and Irani (1992), Loh and Shih (1997), and Lim et al. (2000). In most of the cases, the choice of splitting criteria will not make much difference on the tree performance.

For example, in C4.5 attributes are chosen to maximize the information gain ratio in the split (Quinlan, 1993). This is an entropy measure designed to increase the average class purity of the resulting subsets. The entropy function is defined by

$$Entropy(S) = \sum_{i=1}^{c} - p_i \log_2 p_i \tag{1}$$

where S represents the training data and p_i is the proportion of S classified as class i. Then the information gain *Gain(S,a)* is defined as:

$$\begin{aligned} Gain(S,a) &= Entropy(S) - \sum_{v \in Values(a)} \frac{|S_v|}{|S|} Entropy(S_v) \\ S_v &= \{s \in S : a(s) = v\} \end{aligned} \tag{2}$$

where *Values(a)* represents all possible values for attribute a. *Gain*(S,a) implies the expected information provided about the classification from knowing the value of attribute a, but it tends to favor attributes that have a large number of values. For example, if we have an attribute a that has a distinct value for each instance, then $\sum_{v \in Values(a)} \frac{|S_v|}{|S|} Entropy(S_v) = 0$, thus *Gain(S,a)* is maximal. To compensate for this, C4.5 algorithm uses the following ratio in equation (3) instead of *Gain*, here *SplitInfo*(S,a) is the information due to the split of S on the basis of the value of attribute a. Notice that the *SplitInfo* term discourages the selection of attributes with many uniformly distributed values.

$$\begin{aligned} GainRatio(S,a) &= \frac{Gain(S,a)}{SplitInfo(S,a)} \\ SplitInfo(S,a) &= - \sum_{v \in Values(a)} \frac{|S_v|}{|S|} \log \frac{|S_v|}{|S|} \end{aligned} \tag{3}$$

Algorithms such as C4.5 and CART are computationally efficient and have proven very successful in practice. However, the greedy characteristic of decision trees leads to one disadvantage which is its over-sensitivity to the training data, to irrelevant attributes, and to noise (Quinlan, 1993). Also as decision trees use the "divide and conquer" method, they tend to perform well if a few highly relevant attributes exist, but less so if many complex interactions are present, in some cases the tree will contain several duplications of the same subtree in order to represent the classifier (Pagallo and Huassler, 1990).

As for other classification methods, the quality of decision trees is primarily measured in terms of its accuracy in classifying new data. However, we argue that the complexity of the decision tree is also important and must be controlled. Naturally, decision-makers prefer less complex decision trees, since they may be considered more comprehensible. Usually the tree complexity is measured by one of the following metrics (Rokach and Maimon, 2005): the total number of nodes, total number of leaves, tree depth, and number of attributes used. According to Breiman et al. (1984), the tree complexity has a crucial effect on its accuracy. Furthermore, since interpretability is a major motivation in the use of decision trees the complexity of the tree becomes critical. If the tree is too complex then it may no longer be easily interpretable. Here we measure the complexity of the tree in terms of the total number of nodes, which we refer to as the size of the tree, and the quality of the tree is measured in terms of a combination of the accuracy and the size of the tree. So typically the goal is to find

the optimal decision tree by minimizing the prediction error as well as minimizing the number of nodes.

Unfortunately finding the minimal decision tree consistent with the training set is an NP-hard problem (Hancock et al. 1996). Moreover, it has been shown that constructing a minimal binary tree with respect to the expected number of tests required for classifying an unseen instance is NP-complete (Hyafil and Rivest, 1976). Even finding the minimal equivalent decision tree for a given decision tree (Zantema and Bodlaender, 2000) or building the optimal decision tree from decision tables is known to be NP-hard (Naumov, 1991) and heuristics must be applied. Kennedy et al. (1997) first developed a genetic algorithm for optimizing decision trees. In their approach, a binary tree is represented by a number of unit subtrees, each having a root node and two branches. In more recent work, Fu et al. (2003a; 2003b; 2004) also used genetic algorithms for this task. Their method uses C4.5 to generate *K* trees as the initial population, and then exchanges the subtrees between trees (crossover) or within the same tree (mutation). At the end of a generation, logic checks and pruning are carried out to improve the decision tree. They show that the resulting tree performs better than C4.5 and the computation time only increases linearly as the size of the training and scoring combination increases. Furthermore, creating each tree only requires a small percent of data to generate high-quality decision trees. In related work, Dhar et al. (2000) use an adaptive resampling method where instead of using a complete decision tree as

the chromosomal unit, a chromosome is simply a rule, that is, any complete path from the root node of the tree to a leaf node.

Previous research using genetic algorithms to optimize decision trees did not address controlling the growth of the tree, because the genetic algorithm does not evaluate the size of the tree, only the accuracy. GA may lead to a tree that becomes either overly complex or the search may settle to a too simple tree. To address this, Niimi and Tazaki (2000) combined genetic programming with an association rule algorithm for decision tree construction. In this approach, rules generated by the Apriori association rule discovery algorithm (Agrawal et al., 1993) were used as the initial individual decision trees for a subsequent genetic programming algorithm. Another approach to improve the optimization of the decision tree is to improve the fitness function used by the genetic algorithm. Traditional fitness functions use the mean accuracy as the performance measure. Fu et al. (2003b) investigated the use of various percentiles of the distribution of classification accuracy, in place of the mean, and developed a genetic algorithm that simultaneously considers two fitness criteria. In other work, the utilization of a fitness function based on the *J*-Measure, which determines the information content of a tree, was used as a preference criterion to find the decision tree that classifies a set of instances in the best way (Folino et al., 2001).

Previous research on optimizing decision trees is mainly focused on inducing good decision trees with high accuracy from given fixed data. However, recently a few researchers have considered optimizing the tree through instance selection. In particular, Chauchat and

Rakotomalala (2001) proposed a sampling strategy to build decision trees from a very large database with many continuous attributes by determining the sufficient sample size to obtain a decision tree as efficient as that built using all of the data. Yoon et al. (2001) employed tree-based sampling for incremental classification. In their method, the class distribution is represented by the weighted samples, which are extracted from the nodes of intermediate decision trees using a clustering technique. An intermediate classifier is built only on the incremental portion of the data. This approach is independent of data distribution and can be applied to large datasets. Cano et al. (2004) applied stratified CHC to the original training dataset and analyzed the selected training sets quality by C4.5 trees from the precision and interpretability perspectives. However, this work did not consider the size of the trees, which as we have noted before is an important factor to measure the interpretability of the tree. Moreover, it used the entire training dataset in the algorithm, which leads to the long processing time and may make it difficult to apply this approach to very large datasets. On the other hand, Endou and Zhao (2002) divided the training data into several subsets and then used a GA to evolve a small dataset that can cover the domain knowledge with reasonable accuracy. From this dataset, a small but good decision tree can be designed. While this method increases the efficiency of the algorithm, a key limitation is quite similar to the previous research, that is, by only pursuing the accuracy of the decision trees it does not control the growth of the trees. Therefore, during the search process the tree may become overly deep and complex, even though it has rather high accuracy. Another problem is that

Endou and Zhao's algorithm tries to find the best subset, but sometimes this is not possible, so this approach is only valid for redundant datasets, which limits its applications.

In our research, the goal is to optimize decision trees based on instance selection, but with the main objective of determining how such optimization can be applied to design *small, more interpretable and high-quality* decision trees. At the same time we expect that effective instance selection will increase the ability of a decision tree induction algorithm to deal with very large amounts of data. The methodology utilizes a genetic algorithm (GA) approach, which is similar to previous work in this area, but we also introduce new formulations for representing the solutions, defining the fitness function and choosing the various outputs.

1.2 Hypotheses

We will use the following notation in stating the hypotheses of this thesis:

T	entire dataset
S	instance subset of T
$\|S\|$	size of S, i.e., number of instances in subset S
$\psi(S)$	decision tree based on S
$r(i)$	number of instances on leaf node i
$l(\psi(S))$	number of the leaf nodes in $\psi(S)$
$R(\psi(S))$	Average Leaf Ratio (ALR) - average number of instances classified in leaf nodes
$\hat{e}(\psi(S))$	estimated error rate of decision tree

Hypothesis 1

GA-based instance selection will produce smaller and more interpretable decision trees while maintaining an acceptable level of accuracy.

Since interpretability is a major issue in the use of decision trees, the size of the decision trees must be controlled, and to improve interpretability it may be necessary to reduce the tree sizes. We define improvements of instance selection as a reduction in the size of the decision tree, which is measured by the number of the nodes in the decision tree. Reductions in size should result in decision trees that are more easily interpreted, while maintaining adequate prediction accuracy. To formulate this problem mathematically, we let $T = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ be a training dataset of n instances, and the objective is to select the smallest subset $S \subseteq T$ such that the tree $\psi(S)$ induced on this subset has the smallest tree size while at the same time maintains good accuracy. Therefore, generally speaking, instance selection can be formulated as a nonlinear integer programming multi-objective optimization problem. The decision variables determine which instances are selected

$$x_i = \begin{cases} 1 & \text{if instance } i \text{ is selected} \\ 0 & \textit{otherwise} \end{cases}$$

and the optimization problem then becomes:

$$\begin{aligned} \min \quad & size(\psi(S)) \; and \; \sum x_i \\ \text{s.t.} \quad & 1 - \hat{e}(\psi(S)) \geq [1 - \hat{e}(\psi(T)] \times (1 - \varepsilon) \\ & x_i = 0, 1 \end{aligned} \tag{4}$$

where $\hat{e}(\psi)$ is the estimate of the error rate for the decision tree ψ, so $[1\text{-}\hat{e}(\psi)]$ will be the estimated accuracy of the decision tree, and ε is a parameter related to error tolerance, i.e.

we could set ε=5~10%. In order to solve this optimization problem, we have considered several different approaches as illustrated in Figure 1:

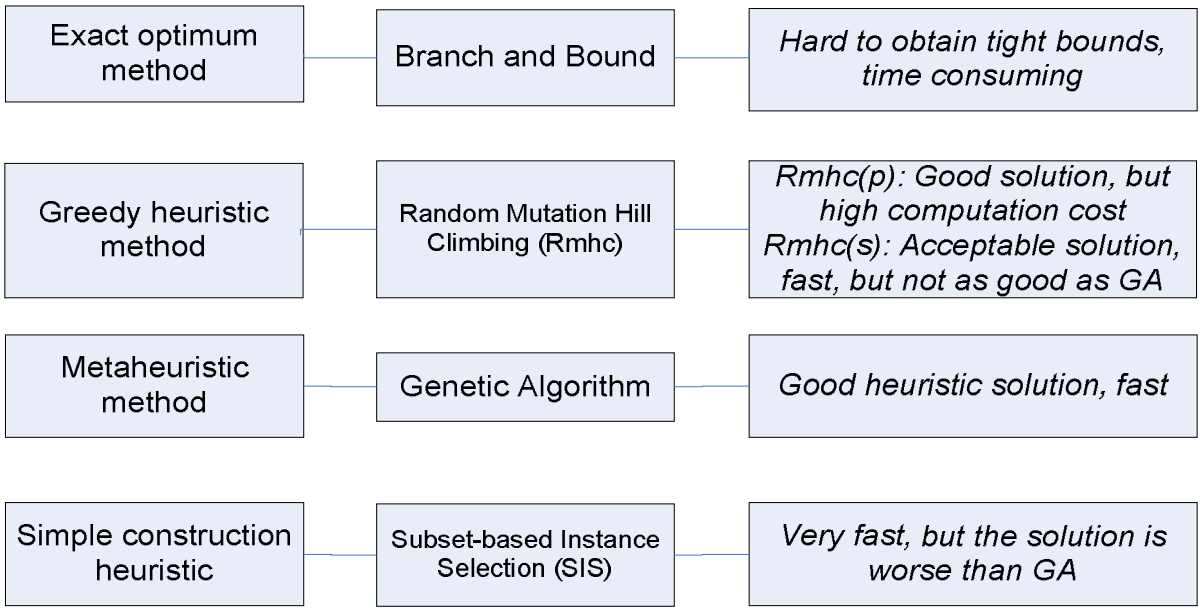


Figure 1 Different approaches for instance selection

As mentioned above, instance selection is a multi-objective non-linear integer programming problem. Given that we want to minimize the size of instance subset as well as the size of the decision tree, traditional integer programming methods such as branch and bound are not suitable for this problem. This is due to the large number of possible branches (2^n) and more importantly, the inability to find a good relaxed problem because of the complexity of the decision trees. The implication is that it is difficult to acquire tight bounds, making branch-and-bound almost infeasible for relatively large datasets. In order to reduce the search space, we have developed a 2-phase *Rmhc* method (described in Chapter 3). The point-based *Rmhc* method is able to select a relatively best instance subset in finite steps but it is still quite time consuming. The subset-based *Rmhc* method greatly increases the speed but the

solution quality is worse than GA. It will be shown in Chapter 3 that a GA-based instance selection produce similar results as the point-based *Rmhc* method with much less computation cost (i.e., it is more efficient). On the other hand, simple construction heuristic is quite fast, but it is not as good as GA, especially for some large datasets, the tree's size will be much larger than that from GA approach.

Finally, GA-based instance selection turns out to be the best option in balancing the quality of the solutions and the speed of the algorithm. This is shown through the application of GA-based instance selection to several test problems, and a comparison of the original decision tree developed on the entire dataset with the tree from the selected instances. Using this approach, we have shown in Chapter 2 that for C4.5 decision trees, the size of the tree can be significantly reduced using instance selection, while the predictive accuracy is as good.

Hypothesis 2

Optimization-based instance selection prevents overfitting in decision tree learning.

Overfitting is possible even when the training data are noise-free, especially when small numbers of instances are associated with leaf nodes. In this case, it is quite possible for coincidental regularities to occur (Mitchell, 1997), in which some attribute happens to partition the instances very well, despite being unrelated to the actual class attribute.

Whenever such coincidental regularities exist, there is a risk of overfitting. We introduce a measure average leaf ratio (ALR) $R(\psi(S))$ that measures the average fraction of instances per leaf node, which is given by

$$R(\psi(S)) = \frac{\sum_{i=1}^{l(\psi(S))} \frac{r(i)}{|S|}}{l(\psi(S))} \times 100\% . \quad (5)$$

It is expected that the decision tree after instance selection should have higher value of $R(\psi(S))$ than the decision tree based on original training data. With more number of instances associated with leaf nodes, the tree will stop growing earlier before it reaches the point where it perfectly classifies the training data, so there will be less chance for the occurring of overfitting. Another approach to avoiding overfitting in decision tree learning is to allow the tree to overfit the data, and then post-prune the tree. In Chapter 4, we will compare instance selection with several post-pruning techniques to show that instance selection can be used as an effective alternative for decision tree pruning.

Hypothesis 3

Number of instances, number of class values, and number of attributes will affect the performance of instance selection for improving decision tree.

Datasets for data mining applications are usually large and may involve several million instances. Furthermore, each instance typically consists of ten to hundreds attributes. Using large datasets usually improves the accuracy of the classifier, but the enormity and complexity of the data involved in these applications makes the classification task

computationally intensive. Decision trees, for example, require several passes over the entire dataset and efficient computation using this method requires all the instances be stored in the main memory. This limitation and the interpretability issue generally limit decision trees to classifying small datasets.

Therefore, empirical studies were performed to compare decision trees with and without instance selection. Parameters including number of instances, number of class values, and number of attributes influence were studied to determine their effect(s) on the performance of the algorithm in terms of computation cost, the decision tree's size, the classification accuracy and the average leaf ratio (ALR).

The remainder of the dissertation is organized as follows. In Chapter 2, we formulate the optimal instance selection problem, propose a genetic algorithm for finding heuristic solutions to this problem, and determine guidelines for how the GA approach should be implemented and when this approach works best. In Chapter 3, we compare the results between different instance selection approaches to demonstrate the effectiveness of GA-based instance selection. In Chapter 4, we illustrate how instance selection can be applied to decision tree pruning and provide a case study to show the benefits from instance selection. Finally, Chapter 5 contains some concluding remarks and future research directions.

CHAPTER 2. METAHEURISTIC INSTANCE SELECTION

2.1 Metaheuristic Method and Genetic Algorithm

A metaheuristic is a heuristic method for solving a very general class of computational problems by combining user given black-box procedures--usually heuristics themselves--in an efficient way (Blum and Roli, 2003). Metaheuristics are commonly used to solve combinatorial optimization problems. The goal of combinatorial optimization is to find a set of discrete values for a set of decision variables that maximize (or minimize) an objective function.

Metaheuristics track the current best solution and the current values for the set of decision variables (i.e., the current state). A new set of values are generated by an interchange or mutator procedure. Such interchanges are often probabilistic procedures. The set of new values produced by the mutator are in the neighborhood of the current set. More sophisticated metaheuristics maintain more than one set of values. Criteria are defined to select which sets will be retained and which sets will be discarded. New sets can be generated by some combination or crossover of two or more sets. Since the set of candidates is usually very large, metaheuristics are usually constrained by the maximum number of iterations. When unconstrained, some exact metaheuristics will eventually check all candidates, and use heuristic methods only to choose the order of enumeration. Therefore,

they will always find the true optimum, if the constraint is large enough. Other metaheuristics give only a weaker probabilistic guarantee, namely that, as the number of iterations approaches infinity, the probability of checking every candidate tends to be one.

Most of the success of metaheuristic methods is due to their ability to exploit the information accumulated about an initially unknown search space (Blum and Roli, 2003). This is their key feature, particularly in large, complex, and poorly understood search spaces, where classical search tools (enumerative, heuristic, etc.) are inappropriate. In such cases, they offer a valid approach to problems requiring efficient and effective search techniques. Some well-known metaheuristics include Genetic Algorithm (Holland, 1975), Simulated Annealing (Kirkpatrick et al., 1983), and Tabu Search (Glover and Laguna, 1997).

Genetic algorithms are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, natural selection, and recombination (or crossover) (Goldberg, 1989). Key characteristics of GA include:

- GA searches from a *population* of points, not a single point.
- GA uses a fitness function (i.e., objective function), not derivatives or other auxiliary knowledge.
- GA uses *probabilistic* transition rules, not deterministic rules.
- GA is robust with respect to local minima or maxima.
- GA works well on mixed discrete/continuous problems.
- GA is stochastic and easy to implement.

A population of abstract representations (chromosomes) represents a solution to the problem. Usually, solutions are represented in binary as a sequence of 0s and 1s, but different encodings are also possible for specific applications. The evolution starts from a population of completely random individuals (i.e, the first generation). Each iteration represents a new generation in which the fitness of the whole population is evaluated; multiple individuals are randomly selected from the current population (based on their fitness) and modified (mutated or recombined) to form a new population, which becomes the next generation. A pseudo-code genetic algorithm is shown below (Goldberg, 1989):

```
CHOOSE INITIAL POPULATION
REPEAT
    Evaluate the individual fitnesses of a certain proportion of the population
    Select pairs of best-ranking individuals to reproduce
    Breed new generation through crossover and mutation
UNTIL TERMINATING CONDITION
```

2.2 Genetic Algorithm for Instance Selection

Our methodology uses GA similar to previous work in this area. The GA greatly reduces the search space in that it is only carried out on the "populations" instead of on individual instance. The objective function in (4) was modified to include the decision tree size and estimated error rate.

2.2.1 Fitness Function

The definition of our fitness function incorporates some ideas from the concept of the information entropy and *minimum description length principle* (MDL, Rissanen 1985). Originally, information entropy proposed by Shannon (1948) can be derived by calculating the mathematical expectation of the amount of information contained in a digit from the information source. Shannon's entropy measure came to be taken as a measure of the uncertainty about the realization of a random variable. It thus served as a proxy capturing the concept of information contained in a message as opposed to the portion of the message that is strictly determined (hence predictable) by inherent structures. There is a longstanding tradition in science that, given a choice of theories that are equally good the simplest theory should be chosen, which is known as *Occam's Razor* (Ariew, 1976), and the minimum description length principle takes the stance that the best theory for a body of data is one that minimizes the size of the theory plus the amount of information necessary to specify the exceptions relative to the theory.

In data mining, theory corresponds to the predictive model, so the size of the theory is the size of the data mining model and the amount of information for exceptions can be roughly measured as the estimated error rates of the data mining model. Therefore, we define the fitness function, $f(S)$, in the form of entropy giving

$$f(S) = -\log\left(\hat{e}(\psi(S))\right) - a\log\left(\frac{size(\psi(S))}{K}\right), \quad a > 1 \tag{6}$$

Here K is an upper bound on the size of the tree (b is a constant, i.e. b=5)

$$K = \max\{\text{tree size in current population}\} + b \tag{7}$$

and we assign higher weight a to the size factor since our main goal is to create smaller and more easily interpreted decision trees. The optimization problem is thus to find the subset $S_{best} \subseteq T$ that maximizes (6) above. Our proposed methodology is to use a GA implementation to find a heuristic solution, $\widetilde{S}_{best}$, to this problem.

To evaluate the performance of an instance subset S using (6), the error rate, or equivalently the accuracy, of $\psi(S)$ must be estimated. We do this using a bootstrapping approach. Before the optimization starts, the original dataset is randomly divided into two separate training data T^* and test data D^*, then T^* is sampled with replacement $n = |T^*|$ times to generate the training set T, and the instances that are not selected form an independent test set $D = T^* \setminus T$. The chance that a particular instances will not be picked for the training set T is $\left(1-\frac{1}{n}\right)^n$ and $\lim_{n\to\infty}\left(1-\frac{1}{n}\right)^n = e^{-1}$, where e = 2.7183, the base of natural logarithms. Thus for a reasonably large dataset $\left(1-\frac{1}{n}\right)^n \approx e^{-1} = 0.368$ and the test set will contain about 36.8% of the instances, and the training set will contain about 63.2% of them, so the estimated error rate is given by (Efron, 1979)

$$\hat{e}(\psi(S)) = 0.632 \cdot e_D(\psi(S)) + 0.368 \cdot e_T(\psi(S)), \tag{8}$$

where $e_D(\psi(S))$ is the actual error when the tree $\psi(S)$ is applied to the test data *D*, and $e_T(\psi(S))$ is the error when it is applied to the training data *T*. Note that we have two different test datasets here, D^* and *D*. *D* is used to evaluate the decision trees generated within the GA algorithm, while D^* is used to validate the decision tree for the final GA solution, $\widetilde{S}_{best}$. This prevents possible bias from applying the decision trees to the same test set.

2.2.2 Solution Representation

The solution space must be defined in terms of what are called the chromosomes, which in most GA applications are binary strings. Here we take a slightly different approach and let g_i denote the position of the *i*th training instance in the training dataset *T*. The chromosomal unit of each subset is defined as a vector $C = [g_1, g_2, \cdots\cdots, g_N]$ of integers, where *N* is the number of training instances in a subset and represents the length of the chromosomal unit. Here the sequence of g_i does not have any influence on the GA process.

Note that one reason for not using the binary representation is that each instance would have a placeholder in the string, creating an intractable representation for large datasets. Another reason for why we do not use binary string is that the instances in each subset may be changeable (this is highly likely because of the GA operations). Therefore, with the use of integer chromosomes, each element in the integer vector chromosome can be easily replaced

by a specific training instance, and it is more natural and easier to implement the evolution process.

2.2.3 GA Operations

The GA search starts with an initial population $P_0 = \{C_1^{(0)}, C_2^{(0)}, ..., C_M^{(0)}\}$ of chromosomes that is selected as follows. Given $n = |T|$ instances we divide T into M subsets by sampling the training set N times without replacement to generate $C_1^{(0)}$, where $N = \lfloor n/M \rfloor$, and then repeating this process M times to generate the remaining subsets. Starting with this initial population, the usual GA operations of selection, crossover, and mutation are applied to improve the population. These operations are described as follows.

The most typical type of selection technique is called proportionate selection (e.g. roulette wheel selection) which is realized as a natural selection and can be defined as below (Davis 1991): In the kth generation, the individual $S(C_i^k)$ is selected by some probability:

$$P[S(C_i^k)] = \frac{f(S(C_i^k))}{Q \times Max[f(S(C_i^k))]}, i = 1,2,...,M \tag{9}$$

where $S\left(C_{[j]}^{(k)}\right) = \left\{\mathbf{x}_i : i \in C_{[j]}^{(k)}\right\}$ is the set of all instances in the subset (chromosome) $C_{[j]}^{(k)}$, j=1,2,…,M and Q is a constant, so the individual with higher fitness value is more likely to be chosen and $(1-c)M$ individuals are selected into the next generation, where c is the crossover rate. Another type of selection is ranking selection, that is, all individuals of current population are ranked according to their fitness values and the fittest ones will be selected into the next generation. The comparison between these two selection techniques is

listed in Appendix E (Table A18 and A19). It can be seen that roulette wheel selection generally obtains much smaller tree size. The reason is that by adding some random factors to selection, GA will have wider searching space and thus it may find better instances.

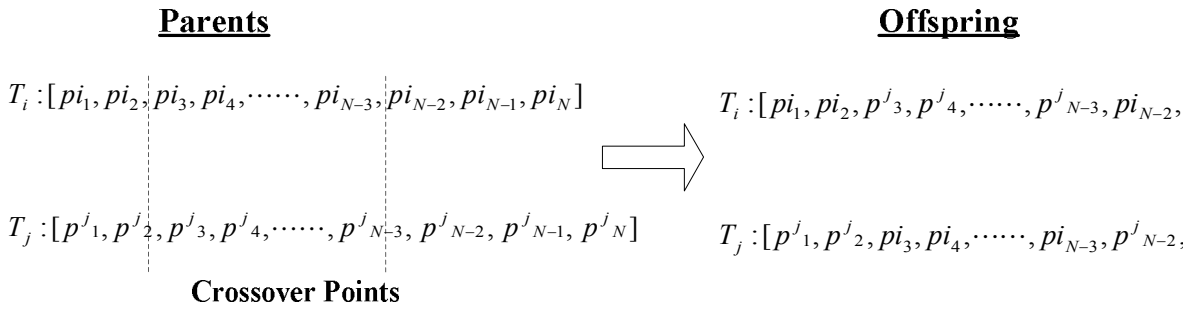
Crossover is an exploitation technique in which two chromosomes in a population exchange a portion of their genes. This allows variations to be introduced to the new population and a heuristic search for a new localized state space. There are different techniques for crossover. The simplest technique is one-point crossover (Davis 1991) that randomly picks a crossover point and swaps the segments to the right of this point between the two chromosomes. Two-point crossover is commonly used to allow a wider range of combinations. This technique selects two crossover points and the segment between the two crossover points in a chromosome is then swapped with the segment with the same position in the other mating chromosome. Another technique that allows all combinations of crossover is uniform crossover (Davis 1991). Individual bits in the chromosome are compared between two parents, then the bits are swapped with a fixed probability, typically 0.5. Comparing the three crossover techniques, one-point crossover has $(N-1)$ possible crossover combinations (where N is the gene length), two-point crossover has $\sum_{i=1}^{N-2} i$ possible combinations, and uniform crossover has 2^N possible combinations (Lam, 1994).

In the past several years, GA researchers have preferred either two-point or uniform crossover. Syswerda (1989) demonstrated that a uniform crossover outperforms a one-point

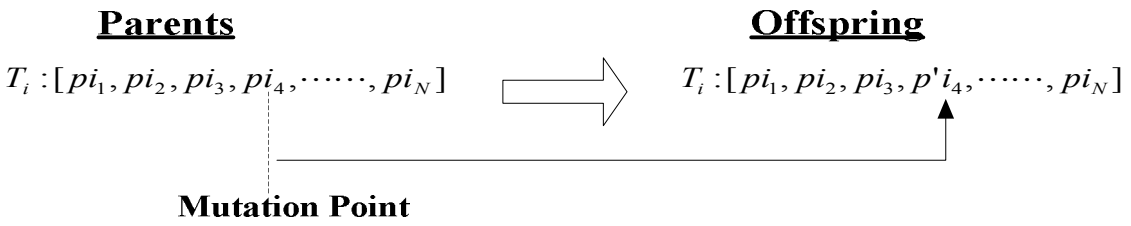
crossover and two-point crossover in maintaining population diversity to search for the global optimum. Spears and DeJong (1991) observed that two-point crossover converges more quickly, but to a lower plateau than uniform crossover which converges more slowly to a better solution. Their results show that uniform crossover is better than two-point crossover for smaller values of the gene length N and the population size M, but they note just the opposite effect as N and M increase. This suggests a way to understand the role of multi-point crossover. With smaller populations and shorter gene length, more disruptive crossover, such as uniform or n-point (n>>2) may yield better results because they help overcome the limited information capacity of smaller populations and shorter gene length and the tendency for more homogeneity. However, with larger populations and longer gene length, less disruptive crossover operators (two-point crossover) are more likely to work better, as suggested by the theoretical analysis (Levine, 1994). As we are dealing with large datasets in instance selection, the gene length tends to be large, so two-point crossover seems a good choice. In Appendix D (Table A15, A16 and A17), we compare the results with different crossover operations, and it can be seen that one-point crossover works well for multi-class problems when only the best subset is chosen, two-point crossover and uniform crossover are pretty close, but two-point crossover tends to obtain the decision trees with slightly higher accuracies and relatively small tree sizes. So we choose two-point crossover.

The proportion of the number of chromosomes involved in crossover operation over the total number of chromosomes is defined as the crossover rate. The crossover rate is problem

dependent and there is no specific method to determine it. Studies of crossover rate suggest that a higher crossover rate yields more chances for chromosomes with better fitness to crossover more than once and hence faster convergence. Further studies show a decreasing crossover rate as the population size increases. Some results have suggested M=50~100 and c=0.6 (DeJong and Spears, 1991), and M=80 and c=0.45 (Grefenstette, 1986) as good values for offline performance. We tried three different crossover rates (0.6, 0.7 and 0.8) and the results are listed in Appendix A (Table A5, A6, A7 and A8).



(a)



(b)

Figure 2 Operations of the genetic algorithm: (a) crossover (b) mutation

In summarization, the crossover operator in our algorithm probabilistically selects $cM/2$ pairs from P_k, chooses two random points and then swaps the parts between crossover points

among parent chromosomes (see Figure 2(a)). Meanwhile the instances corresponding to the position numbers will be exchanged between these two subsets.

Nevertheless, using crossover alone is not an effective search for a global optimum. Particularly, when the population converges, every chromosome in the population is similar to every other chromosome, and hence crossover becomes less productive. So mutation is introduced to complement the weakness of crossover. Mutation is an exploration technique that is used to introduce new values into a chromosome by randomly flipping selected bits. Young (1990) demonstrated that combining mutation and crossover significantly outperformed and more robust than using either mutation or crossover alone. In our algorithm, if the element g_i in the chromosomal unit is chosen to be mutated (the probability of mutation is determined by the mutation rate m), a new random number g_i' will be generated uniformly from $\{1,2,\ldots,|T|\}$ and then replaced for g_i (see Figure 2(b)). Furthermore, the instance in position g_i will be replaced by the new instance in position g_i'. Note that the mutation rate should be kept very low (usually about 0.001) as a high mutation rate will destroy fit strings and degenerate the GA algorithm into a random walk. But Tate and Smith (1993) argue that the optimal mutation rates depend strongly on the choice of encoding, and problems requiring non-binary encoding may benefit from mutation rates much higher than those generally used with binary encodings. Our algorithm uses integer encoding, so we tried three different mutation rates with relatively higher values (0.01, 0.05 and 0.09) and the results are also listed in Appendix A (Table A5, A6, A7 and A8).

2.2.4 Heuristic Solutions

The GA operations are repeated for a given number of G generations, resulting in a final population $P_G = \{C_1^{(G)}, C_2^{(G)}, ..., C_M^{(G)}\}$, which is ranked according to the fitness as in (6).

$$f(S(C_{[1]}^{(k)})) \geq f(S(C_{[2]}^{(k)})) \geq ... \geq f(S(C_{[M]}^{(k)})), \tag{10}$$

The heuristic solution $\widetilde{S}_{best}$ to the best instance subset is then found from this final population. One approach is to simply let it correspond to all of the instances that are contained in the top chromosome, that is,

$$\widetilde{S}_{opt}^{(best)} = \{\mathbf{x}_i : i \in C_{[1]}^{(G)}\}, \tag{11}$$

But this may miss some good instances and we therefore also consider selecting all instances that are contained in at least one of the top y% of subsets (chromosomes), that is,

$$\widetilde{S}_{opt}^{(y\%)} = \left\{\mathbf{x}_i : i \in \bigcup_{j=1}^{m} C_{[j]}^{(G)}, \text{ where } m = \left\lfloor M \cdot \frac{y}{100} \right\rfloor\right\}. \tag{12}$$

In the numerical experiments that follow we will consider $y \in \{25,50,75\}$.

2.3 Experiment Results

2.3.1 Experiment Setups

The basic premise of this research is that instance selection can be used to improve decision tree induction and that the genetic algorithm methodology presented in Section 2.2 is effective in achieving such improvements. Furthermore, we define improvements as reduction in the size of the decision tree, which should result in decision trees that are more

easily interpreted, while maintaining adequate prediction accuracy. In order to validate this premise, some numerical experiments were conducted involving five challenging test problems, four of which are taken from the Machine Learning Repository of the University of California at Irvine, and a scheduling problem adopted from Li and Olafsson (2004). The characteristics of these five datasets are described in Table 2. Also the histograms of these five datasets before and after instance selection are shown in Appendix F (Figure A1~A12).

Table 2 Test datasets

Dataset	Instances	Attributes	Classes	Description
Scheduling	7140	11	2	This dataset is meant to discover the scheduling rules, and represent the result that enables its use for job scheduling.
Sickness	3772	30	2	The objective is to predict whether a patient has thyroid disease.
Splice	3190	62	3	Splice junctions are points on a DNA sequence at which superfluous DNA is removed during the process of protein creation in higher organisms. The problem posed in this dataset is to recognize, given a sequence of DNA, the boundaries between exons (the parts of the DNA sequence retained after splicing) and introns (the parts of the DNA sequence that are spliced out).
Segment	2310	20	7	The instances were drawn randomly from a database of 7 outdoor images. The images were hand-segmented to create a classification. Each instance is a 3x3 region for every pixel.
Letter	20000	17	26	The objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes which were then scaled to fit into a range of integer values from 0 through 15.

All of the data mining algorithms, including the C4.5 decision tree induction algorithm, were implemented in the WEKA environment (Witten and Frank, 2000). The C4.5 algorithm was run using its default parameter settings in WEKA. It is of special interest to note that these settings include using the C4.5 pessimistic pruning with a subtree raising operation. Such pruning is essential to the construction of high quality decision trees. As an example, applying the C4.5 algorithm with the same parameter settings but without pruning to the "Splice" dataset results in a tree of size 3707 (number of nodes) with an estimated accuracy of 91.9% (estimated using 10-fold cross-validation). Adding pruning as described above results in a tree of size 229 with an estimated accuracy of 94.1%, that is, the size of the tree is reduced by an order of magnitude while the accuracy is increased. Similar results were found for the other datasets.

For the genetic algorithm settings the number of generations is fixed as G = 20, the crossover rate is c = 0.6, the mutation rate is m = 0.09, the constant in selection operation is Q=2.5, and the weight for fitness function is a=6. These parameter settings were selected as they appear to perform well for this task as shown in Appendix A (Table A1~A12). The number of subsets in each generation is allowed to vary $M \in \{5,10,15\}$. The original dataset is randomly divided into one training set T^* and one independent test set D^*, for some smaller datasets (n<3000), approximately 1/4 instances are randomly chosen in test set and for larger datasets, 1/3 instances are chosen. This holdout procedure for creating independent test set is commonly used in evaluating the classification.

Instance selection is applied to the training set T^*. The original decision tree from the entire data and the decision tree from the selected instances are both evaluated using the test data D^*. For each experimental setting, ten replications are made and both the average and standard error are reported. The whole process of the experiment is illustrated in Figure 3.

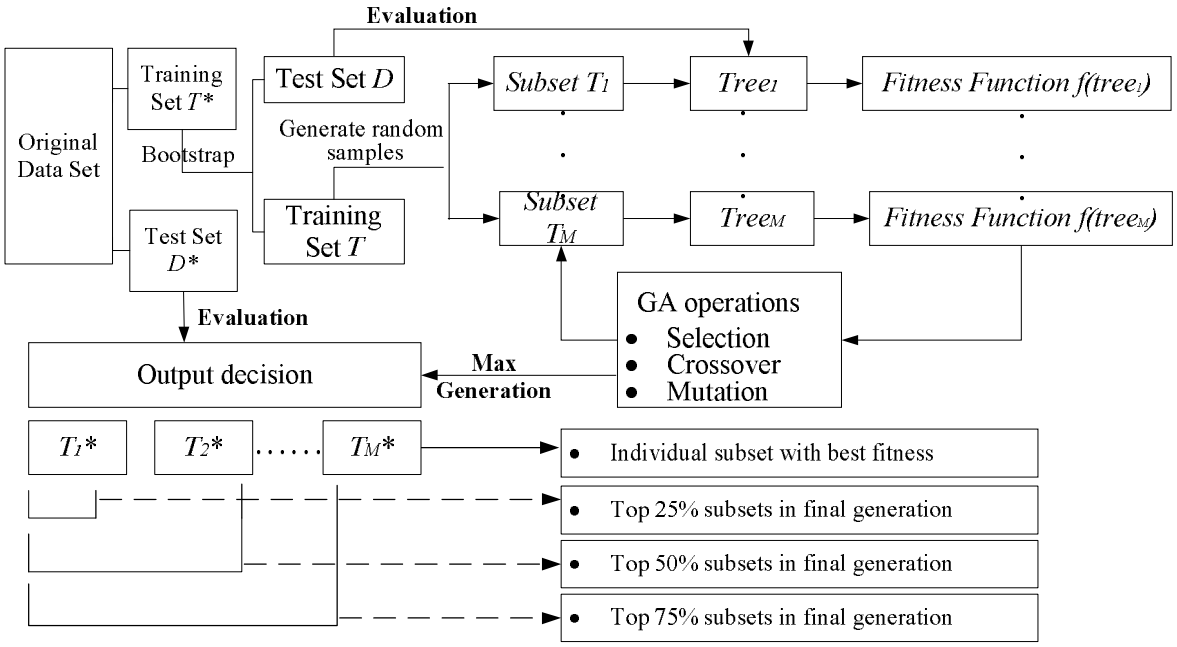


Figure 3 Design of the experiment on GA-based instance selection

2.3.2 Effectiveness of GA-based Instance Selection

We start with only selecting the best instance subset at the end of the GA run, that is, equation (11) is used to select the heuristic solution $\tilde{S}_{best}$ to the instance selection problem. Table 3 compares the original decision trees from the entire data with the decision trees from the selected instances when the number of subsets is set to the average number (M=10).

Table 3 Results for best subset

Dataset			With Instance Selection		
		Original	Avg.	S.E.	Change
Scheduling	Accuracy	99.6	96.3	0.6	3.3%
	Tree size	69.0	8.6	1.7	87.5%
	ALR	2.9	22.6	3.5	679.3%
Splice	Accuracy	94.1	60.3	11.6	35.9%
	Tree size	229.0	14.4	16.5	93.7%
	ALR	0.7	65.0	50.0	9185.7%
Segment	Accuracy	96.9	88.4	1.2	8.8%
	Tree size	77.0	17.0	1.4	77.9%
	ALR	2.3	11.8	0.8	413.0%
Letter	Accuracy	88.0	62.4	2.5	29.1%
	Tree size	2451.0	302.2	22.6	87.7%
	ALR	0.1	0.7	0.1	600.0%
Sick	Accuracy	98.8	95.3	1.2	3.5%
	Tree size	61.0	3.8	2.4	93.7%
	ALR	4.8	61.2	35.4	1175.0%

We observe that the reduction in the size of the decision tree ranges from 77.9% for the "Segment" dataset to 93.7% for the "Splice" dataset. Substantial reductions in the size of the decision tree are therefore obtained for all of the datasets. Meanwhile the reduction in accuracy is less than 10% for three of the five datasets. For two datasets the accuracy loss is clearly unacceptable ("Letter" dataset and "Splice" dataset), for one dataset the loss may be considered marginal ("Segment" datasets), and for two datasets the loss is relatively minor ("Scheduling" and "Sick" datasets). We note that the significant reduction in size of the decision trees is in addition to the reduction already achieved through pruning the tree (e.g., for the "Splice" dataset from 3707 for an unpruned tree, to 229 for a pruned tree and an average of 14.4 for a pruned tree with instance selection).

Varying certain key parameters of the GA algorithm may be expected to affect the performance of the approach, and in Table 4 the results from varying the number of subsets $M \in \{5,10,15\}$ are reported.

Table 4 Results when number of subsets (*M*) is varied

Dataset		*M*=5		*M*=10		*M*=15	
		Avg.	S.E.	Avg.	S.E.	Avg.	S.E.
Scheduling	Accuracy	97.4	0.6	96.3	0.6	94.3	0.7
	Tree size	17.4	2.3	8.6	1.7	3.8	2.4
	ALR	12.3	1.8	22.6	3.5	59.9	35.0
Splice	Accuracy	82.9	3.2	60.3	11.6	52.9	1.6
	Tree size	66.8	7.4	14.4	16.5	1.0	1.8
	ALR	2.1	0.2	65.0	50.0	93.1	2.0
Segment	Accuracy	92.6	1.1	88.4	1.2	82.8	3.3
	Tree size	22.2	2.0	17.0	1.4	13.4	1.0
	ALR	9.5	0.9	11.8	0.8	14.4	0.7
Letter	Accuracy	71.1	3.7	62.4	2.5	54.8	1.7
	Tree size	507.8	73.2	302.2	22.6	210.6	7.9
	ALR	0.5	0.1	0.7	0.1	1.0	0.1
Sick	Accuracy	97.7	0.9	95.3	1.2	94.2	2.1
	Tree size	6.6	0.8	3.8	2.4	2.2	1.2
	ALR	29.3	3.6	61.2	35.4	73.5	26.3

In order to find the changing trends of accuracy, tree size and ALR with different number of subsets *M*, we conducted a set of statistical two sample *t*-tests as shown in Appendix C (Table A14). The two-sample *t*-test (Snedecor and Cochran, 1989) is applied in testing the hypothesis concerning differences between the means of two populations, that is, we want to test the null hypothesis $u_1 - u_2 = \delta$, where δ is a given constant, against one of the alternatives $u_1 - u_2 \neq \delta$, $u_1 - u_2 > \delta$ or $u_1 - u_2 < \delta$. Suppose that we are dealing with independent random samples of size *m* and *n* from two normal populations and *m* and *n* are small (*m*<30 and *n*<30) with unknown variances $\sigma_1 \neq \sigma_2$, we have:

$$t = \frac{\overline{x_1} - \overline{x_2} - \delta}{\sqrt{\frac{s_1^2}{m} + \frac{s_2^2}{n}}} \quad \text{and} \quad v = \frac{\left(\frac{s_1^2}{m} + \frac{s_2^2}{n}\right)^2}{\frac{\left(s_1^2/m\right)^2}{m-1} + \frac{\left(s_2^2/n\right)^2}{n-1}} \tag{13}$$

where $\overline{x_1}$ and $\overline{x_2}$ are the means of the two samples and s_1 and s_2 are the standard deviations. This expression for t is a value of a random variable having the t-distribution with v degrees of freedom. Thus, the appropriate critical regions of size a for testing the null hypothesis $u_1 - u_2 = \delta$ against the alternatives $u_1 - u_2 \neq \delta$, $u_1 - u_2 > \delta$ or $u_1 - u_2 < \delta$ under the given assumptions are, respectively, $|t| \geq t_{\alpha/2,v}$, $t \geq t_{\alpha,v}$, and $t \leq -t_{\alpha,v}$.

It can be quickly noted that as a rule from the statistical significance results in Appendix C (Table A14) that increasing M results in smaller trees that also have lower accuracy and higher ALR. This is quite intuitive. If the original training data is divided into more subsets, the number of instances in each subset will decrease, and with less training instances the decision trees will tend to be smaller but less accurate and the average leaf ratio will be larger. The best value of M appears to be application dependent. For the "Scheduling" and "Sick" datasets the smallest decision trees are obtained at M = 15 with only a minimal degradation in accuracy. For the "Splice" and "Segment" datasets, however, using M=5 results in significantly smaller trees on the average with relatively little loss in accuracy, while increasing M leads to substantial degradation in accuracy for more marginal reduction in the size of the tree. Another factor in the selection of M is the computation time, which will be discussed next.

The computational overhead of the instance selection is an important measure for the performance of our approach. The computation time in seconds is shown in Table 5 as a function of the number of subsets used (*M*) and the number of GA generations (*G*). In order to view the trends more intuitively, the computation time in seconds is also shown in Figure 4 and Figure 5 for four datasets except "letter" data because of the scales of the values.

Table 5 Calculation time with different *M* and *G*

Dataset	M	Calculation time (*G*=20)	G	Calculation time (*M*=10)
Scheduling	5	29.3 sec	20	47.9 sec
	10	47.9 sec	30	67.4 sec
	15	72.3 sec	40	85.8 sec
Splice	5	53.3 sec	20	55.7 sec
	10	55.7 sec	30	65.0 sec
	15	72.7 sec	40	74.0 sec
Segment	5	25.3 sec	20	34.4 sec
	10	34.4 sec	30	48.1 sec
	15	46.9 sec	40	64.6 sec
Letter	5	215.5 sec	20	329.0 sec
	10	329.0 sec	30	641.8 sec
	15	541.3 sec	40	974.4 sec
Sick	5	24.9 sec	20	37.8 sec
	10	37.8 sec	30	53.8 sec
	15	54.6 sec	40	71.1 sec

For scheduling data, as an example, the time it takes to solve the problem is on the scale of approximately half of a minute to about 2 minutes, depending on the setting of *M* and the number of GA generations (*G*).

Figure 4 Computation time with different number of generations (*G*)

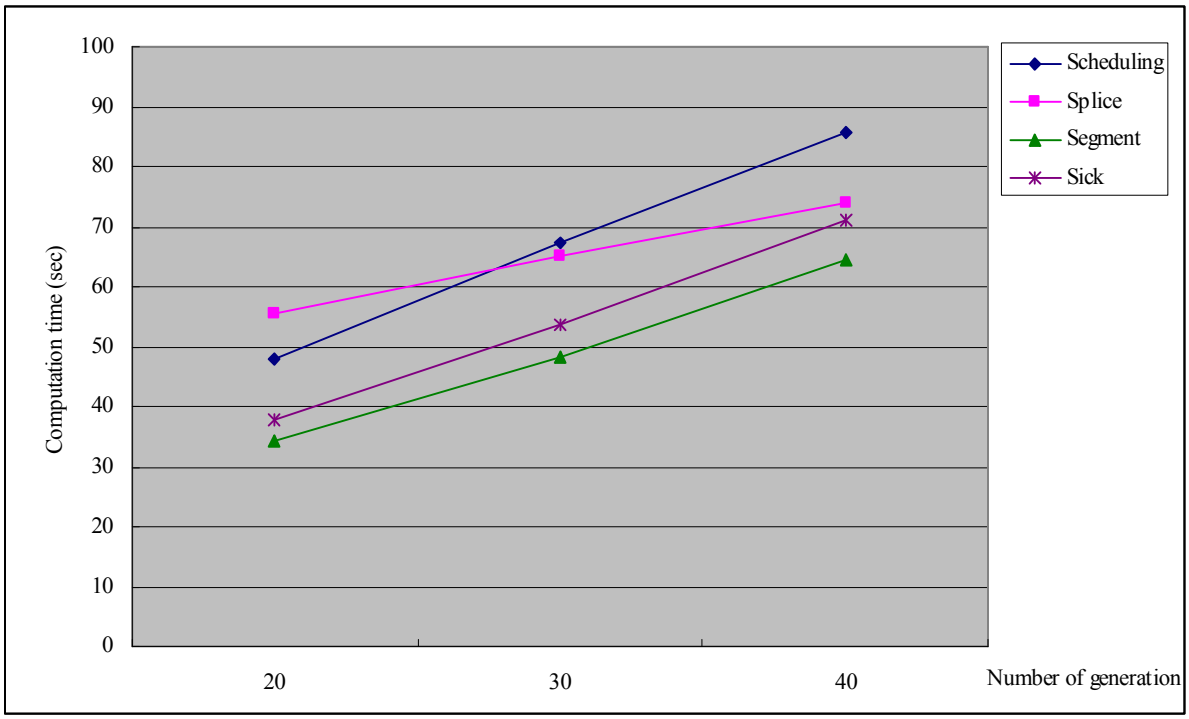
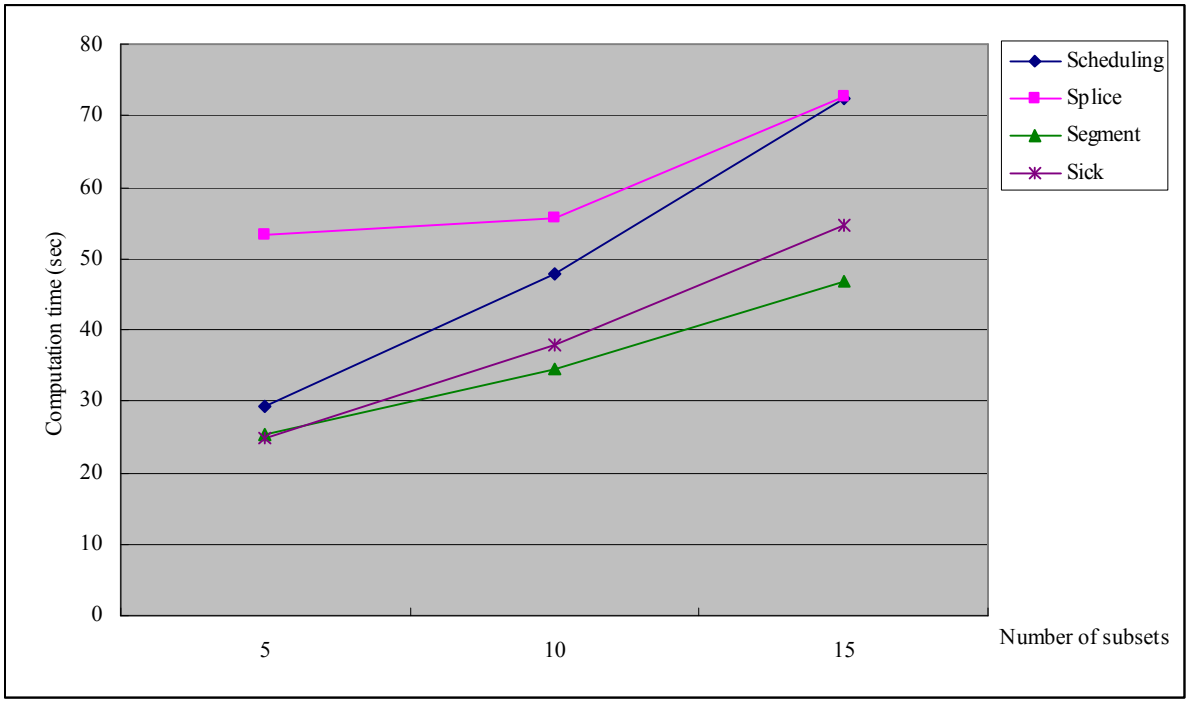


Figure 5 Computation time with different number of subsets (*M*)



As expected, the computation time increases in both variables as shown in Figure 4 and 5, with a linear growth in the number of generations G and an apparently faster growth in the number of subsets M. This is intuitive, since more subsets imply larger space for the GA algorithm to explore, and hence it can be expected that this is one of the main factors determining the computation time. In particular, the computation time is quite similar for M = 5 and M = 10, but if M = 15, the computation time will increase obviously, especially for the larger dataset. Therefore, it is recommended to choose M as relatively small, e.g. $M \leq 10$. In our following experiments, M is set as 10.

Another factor that will affect the final results is the part of the final GA population that is selected to induce the final decision tree. In the results reported above we simply select the best subset and use only this subset, that is, $\widetilde{S}_{best} = \widetilde{S}_{best}^{(opt)}$ according to equation (11). However, as noted above it might be beneficial to use instances that are included in several of the best subsets. Table 6 shows the results for the decision trees when the instances are varied from just the best subset to all the instances that are contained in the top y% of subsets, that is, $\widetilde{S}_{best} = \widetilde{S}_{best}^{(y\%)}$ according to equation (12), where $y \in \{25,50,75\}$.

Table 6 Results when fraction of selected instances is varied

Dataset		Best		Top 25%		Top 50%		Top 75%	
		Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.
Scheduling	Accuracy	96.3	0.6	98.0	0.5	98.6	0.4	98.8	0.4
	Tree size	8.6	1.7	27.8	3.8	41.0	3.5	45.4	4.2
	ALR	22.6	3.5	7.1	0.9	4.8	0.4	4.3	0.4
Splice	Accuracy	60.3	11.6	85.7	2.0	91.2	1.3	92.8	0.8
	Tree size	14.4	16.5	71.8	19.0	116.2	9.3	146.2	6.7
	ALR	65.0	50.0	1.9	0.5	1.1	0.1	0.9	0.0
Segment	Accuracy	88.4	1.2	92.6	0.8	94.8	1.3	95.2	0.4
	Tree size	17.0	1.4	33.4	3.3	47.8	2.3	53.4	4.7
	ALR	11.8	0.8	5.9	0.6	4.1	0.2	3.7	0.3
Letter	Accuracy	62.4	2.5	73.9	0.8	80.2	0.6	82.3	0.4
	Tree size	302.2	22.6	623.0	7.9	1071.2	29.7	1244.3	15.6
	ALR	0.7	0.1	0.3	0.0	0.2	0.0	0.2	0.0
Sick	Accuracy	95.3	1.2	98.0	0.7	98.5	0.5	98.6	0.4
	Tree size	3.8	2.4	12.4	2.4	15.6	4.5	23.4	5.6
	ALR	61.2	35.4	14.2	3.3	11.6	4.6	7.5	1.5

The statistical significance results in Appendix C (Table A14) show that as y is increased, the average accuracy improves but the tree size increases as well, moreover, the average ratio drops. These results are not unexpected, as more instance are included the accuracy increases but the size of the decision tree will also grow. Therefore, sometimes it is beneficial to use an aggregation of subsets instead of just using the best one as the selected instance subset, that is, $\widetilde{S}_{best} = \widetilde{S}_{best}^{(y\%)}$ in order to obtain higher accuracy. It should be noted, however, that the number of distinct instances in these subsets is typically much less than the total size of the subsets. This can be observed from Table 7, which show the number of distinct instance in $\widetilde{S}_{best} = \widetilde{S}_{best}^{(y\%)}$ for $y \in \{25,50,75\}$. We also note that $y = 50$ seems to work well for these test

problems, since this results in decision trees with high accuracy (that is, close to the accuracy of the trees induced from original dataset) that are also relatively small.

Table 7 Distinct instances in the output

Dataset	M	Fraction of Selected Instances (G=20)		
		Top 25%	Top 50%	Top 75%
Scheduling	5	857.0	1564.2	2146.6
	10	860.1	1865.1	2368.5
	15	859.5	1775.8	2492.3
Splice	5	384.0	707.5	968.6
	10	381.2	837.4	1079.2
	15	384.3	789.5	1103.8
Segment	5	316.7	574.2	783.5
	10	314.9	683.3	873.2
	15	311.8	643.6	900.3
Letter	5	2401.5	4377.0	5996.1
	10	2426.2	5272.3	6735.0
	15	2419.0	4978.6	6913.8
Sick	5	456.1	825.8	1126.6
	10	456.3	986.2	1262.7
	15	451.8	928.1	1301.5

It should be noted that while the GA optimization algorithm is always able to reduce the size of the decision trees, the quality of the trees is very much application dependent. For example, we note from Table 3 that for the "Scheduling" dataset it is possible to find a decision tree using only 10% of the instances where the accuracy is almost as good as the decision tree designed directly with all training instances, but the average size of the tree is 8.6 versus 69.0 nodes. Similarly, the approach performs very well for the "Sick" dataset. One characteristic that these dataset have in common is that the class attribute only takes two values. On the other hand, our approach performs much worse for the "Letter" dataset where

the class attribute has 26 possible values. For these problems, it is beneficial to use an aggregation of subsets instead of just using the best one as the selected instance subset, that is, $\widetilde{S}_{best} = \widetilde{S}_{best}^{(y\%)}$. It appears reasonable to speculate that more class values, more data instances are needed to induce good decision trees. We will provide further discussions in 2.3.4.

It is interesting to explore further on the average leaf ratio, since this measure provides some insights of the benefits from instance selection. The values of ALR before and after instance selection are compared in Table 8.

Table 8 Results on ALR before/after instance selection

Dataset	Original data		Instance selection		
	$\|T\|$	$R(\psi(T))$	Output	Average $R(\psi(S))$	**Change**
Letter	13333	0.1	Best	0.7	**600.0%**
			y=25	0.3	**200.0%**
			y=50	0.2	**100.0%**
			y=75	0.2	**100.0%**
Splice	2126	0.7	Best	65.0	**9185.8%**
			y=25	1.9	**171.4%**
			y=50	1.1	**57.1%**
			y=75	0.9	**28.6%**
Segment	1732	2.3	Best	11.8	**413.0%**
			y=25	5.9	**156.5%**
			y=50	4.1	**78.2%**
			y=75	3.7	**60.9%**
Scheduling	4760	2.9	Best	22.6	**679.3%**
			y=25	7.1	**144.8%**
			y=50	4.8	**65.5%**
			y=75	4.3	**48.3%**
Sick	2514	4.8	Best	61.2	**1175.0%**
			y=25	14.2	**195.8%**
			y=50	11.6	**141.7%**
			y=75	7.5	**56.3%**

From Table 8, it can be seen that the greatest improvement in Average Leaf Ratio occurs in the best subset, the change is from 413.0% for "Segment" data to 9185.8% for "Splice" data. However, with more instances included in the final output, the improvement in ALR drops quickly, when y=75%, the change is less than 100%. Therefore, with more instances in the output, there will be less benefit in ALR from instance selection. Furthermore, as noted above, for the multi-class problems such as "Letter" and "Splice" data, we need to use the aggregation of the subsets to obtain good accuracy, so for these problems, even though instance selection is still able to reduce the tree's size and achieve acceptable accuracy, it does not act as well as the two-class problems for which the best subset is good enough.

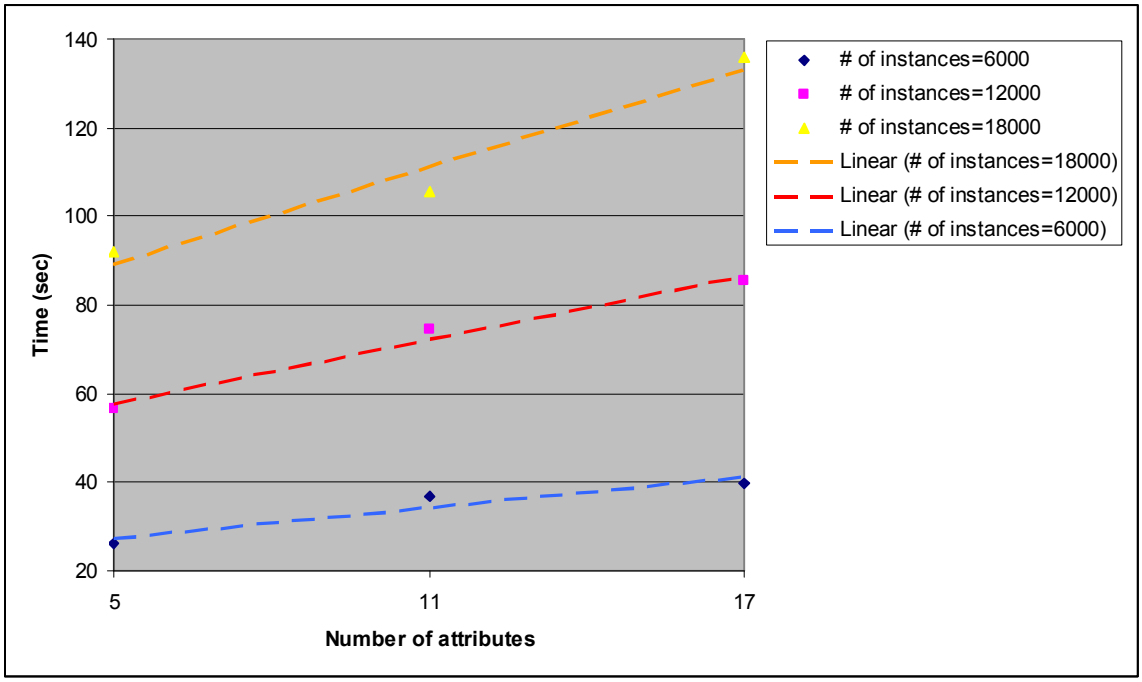
2.3.3 GA-Based Instance Selection for Large Datasets

The application of instance selection to large datasets is essential in evaluating the value of this approach. To investigate the performance of the genetic algorithm for optimal instance selection in different scenarios (i.e., different number of attributes and different number of instances), we will for the purpose of limiting repetition focus only on the "Letter" and "Scheduling" problem. We note, however, that similar observations hold for the other test problems. Here the most important measure is computation cost since one concern is that the GA process may be slow in dealing with large datasets.

Figure 6 shows the computation time with different number of attributes randomly selected from the "Letter" data and the number of attributes is varied from 5 to 17.

Meanwhile, for each setting, we tested different number of instances. For example, when the number of attributes is 5, the number of randomly selected instances is varied from 6000 to 18,000. As expected, the computation time grows with more attributes. And the growth of computation time with different number of attributes is approximately linear when the number of instances is less than 12,000, but as the number of instance reaches 18,000, the growth of computation time is much faster.

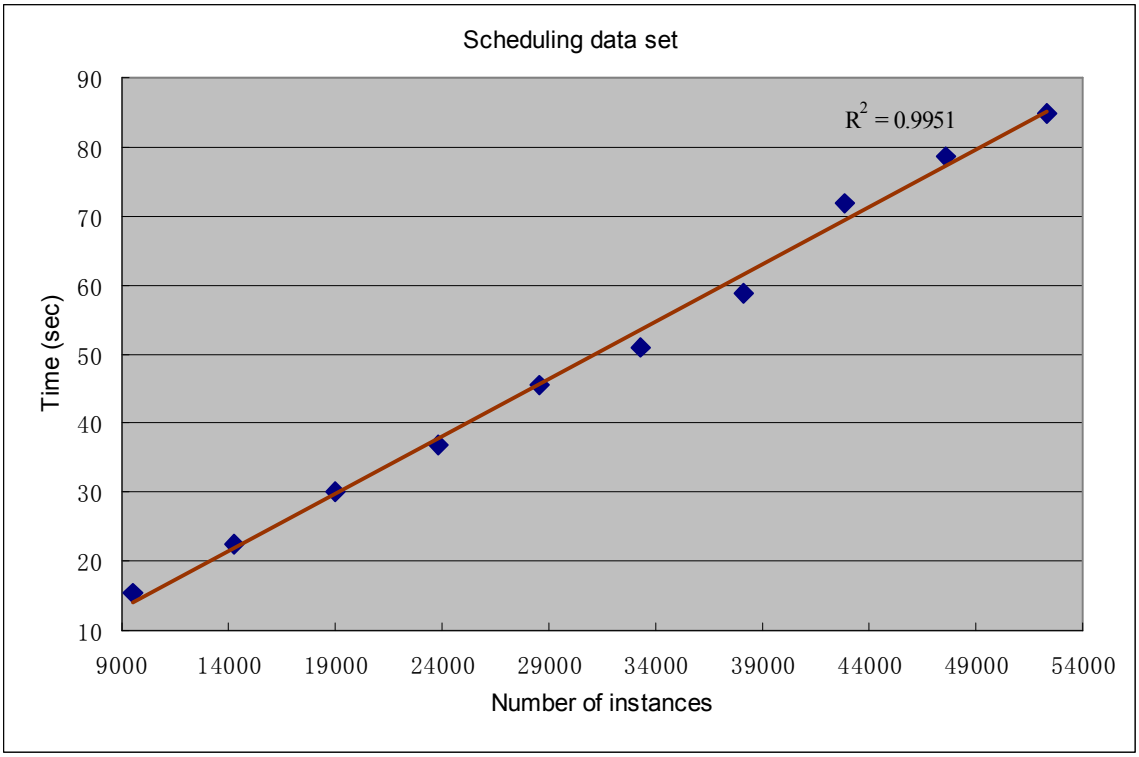
Figure 6 Computation time with different number of attributes



In order to verify the performance of our approach under some extreme situation, we use "Scheduling" data as an example. An attractive property of using the "Scheduling" dataset for this test of computation time is that we are able to generate any number of instances for this dataset while maintaining the structure. Figure 7 shows the effect increasing the number of the instances has on the computation time. It can be seen that increasing the number of

instances will increase the computation time, but this increase still appears to occur at a linear rate with the value of R^2 close to 1. So our approach is capable of handling large datasets with acceptable computation cost.

Figure 7 Computation time with different number of instances



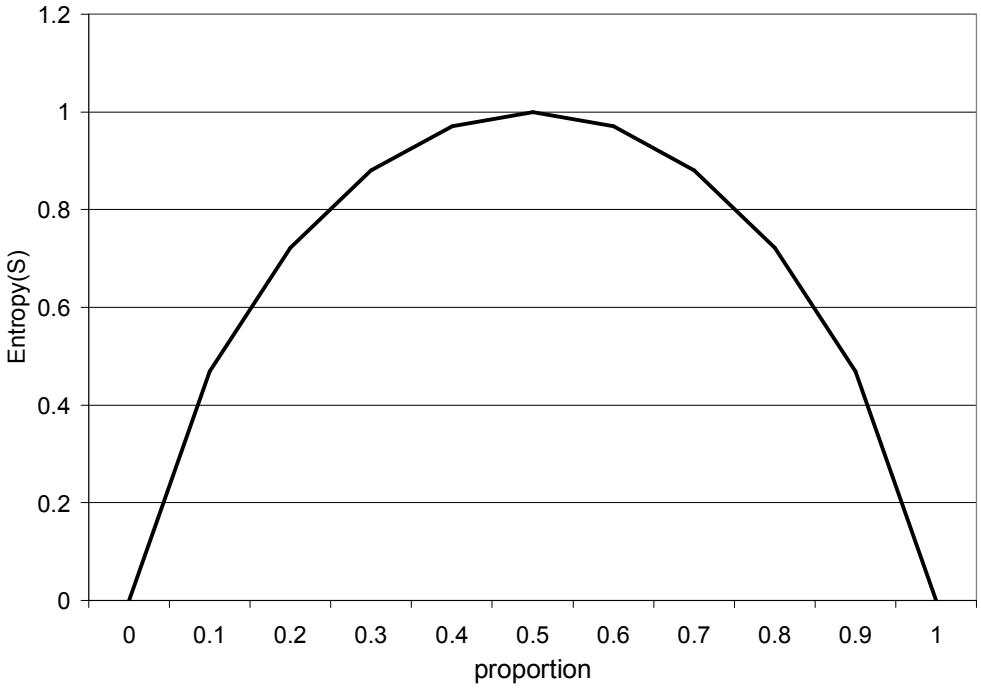
2.3.4 The Influence of the Instance Entropy on Instance Selection

From previous sections, we know that the number of class values has close relationship with the performance of instance selection. In this section, we will explore this further to investigate what is the major factor that influences the instance selection. Here we will use the measure of instance entropy $Entropy(S)$ which is defined as:

$$Entropy(S) = \sum_{i=1}^{c} - p_i \log_2 p_i \qquad (14)$$

where c is the number of classes and p_i is the proportion of dataset S classified as class i. Note that the entropy is 0 if all members of S belong to the same class and the entropy is maximum when all classes are equally likely. Figure 8 shows the form of the entropy function relative to a boolean classification (c=2), as the proportion of one class varies between 0 and 1.

Figure 8 The entropy function relative to a boolean classification



One interpretation of entropy from information theory is that it specifies the minimum number of bits of information needed to encode the classification of an arbitrary member of S (Mitchell, 1997). So $Entropy(S)$ provides a measure of impurity in the dataset S. The higher the entropy is, the more diversity the data contains. Therefore, it is obvious that multi-class problem will have larger entropy than two-class problem. Next we will show that with higher

entropy, that is, with more impurity in the data, there will be less benefit from instance selection. Here we use top 25% subsets, since the results of top 25% subsets are more stable than those of the best subset with less deviation in ALR, accuracy and tree size. From Table 9, it can be seen that though instance selection is able to obtain high ALR improvement and significant tree size reduction for multi-class problems, the decrease in accuracy is much higher than two-class problems, so in order to describe more diversity in multi-class problem, we will need more instances to induce good decision tree with acceptable accuracy.

Table 9 The influence of entropy on different datasets

Dataset	Number of class values	Entropy	ALR improvement for Top 25% subset	Accuracy decrease for Top 25% subset	Tree size decrease for Top 25% subset
Letter	26	4.70	200.0%	16.0%	74.6%
Splice	3	1.48	174.4%	8.9%	68.6%
Segment	7	2.81	156.5%	4.4%	56.6%
Scheduling	2	0.32	144.8%	1.6%	59.7%
Sick	2	0.33	195.8%	0.8%	79.7%

Furthermore, it is interesting to investigate for a specific problem, how entropy will influence the performance of instance selection. Here we resample the instances to get different entropy values. The resample bias determines whether to use bias towards a uniform class. A value of 0 leaves the class distribution as it is, while a value of 1 ensures the class distribution is uniform in the output data. Considering that with more class values, there will be less difference in the entropy values out of resampling, for example, the entropy of the "Segment" dataset with seven class values is around 2.81 no matter how the resample bias is changed, our experiments are only focused on "Sick", "Scheduling" and "Splice" data with no more than three class values. Table 10 and 11 list the results from different entropies.

Table 10 The influence of entropy on two-class problem (scheduling and sick datasets)

Resample Bias	Entropy	ALR improvement for Top 25% subset (higher is better)		Accuracy decrease for Top 25% subset (lower is better)		Tree size decrease for Top 25% subset (higher is better)	
		Scheduling	Sick	Scheduling	Sick	Scheduling	Sick
0	0.33	144.8%	195.8%	1.6%	0.8%	59.7%	79.7%
0.2	0.60	77.1%	180.4%	1.6%	1.8%	53.9%	76.7%
0.4	0.79	70.0%	178.9%	1.6%	2.1%	51.0%	74.4%
0.6	0.91	67.7%	124.4%	1.6%	2.1%	49.3%	68.8%
0.8	0.98	66.7%	102.1%	1.9%	2.4%	44.7%	67.9%
1.0	1.00	66.4%	93.8%	2.1%	2.9%	44.0%	65.6%

Table 11 The influence of entropy on multi-class problem (splice dataset)

Resample Bias	Entropy	ALR improvement for Top 25% subset (higher is better)	Accuracy decrease for Top 25% subset (lower is better)	Tree size decrease for Top 25% subset (higher is better)
0	1.48	174.4%	8.9%	68.6%
0.2	1.51	164.3%	8.8%	62.0%
0.4	1.55	157.1%	8.8%	61.6%
0.6	1.56	145.7%	9.0%	61.6%
0.8	1.58	104.3%	9.1%	61.3%
1.0	1.59	102.9%	9.5%	58.0%

Note that it is better to have larger increase in ALR, less decrease in accuracy, and more reduction in tree size. It can be seen from Table 10 and 11 that with higher entropy in the data, there will be less improvement in the average leaf ratio and less reduction in the tree size as well as higher degradation in the classification accuracy. So generally speaking, higher entropy in the data will lead to less benefit from instance selection.

CHAPTER 3. HEURISTIC INSTANCE SELECTION

The goal of selecting an optimization algorithm for solving problems such as the instance selection problem is to find algorithms with provably low computation cost and good or optimal solution quality. However, sometimes these goals cannot be achieved simultaneously and therefore a heuristic is proposed as an algorithm that gives up one or both of these goals. For example, it may find pretty good solutions, but the search process is time consuming. On the other hand, some heuristic runs reasonably fast, but it may produce bad results. For many practical problems including instance selection problem, when finding the exact optimal solution is impossible, heuristic algorithm may be the only way to get good solutions in a reasonable amount of time.

3.1 Greedy Heuristic Method

A greedy heuristic method solves an optimization problem by finding locally optimal solutions. The algorithm is called "greedy" because it always takes the best immediate, or local, solution while finding an answer. As for our instance selection problem, using enumeration will find the optimal solution, but it will need to run 2^n steps (n is the number of instances in original dataset T) to check all possible candidates since for each instance there will be two choices: select it or not. For some really small datasets (n<50), it is possible to use enumerative method, but as the number of instances increases, the computational cost

will grow up dramatically, which makes enumeration infeasible. Also branch-and-bound cannot be applied to this problem as discussed in Section 1.2. Therefore, greedy heuristic method may be a good way to obtain good solution. Here we have developed a new 2-phase *Rmhc* heuristic method for instance selection problem. The original Random Mutation Hill Climbing method (Mitchell et al., 1992) contains four steps as following:

1. Choose a candidate at random. Call this "best-evaluated".
2. Choose a locus at random to flip. If the flip leads to an equal or higher objective value, then set "best-evaluated" to the resulting solution.
3. Go to step 2 until an optimal solution has been found or a maximum number of evaluations has been performed.
4. Return the current value of "best-evaluated".

Rmhc is a greedy heuristic since it always tried to find better objective value. As found by Mitchell et al. (1992), *Rmhc* outperforms genetic algorithm in some difficult optimization problems. We follow some basic ideas of *Rmhc* method and the main novelty of our method is trying to decompose the original optimization problem instead of trying to obtain the optimal solutions for the two objectives (minimizing the tree size and the instance subset size) simultaneously. As is shown in Figure 9, in Phase I, we start with randomly selecting a subset *S* from original dataset *T* to satisfy the accuracy constraint, that is, the decision tree built from this subset *S* should have good accuracy compared with the tree built from entire data *T*. This subset is the initial feasible solution. In each following iteration, the algorithm

interchanges an instance from *S* with another from *T/S*. The change is accepted if it offers smaller tree size and satisfies the accuracy constraint. Then in Phase II, we will gradually reduce the subset's size without deteriorating the tree's size and accuracy. The backward selection continues until no instance can be deleted from the current subset.

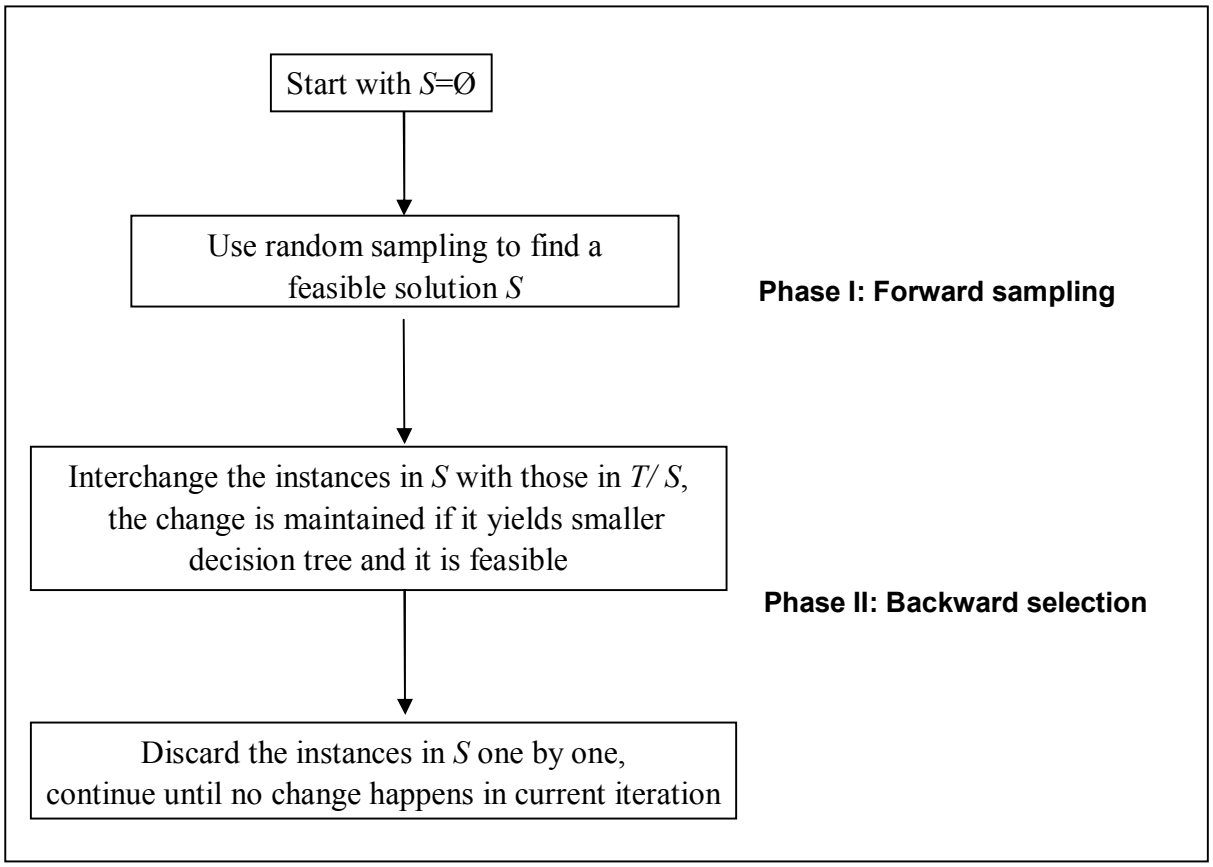


Figure 9 Main steps of 2-phase *Rmhc*

Previous research on applying *Rmhc* to instance selection problem normally stops after Phase I, while our 2-phase *Rmhc* continues in Phase II to search for smaller instance subset to obtain better solution. Some major steps of 2-phase *Rmhc* are described below:

Phase I: Forward sampling

1) Initialization: $S = \varnothing$, best instance subset $S_0 = \varnothing$, best tree size s_0=0, initial subset size p=0;

2) Sample without replacement to generate subset S;

3) Check the accuracy of the decision tree from subset S to verify if it is a feasible solution: *If* $1 - \hat{e}(\psi(S)) \geq [1 - \hat{e}(\psi(T)] \times (1 - \varepsilon)$, the solution is feasible, let $p = |S|$, S_0=S, and $s_0 = size(\psi(S))$, go to step 4; else, go back to step 2 and acquire more samples;

4) Interchange the instances in S with those out of S one at a time: Denote $I_i(S)$ as the ith instance in S and $I_j(S^-)$ as the jth instance in T/S. Let $I_i(S) = I_j(S^-)$, $i = 1,2,...,p, j = 1,2,...,|T| - p$;

5) Accept the interchange if $size(\psi(S)) < s_0$ and $1 - \hat{e}(\psi(S)) \geq [1 - \hat{e}(\psi(T)] \times (1 - \varepsilon)$, update $S_0 = S$ and $s_0 = size(\psi(S))$, go back to step 4; else reject the interchange, restore $S = S_0$, go back to step 4.

Phase II: Backward selection

6) Discard the instances in S one by one, that is, remove $I_i(S)$ from S, i=1,2,…,p;

7) Accept the deletion if $size(\psi(S)) \leq s_0$ and $1 - \hat{e}(\psi(S)) \geq [1 - \hat{e}(\psi(T)] \times (1 - \varepsilon)$, update $p = |S|$, $S_0 = S$ and $s_0 = size(\psi(S))$, go back to step 6; else reject the deletion, restore $S = S_0$, go back to step 6;

8) Continue until no changes happen in current iteration, output s_0 and S_0.

This heuristic method only searches part of the whole solution space so it does not guarantee finding the optimum. In Phase I, it runs $p \times (n-p)$ steps. In phase II, under the best situation, that is, no instance is deleted at the first iteration, this method only needs to run p steps. Even considering the worst situation (only one instance is discarded in each iteration), it runs $p + (p-1) + \ldots + 1 = \frac{p \times (p+1)}{2}$ steps. So the total search steps of 2-phase *Rmhc* are reduced from 2^n in enumeration to $[\, p \times (n-p) + p,\ p \times (n-p) + \frac{p \times (p+1)}{2}\,]$, which converts the original NP-hard problem to a solvable polynomial problem, but it can be expected that this 2-phase *Rmhc* method is only good for median datasets, since for large datasets (n>5000), the computation cost is still too high.

In order to increase the speed of 2-phase *Rmhc* method, it can be modified for individual subset instead of individual instance. For the subset-based *Rmhc* method, the original dataset T is randomly divided into M subsets in Phase I, and these subsets are sampled without replacement to generate the initial feasible solution containing P subsets. In each following iteration, the algorithm interchanges a subset from S with another from T/S instead of interchanging a single instance. Similarly, the change is accepted if it offers smaller tree size and satisfies the accuracy constraint. Then in Phase II, the backward selection continues until no subset can be deleted. Thus the search steps of subset-based *Rmhc* method are $\left[P \times (M-P) + P,\ P \times (M-P) + \frac{P \times (P+1)}{2} \right]$. Since the values of P and M are much smaller than the values of p and n, the modified subset-based *Rmhc* method should run much faster

than the original point-based *Rmhc* method. We will compare these two approaches in Section 3.3.

3.2 Simple Construction Heuristic

The main idea of simple construction heuristic is starting with no instances selected and then trying to add instance subsets gradually based on the value of the objective function, which is defined as:

$$f(S) = \log\left(1 - \hat{e}(\psi(S))\right) + a\log\left(\frac{1}{size(\psi(S))}\right), \quad a > 1. \tag{15}$$

The objective function here is similar to the fitness function used in genetic algorithm, which is also the combination of the tree's accuracy and its size. Figure 10 shows some major steps of this approach.

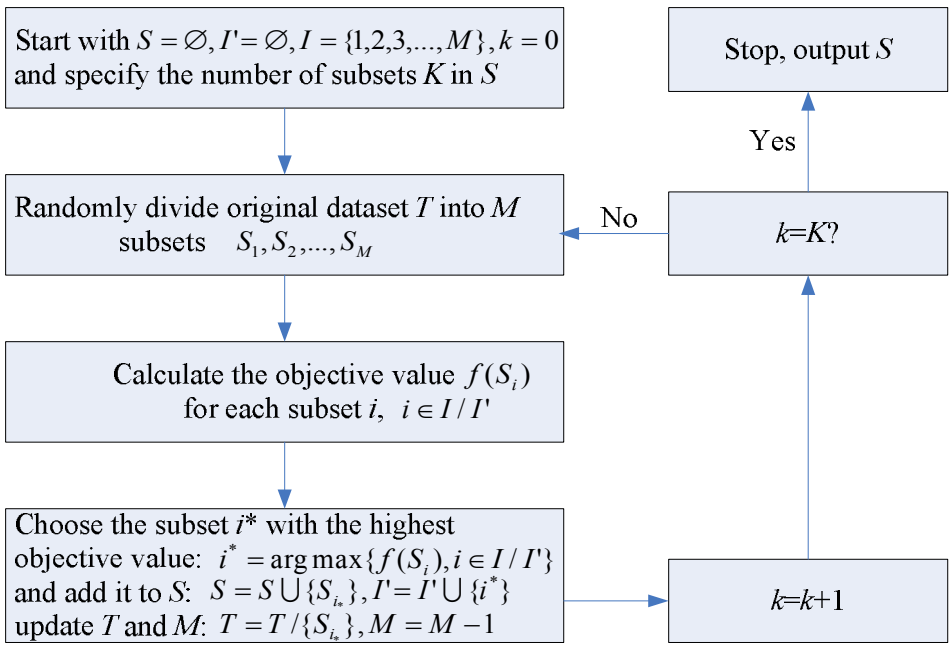


Figure 10 Main steps of simple construction heuristic method

As shown in Figure 10, in each iteration, we randomly divide the remaining instances into some subsets, build decision trees from each subset, calculate the objective values for all subsets, and then select the subset with the highest objective value until we have enough subsets. Next in Section 3.3, we will compare genetic algorithm with greedy heuristic and simple construction heuristic to show the strength of genetic algorithm.

3.3 Experiment Results

We have shown that GA is quite effective in Chapter 2, but how good is it compared to the other heuristic methods? Here we will compare GA search with 2-phase *Rmhc* greedy heuristic and simple construction heuristic. The objective function $f(S)$ is taken from equation (15) and higher objective value is better. We start with the comparison between GA and 2-phase *Rmhc*. Some results including the tree size, the selected subset size, the decision tree's accuracy, the objective value and the computation time are listed in Table 12, and it can be seen that point-based *Rmhc*(p) finds better instance subset with smaller tree size thus resulting in higher objective value than GA considering its much wider search space. However, the strength of GA lies in its ability to obtain the close results with much less computation time. On the other hand, the subset-based *Rmhc*(s) is faster than GA as expected, but GA is better in obtaining smaller trees and instance subsets.

Table 12 GA vs. 2-phase *Rmhc*

Dataset (size)	2-phase *Rmhc(p)*			2-phase *Rmhc(s)*			GA Search		
	Subset size	Tree size	Accuracy	Subset size	Tree size	Accuracy (%)	Subset size	Tree size	Accuracy
Letter	3966.0	1169.0	86.2	7998.0	1365.4	85.9	6735.0	1244.3	82.3
Segment	94.3	23.7	93.9	692.0	37.8	94.3	381.2	33.4	92.6
Scheduling	88.7	9.0	96.6	856.8	18.6	97.5	344.1	8.6	96.3
Sick	5.0	3.0	96.6	251.0	7.0	97.1	251.0	3.8	95.3
Splice	164.0	98.7	91.3	763.2	117.8	91.8	381.2	116.2	91.2

Dataset (size)	2-phase *Rmhc(p)*		2-phase *Rmhc(s)*		GA Search	
	Value of $f(S)$ (higher is better)	Time	Value of $f(S)$ (higher is better)	Time (seconds)	Value of $f(S)$	Time (seconds)
Letter	-61.36	4 months	-62.72	148.0	-61.95	329.0
Segment	-27.49	19.1 hours	-31.53	3.0	-30.48	34.4
Scheduling	-19.07	23.7 hours	-25.34	1.0	-18.68	47.9
Sick	-9.56	15.5 minutes	-16.89	1.0	-11.63	37.8
Splice	-39.88	56.7 hours	-41.40	3.0	-41.30	55.7

From the results reported before it is clear that instance selection has a significant effect on the decision trees, and in particular for many datasets it is possible to significantly reduce the size of the tree without sacrificing much of the accuracy. A natural question to ask is how much of this is due to the size of selected instance subset and how much is due to the method by which they are selected (here the GA optimization). We thus compare the instance subsets selected using the GA approach to those selected using simple construction heuristic and simple random sampling. The results are reported in Table 13 and Table 14 for $M = 10$ and $y = 0.5$, that is, the size of the data is reduced to half of the original size. It can be seen that simple construction heuristic runs very fast, but the quality of the solution is unstable compared with simple random sampling. Particularly, it obtains smaller tree size on two datasets ("Letter" and "Scheduling") but larger tree size on the other three datasets.

Table 13 GA vs. simple construction heuristic

Dataset	GA Search		Simple construction heuristic	
	Tree size	Accuracy	Tree size	Accuracy
Letter	**1244.3**	82.3	**1526.6**	87.7
Segment	**53.4**	95.2	**60.2**	96.0
Scheduling	**27.8**	98.0	**56.2**	99.3
Sick	**12.4**	98.0	**32.2**	98.7
Splice	**146.2**	92.8	**810.0**	93.3
Dataset	GA Search		Simple construction heuristic (SIS)	
	Value of $f(S)$	Time (seconds)	Value of $f(S)$	Time (seconds)
Letter	-61.95	329.0	-63.65	53.0
Segment	-34.50	34.4	-35.53	2.8
Scheduling	-28.81	47.9	-34.89	1.3
Sick	-21.82	37.8	-30.07	1.5
Splice	-43.26	55.7	-58.07	2.5

Table 14 GA vs. simple random sampling

	Original			GA Search			Random Sampling		
	Tree size	Accuracy	Value of $f(S)$	Tree size	Accuracy (%)	Value of $f(S)$	Tree size	Accuracy	Value of $f(S)$
Scheduling	69.0	99.6	-36.65	27.8	98.0	-28.81	69.1	99.2	-36.68
Splice	229.0	94.1	-47.12	146.2	92.8	-43.26	229.0	91.8	-47.16
Segment	77.0	96.9	-37.65	53.4	95.2	-34.50	57.2	95.0	-35.10
Letter	2451	88.0	-67.74	1244.3	82.3	-61.95	1579.4	83.7	-64.00
Sick	61.0	98.8	-35.60	12.4	98.0	-21.82	24.0	98.1	-27.54

Moreover, the results in Table 13 and 14 indicate that simple construction heuristic and simple random sampling have similar effects as the GA search, that is, for four out of the five test datasets the size of the tree is reduced if a random subset of half the instance is used instead of the whole set. Part of the effect of GA-based instance selection thus must be contributed to the simple fact that fewer instances are used after instance selection than before instance selection. However, the method by which the instances are selected is also clearly very important. For each of the datasets the GA search results in instance subsets that on the average give significantly smaller trees and higher objective values.

CHAPTER 4. INSTANCE SELECTION FOR DECISION TREE PRUNING

4.1 Decision Tree Pruning Techniques

Decision tree pruning methods originally proposed in Breiman et al. (1984) have been shown in various studies that they can improve the generalization performance of a decision tree, especially in noisy domains. Another key motivation of pruning is "trading accuracy for simplicity" as presented in Bratko and Bohanec (1994). When the goal is to produce a sufficiently accurate compact decision tree, pruning is highly useful. There are various techniques for pruning decision trees, including cost-complexity pruning (Breiman et al., 1984), minimum error pruning (Olaru and Wehenkel, 2003), pessimistic pruning (Quinlan, 1993), optimal pruning (Bratko and Bohanec, 1994), minimum description length (MDL) pruning (Mehta et al. 1996), minimum message length pruning (Wallace and Patrick, 1993), and critical value pruning (Mingers, 1989). Several studies aim to compare the performance of different pruning techniques (Quinlan 1987; Mingers, 1989; Esposito et al. 1997). The results indicate that there is no pruning method that in any case outperforms other pruning methods.

Generally speaking, these pruning techniques can be divided into two categories: Prepruning and Postpruning. Prepruning will try to decide when to stop developing subtrees

during the tree-building process, while postpruning is adopted by building the complete tree and pruning it afterward. It seems that prepruning is quite attractive because it may avoid all the work of developing unnecessary subtrees, however, postpruning does offer some advantages over prepruning in some "combination-lock" situation, that is, the situation where the correct combination of the two attribute values is very informative whereas the attributes taken individually are not. So most decision tree builders use postpruning techniques and it is still an open question whether prepruning strategies can perform as well.

4.1.1 Postpruning Operations

Two different operations have been involved for postpruning: subtree replacement and subtree raising (Witten and Frank, 2000). At each node, the learning scheme will decide whether to perform subtree replacement, subtree raising or leave the subtree unpruned. Subtree replacement is implemented by proceeding from the leaves and working back up toward the root as shown in Figure 11, (a) is the original decision tree and (b) shows the pruned tree. The subtree C in the original decision tree has three leave nodes: L1, L2, and L3. After subtree replacement, this subtree is replaced by a single leaf node L1.

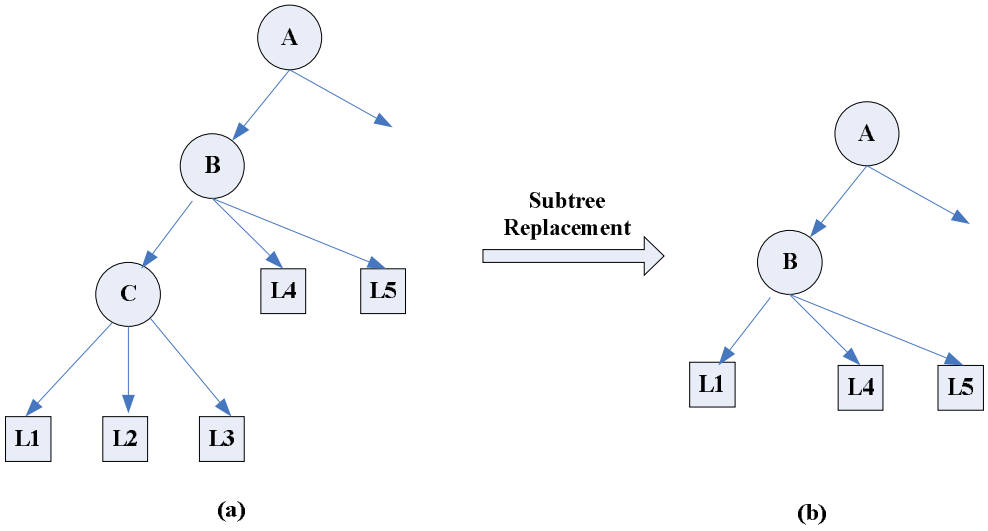


Figure 11 Example of subtree replacement

The second postpruning operation, subtree raising, is more complex. As illustrated in Figure 12, the entire subtree from C downward has been raised to replace the subtree B. Note that in this raising operation, it is necessary to reclassify the instances at the nodes L4 and L5 into the new subtree C. So the children of that node L1', L2' and L3' include not only the original children L1, L2 and L3 but also the instances covered by L4 and L5.

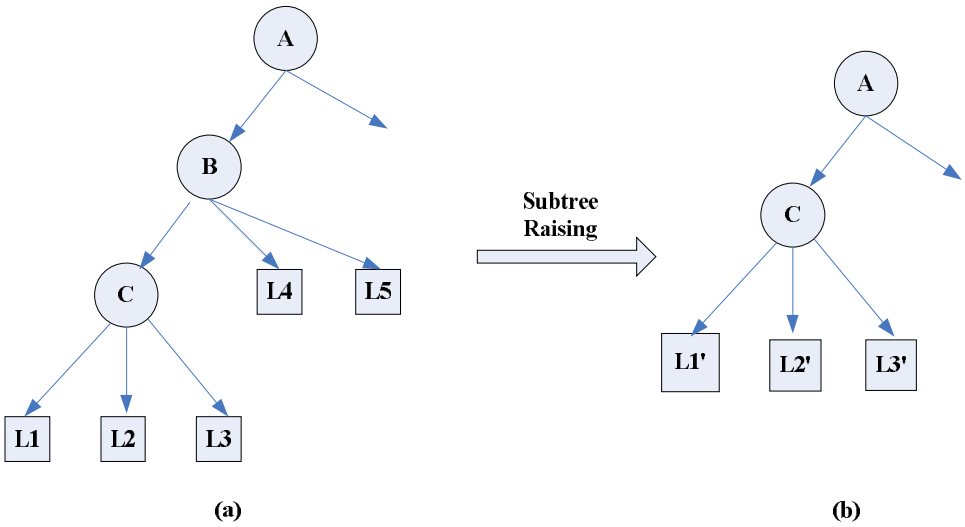


Figure 12 Example of subtree raising

4.1.2 Frequently-used Postpruning Algorithms

There are two broad classes of postpruning algorithms. The first class includes algorithms that use a separate set of instances for pruning, distinct from the dataset used for building the decision tree. For example, Quinlan (1987) proposed a reduced-error pruning technique that simply involves holding back some data and using it as an independent test set to estimate the error at each node. The subtree will be pruned if it provides better estimated accuracy on the test data. The obvious disadvantage of this approach is that the actual tree is built on less data. Another pruning technique, minimal cost-complexity pruning (Breiman et al. 1984), tries to find a series of trees that minimize a function that linearly combines the classification error and the number of leaves in the tree and cross-validation is used to select the best tree. In addition to the ad-hoc nature of cross-validation, this approach also suffers from the drawback that multiple candidate trees need to be generated, which can be computationally expensive.

The second class of decision tree pruning algorithms, which include C4.5 pessimistic pruning (Quinlan, 1993) and MDL pruning (Mehta et al. 1996), uses the whole training data for decision tree generation and pruning. C4.5 pessimistic pruning tries to make some estimate of the error based on the training data itself. The idea is to consider the task of classification as a binomial experiment. Given a test set that contains n instances, let X be the number of instances incorrectly predicted by a model and q be the true error rate of the model. By modeling the prediction task as a binomial experiment, X has a binomial distribution with

mean $n \cdot q$ and variance $n \cdot q \cdot (1-q)$. It can be shown that the empirical error rate, $f = X/n$, also has a binomial distribution with mean q and variance $q \cdot (1-q)/n$ (Miller and Miller, 2004). Although the binomial distribution can be used to estimate the confidence interval for f, it is often approximated by a normal distribution when n is sufficiently large. Based on the normal distribution, the following confidence interval for f can be derived as (Miller and Miller, 2004):

$$P\left[-Z_{\alpha/2} \le \frac{f-q}{\sqrt{q(1-q)/n}} \le Z_{1-\alpha/2}\right] = 1-\alpha \tag{16}$$

where $Z_{\alpha/2}$ and $Z_{1-\alpha/2}$ are the upper and lower bounds obtained from a standard normal distribution at confidence level $(1-\alpha)$. Since a standard normal distribution is symmetric around $Z = 0$, it follows that $Z_{\alpha/2} = Z_{1-\alpha/2}$. Rearranging this inequality leads to the following confidence interval for q:

$$\frac{f + \frac{Z_{\alpha/2}^2}{2n} \pm Z_{\alpha/2}\sqrt{\frac{f}{n} - \frac{f^2}{n} + \frac{Z_{\alpha/2}^2}{4n^2}}}{1 + \frac{Z_{\alpha/2}^2}{n}} \tag{17}$$

which for default α=25% used in C4.5 pessimistic pruning, $Z_{\alpha/2}$=1.15. Higher α will lead to higher classification accuracy but larger tree size, and α=25% seems an appropriate default value which is able to obtain good accuracy without over-pruning from some research results (Quinlan, 1993). And the upper bound for q is used as a *pessimistic* estimate for the

error rate e at the node as $e = \frac{f + \frac{Z_{\alpha/2}^2}{2n} + Z_{\alpha/2}\sqrt{\frac{f}{n} - \frac{f^2}{n} + \frac{Z_{\alpha/2}^2}{4n^2}}}{1 + \frac{Z_{\alpha/2}^2}{n}}$. Similarly the subtree will be pruned if the pruning tree yields lower estimated error rate.

MDL pruning, on the other hand, considers the following scenario: given the description of the complete decision tree and the description of all training instances in terms of values for all attributes and the class label, find the pruned tree that minimizes the description length of the remaining structure of the tree plus the description length of the classification of all instances given the pruned tree. Therefore, the subtree is pruned if

$$MDL(node) = Prior\_MDL(node) - Post\_MDL(subtree(node)) \le 0 \quad (18)$$

The algorithm proceeds bottom-up from leaves toward the root of the tree and for each internal node makes a decision whether to prune the subtree or not by the $MDL(node) \le 0$ criterion. A simple MDL measure is given by Kononenko (1995) as

$$Prior\_MDL(node) = \log\binom{n}{n_1,\ldots,n_c} + \log\binom{n+c-1}{c-1} \quad (19)$$

where n_i is the number of instances from the i-th class and c is the number of class values. The first term represents the encoding length of class of n instances, i.e. the encoding length of training data and the second term represents the encoding length of the class frequency. For example, if every class label occurs at least once, the possible combinations will be $\binom{n-1}{c-1}\cdot\binom{c}{0}$; if one class label never occurs, the possible combinations will be $\binom{n-1}{c-2}\cdot\binom{c}{1}$,

and so forth. Finally we have the total possible class occurrence counts as $\sum_{r=0}^{c-1}\left[\binom{n-1}{c-1-r}\cdot\binom{c}{r}\right]=\binom{n+c-1}{c-1}$ as shown in equation (19). So the second term can be viewed as the model cost of the model class involved (Mehta et al. 1996). There are some other formulas for the MDL code length as described by Krichevsky and Trofimov (1983) and Mehta et al. (1996). When compared to other algorithms such as pessimistic pruning that do not use separate dataset for pruning, the MDL-based pruning algorithm tends to produce trees that are significantly smaller in size.

In next section, we will compare the results from instance selection without pruning on obtaining small decision trees with the results from two widely-used pruning techniques including reduced error pruning and C4.5 pessimistic pruning. For the MDL pruning, we will report some results from Mehta's paper (Mehta et al. 1996).

4.2 Instance Selection for Pruning Decision Trees

Table 15 lists some main results for different decision tree pruning techniques with different number of minimum instances on leaf nodes (the complete results are listed in Appendix B, Table A13). With larger number of minimum instances on leaf nodes, the decision tree is forced to stop growing earlier so the tree size is smaller; but the accompanying problem is that the tree may not be developed well to fit the training data so it is less accurate. Considering the MDL pruning presented by Mehta et al. (1996) with the minimum number of leaf instances is 5, for the "Letter" dataset, the tree size after MDL pruning is 1174.8 and the

accuracy is 84.2%; for the "Segment" dataset, the tree size is 56.2 and the accuracy is 94.5%. As a comparison, for the "Letter" dataset, using reduced error pruning with subtree raising, the tree size is 965 and the accuracy is 82.8%; using C4.5 pessimistic pruning with subtree raising, the tree size is 1499 and the accuracy is 85.9%; using instance selection without pruning, the tree size is 857 and the accuracy is 81.6%. This example indicates that even for the decision tree without any pruning, instance selection still works well in obtaining good accuracy and small tree size compared to other pruning techniques. Therefore, instance selection can be used as a good alternative for tree pruning.

Table 15 Major results for different pruning techniques

Dataset	Number of minimum leaf instances	Minimum leaf instance setting without pruning		**Instance selection without pruning** (Top 50%/75%)*		Reduced error pruning with subtree raising		C4.5 pruning with subtree raising	
		Tree size	Accuracy	Tree size	Accuracy	Tree size	Accuracy	Tree size	Accuracy
Letter	5	1637	85.9	857.0	81.6	965	82.8	1499	85.9
	10	987	83.2	509.0	76.7	647	80.1	931	83.2
	15	755	81.3	348.6	73.0	409	75.7	589	79.2
Scheduling	5	61	99.4	33.0	98.6	43	99.1	55	99.4
	10	47	99.1	24.2	98.1	37	98.7	41	99.1
	15	43	98.9	18.2	97.6	29	97.9	35	98.6
Splice	5	332	92.7	172.0	90.5	154	92.7	171	94.4
	10	213	91.2	116.2	87.9	142	90.9	134	92.5
	15	156	90.3	66.4	85.3	74	86.5	108	89.9
Segment	5	75	96.0	36.2	94.6	43	95.0	59	96.0
	10	47	95.1	25.0	92.4	39	94.9	47	95.0
	15	39	95.0	20.6	90.4	31	92.8	33	94.3
Sick	5	50	98.9	15.6	98.1	39	98.3	34	98.8
	10	37	98.5	10.0	97.9	17	98.0	28	98.6
	15	24	98.3	8.6	97.5	7	97.9	14	98.1

*: Scheduling, segment and sick: use top 50% selected instances; Letter and splice: use top 75% selected instances to obtain close accuracies

4.3 Case Study: Instance Selection for Sick Dataset

Exploring the selected instances will provide more insights for our instance selection approach. There are several ways in which instance selection could be helpful: eliminate outliers, eliminate missing values, and select the most useful instances for separating classes. However, there is also a concern that instance selection may eliminate or vastly reduce minority classes in unbalanced datasets. All these issues are explored in this section for "Sick" dataset. We choose this dataset because it is the only one with the required characteristics: missing values, outliers, and unbalanced class. First of all, Figure 13 and 14 show the decision trees before and after instance selection, respectively.

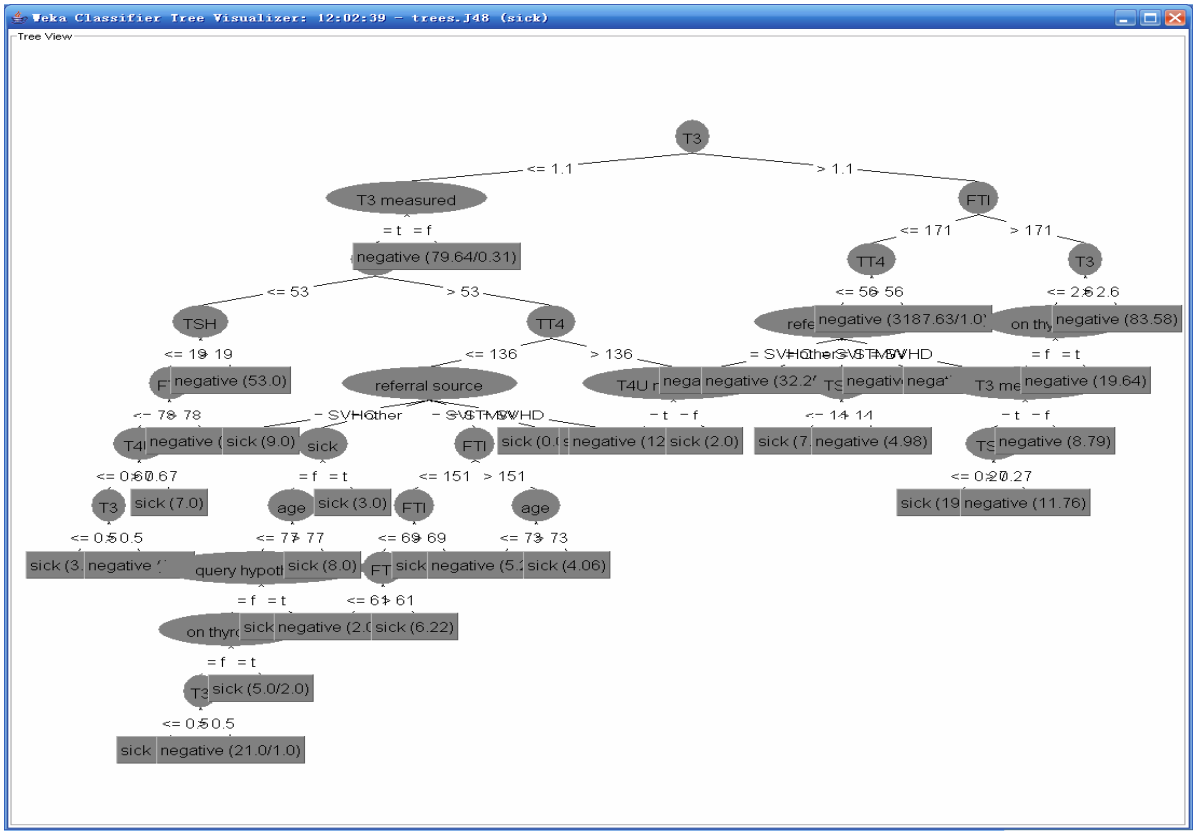


Figure 13 Decision tree before instance selection on sick dataset

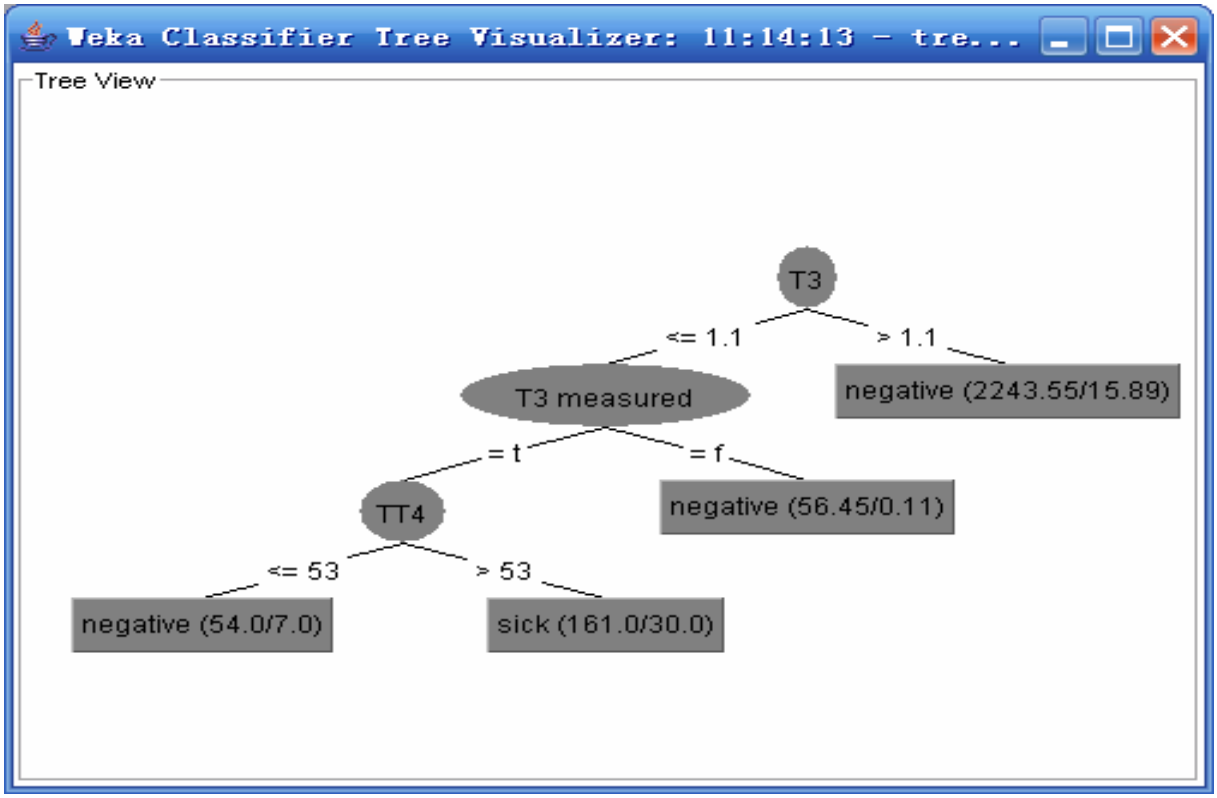


Figure 14 Decision tree after instance selection on sick dataset

Before instance selection, the decision tree has 61 nodes, 34 leaves and uses 12 distinct attributes for splits ("T3", "T3 measured", "TT4", "TSH", "FTI", "T4U", "referral source", "sick", "age", "query hypothyroid", "on thyroxine" and "T4U measured"). After instance selection, the decision tree contains only 7 nodes, 4 leaves and 3 split attributes ("T3", "T3 measured" and "TT4"). More importantly, the tree is much more interpretable. From medical knowledge (Fu, 2007), the values of "T3" and "TT4" are two most important measures to reflect the patient's thyroid situation. The normal "T3" value is 1.23~3.39 nmo1/L, if the value of "T3" is less than 1.23 and the value of "TT4" is over 30 times than "T3" value, it is highly possible that the patient has thyroid disease. The decision tree in Figure 14 shows that

if “T3” value is less than 1.1 and “TT4” value is higher than 53, the patient has thyroid disease, which is quite consistent with the medical knowledge.

Moreover, 2-dimensional projection tour plots from GGobi (Swayne et al. 2003) are used to display the distribution of the class attribute before and after instance selection. Mathematically, a 2-dimensional projection of data is computed by multiplying an $n \times p$ data matrix X by an orthonormal $p \times 2$ projection matrix A. These tour plots are useful in revealing interesting data structures, such as clusters of points and outliers. For data mining models, tour plots can help to understand how the models work in a particular problem (Cook et al., 1995). For a 2-dimensional tour plot, the *x*-axis corresponds to the values of data matrix X projected to one dimension and the *y*-axis corresponds to the values of data matrix X projected to the other dimension. For example, suppose the data matrix X and projection A_1 were these (Cook et al., 1995):

$$X = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 15 \\ 0 & 15 & 0 \\ 0 & 15 & 15 \\ 15 & 0 & 0 \\ 15 & 0 & 15 \\ 15 & 15 & 0 \\ 15 & 15 & 15 \end{bmatrix}_{8\times 3} \quad \text{and } A_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}_{3\times 2} \quad \text{then } XA_1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 15 \\ 0 & 15 \\ 15 & 0 \\ 15 & 0 \\ 15 & 15 \\ 15 & 15 \end{bmatrix}_{8\times 2}$$

is the first two columns of the data matrix X. If instead

$$A_2 = \begin{bmatrix} 0.71 & -0.42 \\ 0.71 & 0.42 \\ 0 & 0.84 \end{bmatrix}_{3\times 2} \quad \text{then } XA_1 = \begin{bmatrix} 0 & 0 \\ 0 & 12.60 \\ 10.65 & 6.30 \\ 10.65 & 18.90 \\ 10.65 & -6.30 \\ 10.65 & 6.30 \\ 21.30 & 0 \\ 21.30 & 12.60 \end{bmatrix}_{8\times 2}$$ is a combination of all three columns

of the data matrix X. These projections are illustrated in Figure 15, which shows the data projections, XA_1 and XA_2, respectively. More examples of these 2-dimensional tour plots can be found in Appendix G (Figure A13, A15, A17 and A19).

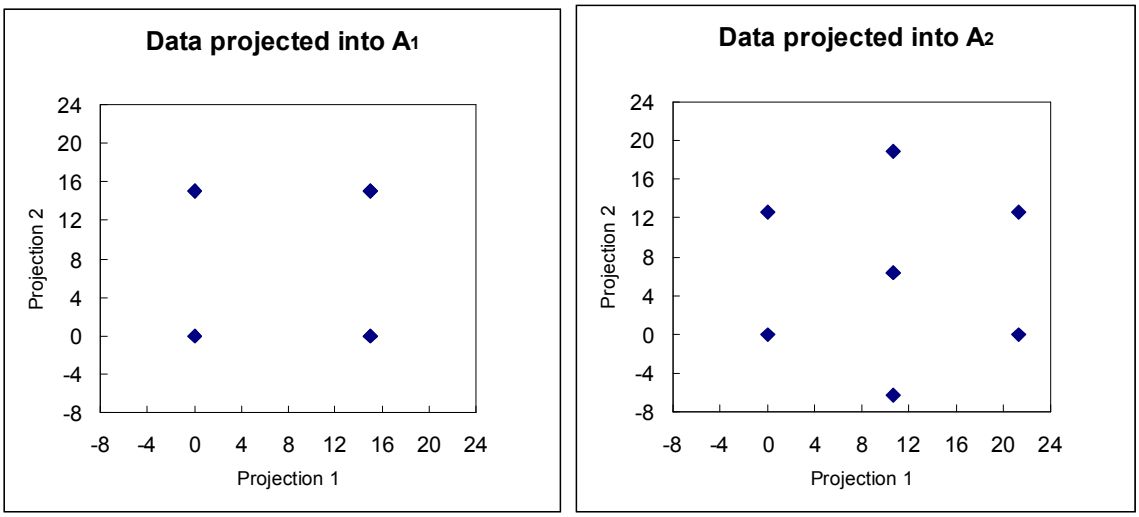


Figure 15 Two 2-dimensional data projections

Figure 16 and 17 show the tour plots on "Sick" dataset before and after instance selection. The green cross points represent the "negative" class while the blue circle points represent the "sick" class. Figure 16 (a) and Figure 17 use the three attributes in the decision tree after instance selection, that is, "T3", "T3 measured" and "TT4", while Figure 16 (b) uses the

twelve attributes in the decision tree before instance selection including “T3”, “T3 measured”, “TT4”, “TSH”, “FTI”, “T4U”, “referral source”, “sick”, “age”, “query hypothyroid”, “on thyroxine” and “T4U measured”.

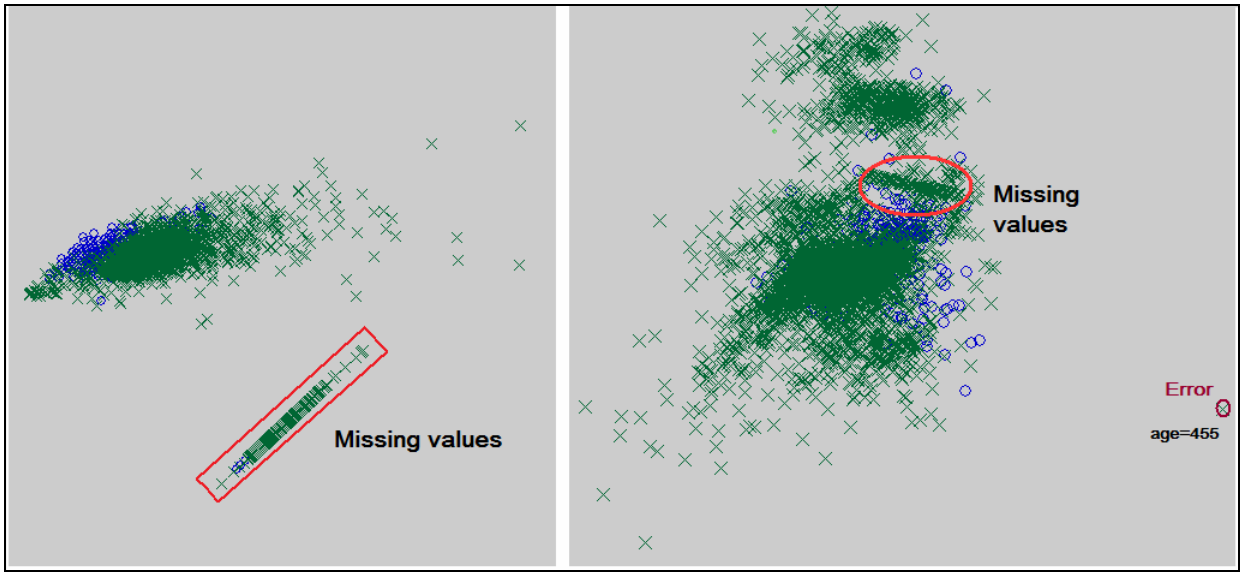


Figure 16 Data visualization before instance selection
(a) using three attributes; (b) using twelve attributes

As is known, a decision tree can be viewed as a partitioning of the instance space and the instance space can only be partitioned in boxes parallel to axes of the space. Each partition, represented by a leaf, contains the instances that are similar in relevant respects and thus are expected to belong to the same class. So if the instances from the same class are close to each other and well separated from the instances from other classes, the decision tree can find the partition quickly hence the size of the tree is better controlled. From Figure 16, it can be seen that before instance selection there is a lot of overlapping on the distribution for the classes no matter we use three or twelve attributes, so while it is quite possible to grow a tree that fits

the training set well it may become too elaborate. On the other hand, the distribution of the classes is much more separated after instance selection as shown in Figure 17, thus it is easier for the decision tree to find a good partition of the whole space and helps to cut down the tree's size.

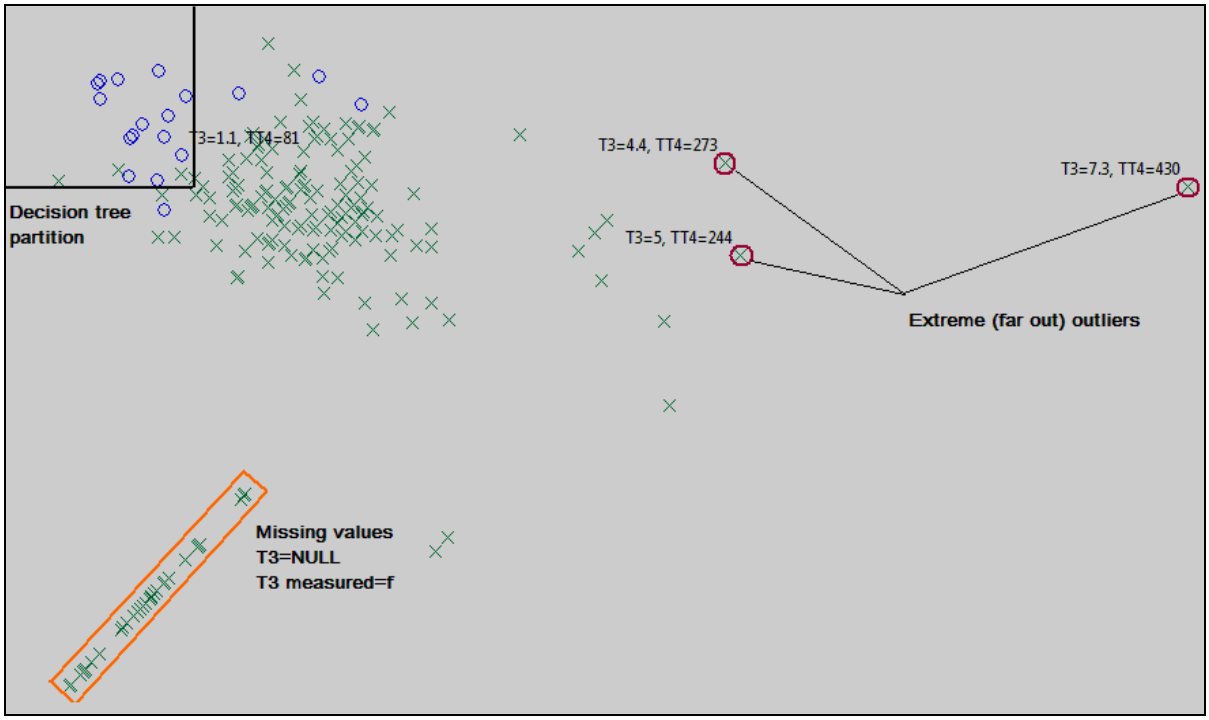


Figure 17 Data visualization after instance selection

As a comparison, Figure 18 shows the scatter plot matrix on selected instances. It can be seen that even though Figure 18 does show some relationship between those three important attributes with the classification, it is still difficult to determine the exact separation of the two classes from the pairwise plots without the combination of the three attributes. However, by using a tour plot, the separation of the two classes can be drawn using all three attributes as shown in Figure 17. The upper left corner is separated as "sick" class and the other space

is separated as “negative” class. It can be seen that there are three instances (plotted as cross) incorrectly classified as “sick” which are very close to the border and there are four instances (plotted as circle) incorrectly classified as “negative”. These seven points are the errors from decision tree. So the tour plot clearly displays the tree solution to this classification problem.

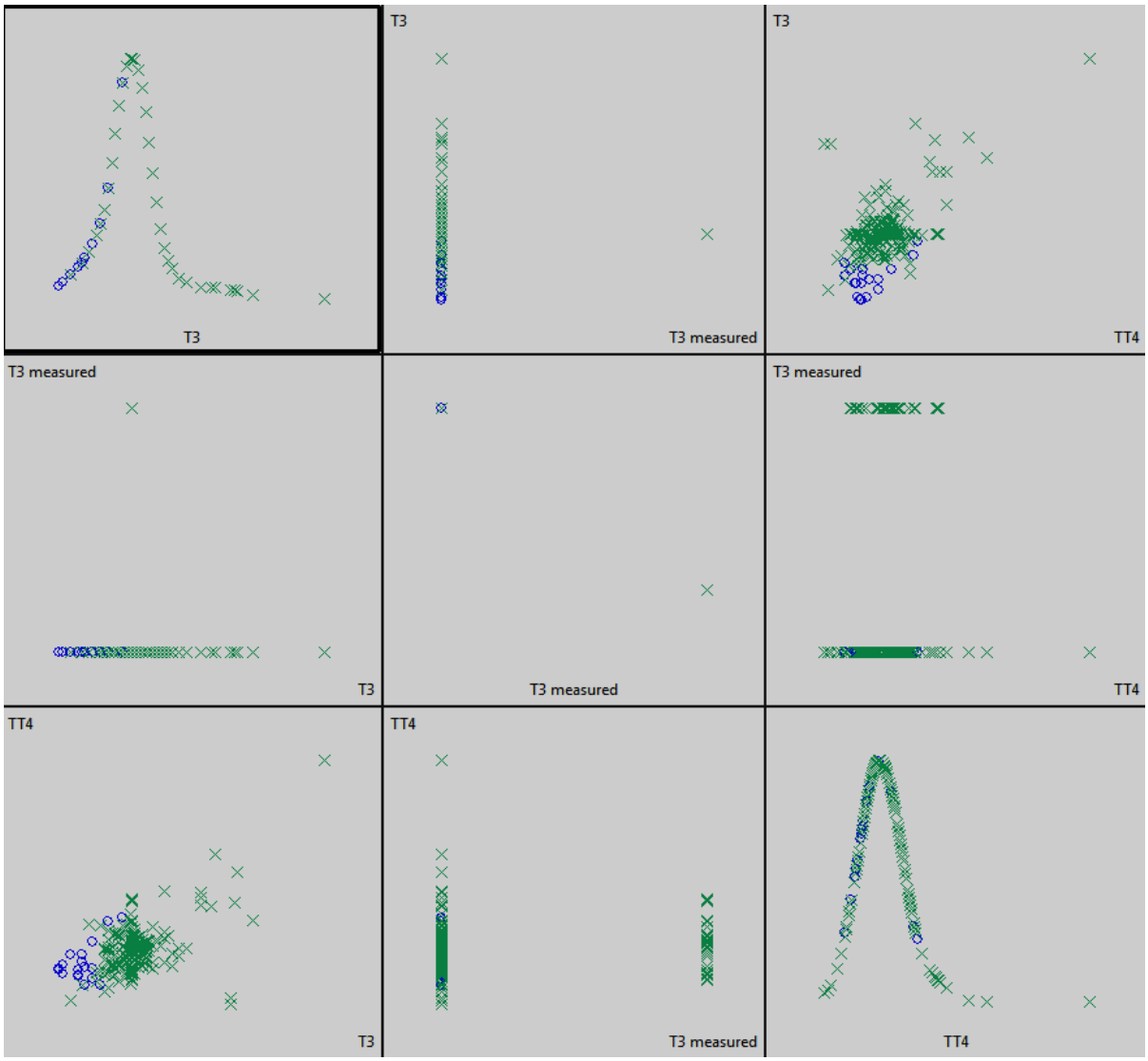


Figure 18 Scatter plot matrix of three important attributes

Furthermore, from Figure 16, it can be easily identified one error in the original data. This error has the value of 455 in "age" attribute which is impossible for a patient. However, after instance selection, this instance is excluded from the selected subset as shown in Figure 19, which indicates that instance selection may get rid of some errors in the original data.

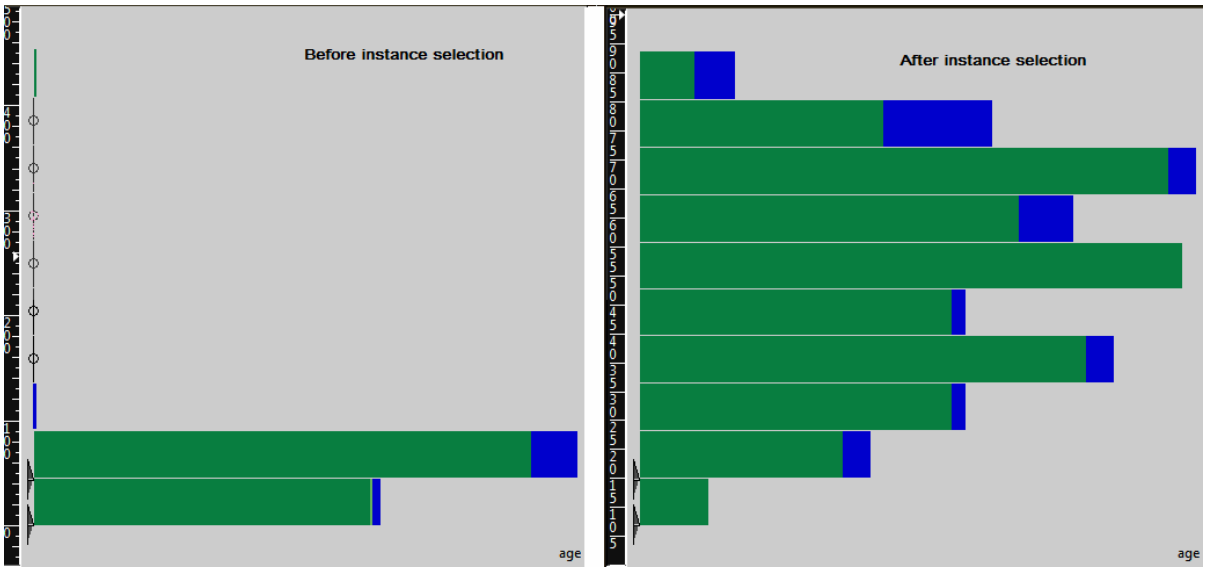


Figure 19 Bar charts on age attribute before (left) and after (right) instance selection

There are some further interesting insights that can be read from Figure 16 and 17. Both of these figures have some data points arranged in straight line because these instances have missing values in attribute "T3" and identical values in attribute "T3 measured". The presence of missing values in a dataset can affect the performance of a classifier constructed using that dataset as a training data. Rates of less than 1% missing data are generally considered trivial, 1~5% manageable. However, 5~15% require sophisticated methods to handle, and more than 15% may severely impact any kind of interpretation (Little and Rubin, 2002). Before instance selection, the number of these missing values is 769 (the rate of the

number of missing values over the total number of instances is 20.4%) while after instance selection, the number of these missing values is down to 31 (the rate drops to 12.4%). Instance selection thus helps to reduce the number of missing values, which is certainly beneficial for the decision tree even though C4.5 decision tree is able to deal with missing values.

Besides missing values, the sick dataset also contains some possible outliers. For example, considering attribute "TT4", there appear to be some data points on the far top in the bar charts as shown in Figure 20. For quantifiable methods, modified z-score method (Barnett, 1985) and box plot rule can be used to identify the possible outliers.

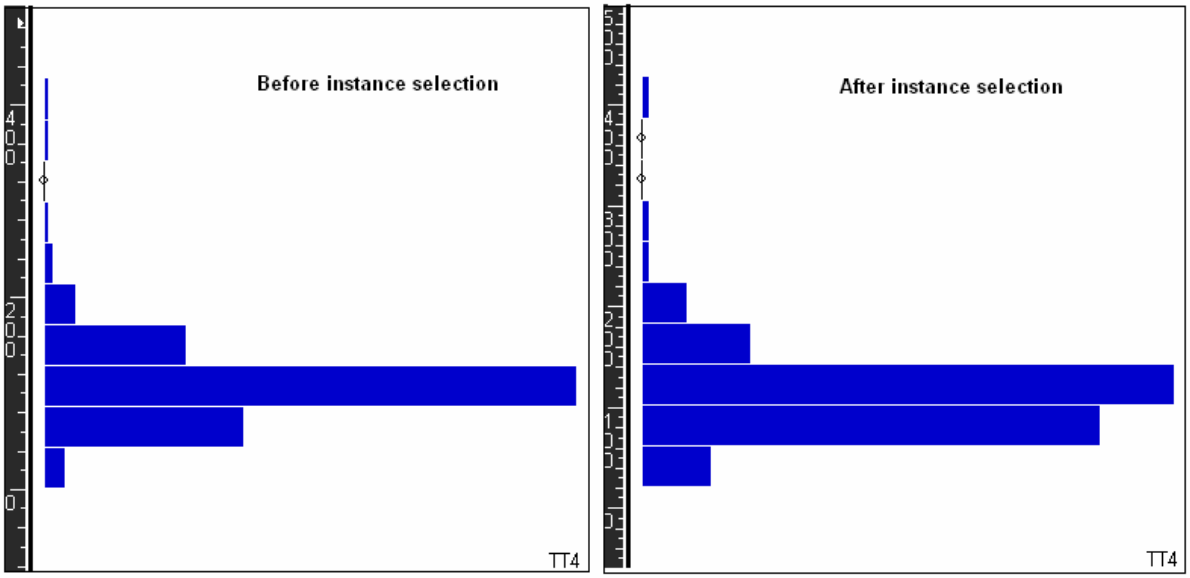


Figure 20 Bar charts on TT4 attribute before (left) and after (right) instance selection

Z-score method is based on robust regression methods (Rousseeuw, 1987). In the original z-score test, the mean $\bar{x}$ and standard deviation s of the dataset are used to obtain a

z-score $z_i = \frac{x_i - \bar{x}}{s}$ for each observation. Donzenis and Rakow (1987) suggested that absolute z-score higher than 2.70 should be considered "outside" and higher than 4.72 should be considered "far out". However, this method is not reliable because both the mean and standard deviation are influenced by the outliers. To address this problem, the modified z-score method uses the median of absolute deviation about the sample median (MAD) to replace the standard deviation s in z-score calculations. The MAD is defined as $MAD = median\{|x_i - x_m|\}$, where x_m is the median of entire data. Table 16 lists the results from modified z-score method for attribute "TT4" before and after instance selection.

Table 16 Modified z-score method for identifying outliers

Attribute TT4	Before instance selection	After instance selection
Sample mean $\bar{x}$	108.3	111.3
Sample median x_m	103	109.5
MAD value	18	22
Sample size (without missing values)	3541	240
Number of missing values	231 (6.1%)	11 (4.4%)
Number of outside outliers (z-score> 2.70)	435 (12.3%)	20 (8.3%)
Number of far-out outliers (z-score> 4.72)	123 (3.5%)	3 (1.2%)

It can be seen from Table 16 that before instance selection, there are 435 potential outliers in "TT4" which takes up 12.3% of the entire data including 123 "far out" outliers which takes up 3.5% of the entire data. However, in the selected instances, there are only 20 possible outliers in "TT4" with 3 "far out" outliers, and the rates for "outside" and "far out" outliers drop from 12.3% to 8.3% and 3.5% to 1.2% respectively. Again Table 16 shows that

instance selection greatly reduces the missing values. 220 missing values in original data are excluded from the selected instances.

Modified z-score method is mainly applied to normally distributed data. But the results from Kolmogorov-Smirnov normality test (Kolmogorov, 1933) indicate that the normal distributional assumption may not be satisfied, so the modified z-score method may mislabel or miss some outliers. Box plot rule, on the other hand, does not require the normal distributional assumption. Box plot has become the standard technique for presenting the 5-number summary which consists of the minimum and maximum range values, the 25th percentile (Q_1), the 75th percentile (Q_3) and the median. The rule states that an observation is labeled as a "*mild*" outlier if its value is less than $[Q_1 - 1.5 \times (Q_3 - Q_1)]$ or larger than $[Q_3 + 1.5 \times (Q_3 - Q_1)]$; it is an "*extreme*" outlier if its value is less than $[Q_1 - 3 \times (Q_3 - Q_1)]$ or larger than $[Q_3 + 3 \times (Q_3 - Q_1)]$ (Tukey, 1977). Table 17 lists the results from box plot rule.

Table 17 Box plot rule for identifying outliers

Attribute TT4	Before instance selection	After instance selection
25th Percentile (Q_1)	87	85.5
Median	103	109.5
75th percentile (Q_3)	124	127
Sample size (without missing values)	3541	240
Number of mild outliers	495 (13.9%)	32 (13.3%)
Number of extreme outliers	71 (3.5%)	3 (1.3%)

Comparing the results from modified z-score method in Table 16 and box plot rule in Table 17, it can be seen that box plot rule identifies more mild outliers than modified z-score method but equal or less extreme outliers. Box plot rule finds 495 mild outliers in original data and 71 of them are extreme outliers; after instance selection, the number of mild outliers

is down to 32 and 3 of them are extreme outliers. It is interesting that the extreme or far out outliers from modified z-score method and box plot rule turn out to be the same. These three outliers are also highlighted in Figure 17. Nevertheless, the results from Table 16 and 17 consistently reveal another benefit from instance selection, that is, it may help to reduce the possible outliers in the data, especially the "far out" or extreme outliers, which is beneficial to the development of better data mining models since in the presence of outliers, any statistical analysis based on sample means and variances can be distorted.

Last but not least, Figure 16 and 17 show that the two class values are not balanced in the original dataset as well as the selected instance subset, so there is a concern that instance selection would eliminate a minority class and the overall accuracy of the decision tree may be misleading. In particular, the original sick dataset has 3541 "negative" values but only 231 "sick" values (the ratio between the number of the minority class and majority class is 6.5%); while in the selected instance subset, there are 228 "negative" values but only 23 "sick" values (the ratio is 10.1%). When a dataset is unbalanced (the number of instances in different classes varies greatly), the error rate of a classifier may not be representative of the true performance of the classifier. A confusion matrix (Kohavi and Provost, 1998) is helpful under this situation. This matrix contains information about actual and predicted classifications done by a classification model. For example, considering a binary classification problem (positive or negative), the confusion matrix will have four categories as shown in Figure 21 (a): True positives (*TP*) are instances correctly classified as positives;

False positives (*FP*) are negative instances incorrectly classified as positive; True negatives (*TN*) correspond to correctly classified negatives; finally, false negatives (*FN*) refer to positive instances incorrectly classified as negative. Figure 21 (b) gives some definitions of different evaluation measures from the confusion matrix (Davis and Goadrich, 2006). The True Positive Rate measures the fraction of positive instances that are correctly classified. The False Positive Rate measures the fraction of negative instances that are misclassified as positive. Recall is the same as True Positive rate, whereas Precision measures the fraction of instances classified as positive that are truly positive.

	Actual positive	Actual negative
Predicted positive	*TP*	*FP*
Predicted negative	*FN*	*TN*

(a)

Recall $= \frac{TP}{TP+FN}$

Precision $= \frac{TP}{TP+FP}$

True Positive Rate $= \frac{TP}{TP+FN}$

False Positive Rate $= \frac{FP}{FP+TN}$

(b)

Figure 21 Confusion matrix (a) and related evaluation measures (b)

Our concern here is that instance selection would vastly reduce the minority class then the decision tree from the selected instances may have high overall accuracy but poor recall on the minority class since the majority class dominates the classification results. We want the decision tree to have good performance both on the majority and minority class, so the value of recall or true positive rate on the minority class is of special interest.

Table 18 Performance of the decision tree before/after instance selection

Confusion matrix	Before instance selection (10-fold cross validation)		After instance selection			
			10-fold cross validation		**Independent test set**	
	Negative	Sick	Negative	Sick	Negative	Sick
Classified as negative	3523	27	225	4	1173	6
Classified as sick	18	204	3	19	12	67
True positive rate	99.5%	88.3%	98.7%	82.6%	99.0%	91.8%
False positive rate	11.7%	0.5%	17.4%	1.3%	8.2%	1.0%
Precision	99.2%	91.9%	98.3%	86.4%	99.5%	84.8%
Recall	99.5%	88.3%	98.7%	82.6%	99.0%	**91.8%**
Kappa statistic	0.89		0.83		0.87	
Overall accuracy	98.8%		97.2%		98.6%	

Table 18 includes the confusion matrix and different performance measures for the decision trees. Using 10-fold cross-validation, the decision tree built on the entire data has the recall of 99.5% for the "negative" class and 88.3% for the "sick" class. After instance selection, the decision tree from the selected subset has the recall of 98.7% for the "negative" class and 82.6% for the "sick" class. As expected, the decision trees have higher accuracies on the majority class, but the accuracies on the minority class are still acceptable with more than 80%. More importantly, as in all of the above experiments, the decision tree from the selected subset is applied to another independent test data (1185 "negative" instances and 73 "sick" instances) to obtained an unbiased estimate of the accuracy. The tree obtains the overall accuracy of 98.6%, the recall of 99.0% on "negative" class and the recall of 91.8% on "sick" class. Instance selection therefore actually increases the recall for the minority class on the test data. Also the decision trees before and after instance selection both have high Kappa statistic values (over 0.8), which represents the correlation between the classification and the actual data. The value of Kappa statistic ranges from 0 to 1. A perfect classification will have

a Kappa value of 1. Typically, values greater than 0.8 represent strong agreement between classification and the actual data while values between 0.4 and 0.8 represent moderate agreement. Anything below 0.4 is indicative of poor agreement (Congalton and Green, 1999). So the high Kappa statistic values in Table 18 indicate that the classification predicted by decision trees has good agreement with the actual data and the unbalanced structure of this dataset does not have great impact on the performance of the decision tree.

Similar analysis is applied to the other two datasets ("Scheduling" and "Splice" data) that also have unbalanced class values and the results are listed in Appendix H (Table A20 and A21). It can be seen that after instance selection, the recalls of the "Splice" data are quite good for those two minority classes ("EI" and "IE") with over 85% while the recall of the "Scheduling" data is unacceptable for the minority class ("no") with only 41.7%. So for "Scheduling" data, the unbalanced classes do affect the performance of the decision tree from the selected instances. One possible way to address this concern is to incorporate the recall for the minority class into the objective function, for example, the objective function might be modified as follows:

$$f(S) = -\log\left(\hat{e}(\psi(S))\right) - \log(\text{recall for minority class}) - a\log\left(\frac{size(\psi(S))}{K}\right), \quad a > 1. \qquad (20)$$

With the modified objective function, the optimization is forced to search the instances from which the relatively high recall on the minority class can be obtained. Appendix H (Table A20) also includes the results from the modified objective function for comparison. It can be

seen that six more instances are selected from the minority class resulting in higher recall value. Even the overall accuracy is reduced slightly from 97.1% to 95.9%, the recall on "no" class is greatly increased from 41.7% to 83.3%. Thus, the modified objective function acts quite well in improving the decision tree's performance on minority class. It is also interesting that the accuracies on different classes have similar patterns before and after instance selection. For example, the decision tree from original "Sick" dataset has higher accuracy on "negative" class than "sick" class, then after instance selection, the decision tree from the instance subset still has higher accuracy on "negative" class. Similar conclusions can also be reached for "Scheduling" and "Splice" datasets from Appendix H (Table A20 and A21), which implies that instance selection tends to keep the instances doing well in classification.

As a brief summary for this section, the benefits from instance selection may include:

- Reduce the size of decision tree and the amount of data needed for inducing good decision tree.
- Reduce some missing values from the original data.
- Reduce some possible outliers, especially the extreme outliers, from the original data.
- Keep the instances that help to obtain good classification accuracy.

CHAPTER 5. CONCLUSIONS

Facing the mounting challenges of enormous amounts of data, much of the current research concerns itself with scaling up data mining algorithms. Instance selection, which is dealing with scaling down the data, provides an alternative to the algorithm scaling-up and has drawn more and more attention recently. As data mining is applied to larger datasets, effective instance selection can be expected to grow in importance. In this dissertation we present an optimization-based approach to instance selection for improving decision tree. We provide in the following sections a summary of main results, general conclusions and some directions for future research.

5.1 Summary of Results

The objective of instance selection is to select an instance subset that results in smaller and more easily interpretable decision trees without losing predictive accuracy. To obtain heuristic solutions to this problem we used a genetic algorithm (GA) implementation. Section 2.2 describes the details of the genetic algorithm. The genetic algorithm incorporates decision tree's accuracy and size into its objective (fitness) function, thus during the search process, the algorithm tries to find the instance subset from which the small and high quality decision tree can be developed. The decision tree from the entire dataset and the selected instances are both evaluated using an independent test data. Through the computational experiments on

five different datasets from UCI Machine Learning Repository, we have shown in Section 2.3.2 that for C4.5 decision trees, the size of the tree can be significantly reduced using instance selection, while the predictive accuracy is good. One concern for this approach is that it may be time consuming for large datasets since lots of decision trees have to be built and evaluated during the GA process. So in Section 2.3.3, GA-based instance selection is applied to a large dataset with over 50,000 instances and the computation time grows only at linear rate, which indicates that our approach is able to find good instance subsets with acceptable computation cost. Then a natural question for GA-based instance selection is when this approach is the most likely to work well. A study of data entropy in Section 2.3.4 leads to the conclusion that GA-based instance selection works best for the dataset with relatively low data entropy. For example, the dataset with only two class values has much lower entropy than the dataset with multiple class values, then for two-class problem, we usually only need a small fraction of the instances to build a good decision tree, while for multi-class problem, much more instances are needed hence there is less benefit from instance selection.

In addition to showing the effectiveness of GA-based instance selection, Chapter 3 compares the results from genetic algorithm and other heuristic methods including *Rmhc* greedy heuristic and simple construction heuristic. The point-based *Rmhc* method has the best solution due to its widest search space, but it is very heavyweight in computational resources. On the other hand, the subset-based *Rmhc* and simple construction heuristic have

the same search space as GA and they both run very fast, but they are not as good as GA in terms of solution quality. Considering the speed of the algorithm and the quality of the solution, GA is obviously the winner with good efficiency and effectiveness.

The dissertation further investigates other decision tree pruning techniques that also try to reduce the tree's size and improve its interpretability. Chapter 4 reviews some frequently used pruning techniques, including reduced error pruning, C4.5 pessimistic pruning and Minimum Description Length (MDL) pruning. The comparison between instance selection and these pruning techniques demonstrates that even without pruning, the decision tree from selected instances has small size and comparable accuracy, so instance selection can be used as a good alternative for decision tree pruning. Also the case study reported in Section 4.3 reveals some insights from instance selection. More specifically, the study shows that instance selection effectively reduces the missing values and potential outliers from the original data while at the same time keeps the instances beneficial for classification. Another concern for instance selection is that it may eliminate or greatly reduce the minority class for the dataset with unbalanced classes. With the analysis of confusion matrices on three different datasets, it is found that one dataset did have this problem, that is, the decision tree from the selected instances has low recall on the minority class even though the overall accuracy is high. A possible modification of the objective function, that is, adding the recall on minority class, is proposed to address this problem and it does perform well in improving the decision tree's performance on the minority class.

5.2 Hypotheses Revisited and General Conclusions

Our work illustrates the usefulness of using instance selection as pre-processing before decision tree induction, and the effectiveness of using genetic algorithm for obtaining heuristic solutions to this problem. Some major conclusions are summarized below. Also the three hypotheses proposed in Section 1.2 are quoted below for ease of reference.

The first hypothesis addresses the effectiveness of genetic algorithm for instance selection:

Hypothesis 1

GA-based instance selection will produce smaller and more interpretable decision trees while maintaining an acceptable level of accuracy.

The following conclusions are based on the study of GA-based instance selection in Chapter 2:

- GA-based instance selection reduces the size of the decision tree by an order of magnitude.
- Using different aggregation of the instance subsets, GA-based instance selection maintains good prediction accuracy.

In addition to these results, which are related to the effectiveness of GA-based instance selection, the discussion in Chapter 3 concludes that compared with other heuristic

approaches such as *Rmhc* greedy heuristic and simple construction heuristic, GA-based instance selection performs better in balancing computation cost and solution quality.

The second hypothesis addresses the benefit of instance selection for preventing overfitting in decision tree induction:

Hypothesis 2

Optimization-based instance selection prevents overfitting in decision tree learning.

The following conclusions are based on the study of average leaf ratio described in Chapter 2:

- Instance selection is beneficial for decision tree in increasing the average number of instances (ALR) associated with leaf nodes, thus helps to avoid the overfitting of decision tree.
- With more instances selected, there is less improvement in ALR hence less benefit from instance selection.

Furthermore, in Chapter 4, we compare instance selection with different decision tree pruning techniques. The results show that instance selection can be used as an effective alternative for decision tree pruning. And through the case study on "Sick" dataset, we find that instance selection helps to reduce the errors, potential outliers and missing values from the data while it maintains the instances that perform well in classification.

The third hypothesis addresses the effects of different parameters in the performance of instance selection:

Hypothesis 3

Number of instances, number of class values, and number of attributes will affect the performance of instance selection for improving decision tree

The following conclusions are based on the study described in Chapter 2:

- Number of class values, or more generally, data entropy has great impact on instance selection. Instance selection works best for low entropy problems; with higher entropy more instances are needed to account for the diversity in the data to induce good decision trees and hence there is less benefit from the instance selection approach.

- Number of instances and attributes will influence the speed of genetic algorithm. The computation time grows only at linear rate with more instances, while it grows faster with more attributes for large datasets.

5.3 Future Work

It should be noted that even though GA-based instance selection is mainly focused on improving C4.5 decision trees in this dissertation, it can be applied to other interesting problems. For example, Li and Olafsson (2005) used instance selection to identify good scheduling practices and the fitness function was modified to include the scheduling

performance measure such as maximum lateness. The results showed that instance selection greatly reduced the maximum lateness and provided more scheduling insights. Moreover, instance selection has been successfully applied to identity disclosure protection (Zhu and Wu, 2006). Identity disclosure is one of the most serious privacy concerns nowadays. Instance selection was used to reconstruct the original data so that the most important instances for classification can be protected.

Future research will take into account more complicated situations, and better characterize dataset where this approach is the most likely to work well. In particular, we will consider how different structures of data such as unbalanced classes, high entropies and existence of outliers influence the performance of the algorithm. And it will be interesting to further explore the selected instances since it may reveal some insights of instance selection. Another important issue in instance selection is how to choose the appropriate number of subsets. Currently the number of subsets is determined by empirical study, but it will be beneficial to develop some heuristic for this. We will also consider other choice for defining a fitness function that balances the size reduction objective with maintaining high predictive accuracy. Finally, for better data reduction, it is natural to investigate if this work can be combined with other research in solving the problem of huge amounts of data, such as algorithm scaling-up, attribute selection and attribute construction. It is still a big challenge to integrate these different techniques in achieving the goal for effective and efficient data mining.

APPENDIX A DIFFERENT PARAMETERS FOR GA-BASED INSTANCE SELECTION

TABLE A1-A12 (PAGE 91-102)

Table A1 Results of best subset when fitness weight *a* and selection constant *Q* are varied

Dataset		α=2.0						α=4.0						**α=6.0**			
		Q=1.5		Q=2.0		Q=2.5		Q=1.5		Q=2.0		Q=2.5		Q=1.5		Q=2.0		**Q=2.5**	
		Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.
Scheduling	Accuracy	95.97	0.94	96.12	0.74	96.26	0.39	95.27	0.44	95.16	0.85	95.28	0.82	95.78	0.60	94.99	0.84	95.86	0.51
Best subset	Tree size	10.20	2.40	10.20	2.71	11.40	1.50	7.80	2.04	9.80	2.04	8.60	2.33	7.80	2.04	9.00	1.26	10.60	0.80
	ALR	19.43	3.20	19.90	4.51	17.25	2.26	25.35	6.07	20.16	3.88	23.37	6.49	25.24	6.01	21.44	2.97	18.22	1.42
Splice	Accuracy	71.65	10.15	73.65	10.59	68.27	13.24	69.32	8.69	73.72	12.36	68.69	10.38	68.92	9.71	76.41	2.63	76.25	4.95
Best subset	Tree size	25.80	13.06	27.40	15.44	19.80	15.53	23.60	12.27	27.80	13.63	22.60	13.04	26.40	12.95	39.00	3.87	32.00	4.24
	ALR	23.80	39.31	24.32	40.59	44.65	49.66	24.31	39.31	24.33	41.10	25.60	41.04	24.17	40.39	3.38	0.31	4.19	0.59
Segment	Accuracy	88.12	3.08	90.63	1.11	89.52	0.79	82.44	5.19	87.19	3.28	85.92	3.47	84.42	4.63	80.64	3.60	84.89	3.51
Best subset	Tree size	17.40	2.33	17.80	1.19	18.60	1.57	14.60	1.60	17.00	1.39	16.60	2.03	15.80	1.17	15.40	1.57	16.20	2.12
	ALR	11.50	1.53	11.10	0.68	10.79	0.87	13.65	1.31	11.64	0.86	12.24	1.47	12.60	0.92	12.90	1.06	12.49	1.48
Letter	Accuracy	58.89	2.64	58.67	3.05	60.63	1.81	57.69	4.64	60.24	3.03	57.53	4.00	57.28	3.41	57.36	3.26	60.58	2.41
Best subset	Tree size	257.0	23.43	262.6	11.19	269.4	17.43	264.6	27.35	279.4	21.38	262.6	19.27	259.0	12.48	264.6	23.90	285.0	20.98
	ALR	0.82	0.07	0.80	0.04	0.78	0.06	0.80	0.09	0.75	0.05	0.81	0.06	0.81	0.04	0.80	0.07	0.74	0.05
Sick	Accuracy	95.26	1.40	96.01	1.68	94.19	1.22	96.11	1.84	96.43.	1.18	95.56	1.32	97.30	1.01	96.62	1.34	96.41	1.37
Best subset	Tree size	3.00	2.53	4.40	3.02	2.60	2.11	5.80	1.18	5.80	1.09	5.00	0.47	6.20	1.03	5.60	2.54	6.40	1.40
	ALR	75.70	36.92	58.80	40.19	77.80	34.83	32.14	4.63	31.28	4.15	34.86	0.09	29.67	3.95	42.45	32.87	28.14	6.70

Table A2 Results of top 25% subsets when fitness weight *a* and selection constant *Q* are varied

Dataset		α=2.0						α=4.0						**α=6.0**					
		Q=1.5		Q=2.0		Q=2.5		Q=1.5		Q=2.0		Q=2.5		Q=1.5		Q=2.0		**Q=2.5**	
		Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.
Scheduling	Accuracy	97.25	0.50	97.56	0.34	97.58	0.31	97.52	0.62	97.47	0.40	97.73	0.39	97.47	0.37	97.89	0.82	97.47	0.68
Top 25%	Tree size	23.80	2.99	29.80	2.71	27.00	3.79	23.80	4.66	27.40	6.37	28.60	3.67	26.20	0.98	27.00	4.56	22.20	3.92
	ALR	8.19	1.05	6.55	0.62	7.27	0.96	8.37	1.66	7.36	1.42	6.86	0.82	7.36	0.27	7.37	1.42	8.93	1.84
Splice	Accuracy	85.66	1.19	85.96	1.48	85.08	1.33	85.56	1.55	85.13	1.41	86.18	2.39	86.22	1.71	86.05	1.03	83.63	2.39
Top 25%	Tree size	81.00	9.40	77.40	11.65	70.00	6.16	82.40	10.97	72.40	5.87	76.20	8.59	87.40	16.70	79.40	5.86	81.00	10.58
	ALR	1.56	0.17	1.65	0.27	1.79	0.16	1.54	0.20	1.72	0.15	1.65	0.18	1.48	0.32	1.57	0.13	1.56	0.19
Segment	Accuracy	91.73	1.30	91.59	1.23	91.70	0.71	91.94	1.84	90.66	1.00	92.08	1.57	90.76	0.85	91.66	0.57	91.14	0.51
Top 25%	Tree size	29.80	3.71	29.00	3.89	30.20	2.22	29.00	4.06	29.40	2.80	31.00	3.66	33.80	6.82	31.40	3.63	30.60	3.31
	ALR	6.59	0.79	6.78	0.91	6.44	0.43	6.79	0.93	6.63	0.61	6.32	0.67	5.96	1.09	6.24	0.64	6.39	0.65
Letter	Accuracy	72.70	0.72	73.57	1.16	72.25	1.14	72.59	0.68	73.53	0.94	73.51	0.60	72.25	0.99	72.46	1.32	72.80	0.63
Top 25%	Tree size	629.0	23.82	635.0	19.01	638.6	17.01	619.4	32.06	620.6	19.57	619.0	7.94	637.8	28.3	637.4	12.45	609.0	17.18
	ALR	0.32	0.01	0.31	0.01	0.31	0.01	0.32	0.02	0.32	0.01	0.32	0.00	0.31	0.01	0.31	0.01	0.33	0.01
Sick	Accuracy	97.44	0.31	98.14	0.36	98.20	0.37	98.09	0.34	97.54	0.40	98.06	0.32	98.12	0.50	97.69	0.50	97.63	0.41
Top 25%	Tree size	13.20	4.12	13.80	2.72	10.20	3.51	8.80	2.02	11.40	4.08	10.80	3.37	12.40	3.81	11.40	4.75	10.00	2.23
	ALR	12.33	2.95	12.44	3.81	18.50	6.24	20.50	4.58	17.04	5.99	17.86	5.19	14.75	5.49	17.82	7.00	18.05	4.06

Table A3 Results of top 50% subsets when fitness weight *a* and selection constant *Q* are varied

Dataset		α=2.0						α=4.0						**α=6.0**			
		Q=1.5		Q=2.0		Q=2.5		Q=1.5		Q=2.0		Q=2.5		Q=1.5		Q=2.0		**Q=2.5**	
		Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.
Scheduling	Accuracy	98.41	0.26	98.38	0.12	98.43	0.39	98.83	0.20	98.73	0.14	98.39	0.19	98.34	0.22	98.55	0.24	98.56	0.17
Top 50%	Tree size	44.20	3.25	37.80	4.12	39.00	6.32	37.80	3.49	40.20	2.04	42.60	6.62	43.00	3.35	40.20	2.04	40.60	2.33
	ALR	4.45	0.33	5.21	0.52	5.12	0.78	5.20	0.47	4.87	0.24	4.70	0.73	4.57	0.36	4.87	0.24	4.82	0.28
Splice	Accuracy	90.32	0.70	91.24	0.73	90.15	0.93	91.69	1.30	90.73	0.84	90.58	1.03	91.37	0.64	89.87	2.10	90.11	0.91
Top 50%	Tree size	136.0	8.49	124.8	8.04	128.4	6.83	135.6	13.89	136.4	5.98	117.2	20.48	130.4	4.68	124.0	12.06	138.2	11.86
	ALR	0.92	0.06	1.00	0.06	0.97	0.05	0.93	0.10	0.91	0.04	1.09	0.19	0.96	0.03	1.01	0.10	0.91	0.07
Segment	Accuracy	93.39	0.25	94.26	0.92	94.50	0.70	94.29	0.73	93.77	1.11	93.56	1.17	93.49	0.68	94.29	0.96	93.84	0.79
Top 50%	Tree size	42.20	5.15	46.60	3.59	53.40	5.33	45.40	3.64	45.00	3.45	47.00	5.13	46.60	3.11	46.20	3.58	45.00	6.86
	ALR	4.70	0.59	4.22	0.29	3.71	0.37	4.33	0.31	4.37	0.30	4.21	0.44	4.22	0.26	4.26	0.32	4.45	0.68
Letter	Accuracy	79.63	0.58	79.87	0.55	79.37	0.46	79.43	0.53	80.07	0.73	80.08	0.52	79.44	1.04	79.30	0.50	80.18	0.54
Top 50%	Tree size	1031	14.31	1059	24.76	1058	26.85	1050	26.74	1054	17.66	1065	14.92	1077	40.16	1081	25.99	1048	15.17
	ALR	0.19	0.00	0.19	0.00	0.19	0.00	0.19	0.00	0.19	0.00	0.19	0.00	0.19	0.01	0.18	0.00	0.19	0.00
Sick	Accuracy	97.98	0.23	98.43	0.27	98.33	0.33	98.19	0.34	97.63	0.42	98.11	0.32	98.22	0.51	97.95	0.40	97.76	0.38
Top 50%	Tree size	16.80	5.00	20.20	2.72	15.80	4.37	15.60	3.84	17.00	4.56	12.60	5.69	14.00	5.43	14.40	4.70	15.60	2.45
	ALR	10.25	2.67	8.67	1.38	11.58	4.36	11.98	4.25	11.02	4.55	16.54	6.58	14.19	6.23	13.48	5.35	11.35	2.42

Table A4 Results of top 75% subsets when fitness weight *a* and selection constant *Q* are varied

Dataset		α=2.0						α=4.0						**α=6.0**			
		Q=1.5		Q=2.0		Q=2.5		Q=1.5		Q=2.0		Q=2.5		Q=1.5		Q=2.0		**Q=2.5**	
		Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.
Scheduling	Accuracy	98.61	0.20	98.72	0.24	98.68	0.33	98.92	0.15	98.82	0.16	98.50	0.24	98.75	0.20	98.87	0.22	98.73	0.20
Top 75%	Tree size	48.60	4.08	45.40	3.20	48.60	5.43	48.20	6.14	47.80	3.71	49.80	2.99	47.40	5.28	44.60	3.20	45.80	8.45
	ALR	4.06	0.37	4.33	0.30	4.08	0.42	4.12	0.47	4.12	0.29	3.95	0.22	4.18	0.42	4.41	0.31	4.42	0.83
Splice	Accuracy	91.15	0.83	92.20	0.42	91.07	0.90	92.89	0.59	91.07	0.77	91.62	0.66	92.39	0.47	91.47	0.64	91.02	0.83
Top 75%	Tree size	141.8	3.76	149.2	15.94	131.0	7.51	148.8	9.01	147.2	11.06	130.4	14.13	147.0	6.48	140.8	9.31	148.8	7.02
	ALR	0.88	0.02	0.84	0.08	0.95	0.05	0.84	0.05	0.85	0.06	0.96	0.10	0.85	0.03	0.89	0.05	0.84	0.04
Segment	Accuracy	93.70	0.85	95.22	0.78	94.50	0.70	95.50	0.76	94.36	0.92	95.12	0.57	93.60	1.23	94.98	0.55	95.05	0.99
Top 75%	Tree size	48.20	1.60	52.60	2.71	53.40	5.33	55.80	2.29	53.40	8.07	51.00	4.90	49.40	4.00	54.20	5.53	52.60	3.60
	ALR	4.07	0.13	3.74	0.18	3.71	0.37	3.53	0.13	3.76	0.59	3.88	0.38	3.99	0.32	3.66	0.39	3.75	0.25
Letter	Accuracy	81.42	0.68	81.78	0.72	81.49	0.45	81.14	0.57	81.88	0.73	81.60	0.64	81.72	0.48	81.61	0.45	82.15	0.53
Top 75%	Tree size	1239	14.61	1244	13.47	1226	11.12	1233	30.48	1265	29.89	1217	17.58	1247	28.50	1251	38.57	1250	22.43
	ALR	0.16	0.00	0.16	0.00	0.16	0.00	0.16	0.00	0.16	0.00	0.16	0.00	0.16	0.00	0.16	0.00	0.16	0.00
Sick	Accuracy	98.03	0.45	98.27	0.48	98.51	0.34	98.20	0.32	97.55	0.48	98.14	0.33	98.29	0.55	98.11	0.42	98.16	0.38
Top 75%	Tree size	17.40	4.72	24.40	2.83	22.40	2.68	18.80	6.83	20.00	2.14	16.80	4.58	17.20	3.33	21.00	6.31	21.20	3.20
	ALR	10.15	2.21	7.11	0.70	8.10	1.16	10.72	4.08	8.38	0.63	11.27	3.62	10.64	3.06	8.99	2.90	8.05	0.93

Table A5 Results of best subset when crossover rate *c* and mutation rate *m* are varied

Dataset		***c*=0.6**						*c*=0.7						*c*=0.8					
		m=0.01		*m*=0.05		***m*=0.09**		*m*=0.01		*m*=0.05		*m*=0.09		*m*=0.01		*m*=0.05		*m*=0.09	
		Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.
Scheduling	Accuracy	95.43	0.47	95.83	0.98	95.53	0.71	95.83	0.56	95.66	1.09	96.01	0.77	95.04	0.77	95.21	0.62	95.05	1.13
Best subset	Tree size	9.40	2.65	11.80	1.22	6.20	1.67	7.80	2.52	9.00	2.63	8.20	2.16	8.20	2.14	8.20	2.14	9.80	1.18
	ALR	22.00	7.17	16.48	1.38	30.51	5.86	25.89	7.47	22.55	6.56	24.22	6.04	24.15	6.14	24.24	6.05	19.54	1.69
Splice	Accuracy	72.77	7.80	72.17	11.15	79.50	3.46	68.83	10.18	72.26	10.56	62.96	13.25	70.81	10.75	69.68	10.42	71.35	9.80
Best subset	Tree size	27.00	11.15	26.80	13.83	33.00	2.68	28.00	13.89	27.20	13.88	13.80	15.82	27.60	13.60	23.20	12.09	26.60	13.40
	ALR	7.45	6.74	24.17	40.39	4.01	0.19	24.26	40.87	23.85	39.78	64.75	49.67	24.68	41.74	24.93	40.54	24.09	40.17
Segment	Accuracy	84.61	2.51	84.05	1.42	86.68	4.26	86.26	2.82	85.36	2.98	85.30	2.87	84.18	2.06	84.10	2.29	86.10	2.88
Best subset	Tree size	16.60	0.80	14.20	1.06	17.00	1.35	17.00	2.25	17.00	1.43	17.00	2.26	13.80	1.19	14.60	1.57	15.80	2.12
	ALR	11.96	0.68	13.95	1.05	11.80	0.85	11.84	1.83	11.86	0.80	11.76	1.12	14.30	0.89	13.80	1.41	12.89	1.69
Letter	Accuracy	62.02	1.41	59.82	2.10	62.50	1.82	58.46	3.22	59.46	2.34	59.35	2.09	54.25	3.04	56.17	1.04	57.16	2.74
Best subset	Tree size	302.2	8.06	288.6	13.53	311.8	25.87	277.8	25.36	278.6	21.97	275.0	14.07	240.2	21.74	253.4	5.95	238.6	13.86
	ALR	0.70	0.02	0.73	0.03	0.68	0.06	0.76	0.07	0.76	0.06	0.76	0.04	0.88	0.08	0.83	0.02	0.88	0.05
Sick	Accuracy	95.44	1.40	95.64	1.62	96.38	1.43	95.29	2.15	96.22	2.05	94.98	1.45	94.53	0.62	95.60	1.51	95.43	1.30
Best subset	Tree size	5.60	2.80	5.00	2.32	5.40	2.43	4.20	2.16	4.60	1.03	3.20	2.89	2.20	2.52	3.80	2.50	4.20	1.75
	ALR	43.39	33.18	45.49	30.26	44.19	32.00	51.10	28.91	38.68	6.84	73.01	38.75	89.16	31.72	61.68	35.89	48.87	27.21

Table A6 Results of top 25% subsets when crossover rate *c* and mutation rate *m* are varied

Dataset		***c*=0.6**						*c*=0.7						*c*=0.8					
		m=0.01		*m*=0.05		***m*=0.09**		*m*=0.01		*m*=0.05		*m*=0.09		*m*=0.01		*m*=0.05		*m*=0.09	
		Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.
Scheduling	Accuracy	97.43	0.78	97.79	0.66	97.25	0.52	98.12	0.46	97.60	0.55	97.29	0.51	97.45	0.45	97.48	0.43	98.08	0.49
Top 25%	Tree size	27.40	5.12	27.00	3.93	25.40	3.32	28.60	3.94	26.20	3.37	23.40	2.47	27.00	4.44	29.00	4.48	26.60	4.55
	ALR	7.27	1.30	7.28	1.03	7.68	0.90	6.88	0.96	7.46	0.86	8.27	0.77	7.32	1.15	6.81	1.00	7.46	1.36
Splice	Accuracy	84.04	2.11	85.62	1.29	82.14	2.53	85.09	1.21	84.96	2.31	83.44	1.61	85.08	1.36	83.89	1.86	85.15	1.73
Top 25%	Tree size	76.60	6.22	72.00	6.76	78.40	7.69	71.40	16.47	82.60	6.95	58.80	12.77	73.00	10.85	70.40	7.82	76.00	14.50
	ALR	1.64	0.14	1.74	0.16	1.61	0.18	1.82	0.32	1.52	0.13	2.21	0.41	1.74	0.25	1.79	0.21	1.70	0.35
Segment	Accuracy	92.87	0.95	90.87	1.33	92.87	0.73	91.45	0.87	90.45	1.26	92.98	0.96	92.04	1.57	92.21	0.89	92.53	1.22
Top 25%	Tree size	28.60	3.44	33.40	3.97	33.00	2.00	31.80	2.79	31.80	2.81	27.40	1.67	31.80	2.77	31.40	3.95	27.40	2.15
	ALR	6.86	0.85	5.88	0.58	5.90	0.31	6.14	0.50	6.14	0.55	7.06	0.38	6.14	0.50	6.26	0.75	7.07	0.46
Letter	Accuracy	74.02	0.66	72.69	0.83	73.36	0.78	72.57	1.02	72.93	0.64	73.59	0.78	72.71	0.89	73.62	0.73	72.01	0.63
Top 25%	Tree size	629.0	10.43	609.4	10.10	628.2	19.99	624.6	17.25	627.4	15.83	619.8	15.52	629.8	16.52	633.8	26.79	610.2	11.02
	ALR	0.32	0.01	0.33	0.01	0.32	0.01	0.32	0.01	0.32	0.01	0.32	0.01	0.32	0.01	0.32	0.01	0.33	0.01
Sick	Accuracy	98.41	0.26	98.55	0.33	98.08	0.39	97.89	0.53	97.38	0.60	97.49	0.64	97.74	0.67	97.55	0.45	97.57	0.39
Top 25%	Tree size	12.60	3.56	9.20	1.81	11.20	3.48	11.20	3.41	10.80	3.75	10.80	2.30	8.00	1.44	10.00	1.64	10.00	1.24
	ALR	14.08	5.68	19.50	4.00	15.53	4.11	16.08	5.65	17.08	6.62	15.39	3.20	21.86	4.25	17.71	2.80	16.21	3.16

Table A7 Results of top 50% subsets when crossover rate *c* and mutation rate *m* are varied

Dataset		***c*=0.6**						*c*=0.7						*c*=0.8					
		m=0.01		*m*=0.05		***m*=0.09**		*m*=0.01		*m*=0.05		*m*=0.09		*m*=0.01		*m*=0.05		*m*=0.09	
		Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.
Scheduling	Accuracy	98.57	0.32	98.50	0.73	98.13	0.59	98.51	0.48	98.39	0.42	97.29	0.51	98.40	0.37	98.24	0.54	98.61	0.48
Top 50%	Tree size	35.00	4.73	37.40	5.93	34.20	4.07	42.60	3.37	37.80	2.83	36.60	3.95	39.40	3.98	43.80	4.91	42.20	3.40
	ALR	5.65	0.71	5.32	0.76	5.75	0.61	4.61	0.36	5.18	0.34	5.38	0.58	5.00	0.46	4.52	0.55	4.66	0.34
Splice	Accuracy	90.73	1.01	91.17	0.70	89.49	1.07	90.66	0.83	90.66	0.89	89.29	0.60	88.85	1.06	89.38	1.59	89.94	0.83
Top 50%	Tree size	132.4	14.83	132.8	7.32	119.8	11.03	133.6	11.10	125.0	10.52	124.6	15.40	115.4	19.02	130.0	9.43	128.0	14.96
	ALR	0.95	0.11	0.94	0.05	1.05	0.09	0.94	0.08	1.00	0.08	1.01	0.12	1.11	0.17	0.96	0.07	0.98	0.10
Segment	Accuracy	95.61	0.50	92.56	1.11	93.88	1.22	94.78	1.44	92.91	1.43	94.46	1.01	94.50	0.69	94.33	0.80	93.67	0.78
Top 50%	Tree size	46.20	5.46	43.00	3.73	49.80	5.52	49.00	7.34	47.00	4.37	42.20	5.53	45.40	5.53	52.20	7.51	44.60	5.56
	ALR	4.30	0.55	4.57	0.36	3.98	0.40	4.09	0.61	4.20	0.41	4.71	0.61	4.37	0.48	3.84	0.62	4.45	0.56
Letter	Accuracy	79.81	1.04	79.57	1.02	79.85	0.84	79.47	0.73	79.96	0.44	79.96	0.43	79.63	0.83	80.00	0.49	78.97	0.62
Top 50%	Tree size	1069	19.68	1040	18.55	1069	33.22	1041	29.84	1059	12.08	1044	28.99	1058	25.93	1034	25.01	1040	24.26
	ALR	0.19	0.00	0.19	0.00	0.19	0.01	0.19	0.01	0.19	0.00	0.19	0.01	0.19	0.00	0.19	0.00	0.19	0.00
Sick	Accuracy	98.81	0.15	98.89	0.30	98.47	0.34	98.31	0.34	97.69	0.50	98.16	0.41	97.93	0.50	97.73	0.33	97.73	0.35
Top 50%	Tree size	20.00	3.35	20.80	2.20	18.80	2.57	19.00	7.36	13.40	3.65	23.00	5.93	20.60	8.47	16.20	2.06	18.40	1.63
	ALR	8.48	1.10	8.12	0.66	9.37	1.06	11.41	6.98	12.90	3.96	8.00	1.76	10.68	5.62	10.36	1.30	8.97	0.62

Table A8 Results of top 75% subsets when crossover rate *c* and mutation rate *m* are varied

Dataset		***c*=0.6**						*c*=0.7						*c*=0.8					
		m=0.01		*m*=0.05		***m*=0.09**		*m*=0.01		*m*=0.05		*m*=0.09		*m*=0.01		*m*=0.05		*m*=0.09	
		Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.
Scheduling	Accuracy	98.70	0.20	98.88	0.52	98.50	0.52	98.91	0.29	98.80	0.40	98.73	0.36	98.79	0.45	98.26	0.56	98.68	0.44
Top 75%	Tree size	45.00	3.35	45.80	4.39	40.60	2.81	48.20	4.79	42.60	4.38	44.20	5.39	42.60	7.05	47.80	3.89	46.20	4.21
	ALR	4.37	0.31	4.31	0.39	4.83	0.30	4.10	0.39	4.64	0.50	4.48	0.52	4.71	0.74	4.12	0.31	4.27	0.40
Splice	Accuracy	91.03	0.82	92.14	0.73	90.15	0.55	91.80	0.56	90.75	0.81	90.66	0.70	90.77	0.76	90.53	1.21	91.43.	0.52
Top 75%	Tree size	138.8	13.33	147.4	13.32	136.4	5.23	151.2	12.54	148.4	12.82	141.6	6.42	139.2	9.31	141.0	9.67	144.2	13.89
	ALR	0.90	0.08	0.85	0.07	0.91	0.03	0.83	0.06	0.84	0.07	0.88	0.04	0.90	0.06	0.89	0.06	0.87	0.08
Segment	Accuracy	95.92	0.86	93.56	0.98	95.54	0.95	94.86	0.50	94.01	0.85	95.05	0.77	94.64	1.01	94.36	0.85	94.71	1.13
Top 75%	Tree size	57.00	4.38	51.00	5.59	57.40	3.82	54.20	5.23	52.20	5.10	49.80	4.60	52.20	3.14	56.20	8.94	49.80	3.54
	ALR	3.47	0.26	3.89	0.42	3.44	0.22	3.66	0.37	3.79	0.35	3.97	0.35	3.77	0.21	3.59	0.63	3.95	0.25
Letter	Accuracy	82.00	0.59	81.87	0.72	82.02	0.60	81.27	0.65	81.75	0.67	81.82	0.59	81.81	0.68	82.11	0.53	80.72	0.91
Top 75%	Tree size	1284	44.48	1228	28.60	1237	35.07	1209	25.32	1240	20.78	1228	26.39	1230	13.44	1228	26.54	1263	41.74
	ALR	0.16	0.01	0.16	0.00	0.16	0.00	0.17	0.00	0.16	0.00	0.16	0.00	0.16	0.00	0.16	0.01	0.16	0.01
Sick	Accuracy	98.87	0.22	98.95	0.39	98.55	0.42	98.47	0.35	97.85	0.57	98.17	0.42	98.17	0.37	97.85	0.30	97.73	0.31
Top 75%	Tree size	25.60	4.45	24.80	6.22	29.80	4.54	24.80	6.59	18.60	2.09	28.40	4.50	20.00	7.08	20.80	4.81	19.80	5.32
	ALR	6.89	0.90	7.25	1.45	6.17	0.98	7.32	1.71	9.22	1.14	6.32	1.04	10.03	3.59	8.35	1.50	9.58	3.65

Table A9 Results of best subset when number of subsets *M* and number of GA generations *G* are varied

Dataset		*M*=5						***M*=10**						*M=15*					
		G=20		*G*=30		*G*=40		***G*=20**		*G*=30		*G*=40		*G*=20		*G*=30		*G*=40	
		Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.
Scheduling	Accuracy	97.43	0.62	97.54	0.83	96.72	1.22	96.30	0.61	95.97	1.20	95.99	1.08	94.34	0.71	95.21	1.05	94.76	0.71
Best subset	Tree size	17.40	2.33	18.60	2.44	19.00	3.61	8.60	1.65	9.40	3.02	11.00	2.95	3.80	2.41	5.80	1.25	4.60	2.10
	ALR	12.30	1.82	11.37	1.46	11.44	2.33	22.59	3.49	22.27	7.49	18.63	4.68	59.92	34.99	30.85	4.15	46.54	28.69
Splice	Accuracy	82.86	3.22	82.47	3.90	82.56	2.78	60.26	11.56	60.46	9.43	78.25	4.00	52.94	1.56	72.45	3.11	58.50	8.68
Best subset	Tree size	66.80	7.36	60.80	7.68	67.00	14.14	14.40	16.53	11.60	13.08	33.00	2.97	1.00	1.82	23.00	4.36	8.00	8.75
	ALR	2.08	0.24	2.26	0.32	2.20	0.65	65.01	50.00	64.59	48.82	3.96	0.30	93.11	2.02	5.72	0.91	64.68	47.16
Segment	Accuracy	92.58	1.12	92.05	1.06	90.09	1.33	88.38	1.15	81.27	3.26	82.17	3.73	82.77	3.31	84.69	1.96	80.14	4.40
Best subset	Tree size	22.20	2.04	24.20	1.66	23.40	3.47	17.00	1.42	14.60	0.99	15.40	1.16	13.40	0.96	13.80	0.88	13.00	0.48
	ALR	9.50	0.94	8.88	0.53	9.45	1.69	11.78	0.82	13.61	0.73	12.88	0.66	14.38	0.73	13.89	0.88	14.70	0.37
Letter	Accuracy	71.10	3.65	69.28	2.13	68.77	3.18	62.40	2.46	59.63	0.71	59.52	2.51	54.75	1.74	51.90	3.66	51.58	4.61
Best subset	Tree size	507.8	73.22	487.4	38.48	484.2	39.23	302.2	22.58	279.0	8.56	278.6	23.40	210.6	7.92	204.6	17.32	191.8	12.62
	ALR	0.45	0.08	0.46	0.04	0.46	0.04	0.70	0.06	0.75	0.03	0.76	0.01	0.98	0.04	1.01	0.08	1.07	0.07
Sick	Accuracy	97.68	0.91	97.37	1.51	98.02	0.38	95.31	1.22	94.88	0.86	95.61	1.24	94.16	2.13	94.42	0.43	95.86	1.50
Best subset	Tree size	6.60	0.80	7.80	1.67	8.00	1.31	3.80	2.43	2.60	2.04	4.60	2.03	2.20	1.20	1.00	0.64	3.80	1.72
	ALR	29.31	3.62	25.97	5.97	24.12	4.76	61.15	35.44	76.94	33.88	46.71	28.53	73.53	26.27	92.86	1.53	51.51	26.24

Table A10 Results of top 25% subsets when number of subsets *M* and number of GA generations *G* are varied

Dataset		*M*=5						***M*=10**						*M=15*					
		G=20		*G*=30		*G*=40		***G*=20**		*G*=30		*G*=40		*G*=20		*G*=30		*G*=40	
		Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.
Scheduling	Accuracy	97.71	0.42	97.82	0.58	97.49	0.63	98.01	0.50	97.58	0.36	97.22	0.61	97.67	0.76	97.70	0.83	97.44	0.66
Top 25%	Tree size	24.20	2.71	25.40	2.78	25.40	5.03	27.80	3.78	25.00	4.96	28.20	3.40	26.60	3.32	24.60	3.10	25.80	3.80
	ALR	8.03	0.89	7.66	0.81	7.86	1.50	7.06	0.90	7.96	1.45	6.93	0.74	7.36	0.95	7.92	0.95	7.61	1.05
Splice	Accuracy	86.67	0.73	83.53	1.73	85.28	1.69	85.71	1.96	86.20	1.13	85.38	1.36	86.18	1.35	81.94	1.66	85.30	2.16
Top 25%	Tree size	98.20	6.68	74.80	11.30	74.20	16.94	71.80	18.96	66.80	7.88	75.80	9.18	86.00	15.77	63.60	12.85	83.40	10.72
	ALR	1.27	0.08	1.71	0.28	1.75	0.34	1.87	0.51	1.88	0.22	1.66	0.19	1.49	0.29	2.05	0.51	1.51	0.21
Segment	Accuracy	90.76	1.52	90.97	1.66	91.31	1.89	92.63	0.78	90.21	1.55	89.65	1.57	90.00	1.32	90.90	1.64	89.90	0.82
Top 25%	Tree size	28.60	1.96	31.00	5.17	29.80	4.90	33.40	3.26	29.40	5.52	29.40	3.80	27.00	5.71	30.60	3.37	29.00	3.89
	ALR	6.79	0.44	6.42	1.06	6.63	0.90	5.87	0.55	6.77	1.11	6.68	0.84	7.52	1.89	6.39	0.65	6.78	0.91
Letter	Accuracy	72.62	0.63	73.26	1.17	73.64	0.92	73.87	0.76	73.53	0.71	73.06	0.83	72.30	0.62	74.24	0.98	73.53	1.03
Top 25%	Tree size	631.4	18.97	618.6	14.04	626.2	27.09	623.0	7.94	629.0	25.32	621.8	13.45	629.0	14.59	633.8	15.79	629.4	18.29
	ALR	0.32	0.01	0.32	0.01	0.32	0.01	0.32	0.00	0.32	0.01	0.32	0.01	0.32	0.01	0.32	0.01	0.32	0.01
Sick	Accuracy	97.04	0.40	97.93	0.57	97.42	0.54	98.03	0.69	97.47	0.59	97.33	0.61	97.82	0.50	97.82	0.49	97.92	0.47
Top 25%	Tree size	8.60	1.36	12.40	4.74	11.60	3.18	12.40	2.39	12.40	3.40	11.60	2.38	10.20	3.14	12.00	5.24	12.00	2.84
	ALR	19.71	4.79	15.49	6.23	15.67	5.80	14.17	3.33	14.23	3.88	14.94	4.16	17.86	5.19	16.52	7.19	14.72	4.38

Table A11 Results of top 50% subsets when number of subsets *M* and number of GA generations *G* are varied

Dataset		*M*=5						***M*=10**						*M=15*					
		G=20		*G*=30		*G*=40		***G*=20**		*G*=30		*G*=40		*G*=20		*G*=30		*G*=40	
		Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.
Scheduling	Accuracy	98.23	0.13	98.14	0.58	98.42	0.39	98.58	0.43	98.24	0.39	98.49	0.37	98.57	0.61	98.44	0.52	98.05	0.39
Top 50%	Tree size	39.80	4.12	36.60	3.63	35.40	3.98	41.00	3.47	40.20	5.94	35.40	5.04	37.40	6.67	41.00	4.15	35.40	4.57
	ALR	4.95	0.51	5.37	0.52	5.56	0.64	4.79	0.41	4.95	0.70	5.60	0.76	5.36	0.91	4.81	0.46	5.59	0.80
Splice	Accuracy	90.04	1.19	89.36	0.81	90.02	0.85	91.20	1.30	89.42	0.94	90.90	0.92	88.72	0.90	88.70	1.03	90.28	0.61
Top 50%	Tree size	121.8	8.08	117.2	8.46	119.6	14.74	116.2	9.30	118.2	20.80	135.4	11.66	122.6	11.12	118.6	9.23	131.6	7.49
	ALR	1.03	0.07	1.06	0.07	1.05	0.14	1.08	0.09	1.08	0.19	0.93	0.08	1.02	0.08	1.05	0.09	0.95	0.05
Segment	Accuracy	93.25	0.51	94.22	0.84	92.94	1.16	94.78	1.26	93.56	0.89	92.98	0.77	92.53	0.79	93.32	0.48	92.39	1.18
Top 50%	Tree size	37.80	6.01	43.40	3.07	43.80	3.55	47.80	2.32	46.60	4.15	44.60	5.49	40.60	3.94	38.20	2.23	44.60	6.70
	ALR	5.30	0.97	4.58	0.31	4.49	0.35	4.11	0.17	4.23	0.36	4.45	0.53	4.85	0.43	5.12	0.27	4.49	0.69
Letter	Accuracy	77.69	1.04	78.05	0.69	78.70	0.93	80.17	0.60	79.70	0.75	79.71	0.83	78.62	0.86	80.16	0.52	79.79	0.55
Top 50%	Tree size	937.4	17.77	933.8	7.89	949.8	25.12	1071	29.70	1055	42.17	1042	22.79	1048	26.24	1038	14.30	1039	20.56
	ALR	0.21	0.00	0.21	0.00	0.21	0.01	0.19	0.01	0.19	0.01	0.19	0.00	0.19	0.00	0.19	0.00	0.19	0.00
Sick	Accuracy	97.28	0.29	98.06	0.40	97.80	0.38	98.47	0.52	97.97	0.35	98.00	0.31	97.98	0.43	98.33	0.35	98.51	0.41
Top 50%	Tree size	10.60	3.83	12.00	3.28	12.80	3.58	15.60	4.50	19.80	3.35	17.00	2.51	14.00	5.81	16.20	5.37	21.20	5.20
	ALR	17.68	5.47	14.82	3.60	14.26	4.74	11.56	4.56	8.40	1.33	10.16	2.13	15.40	6.37	11.29	3.62	8.29	1.75

Table A12 Results of top 75% subsets when number of subsets *M* and number of GA generations *G* are varied

Dataset		*M*=5						***M*=10**						*M=15*					
		G=20		*G*=30		*G*=40		***G*=20**		*G*=30		*G*=40		*G*=20		*G*=30		*G*=40	
		Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.
Scheduling	Accuracy	98.56	0.42	98.91	0.27	98.84	0.30	98.82	0.38	98.59	0.42	98.75	0.49	98.82	0.33	98.81	0.32	98.51	0.51
Top 75%	Tree size	42.60	2.94	44.20	4.09	44.60	3.09	45.40	4.15	49.80	4.92	42.60	5.90	45.80	6.83	45.80	4.60	48.20	5.42
	ALR	4.61	0.32	4.46	0.37	4.40	0.28	4.34	0.40	3.98	0.42	4.68	0.69	4.37	0.65	4.31	0.39	4.11	0.46
Splice	Accuracy	91.09	0.77	90.86	1.09	91.62	0.80	92.84	0.80	92.33	1.17	92.33	0.60	90.79	0.95	90.17	0.80	91.52	0.39
Top 75%	Tree size	146.8	12.12	141.8	4.70	148.0	12.07	146.2	6.68	151.2	10.11	152.8	5.91	145.4	9.49	148.0	10.47	138.2	11.5
	ALR	0.85	0.07	0.88	0.03	0.84	0.07	0.85	0.04	0.82	0.06	0.82	0.03	0.86	0.06	0.84	0.05	0.90	0.07
Segment	Accuracy	93.63	0.40	94.71	1.29	94.84	0.99	95.19	0.35	94.26	1.10	93.15	1.41	93.39	0.62	94.53	1.18	93.29	0.72
Top 75%	Tree size	45.00	4.00	44.20	9.02	45.80	5.09	53.40	4.72	53.40	3.69	51.40	6.33	47.80	2.26	52.20	4.74	49.80	2.33
	ALR	4.38	0.39	4.52	0.78	4.32	0.42	3.70	0.31	3.69	0.25	3.87	0.42	4.11	0.17	3.79	0.34	3.94	0.16
Letter	Accuracy	80.54	0.40	80.87	0.66	81.26	1.07	82.28	0.38	81.89	0.58	82.05	0.50	80.77	0.52	82.80	0.84	82.67	0.79
Top 75%	Tree size	1137	33.58	1145	31.29	1174	35.89	1244	15.58	1228	27.61	1232	26.84	1304	36.67	1321	21.41	1298	28.17
	ALR	0.18	0.01	0.17	0.00	0.17	0.01	0.16	0.00	0.16	0.00	0.16	0.00	0.15	0.00	0.15	0.00	0.15	0.00
Sick	Accuracy	97.29	0.24	98.49	0.31	97.82	0.39	98.57	0.42	97.97	0.37	98.14	0.29	98.30	0.34	98.46	0.32	98.59	0.35
Top 75%	Tree size	13.60	4.08	21.80	3.96	14.80	3.80	23.40	5.57	23.40	2.65	21.20	4.75	27.00	9.44	18.20	5.24	21.40	3.69
	ALR	14.48	4.22	8.09	1.06	12.52	4.23	7.49	1.52	7.19	0.61	8.18	1.32	8.54	5.75	10.65	2.78	8.27	1.27

Table A13 Instance selection vs. different decision tree pruning techniques

Dataset	Minimum leaf instances	Minimum leaf instance setting without pruning		**Instance selection without pruning***		Reduced error pruning with subtree raising		Reduced error pruning with subtree replacement		C4.5 pruning with subtree raising		C4.5 pruning with subtree replacement	
		Tree size	Accuracy	Tree size	Accuracy	Tree size	Accuracy	Tree size	Accuracy	Tree size	Accuracy	Tree size	Accuracy
Letter	5	1637	85.9	857.0	81.6	965	82.8	965	82.7	1499	85.9	1519	85.8
	10	987	83.2	509.0	76.7	647	80.1	647	80.1	931	83.2	943	83.2
	15	755	81.3	348.6	73.0	409	75.7	503	76.1	589	79.2	717	81.2
Scheduling	5	61	99.4	33.0	98.6	43	99.1	43	99.1	55	99.4	57	99.4
	10	47	99.1	24.2	98.1	37	98.7	37	98.7	41	99.1	45	99.1
	15	43	98.9	18.2	97.6	29	97.9	33	98.4	35	98.6	41	98.9
Splice	5	332	92.7	172.0	90.5	154	92.7	154	92.7	171	94.4	171	94.4
	10	213	91.2	116.2	87.9	142	90.9	142	90.9	134	92.5	134	92.5
	15	156	90.3	66.4	85.3	74	86.5	94	89.4	108	89.9	136	90.8
Segment	5	75	96.0	36.2	94.6	43	95.0	43	95.0	59	96.0	59	95.9
	10	47	95.1	25.0	92.4	39	94.9	39	94.9	47	95.0	47	95.1
	15	39	95.0	20.6	90.4	31	92.8	31	93.8	33	94.3	39	95.0
Sick	5	50	98.9	15.6	98.1	39	98.3	39	98.3	34	98.8	36	98.7
	10	37	98.5	10.0	97.9	17	98.0	17	98.0	28	98.6	28	98.5
	15	24	98.3	8.6	97.5	7	97.9	14	97.7	14	98.1	24	98.3

*: Scheduling, segment and sick: use top 50% selected instances; Letter and splice: use top 75% selected instances to obtain good accuracies

APPENDIX C STATISTICAL SIGNIFICANCE TESTS

Table A14 Statistical significance tests for different *M* and *y* in instance selection

Dataset	Tests for different *M*	*M*=5 to 10			*M*=10 to 15			*M*=5 to 15		
		t	$t_{\alpha,v}$	Reject?	t	$t_{\alpha,v}$	Reject?	t	$t_{\alpha,v}$	Reject?
Scheduling	Accuracy	4.099	1.734	Y	6.860	1.734	Y	10.633	1.734	Y
	Tree size	9.730	1.740	Y	5.161	1.746	Y	12.938	1.734	Y
	ALR	-8.276	1.771	Y	-3.353	1.833	Y	-4.295	1.833	Y
Splice	Accuracy	5.939	1.812	Y	1.998	1.833	Y	26.517	1.771	Y
	Tree size	9.163	1.782	Y	2.553	1.833	Y	27.322	1.812	Y
	ALR	-3.978	1.833	Y	-1.776	1.833	N	-143.170	1.833	Y
Segment	Accuracy	8.159	1.734	Y	5.043	1.796	Y	8.909	1.796	Y
	Tree size	6.736	1.746	Y	6.617	1.746	Y	12.445	1.771	Y
	ALR	-6.040	1.734	Y	-7.735	1.734	Y	-13.590	1.740	Y
Letter	Accuracy	6.161	1.746	Y	7.950	1.746	Y	12.659	1.771	Y
	Tree size	8.487	1.796	Y	12.099	1.796	Y	12.765	1.833	Y
	ALR	-4.472	1.734	Y	-6.708	1.734	Y	-11.180	1.734	Y
Sick	Accuracy	5.060	1.740	Y	1.438	1.761	N	4.844	1.782	Y
	Tree size	3.500	1.796	Y	1.886	1.771	Y	9.648	1.746	Y
	ALR	-2.835	1.833	Y	-0.882	1.740	N	-5.265	1.833	Y
Dataset	Tests for different *y*	Best to Top 25%			Top 25% to Top 50%			Top 50% to Top 75%		
		t	$t_{\alpha,v}$	Reject?	t	$t_{\alpha,v}$	Reject?	t	$t_{\alpha,v}$	Reject?
Scheduling	Accuracy	6.883	1.740	Y	2.963	1.740	Y	1.118	1.734	N
	Tree size	14.585	1.782	Y	8.080	1.734	Y	2.545	1.740	Y
	ALR	13.563	1.812	Y	7.385	1.782	Y	2.795	1.734	Y
Splice	Accuracy	6.824	1.812	Y	7.291	1.753	Y	3.315	1.753	Y
	Tree size	7.213	1.734	Y	6.637	1.771	Y	8.277	1.746	Y
	ALR	3.991	1.833	Y	4.961	1.812	Y	6.325	1.833	Y
Segment	Accuracy	9.209	1.746	Y	4.558	1.753	Y	0.930	1.796	N
	Tree size	14.467	1.782	Y	11.321	1.746	Y	3.384	1.771	Y
	ALR	18.657	1.740	Y	9.000	1.796	Y	3.508	1.746	Y
Letter	Accuracy	13.854	1.796	Y	19.922	1.740	Y	9.209	1.746	Y
	Tree size	42.373	1.796	Y	46.118	1.812	Y	16.317	1.761	Y
	ALR	12.586	1.833	Y	22.361	1.734	Y	0	1.734	N
Sick	Accuracy	6.146	1.761	Y	1.838	1.746	Y	0.494	1.740	N
	Tree size	8.013	1.734	Y	1.984	1.761	Y	3.433	1.740	Y
	ALR	4.180	1.833	Y	1.452	1.746	N	2.680	1.796	Y

APPENDIX D DIFFERENT CROSSOVER OPERATIONS

Table A15 GA-based instance selection using 1-point crossover

Dataset	1-point crossover	Best		Top 25%		Top 50%		Top 75%	
		Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.
Scheduling	Accuracy	95.9	0.8	97.2	0.6	98.3	0.4	98.4	0.3
	Tree size	12.2	1.0	28.2	4.5	39.8	3.3	41.4	2.3
	ALR	15.9	1.3	8.2	1.6	4.9	0.4	4.7	0.3
Splice	Accuracy	76.3	2.7	85.1	2.3	90.9	1.6	91.7	0.5
	Tree size	35.2	0.6	87.0	11.6	126.8	9.1	153.8	13.5
	ALR	3.7	0.1	1.5	0.2	1.0	0.1	0.9	0.1
Segment	Accuracy	87.3	2.5	89.8	1.5	92.5	1.0	93.4	1.1
	Tree size	17.6	0.9	34.8	5.9	48.8	3.0	58.2	3.2
	ALR	12.0	0.6	6.3	1.2	4.3	0.3	4.1	0.2
Letter	Accuracy	66.8	3.5	72.1	1.0	79.3	0.6	81.4	0.8
	Tree size	317.4	15.2	640.2	23.6	1079.2	37.7	1247.8	30.6
	ALR	0.8	0.0	0.3	0.0	0.2	0.0	0.2	0.0
Sick	Accuracy	95.6	0.7	97.7	0.6	98.0	0.6	98.1	0.6
	Tree size	5.0	0.4	13.6	2.9	16.2	3.1	25.0	9.1
	ALR	35.0	0.4	11.6	1.7	10.8	2.9	8.8	2.8

Table A16 GA-based instance selection using 2-point crossover

Dataset	2-point crossover	Best		Top 25%		Top 50%		Top 75%	
		Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.
Scheduling	Accuracy	96.3	0.6	98.0	0.5	98.6	0.4	98.8	0.4
	Tree size	8.6	1.7	27.8	3.8	41.0	3.5	45.4	4.2
	ALR	22.6	3.5	7.1	0.9	4.8	0.4	4.3	0.4
Splice	Accuracy	60.3	11.6	85.7	2.0	91.2	1.3	92.8	0.8
	Tree size	14.4	16.5	71.8	19.0	116.2	9.3	146.2	6.7
	ALR	65.0	50.0	1.9	0.5	1.1	0.1	0.9	0.0
Segment	Accuracy	88.4	1.2	92.6	0.8	94.8	1.3	95.2	0.4
	Tree size	17.0	1.4	33.4	3.3	47.8	2.3	53.4	4.7
	ALR	11.8	0.8	5.9	0.6	4.1	0.2	3.7	0.3
Letter	Accuracy	62.4	2.5	73.9	0.8	80.2	0.6	82.3	0.4
	Tree size	302.2	22.6	623.0	7.9	1071.2	29.7	1244.3	15.6
	ALR	0.7	0.1	0.3	0.0	0.2	0.0	0.2	0.0
Sick	Accuracy	95.3	1.2	98.0	0.7	98.5	0.5	98.6	0.4
	Tree size	3.8	2.4	12.4	2.4	15.6	4.5	23.4	5.6
	ALR	61.2	35.4	14.2	3.3	11.6	4.6	7.5	1.5

APPENDIX D DIFFERENT CROSSOVER OPERATIONS (CONTINUED)

Table A17 GA-based instance selection using uniform crossover

Dataset	**uniform**	Best		Top 25%		Top 50%		Top 75%	
	crossover	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.
Scheduling	Accuracy	94.9	0.7	97.9	0.5	98.1	0.1	98.6	0.4
	Tree size	8.6	0.8	28.0	2.5	41.4	4.6	47.8	2.7
	ALR	21.9	2.0	8.4	0.8	4.8	0.6	4.1	0.2
Splice	Accuracy	68.9	9.3	84.8	0.8	90.4	1.1	90.8	0.3
	Tree size	29.2	14.5	72.0	7.9	126.6	10.5	147.2	12.1
	ALR	24.0	40.8	1.8	0.2	1.0	0.1	0.9	0.1
Segment	Accuracy	86.0	3.1	91.8	1.0	94.0	1.0	94.7	0.8
	Tree size	16.2	2.9	35.6	3.7	48.2	6.8	55.4	4.7
	ALR	12.5	1.8	6.0	0.6	4.5	0.6	3.8	0.3
Letter	Accuracy	59.9	1.8	72.1	0.6	79.5	0.4	81.2	0.5
	Tree size	286.6	11.7	629.4	14.1	1072.2	23.5	1267.0	35.4
	ALR	0.7	0.0	0.3	0.0	0.2	0.0	0.2	0.0
Sick	Accuracy	95.1	1.9	97.9	0.5	98.2	0.5	98.4	0.4
	Tree size	4.6	2.1	13.2	3.5	18.2	7.0	25.4	4.9
	ALR	47.3	28.6	18.5	6.2	10.8	4.8	8.5	1.3

APPENDIX E DIFFERENT SELECTION OPERATIONS

Table A18 GA-based instance selection using ranking selection and 2-point crossover

Dataset	**Ranking selection**	Best		Top 25%		Top 50%		Top 75%	
		Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.
Scheduling	Accuracy	97.7	0.7	98.6	0.4	99.2	0.3	99.4	0.2
	Tree size	18.3	4.2	31.7	3.1	52.3	5.7	59.7	7.2
Splice	Accuracy	85.9	2.1	89.1	0.8	92.9	0.6	93.0	0.5
	Tree size	49.8	13.9	105.4	4.7	143.6	6.4	163.0	10.1
Segment	Accuracy	91.8	0.8	93.7	0.6	95.9	0.7	96.8	0.7
	Tree size	24.6	1.4	35.4	5.7	44.6	3.1	56.4	6.2
Letter	Accuracy	61.7	4.0	80.0	0.7	86.7	0.4	88.9	0.3
	Tree size	248.6	43.9	810.8	32.7	1221.4	43.9	1547.2	32.9
Sick	Accuracy	97.9	0.2	98.1	0.3	98.4	0.3	98.7	0.2
	Tree size	7.6	1.4	11.3	3.4	17.7	2.9	25.4	2.4

Table A19 GA-based instance selection using roulette wheel selection and 2-point crossover

Dataset	**Roulette wheel selection**	Best		Top 25%		Top 50%		Top 75%	
		Avg.	S.E.	Avg.	S.E.	Avg.	S.E.	Avg.	S.E.
Scheduling	Accuracy	96.3	0.6	98.0	0.5	98.6	0.4	98.8	0.4
	Tree size	8.6	1.7	27.8	3.8	41.0	3.5	45.4	4.2
Splice	Accuracy	60.3	11.6	85.7	2.0	91.2	1.3	92.8	0.8
	Tree size	14.4	16.5	71.8	19.0	116.2	9.3	146.2	6.7
Segment	Accuracy	88.4	1.2	92.6	0.8	94.8	1.3	95.2	0.4
	Tree size	17.0	1.4	33.4	3.3	47.8	2.3	53.4	4.7
Letter	Accuracy	62.4	2.5	73.9	0.8	80.2	0.6	82.3	0.4
	Tree size	302.2	22.6	623.0	7.9	1071.2	29.7	1244.3	15.6
Sick	Accuracy	95.3	1.2	98.0	0.7	98.5	0.5	98.6	0.4
	Tree size	3.8	2.4	12.4	2.4	15.6	4.5	23.4	5.6

APPENDIX F HISTOGRAMS BEFORE AND AFTER INSTANCE SELECTION

FIGURE A1-A12 (PAGE 109-120)

Figure A1 Histograms for sick dataset before instance selection

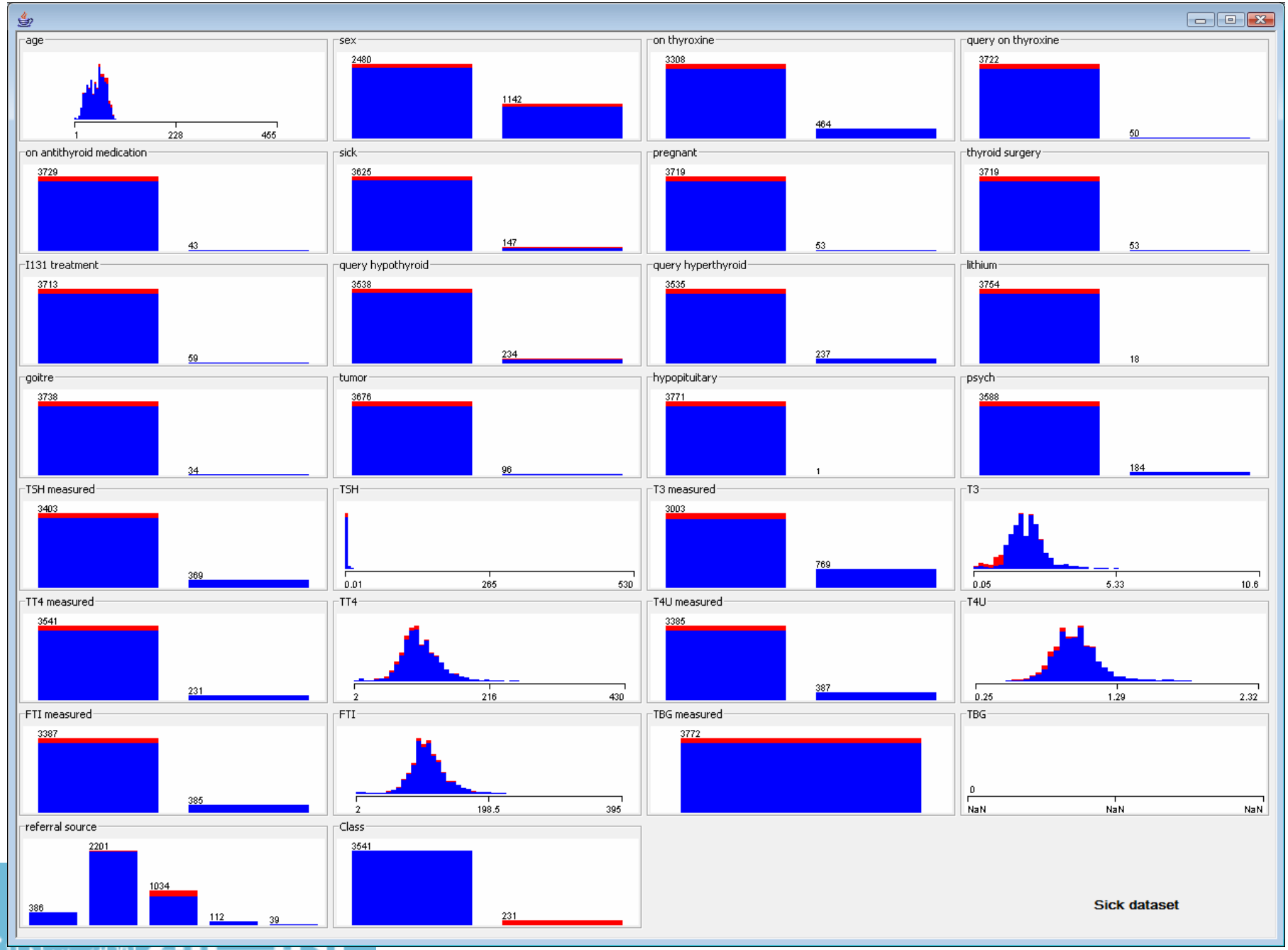


Figure A2 Histograms for sick dataset after instance selection using best subset

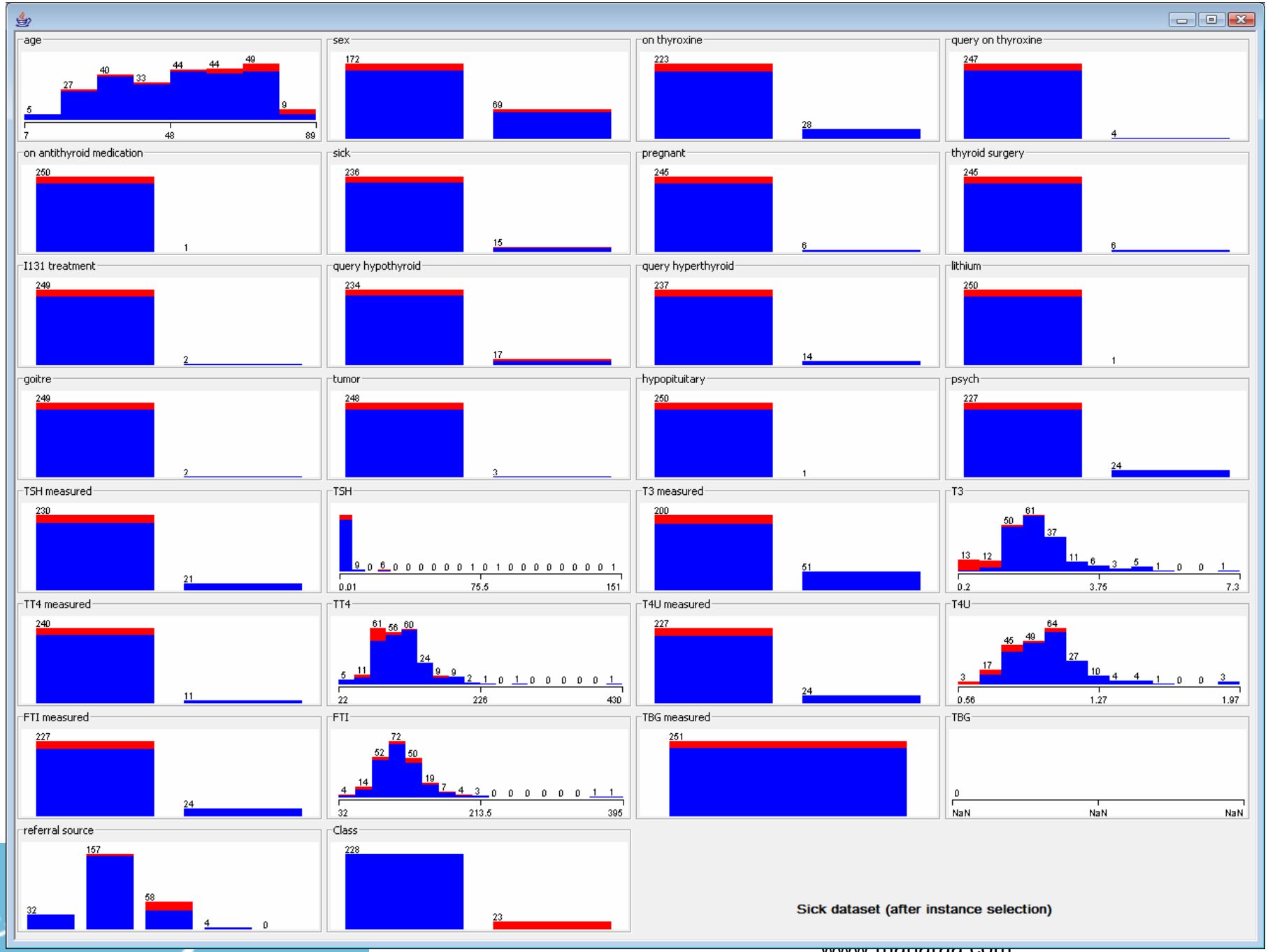


Figure A3 Histograms for letter dataset before instance selection

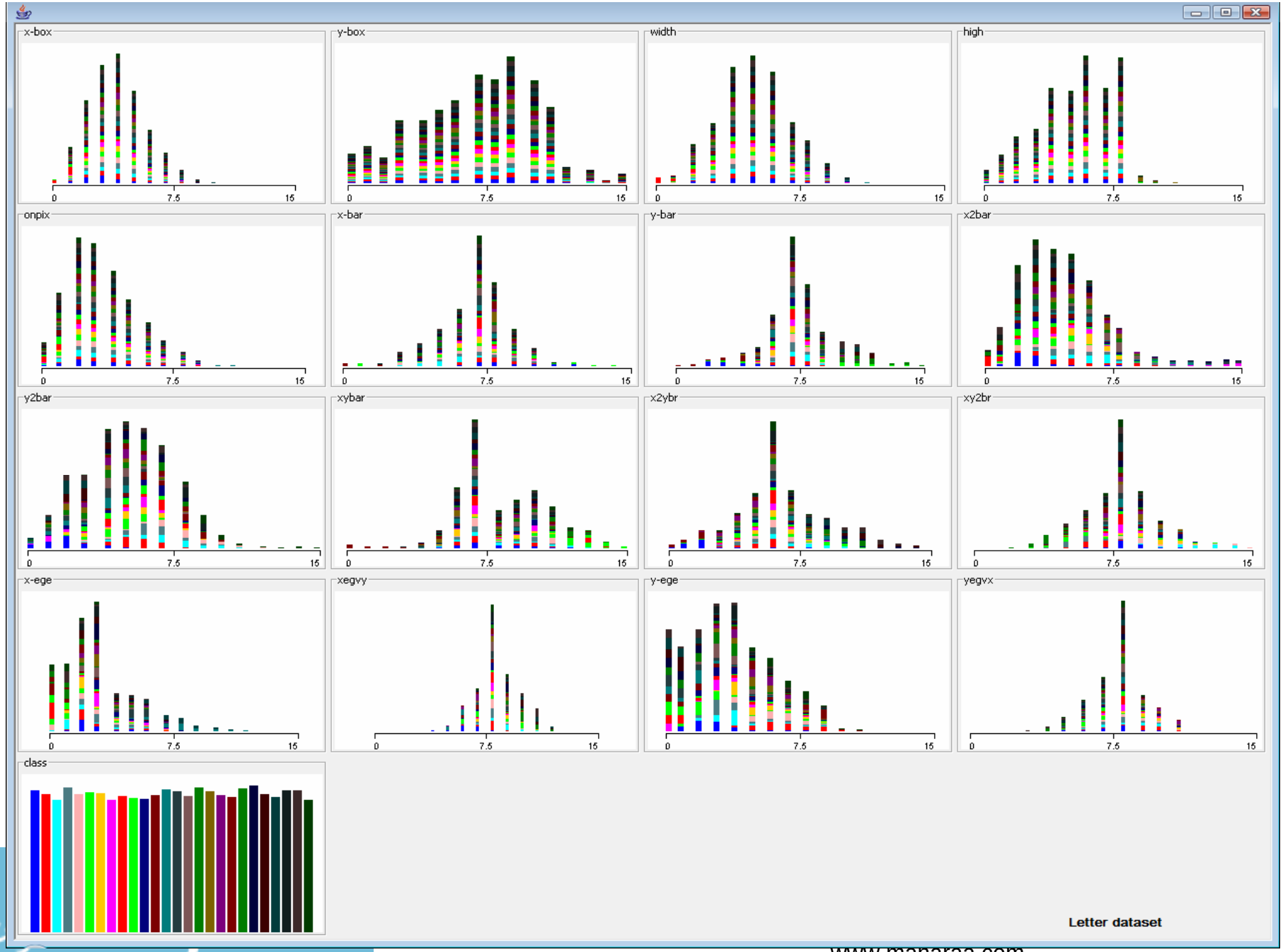


Figure A4 Histograms for letter dataset after instance selection using top 50% subsets

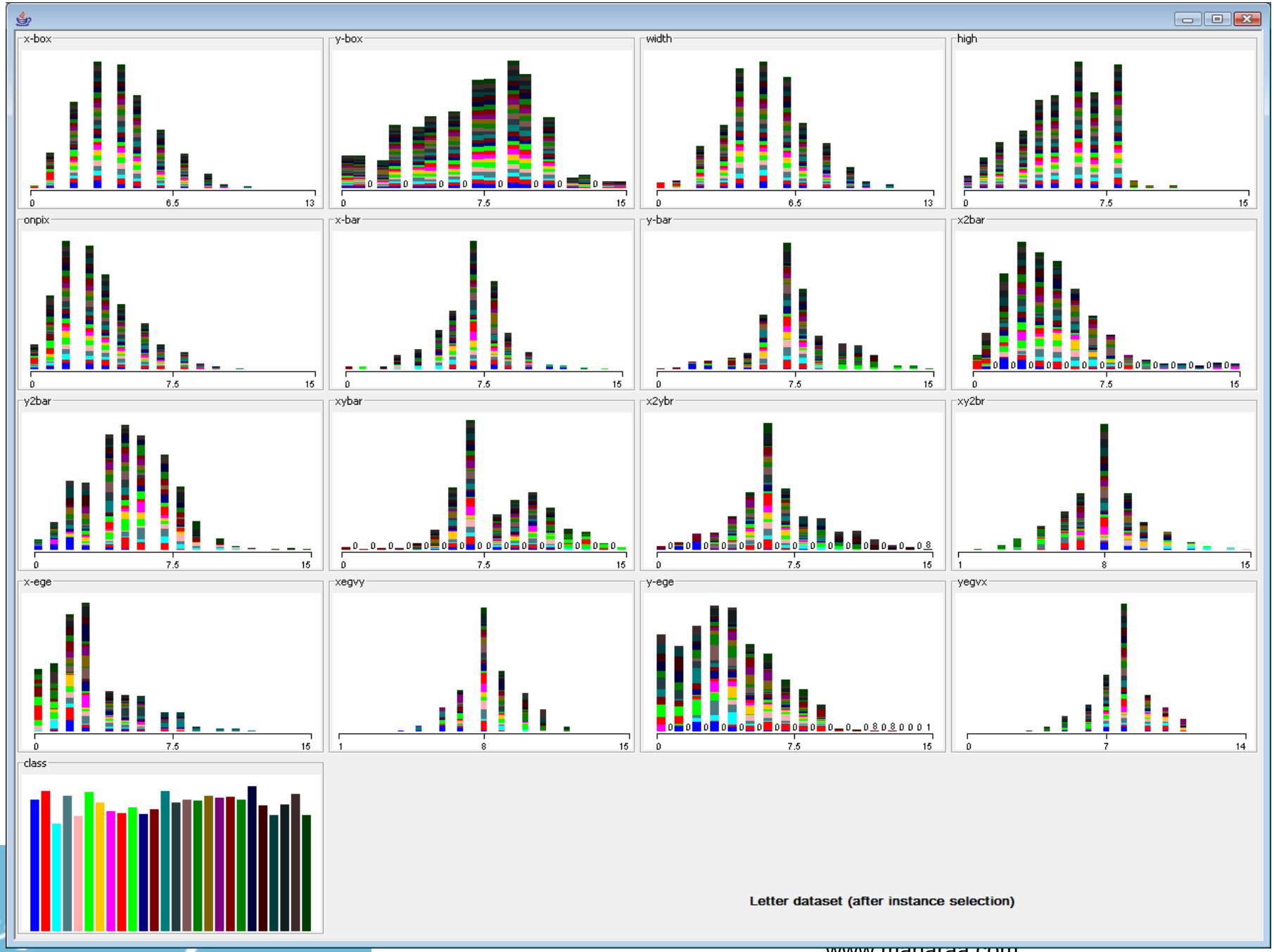


Figure A5 Histograms for scheduling dataset before instance selection

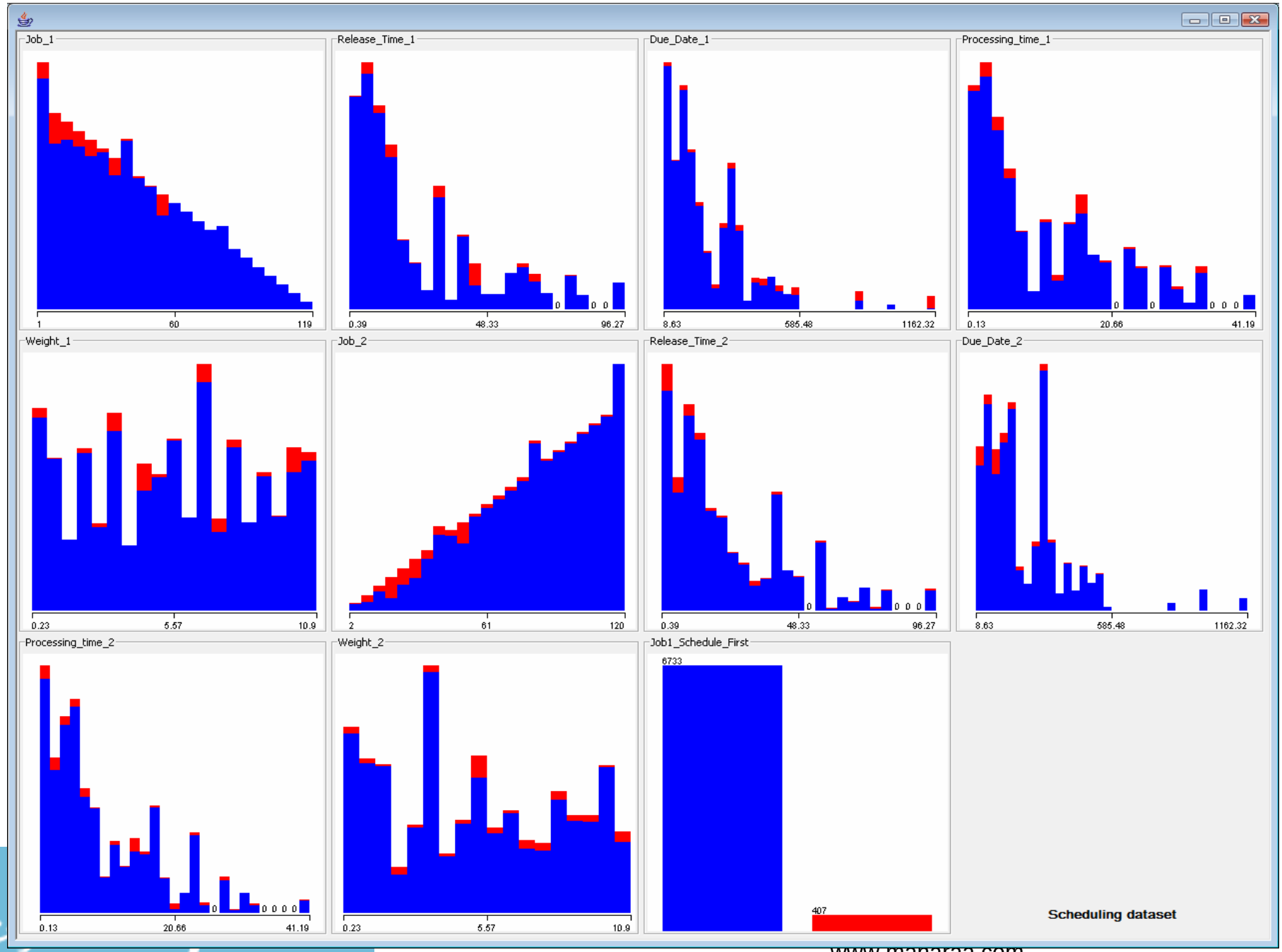


Figure A6 Histograms for scheduling dataset after instance selection using best subset

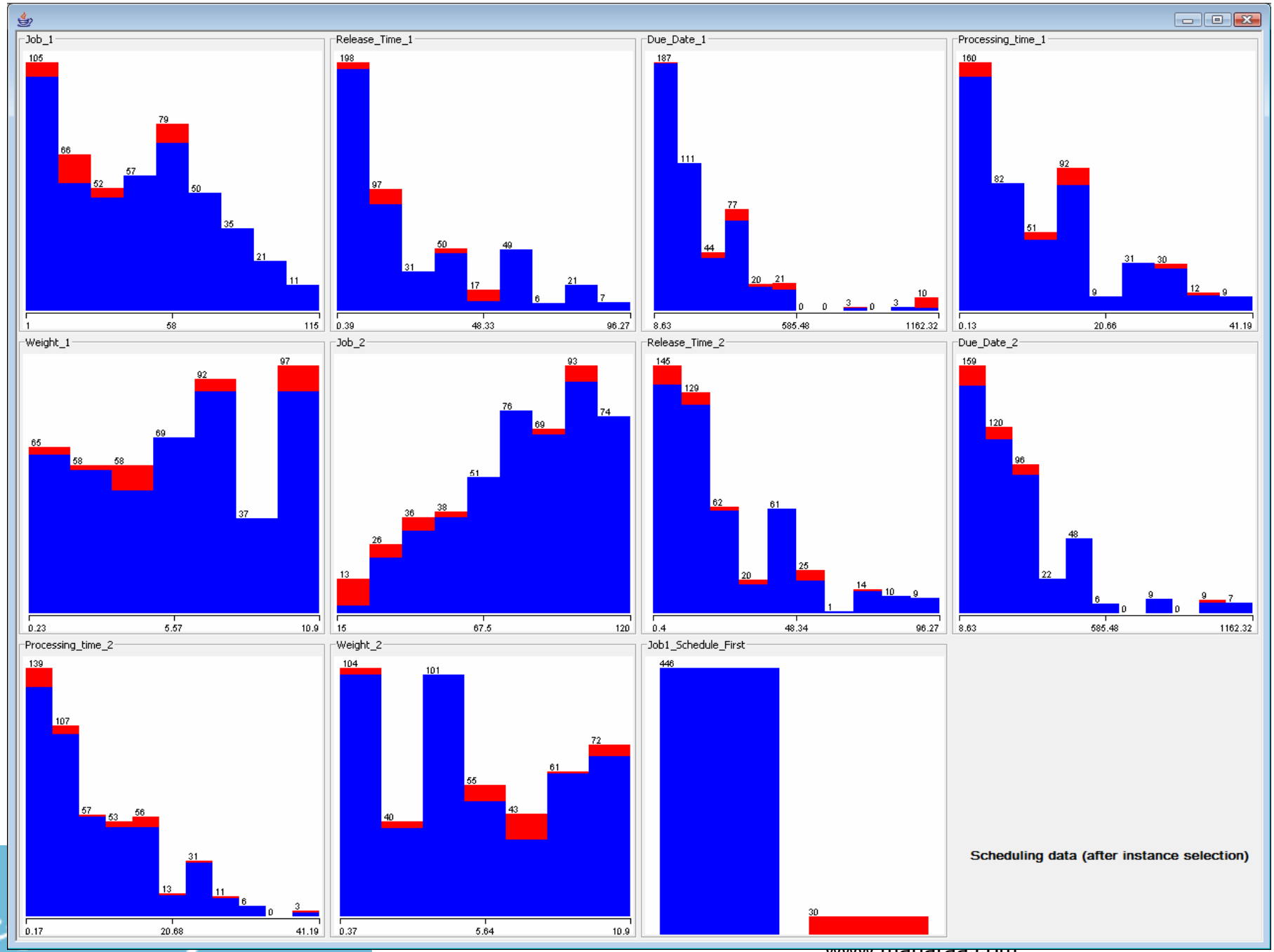


Figure A7 Histograms for segment dataset before instance selection

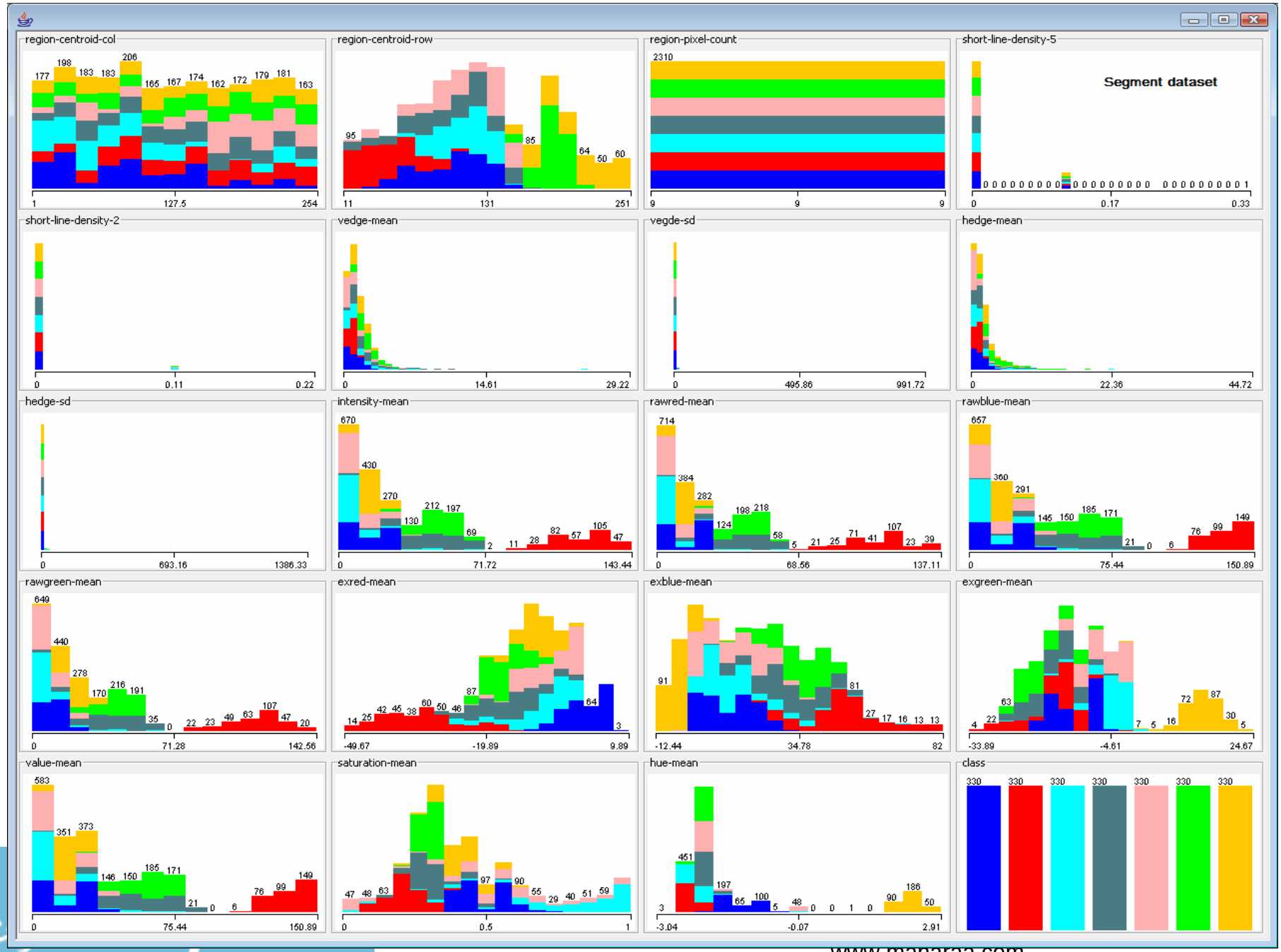


Figure A8 Histograms for segment dataset after instance selection using top 50% subsets

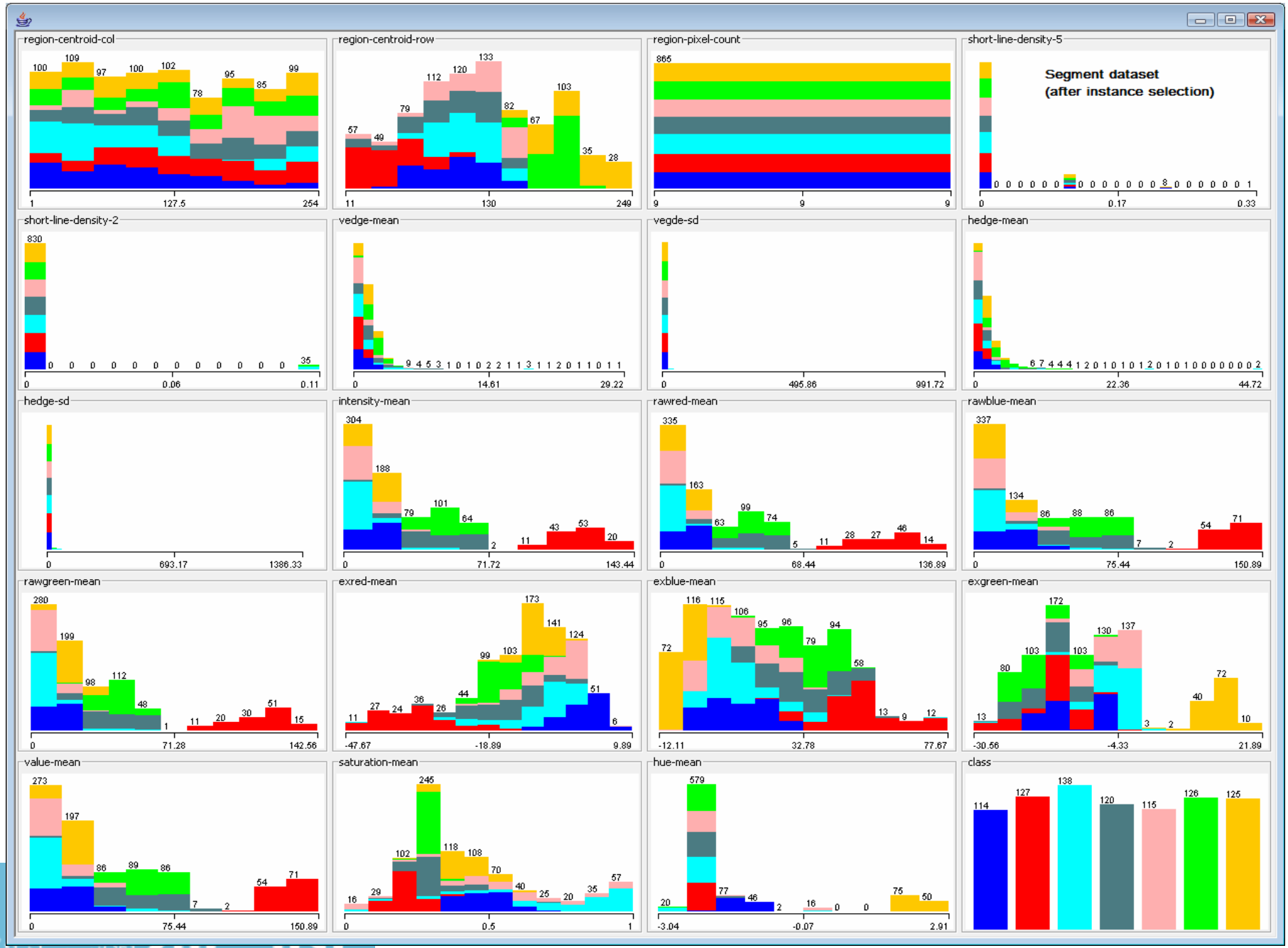


Figure A9 Histograms for splice dataset before instance selection (part I)

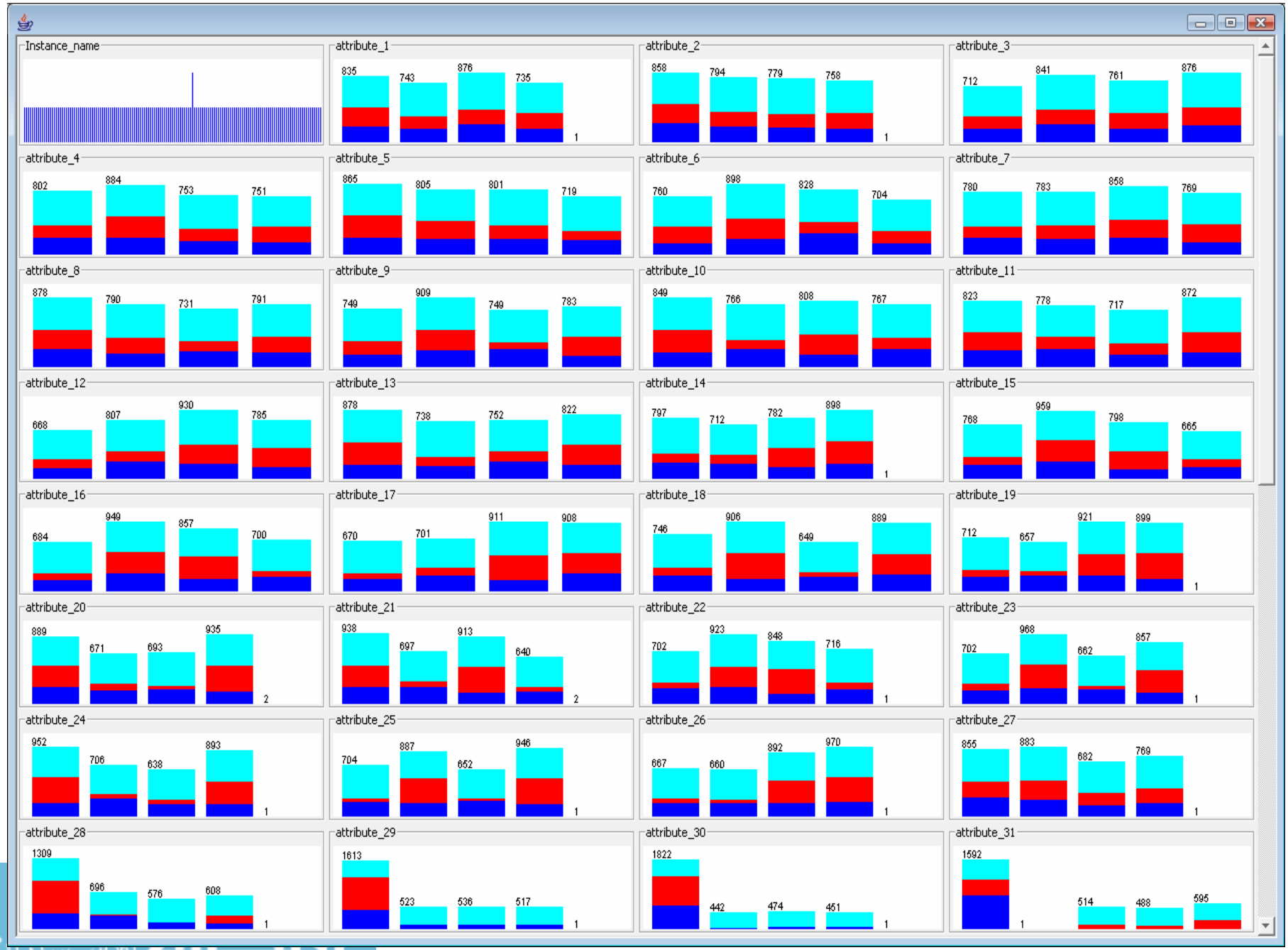


Figure A10 Histograms for splice dataset before instance selection (part II)

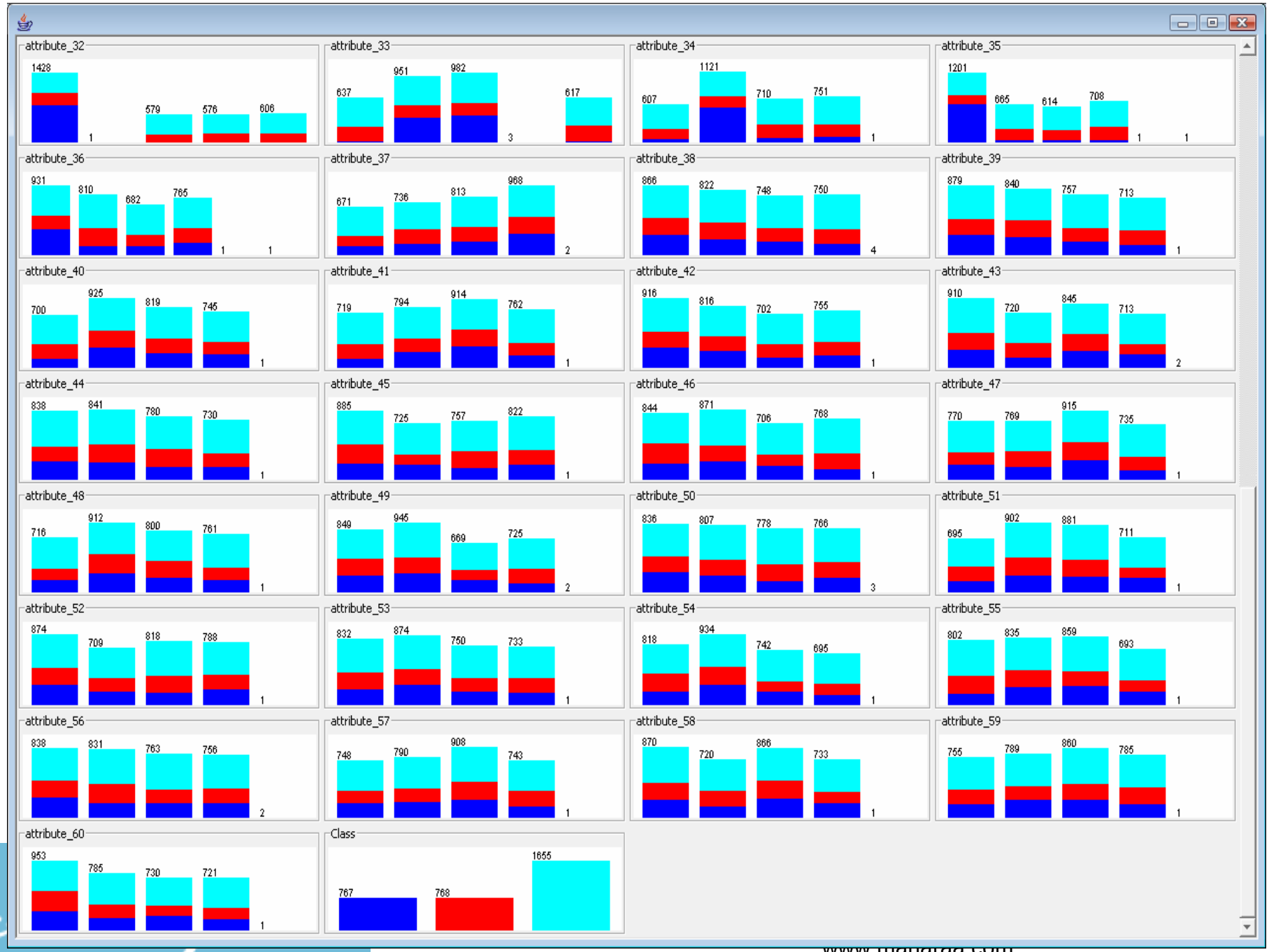


Figure A11 Histograms for splice dataset after instance selection using top 50% subsets (part I)

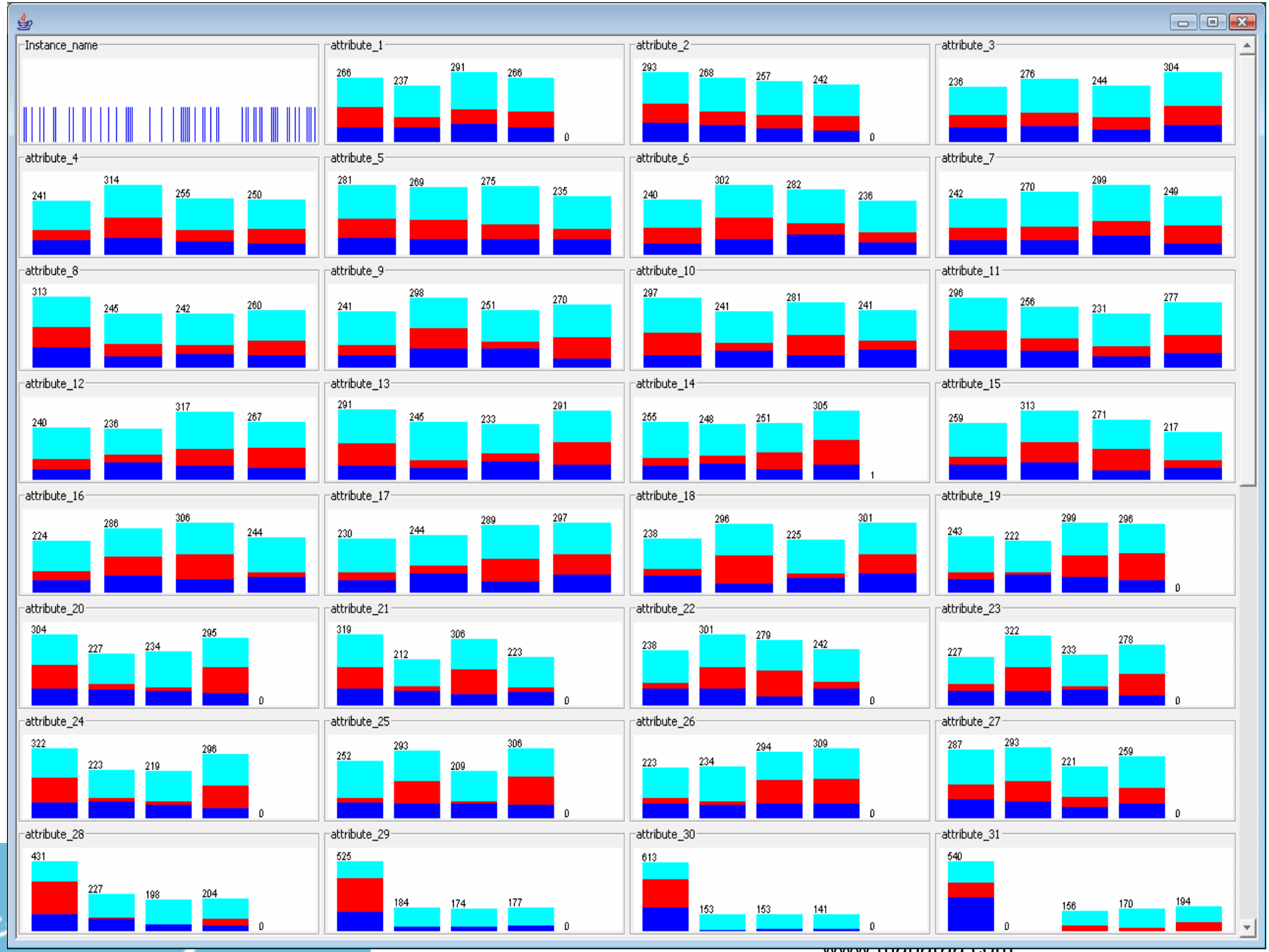
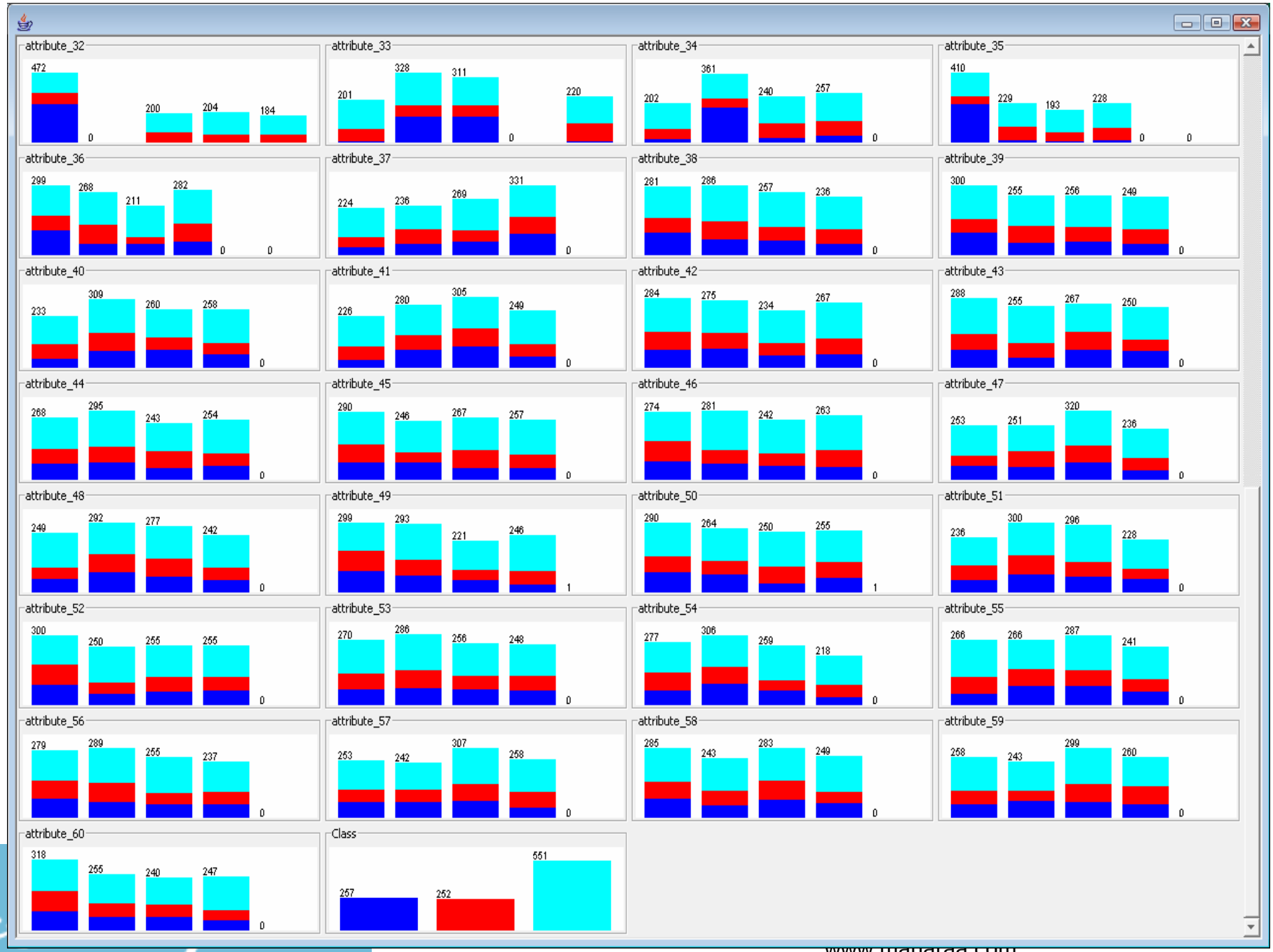


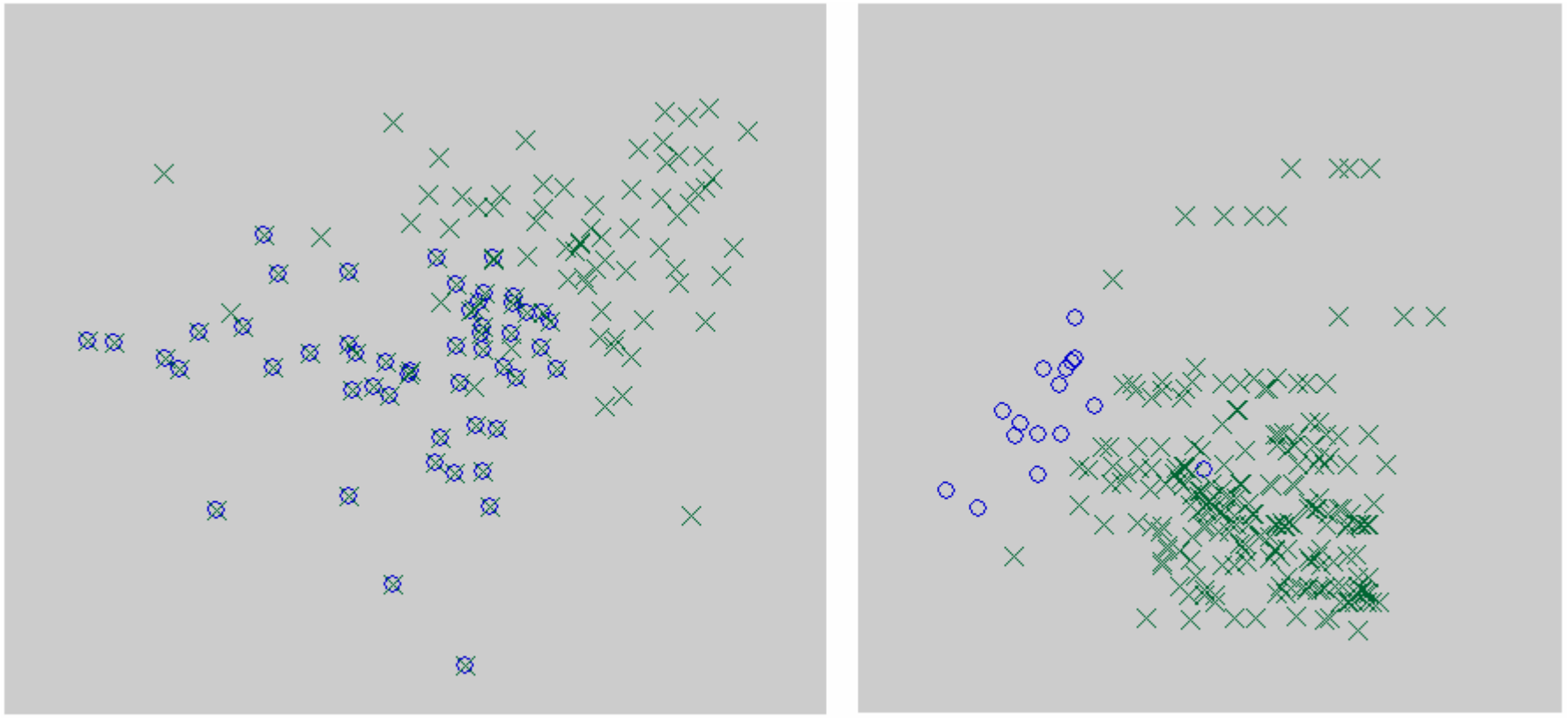
Figure A12 Histograms for splice dataset after instance selection using top 50% subsets (part II)



APPENDIX G TOUR PLOTS AND DECISION TREE VISUALIZATIONS

FIGURE A13-A20 (PAGE 122-129)

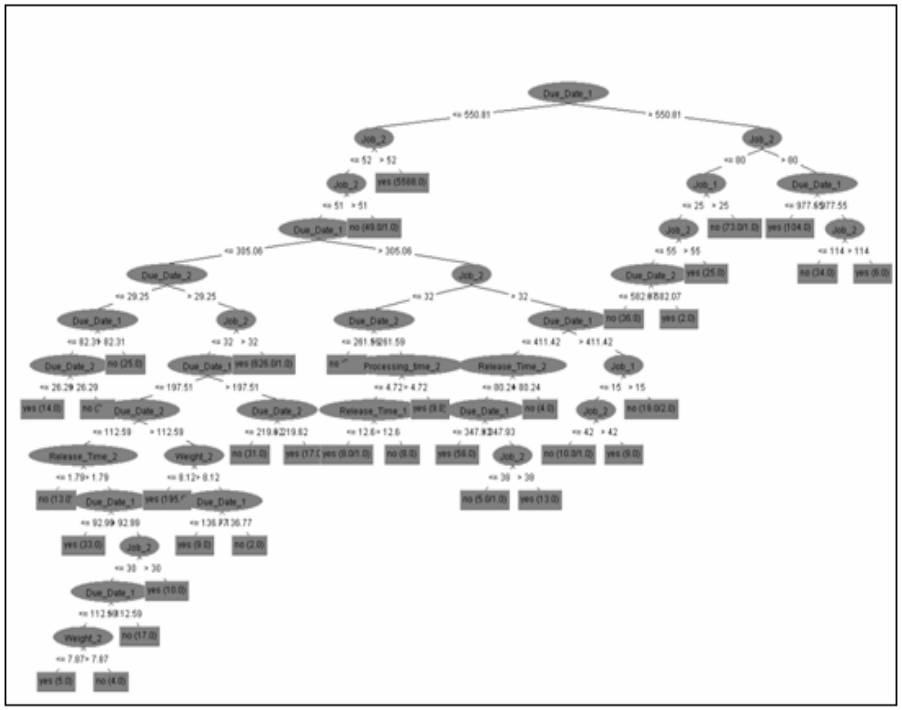
Figure A13 Tour plots for scheduling dataset before/after instance selection



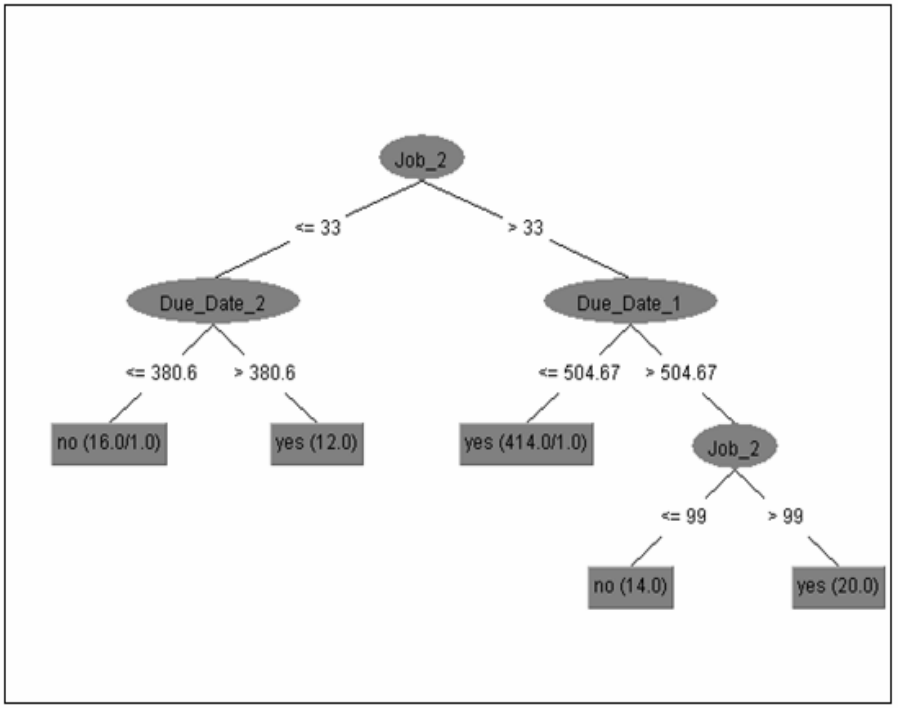
Scheduling dataset: Before instance selection After instance selection

Note: Blue circle points represent "no" class and green cross points represent "yes" class.

Figure A14 Decision trees for scheduling dataset before/after instance selection



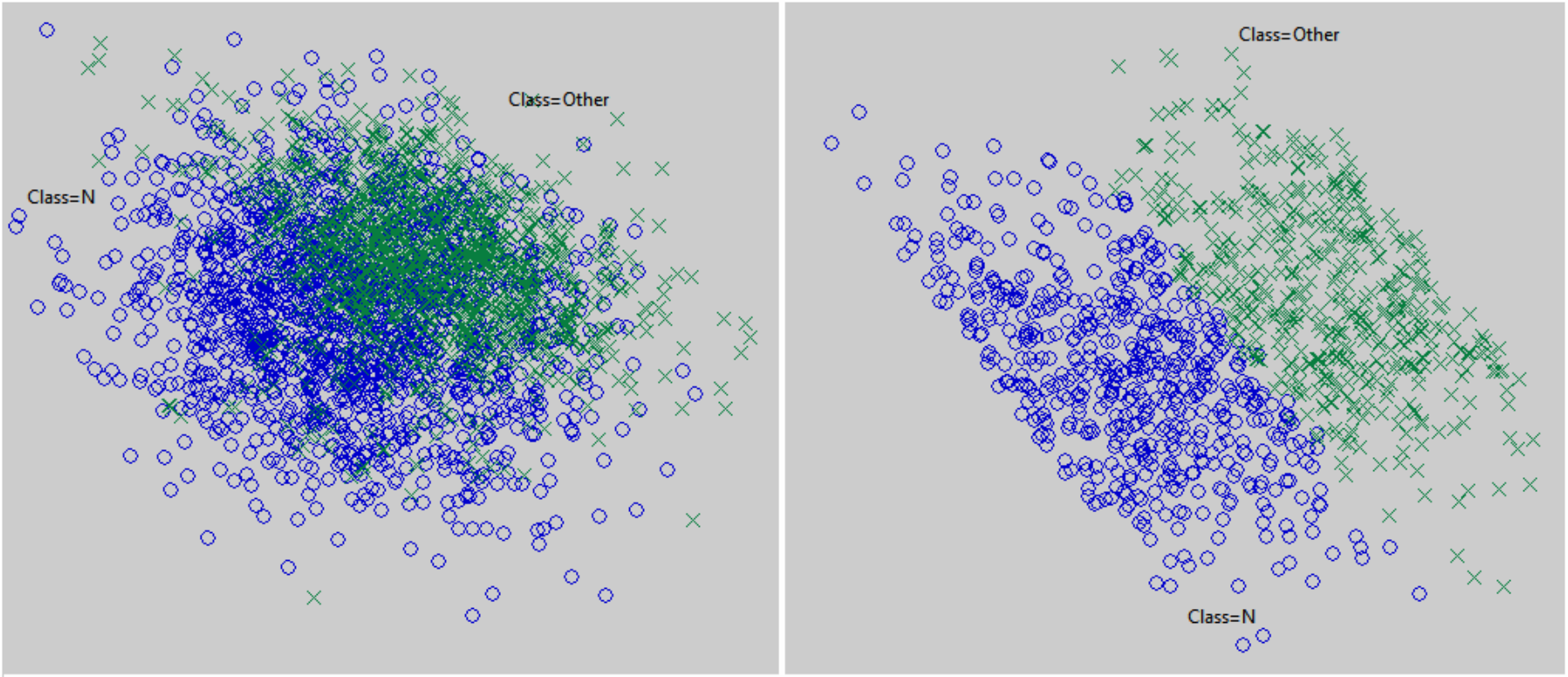
Decision tree from entire dataset



Decision tree from selected instances

- Before instance selection, the decision tree splits on 6 attributes: Job_2, Due_Date_1, Due_Date_2, Release_Time_2, Processing_time_2 and Weight_2; the tree has 35 leaves and 69 nodes.
- After instance selection, the decision tree splits on 3 attributes: Job_2, Due_Date_1 and Due_Date_2; the tree has 5 leaves and 9 nodes. And the two classes are better separated in the tour plot.

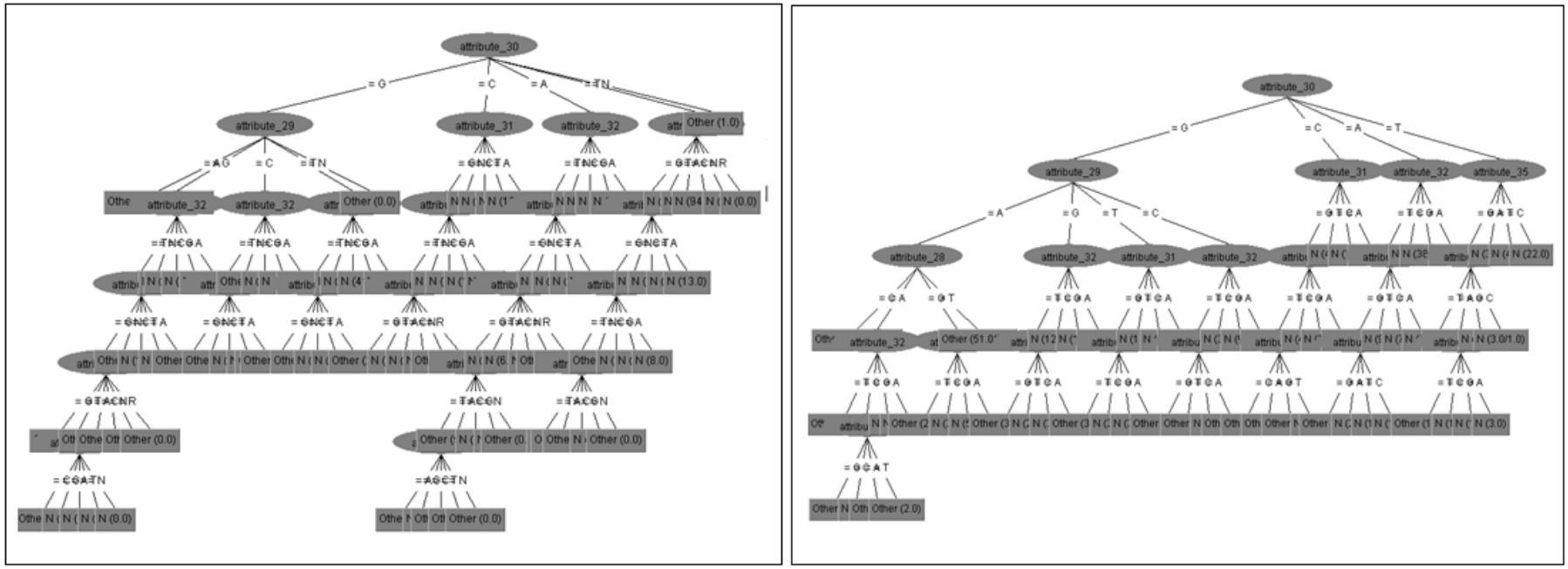
Figure A15 Example tour plots for splice dataset before/after instance selection showing the separation between "N" and other classes



Splice dataset: Before instance selection After instance selection

Note: Blue circle points represent "N" class, green cross points represent the other two classes (EI and IE)

Figure A16 Example decision trees for splice dataset before/after instance selection showing the classification between "N" and other classes

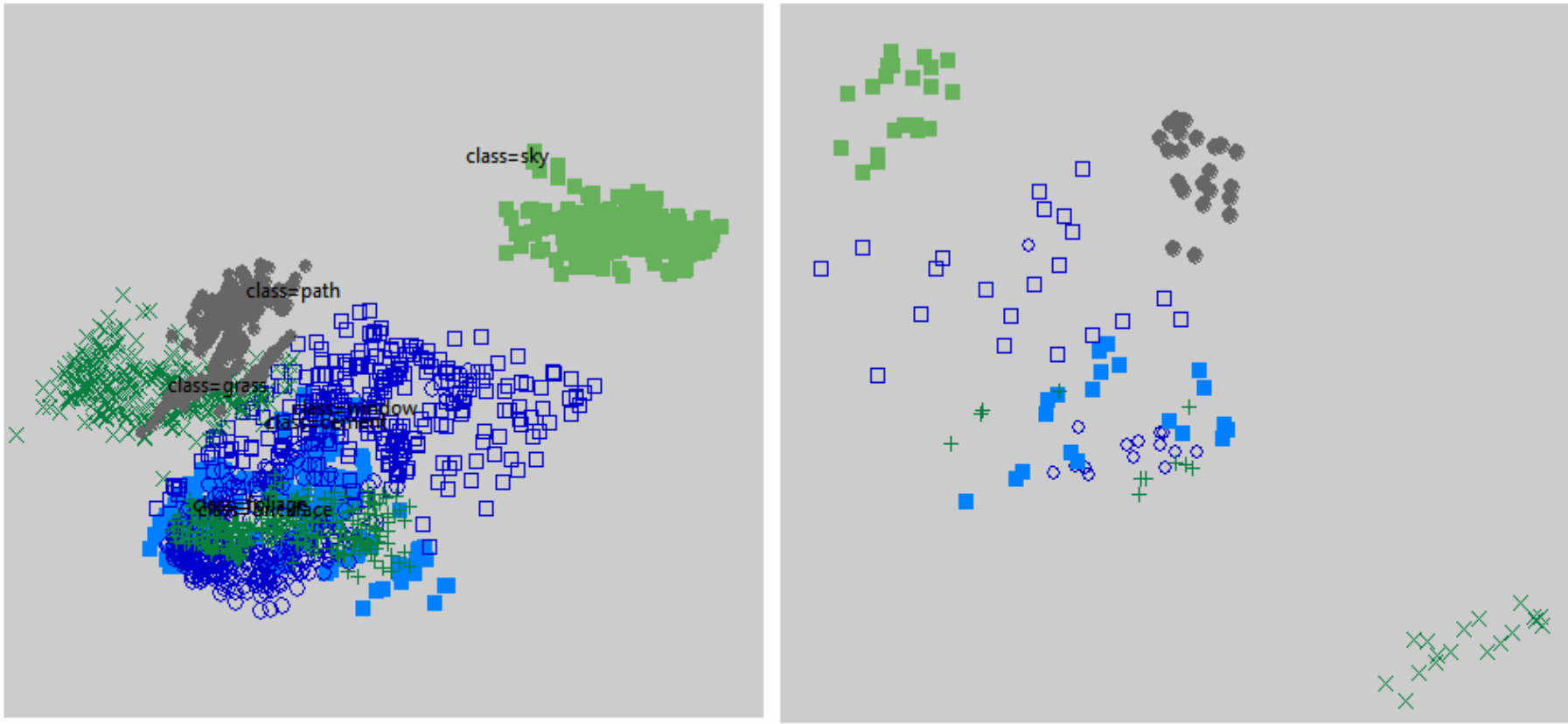


Decision tree from entire dataset

Decision tree from selected instances

- Before instance selection, the decision tree splits on 7 attributes: attribute_30, attribute_29, attribute_32, attribute_31, attribute_35, attribute_20, and attribute_34; the tree has 93 leaves and 115 nodes.
- After instance selection, the decision tree splits on 9 attributes: attribute_30, attribute_28, attribute_29, attribute_32, attribute_18, attribute_5, attribute_34, attribute_35, and attribute_31; the tree has 64 leaves and 85 nodes. Class "N" and the other two classes ("EI" and "IE") are better separated in the tour plot.

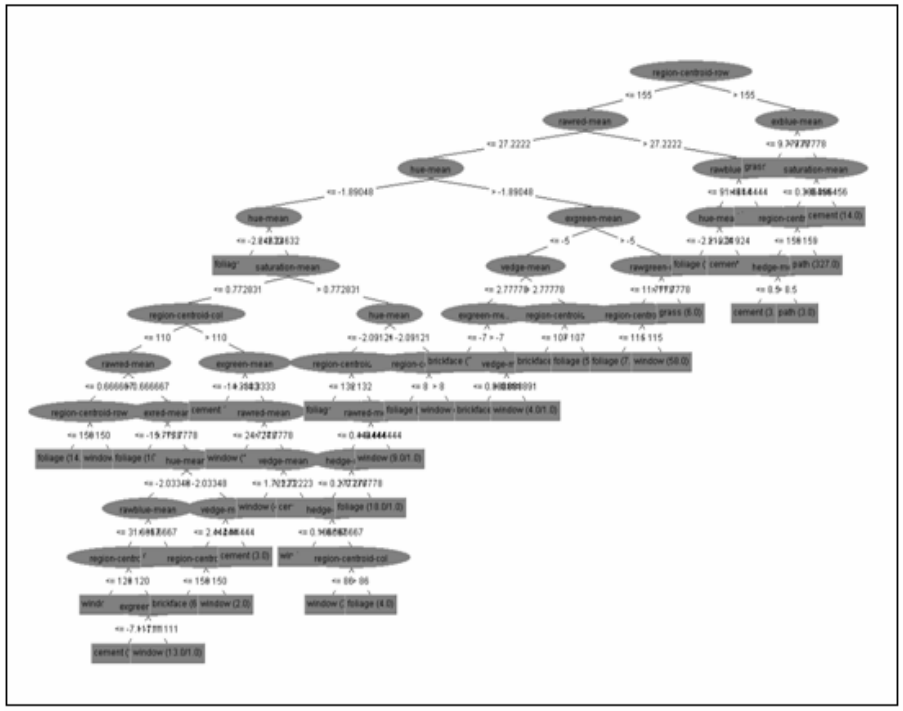
Figure A17 Tour plots for segment dataset before/after instance selection



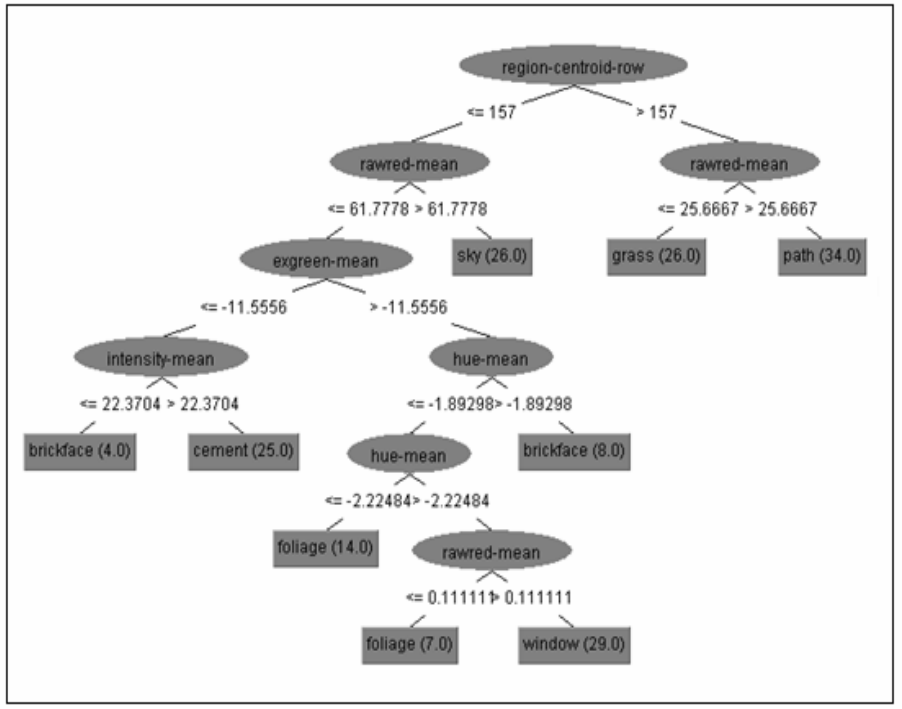
Segment dataset: Before instance selection After instance selection

Note: Blue circle points represent "foliage" class, blue square points represent "cement" class, green cross points represent "grass" class, green plus points represent "brickface" class, grey solid circle points represent "path" class, blue solid square points represent "window" class, and green solid square points represent "sky" class

Figure A18 Decision trees for segment dataset before/after instance selection



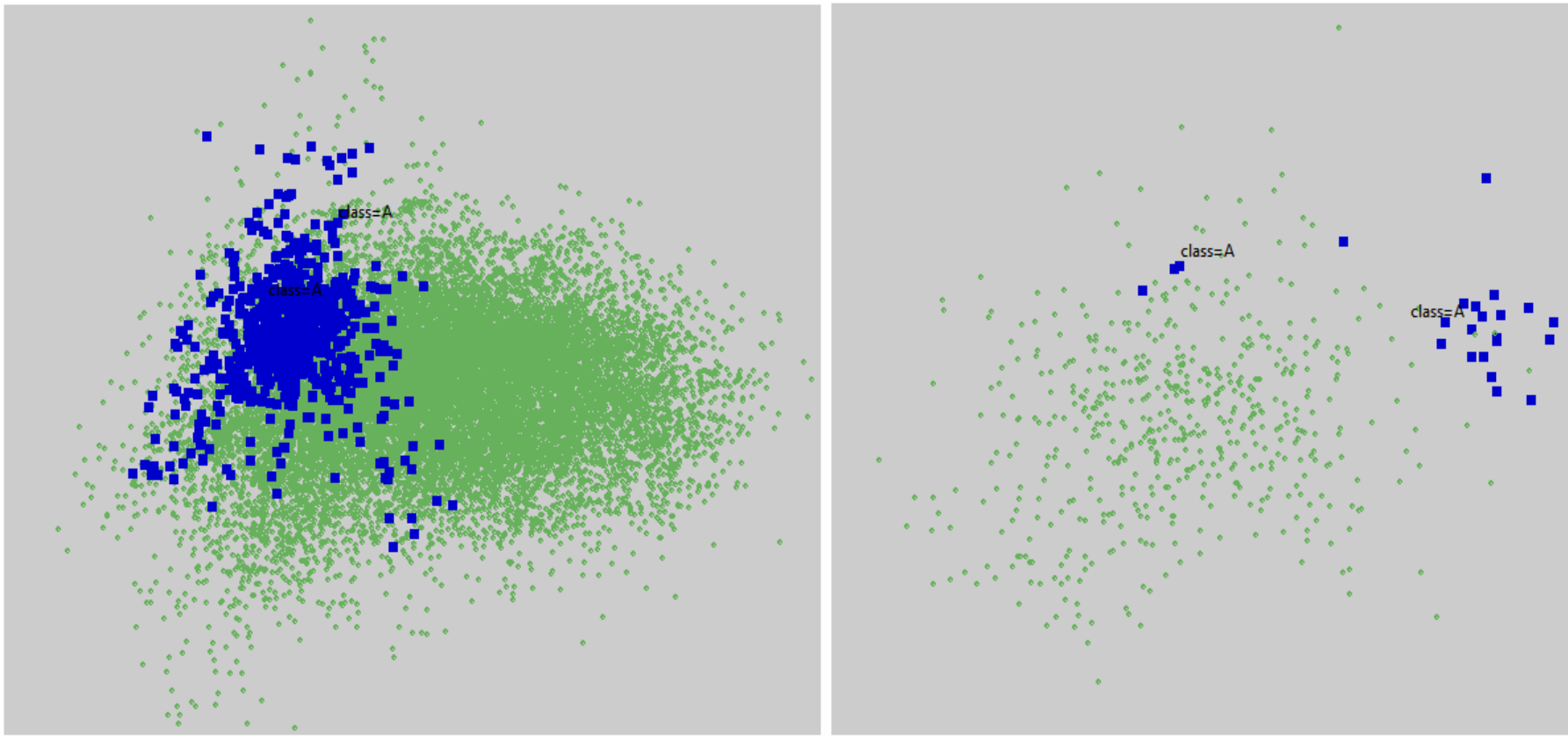
Decision tree from entire dataset



Decision tree from selected instances

- Before instance selection, the decision tree splits on 11 attributes: region-centroid-row, rawred-mean, hue-mean, saturation-mean, region-centroid-col, exred-mean, rawblue-mean, exgreen-mean, vedge-mean, rawgreen-mean and exblue-mean; the tree has 39 leaves and 77 nodes.
- After instance selection, the decision tree splits on 5 attributes: region-centroid-row, rawred-mean, exgreen-mean, intensity-mean and hue-mean; the tree has 9 leaves and 17 nodes. And the three classes are better separated in the tour plot.

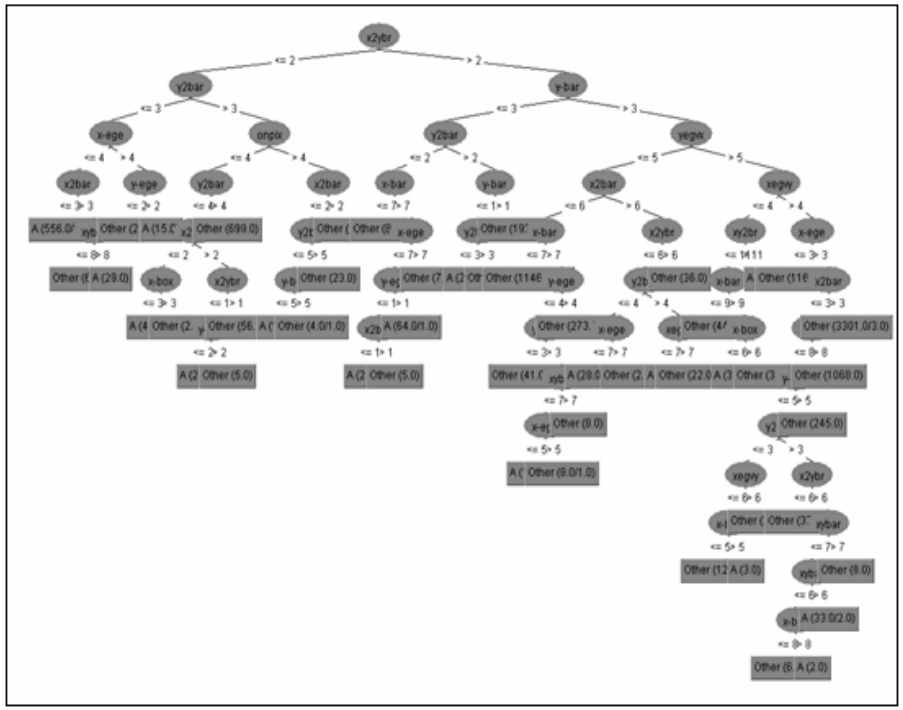
Figure A19 Example tour plots for letter dataset before/after instance selection showing the separation between "A" and other classes



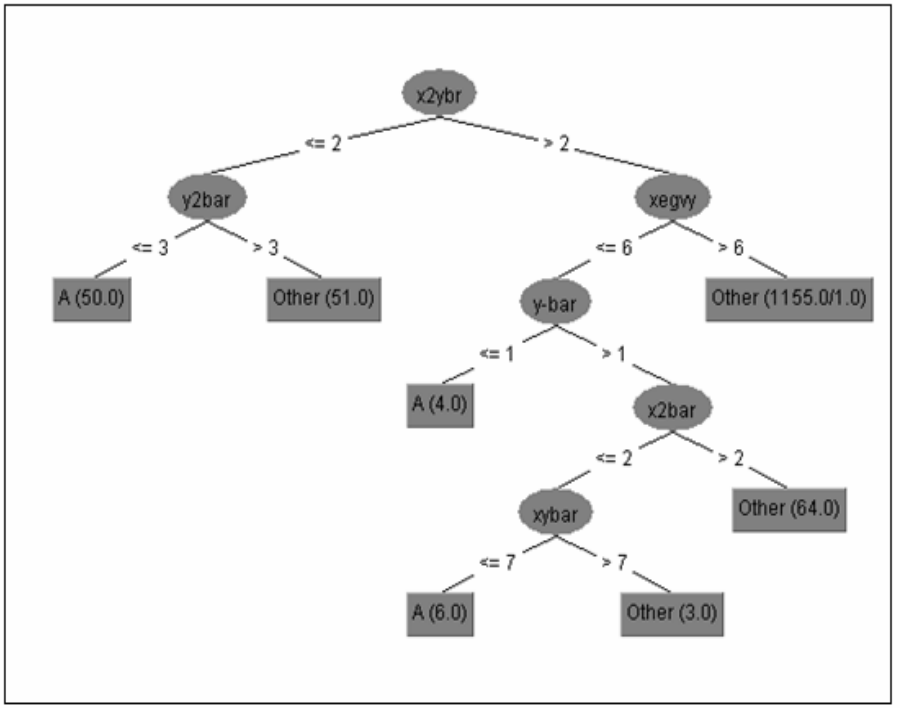
Letter dataset: Before instance selection After instance selection

Note: Blue square points represent "A" class and green dots represent the other 25 classes.

Figure A20 Example decision trees for letter dataset before/after instance selection showing the classification between "A" and other classes



Decision tree from entire dataset



Decision tree from selected instances

- Before instance selection, the decision tree splits on 12 attributes: x2ybr, y2bar, x-bar, y-bar, x-ege, x2bar, xybar, y-ege, onpix, x-box, yegvx and xegvy; the tree has 50 leaves and 99 nodes.
- After instance selection, the decision tree splits on 6 attributes: x2ybr, y2bar, xegvy, y-bar, x2bar and xybar; the tree has 7 leaves and 13 nodes. Class "A" and other classes are better separated in the tour plot.

APPENDIX H DECISION TREE ON UNBALANCED DATA

Table A20 Performance of the decision tree on scheduling data before/after instance selection

Scheduling data confusion matrix	Before instance selection (10-fold cross validation)		After instance selection (using best subset) 10-fold cross validation		**Independent test set**	
	Yes	No	Yes	No	Yes	No
Classified as yes	6719	17	447	5	2265	63
Classified as no	14	390	1	23	7	45
True positive rate (Recall)	99.8%	95.8%	99.8%	82.1%	99.7%	**41.7%**
False positive rate	4.2%	0.2%	17.9%	0.2%	58.3%	0.3%
Precision	99.7%	96.5%	98.9%	95.8%	97.3%	86.5%
Kappa statistic	0.96		0.88		0.55	
Overall accuracy	99.6%		98.7%		97.1%	
Scheduling data Confusion matrix	Before instance selection (10-fold cross validation)		After instance selection (using best subset) *Incorporate recall into GA fitness function* 10-fold cross validation		**Independent test set**	
	Yes	No	Yes	No	Yes	No
Classified as yes	6719	17	440	5	2193	18
Classified as no	14	390	2	29	79	90
True positive rate (Recall)	99.8%	95.8%	99.5%	85.3%	96.5%	**83.3%**
False positive rate	4.2%	0.2%	14.7%	0.5%	16.7%	3.5%
Precision	99.7%	96.5%	98.9%	93.5%	99.2%	53.3%
Kappa statistic	0.96		0.88		0.63	
Overall accuracy	99.6%		98.5%		95.9%	

Table A21 Performance of the decision tree on splice data before/after instance selection

Splice data confusion matrix	Before instance selection (10-fold cross validation)			After instance selection (using top 50% subsets) 10-fold cross validation			**Independent test set**		
	EI	IE	N	EI	IE	N	EI	IE	N
Classified as EI	737	40	34	232	19	24	235	19	11
Classified as IE	14	702	59	14	225	24	4	203	38
Classified as N	16	26	1562	11	8	503	12	14	528
True positive rate (Recall)	96.1%	91.4%	94.4%	90.3%	89.3%	91.3%	**93.6%**	**86.0%**	91.5%
False positive rate	3.1%	3.0%	2.7%	5.4%	4.7%	3.7%	3.7%	5.1%	5.3%
Precision	90.9%	90.6%	97.4%	84.4%	85.6%	96.4%	88.7%	82.9%	95.3%
Kappa statistic	0.90			0.84			0.85		
Overall accuracy	94.1%			90.6%			90.8%		

REFERENCES

[1] Agrawal R., T. Imielinski, and A. Swami (1993). Mining association rules between sets of items in large databases. *Proc. of the ACM SIGMOD Conference on Management of Data*, 207-216.

[2] Aha D. W., D. Kibbler, and M. K. Albert (1991). Instance-based learning algorithms. *Machine Learning*, vol. 6, 37-66.

[3] Ariew R. (1976). *Ockham's razor: A historical and philosophical analysis of Ockham's principle of parsimony*. Champaign-Urbana, University of Illinois.

[4] Attneave F. (1959). *Applications of information theory to psychology*. Holt, Rinehart and Winston.

[5] Baker E. and A. K. Jain (1976). On feature ordering in practice and some finite sample effects. *Proc. of the 3rd International Joint Conference on Pattern Recognition*, San Diego, CA, 45-49.

[6] Barnett V. and T. Lewis (1985). *Outliers in statistical data*. Second edition, John Wiley & Sons.

[7] Blum C. and A. Roli (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys* 35(3), 268-308.

[8] Bratko L. and M. Bohanec (1994). Trading accuracy for simplicity in decision trees. *Machine Learning* 15: 223-250.

[9] Breiman L., J. H. Friedman, R. A. Olsen and C. J. Stone (1984). *Classification and regression trees*. Wadsworth.

[10] Brighton H. and C. Mellish (2001). Identifying competence-critical instances for instance-based learners. H. Liu and H. Motoda (eds.). *Instance Selection and Construction for Data Mining*, Kluwer, 77-94.

[11] Brighton H. and C. Mellish (2002). Advances in instance selection for instance-based learning algorithms. *Data Mining and Knowledge Discovery*, vol. 6, 153-172.

[12] Broadley C. E. (1993). Addressing the selective superiority problem: Automatic algorithm/model class selection. *Proc. of the 10th Machine Learning Conference*, Amherst, MA, 17-24.

[13] Boser B. E., I. M. Guyon, and V. N. Vapnik (1992). A training algorithm for optimal margin classifiers. *Proc. of the 5th Annual ACM Workshop on COLT*, Pittsburgh/ACM Press, 144-152.

[14] Cano J. R., F. Herrera, and M. Lozano (2004). Evolutionary stratified instance selection applied to training set selection for extracting high precise-interpretable classification rules. *IEEE ICDM 2004 Workshop on Alternative Techniques for Data Mining and Knowledge Discovery*, Brighton, UK.

[15] Cano J. R., F. Herrera, and M. Lozano (2003). Using evolutionary algorithms as instance selection for data reduction in KDD: An experimental study. *IEEE Transactions on Evolutionary Computation*, 7(6), 561-575.

[16] Chauchat J. H. and R. Rakotomalala (2001). Sampling strategy for building decision trees from very large databases comprising many continuous attributes. H. Liu and H. Motoda (eds.). *Instance Selection and Construction for Data Mining*, Kluwer, 172-187.

[17] Congalton R. G. and K. Green (1999). *Assessing the accuracy of remotely sensed data: Principles and practices*. Lweis Publishers.

[18] Cook D., A. Buja, J. Cabrera, and H. Hurley (1995). Grand tour and projection pursuit. *Journal of Computational and Graphical Statistics* 2 3, 225-250.

[19] Cover T. M. and P. E. Hart (1967). Nearest neighbor pattern classification. *Institute of Electrical and Electronics Engineers Transactions on Information Theory* 13, 21-27.

[20] Davis L. (1991). *Handbook of genetic algorithms*. Van Nostrand Reinhold.

[21] Davis J. and M. Goadrich (2006). The relationship between precision-recall and ROC curves. *Proc. of the 23rd International Conference on Machine Learning*, Pittsburgh, Pennsylvania, 233-240.

[22] DeJong K. and W. Spears (1991). An analysis of the interacting roles of population size and crossover in genetic algorithms. H. Schwefel and R. Manner (eds.). *Parallel Problem Solving from Nature*, Springer-Verlag, 38-47.

[23] Devijver P. A. and J. Kittler (1982). *Pattern recognition: A statistical approach*. Prentice-Hall.

[24] Dhar V., D. Chou, and F. Provost (2000). Discovering interesting patterns for investment decision making with GLOWER-a genetic learner overlaid with entropy reduction. *Data Mining and Knowledge Discovery* 4, 251-280.

[25] Dietterich T. G., M. Kearns, and Y. Mansour (1996). Applying the weak learning framework to understand and improve C4.5. *Proc. of the 13th International Conference on Machine Learning*, San Francisco: Morgan Kaufmann, 96-104.

[26] Domingo C., R. Gavalda, and O. Watanabe (2001). Adaptive sampling methods for scaling up knowledge discovery algorithms. H. Liu and H. Motoda (eds.). *Instance Selection and Construction for Data Mining*, Kluwer, 134-149.

[27] Donzenis C. and Rakow E. A. (1987). Outliers: A potential data problem. *Annual meeting of the mid-south educational research association*, Mobile, AL.

[28] Duda R. O., P. E. Hart, and D. G. Stork (1997). *Pattern classification and scene analysis*. Wiley.

[29] Efron B. (1979). Bootstrap methods: Another look at the jackknife. *The Annals of Statistics* 7, 1-26.

[30] Endou T. and Q. Zhao (2002). Generation of comprehensible decision trees through evolution of training data. *Proc. of the 2002 Congress on Evolutionary Computation*, Honolulu, Hawaii, 1221-1225.

[31] Esposito F., D. Malerba, and G. Semeraro (1997). A comparative analysis of methods for pruning trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5), 476-492.

[32] Fayyad U. and K. B. Irani (1992). The attribute selection problem in decision tree generation. *Proc. of the 10th National Conference on Artificial Intelligence*, Cambridge, MA: MIT Press, 104-110.

[33] Fayyad U., G. Piatetsky-Shapiro, P. Smith, and R. Uthurusamy (1996). *Advances in knowledge discovery and data mining*. Cambridge, MA: MIT Press.

[34] Ferri C., P. Flach, and J. Hernandez-Orallo (2002). Learning decision trees using the area under the ROC curve. C. Sammut and A. Hoffmann (eds.). *Proc. of the 19th International Conference on Machine Learning*, San Francisco: Morgan Kaufmann, 139-146.

[35] Folino G., C. Pizzuti, and G. Spezzano (2001). Parallel genetic programming for decision tree induction. *Proc. of the 13th IEEE International Conference on Tools with Artificial Intelligence*, Dallas, 129-135.

[36] Fu H. Y. (2007). Interview on thyroid disease diagnosis through blood tests. Guangzhou University of Traditional Chinese Medicine, Guangzhou, China.

[37] Fu Z., B. Golden, S. Lele, S. Raghavan, and E. Wasil (2003a). A genetic algorithm-based approach for building accurate decision trees. *INFORMS Journal of Computing* 15, 3-22.

[38] Fu Z., B. Golden, S. Lele, S. Raghavan, and E. Wasil (2003b). Genetically engineered decision trees: Population diversity produces smarter trees. *Operations Research,* 51(6), 894-907.

[39] Fu Z., B. Golden, S. Lele, S. Raghavan, and E. Wasil (2004). Diversification for smarter trees. *Computers & Operations Research*, 33(11), 3185-3202.

[40] Gates G.W. (1972). The reduced nearest neighbor rule. *IEEE Transactions on Information Theory*, 18(3), 431-433.

[41] Glover F. and M. Laguna (1997). *Tabu Search*. Kluwer, Norwell, MA.

[42] Grefenstette J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1), 122-128.

[43] Hancock T. R., et al. (1996). Lower bounds on learning decision lists and trees. *Information and Computation* 126(2), 114-122.

[44] Hart P. E. (1968). The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14(3), 515-516.

[45] Hettich S. and S. D. Bay (1999). The UCI KDD archive [http://kdd.ics.uci.edu]. Irvine, CA: University of California, Department of Information and Computer Science.

[46] Hisao I., N. Tomoharu, and N. Manabu (2001). Learning of neural networks with GA-based instance selection. *Annual Conference of the North American Fuzzy Information Processing Society*, 4, 2102-2107.

[47] Holland J. H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor.

[48] Hyafil L. and R. L. Rivest (1976). Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1), 15-17.

[49] Ishibuchi H., T. Nakashima, and M. Nii (2001). GA-based instance selection and feature selection. H. Liu and H. Motoda (eds.). *Instance Selection and Construction for Data Mining*, Kluwer, 96-112.

[50] Jankowski N. and M. GrochowskiL (2004). Comparison of instances selection algorithms I: Algorithms survey. Rutkowski et al. (eds.). *Proc. of the 7th International Conference on Artificial Intelligence and Soft*, Zakopane: Springer, 598-603.

[51] Jensen F. V. (2001). *Bayesian networks and decision graphs*. Springer.

[52] Kennedy H., C. Chinniah, P. Bradbeer, and L. Morss (1997). The construction and evaluation of decision trees: A comparison of evolutionary and concept learning methods. Evolutionary Computing, D. Come and J. Shapiro (eds.). *Lecture Notes in Computer Science*, Berlin: Springer-Verlag, 147-161.

[53] Kibbler D. and D. W. Aha (1987). Learning representative exemplars of concepts: An initial case of study. *Proc. of the 4th International Workshop on Machine Learning*, Irvine: Morgan Kaufmann 24-30.

[54] Kirkpatrick S., C. D. Gelatt, and M. P. Vecchi (1983). Optimization by Simulated Annealing. *Science*, 220(4598), 671-680.

[55] Kohavi R. and F. Provost (1998). On applied research in machine learning. *Machine Learning*, 30, 127-132.

[56] Kolmogorov A. N. (1933). On the empirical determination of a distribution function. (Italian) *Giornale dell'Instituto Italiano degli Attuari*, 4, 83-91.

[57] Kononenko I. and I. Bratko (1995). Information based evaluation criterion for classifier's performance. *Machine learning*, 6: 67-80.

[58] Lam K. K. (1994). *Hybridized genetic algorithm and k-nearest neighbor for rainfall prediction*. Minor thesis, RMIT, Department of Computer Science, Melbourne, 19-20.

[59] Lam W., et al. (2001). Instance construction via likelihood-based data squashing. H. Liu and H. Motoda (eds.). *Instance Selection and Construction for Data Mining*, Kluwer, 228-242.

[60] Levine D. (1994). *A parallel genetic algorithm for the set partitioning problem*. PhD thesis, Argonne National Laboratory, Illinois Institute of Technology, 38-39.

[61] Little R. J. and Rubin D. B. (2002). *Statistical analysis with missing data*. John Wiley & Sons.

[62] Li X. and S. Olafsson (2004). Discovering dispatching rules using data mining. *Journal of Scheduling*, 8(6), 515-527.

[63] Li X. and S. Olafsson (2005). Optimal instance selection for learning best scheduling practices. *INFORMS Annual Meeting*, San Francisco, CA.

[64] Lim X., W. Y. Loh, and X. Shih (2000). A comparison of prediction accuracy, complexity and training time of thirty-three old and new classification algorithms. *Machine Learning* 40, 203-228.

[65] Liu H. and H. Motoda (2001). Data reduction via instance selection. H. Liu and H. Motoda (eds.). *Instance Selection and Construction for Data Mining*, Kluwer, 3-20.

[66] Loh W. Y. and X. Shih (1997). Split selection methods for classification trees. *Statistica Sinica*, 7, 815-840.

[67] Lopez de Mantras R. (1991). A distance-based attribute selection measure for decision tree induction. *Machine Learning* 6, 81-92.

[68] Lowe D. G. (1995). Similarity metric learning for a variable-kernel classifier. *Neural Computations*, 7(1), 72-85.

[69] Mehta M., R. Agrawal, and J. Rissanen (1996). SLIQ: A fast scalable classifier for data mining. *Proc. of the 5th International Conference on Extending Database Technology*, Avignon, France, 18-32.

[70] Miller I. and M. Miller (2004). *John E. Freund's mathematical statistics with application*, seventh edition. Pearson.

[71] Mingers J. (1989). An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4(2), 227-243.

[72] Mitchell M., S. Forrest, and J. H. Holland (1992). The royal road for genetic algorithms: Fitness landscapes and GA performance. *Proc. of the 1st European Conference on Artificial Life*, Cambridge, MA: MIT Press, 245-254.

[73] Mitchell T. M. (1997). *Machine learning*. McGraw-Hill.

[74] Naumov G. E. (1991). NP-completeness of problems of construction of optimal decision trees. *Soviet Physics*, Doklady, 36(4), 270-271.

[75] Niimi A. and E. Tazaki (2000). Genetic programming combined with association rule algorithm for decision tree construction. *Proc. of the 4th International Conference on Knowledge-based Intelligent Engineering Systems & Allied Technologies*, Brighton, UK, 746-749.

[76] Olaru C. and L. Wehenkel (2003). A complete fuzzy decision tree technique. *Fuzzy Sets and Systems*, 138(2), 221-254.

[77] Pagallo G. and D. Huassler (1990). Boolean feature discovery in empirical learning. *Machine Learning*, 5(1), 71-99.

[78] Provost F., D. Jensen, and T. Oates (2001). Progressive sampling. H. Liu and H. Motoda (eds.). *Instance Selection and Construction for Data Mining*, Kluwer, 152-169.

[79] Quinlan J. R. (1993). *C4.5: Programs for machine learning*. Morgan-Kaufmann.

[80] Reeves C. R. and D. R. Bush (2001). Using genetic algorithms for training data selection in RBF networks. H. Liu and H. Motoda (eds.). *Instance Selection and Construction for Data Mining*, Kluwer, 339-356.

[81] Reinartz T. (2001). A unifying view on instance selection. H. Liu and H. Motoda (eds.). *Instance Selection and Construction for Data Mining*, Kluwer, 40-54.

[82] Ripley B. D. (1996). *Pattern recognition and neural networks*. Cambridge.

[83] Rissanen J. (1985). The minimum description length principle. Kotz, S. and N. L. Johnson (eds.). *Encyclopedia of Statistical Sciences*, John Wiley, vol. 5, 523-527.

[84] Rokach L. and O. Maimon (2005). Decision trees. *The Data Mining and Knowledge Discovery Handbook*, Springer, 166-187.

[85] Rousseeuw P. J. and A. M. Leroy (1987). *Robust regression and outlier detection*. John Wiley & Sons.

[86] Shannon C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, vol. 27, 379-423 and 623-656.

[87] Skalak D. B. (1994). Prototype and feature selection by sampling and random mutation hill climbing algorithms. *Proc. of the 11th International Conference on Machine Learning*, New Brunswick, NJ: Morgan Kaufmann, 293-301.

[88] Snedecor G. W. and W. G. Cochran (1989). *Statistical Methods*, eighth edition. Iowa State University Press.

[89] Spears W. M. and K. A. DeJong (1991). An analysis of multi-point crossover. *Foundation of Genetic Algorithms*, Morgan Kaufman, 301-315.

[90] Swayne D. F., et al. (2003). GGobi: Evolving from XGobi into an extensible framework for interactive data visualization. *Computational Statistics & Data Analysis*, 43, 423-444.

[91] Syswerda G. (1989). Uniform crossover in genetic algorithms. *Proc. of the 3rd International Conference on Genetic Algorithms*, George Mason University: Morgan Kaufman, 2-8.

[92] Tate D., M. David, and A. E. Smith (1993). Expected allele coverage and the role of mutation in genetic algorithms. *Technical report*, Department of Industrial Engineering, University of Pittsburgh.

[93] Tukey J. W. (1977). *Exploratory data analysis*. Addison-Wesley.

[94] Wallace C. S. and J. Patrick (1993). Coding decision trees. *Machine Learning* 11: 7-22.

[95] Wang H. (2001). Instance selection based on hypertuples. H. Liu and H. Motoda (eds.). *Instance Selection and Construction for Data Mining*, Kluwer, 246-261.

[96] Weiss S. M. and C.A. Kulikowski (1991). *Computer systems that learn: Classification and prediction methods from statistics, neural nets, machine learning, and expert systems*. Morgan Kaufman.

[97] Wilson D. L. (1972). Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems Man, and Cybernetics*, 2(3), 408-421.

[98] Wilson D. R. and T. R. Martinez (1997). Instance pruning techniques. *Proc. of the 14th International Conference on Machine Learning*, Nashville, TN: Morgan Kaufmann, 403-411.

[99] Wilson D. R. and T. R. Martinez (2000). Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38, 257-268

[100] Witten I. H. and E. Frank (2000). *Data mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann.

[101] Wu S. and S. Olafsson (2005). Optimal instance selection for smaller decision trees. *Journal of Optimization Methods and Software*, submitted.

[102] Yoon H., K. AlSabti, and S. Ranka (2001). Incremental classification using tree-based sampling for large data. H. Liu and H. Motoda (eds.). *Instance Selection and Construction for Data Mining*, Kluwer, 190-205.

[103] Young A. (1990). Genetic algorithms and scientific method. *Evolving Knowledge in Natural Science and Artificial Intelligence*, Pitman Publishing, 33-53.

[104] Zantena H. and H. L. Bodlaender (2000). Finding small equivalent decision trees is hard. *International Journal of Foundations of Computer Science*, 11(2), 343-354.

[105] Zhu D., S. Wu, and X. B. Li (2006). A data reconstruction approach for identity disclosure protection. *Proc. of the 6th Security Conference*, Las Vegas, accepted.

ACKNOWLEDGEMENTS

The work with this dissertation has been extensive and challenging, but in the first place exciting, instructive and interesting. Without help, support and encouragement from several persons, I would never have been able to finish this work.

First of all, I would like to take this opportunity to extend my deepest gratitude to my major professor, Dr. Sigurdur Olafsson, for his inspiring and encouraging way to guide me to a deeper understanding of the knowledge on optimization and data mining, and his invaluable comments during the whole work with this dissertation. I am also grateful to other professors in my academic committee for their efforts and contributions to this work, especially Dr. Dan Zhu, who provides me with a great application of my research and Dr. Di Cook, who has a sharp eye for details and superb analytical skill. The informative advice and instructions from the professors have been instrumental and crucial in the progress of my research.

My gratitude also goes to staff, students and colleagues at the Department of Industrial and Manufacturing Systems Engineering for developing a good working atmosphere, especially Dr. Xiaonan Li, Ms. Vuthipadadon Somchan and Mr. Lee Jong-Seok, for their assistance and companionships throughout my work.

Last but not least, I am greatly indebted to my wife, my lovely little daughter and my family for their understanding, patience and support during the entire period of my study.

VITA

NAME OF AUTHOR: Shuning Wu

DATE AND PLACE OF BIRTH: July 9, 1979, Guangzhou, Guangdong, P. R. China

DEGREES AWARDED:

B. S. in Automation, Tsinghua University, Beijing, China, 2001

M. S. in Control Science and Engineering, Tsinghua University, Beijing, China, 2004

HONORS AND AWARDS:

Tsinghua University Outstanding Undergraduate Student Fellowship, 2001

Tsinghua University Outstanding Leader Award, 2003

Iowa State University Graduate College Premium for Academic Excellence Award, 2004

Iowa State University Teaching Excellence Award, 2007

PROFESSIONAL EXPERIENCE:

Research Assistant, Department of Industrial Engineering, Iowa State University, 2004

Research Assistant, Department of Electrical Engineering, Iowa State University, 2006

Research Assistant, Department of LOMIS, Iowa State University, 2006

Teaching Assistant, Department of Industrial Engineering, Iowa State University, 05-07