A Genetic Algorithm-Based Approach for Building Accurate Decision Trees

Zhiwei Fu • Bruce L. Golden • Shreevardhan Lele • S. Raghavan • Edward A. Wasil
Fannie Mae, 4000 Wisconsin Avenue, NW, Washington, DC 20016, USA
Robert H. Smith School of Business, University of Maryland, College Park, Maryland 20742, USA
Robert H. Smith School of Business, University of Maryland, College Park, Maryland 20742, USA
Robert H. Smith School of Business, University of Maryland, College Park, Maryland 20742, USA
Kogod School of Business, American University, Washington, DC 20016, USA
zhiwei_fu@fanniemae.com • bgolden@rhsmith.umd.edu • slele@rhsmith.umd.edu
raghavan@umd.edu • ewasil@american.edu

In dealing with a very large data set, it might be impractical to construct a decision tree using all of the points. Even when it is possible, this might not be the best way to utilize the data. As an alternative, subsets of the original data set can be extracted, a tree can be constructed on each subset, and then parts of individual trees can be combined in a smart way to produce an improved final set of feasible trees or a final tree. In this paper, we take trees generated by a commercial decision tree package, namely, C4.5, and allow them to crossover and mutate (using a genetic algorithm) for a number of generations in order to yield trees of better quality. We conduct a computational study of our approach using a real-life marketing data set. In this study, we divide the data set into training, scoring, and test sets, and find that our approach produces uniformly high-quality decision trees. In addition, we investigate the impact of scaling and demonstrate that our approach can be used effectively on very large data sets.
(*Genetic Algorithm; Decision Tree*)

1. Introduction

A decision tree is one of the most popular methods for discovering meaningful patterns and classification rules in a data set. The rigorous use of decision trees began with Breiman et al. (1984); the reader is referred to Chapter 7 of Ripley (1996) for an introduction to the field. However, in dealing with a very large data set, it might be impractical or even impossible (due to memory limitations) to construct a tree using all of the data points. Even when it is possible, this might not be the best way to utilize all of the data.

1.1. Literature Review

Over the last decade or so, researchers have focused on generating accurate and concise decision trees from large data sets by applying genetic algorithms and a variant, genetic programming. Genetic algorithms (GAs) were introduced and developed by Holland (1975) and have been used in a wide variety of applications including combinatorial optimization and knowledge discovery (see Fayyad et al. 1996). In a GA, an initial population of chromosomes is generated and the quality (fitness) of each chromosome is determined. Parents are selected and mated (crossed over) to form offspring. The offspring are then mutated and the resulting chromosomes are added to the population. A set of fittest chromosomes is allowed to survive to the next generation and the process repeats for a specified number of generations.

A sketch of a typical GA is given in Figure 1. The basic algorithm modifies the original population using three genetic operators: selection, crossover, and

0899-1499/03/1501/0003$05.00
1526-5528 electronic ISSN

```
Choose an initial population of chromosomes;
Evaluate the initial population;
While termination condition not satisfied do
  Select the current population from the previous one;
  If crossover condition satisfied, perform crossover;
  If mutation condition satisfied, perform mutation;
  Evaluate fitness of offspring;
Endwhile
```

Figure 1 Sketch of a Genetic Algorithm

mutation. After many generations (e.g., several hundred), the population evolves to a near-optimal solution (see Michalewicz 1996 for more details).

Genetic programming (GP) is a variant of genetic algorithms that was developed in the early 1990s by Koza (1992). In GP, the individuals are computer programs that are evolved using the genetic operators of reproduction, crossover, and mutation. GP has been applied to a wide variety of problems in science, computer science, and engineering (see Chapter 12 in Banzhaf et al. (1998) for a very nice discussion of GP applications in these areas).

Koza (1991) demonstrated the technique of inducing a decision tree using GP on a small training set of 14 objects taken from Quinlan (1986) when only accuracy was considered. In one run, on the eighth generation, his technique correctly classified all 14 training cases. More recently, Nikolaev and Slavov (1998), Marmelstein and Lamont (1998), Folino et al. (2000), and Bot and Langdon (2000) have generated decision trees using GP.

Ryan and Rayward-Smith (1998) evolved decision trees in two ways—using genetic programming and a genetic algorithm. The genetic programming technique was tested against the popular C4.5 software due to Quinlan (1993) on four data sets taken from the UCI Machine Learning Repository (Blake and Merz 1998). The data sets ranged in size from six attributes and 124 records to 60 attributes and 3,190 records. The data sets were used to build (train) decision trees and accuracy was recorded; a test data set was not used. The computational results were encouraging. On the three smallest data sets, GP evolved trees that were more accurate than C4.5 with execution times that ranged from 4 to 14 minutes on a 433 Mhz DEC Alpha workstation. C4.5 was more accurate on the largest data set (it generated a tree very quickly, whereas the GP technique took nearly 80 minutes).

Ryan and Rayward-Smith also developed a genetic algorithm to evolve decision trees using only small populations. The fitness function took into account both the size of the tree (number of nodes) and accuracy (number of misclassifications). The initial population of trees was generated by using a modified version of C4.5. The GA was tested against C4.5 on six UCI data sets. Here, a training data set and a test data set were used. A crossover operator was applied, but mutation was not used (the authors experimented with several mutation operators but none improved the results of the GA). For the most part, the GA outperformed C4.5 in the computational experiments; GA produced smaller trees with a higher degree of accuracy using only a tiny population. However, the GA was computationally expensive to run as it needed to call C4.5 many times to generate new solutions (the experiment on the largest data set of 14 attributes and 32,561 records took 25 hours). Ryan and Rayward-Smith concluded that the GA's execution time needed to be reduced in order to have a scalable data mining algorithm. In addition, they suggested that it would be interesting to create an ensemble of classifiers consisting of the best tree produced by many runs and compare the accuracy of the ensemble to other techniques.

GP and GAs have been used in a variety of data mining applications like clustering and rule discovery (see Freitas 2002a, b) and there are tradeoffs to consider when choosing between the methods. For example, GAs use a fixed-size string of bits to represent an individual (more about this in the next section) that is suitable for many problems, while GP uses a variable-length program (see Bojarczuk et al. 2000) that can represent complex structures. As pointed out by Bojarczuk et al. (they use GP to discover classification rules in chest pain diagnosis), these structures give GP an advantage over GAs by allowing GP to perform mathematical, logical, and relational operations. However, the infinite space induced by the representation in GP is computationally intensive to search and requires intelligent and adaptive search techniques. In general, GP requires a lot of CPU time and a lot of memory and, in addition, scalability is a major concern.

1.2. Background

The starting point for the approach we develop in this paper is the work of Kennedy et al. (1997). They represent a decision tree as a linear array (chromosome) and apply a genetic algorithm to the optimization of the decision tree with respect to its classification accuracy. Their program (called CALTROP) represents a binary decision tree by a number of unit subtrees (called caltrops), each having a root and two branches (a subtree of a decision tree is the tree obtained by taking an attribute node or leaf node and all of its descendants). Each caltrop is a string of three integers that represent the binary attributes corresponding to the root, the left child, and the right child of the subtree, respectively. Although the authors do not explicitly say so, to be consistent, either the left child or the right child should always correspond to the branch where the condition tested by the binary attribute is satisfied. The integer 0 represents a leaf node of the decision tree. The linear representation is made up of a set of caltrops, the first one representing the root of the tree. After the first one, the order of the caltrops does not matter.

Figure 2 provides an example of a classification tree and Kennedy et al.'s representation of the tree. Note that if there are q attributes that are tested in the decision tree then Kennedy et al.'s representation has $3q$ integers (one caltrop for each binary attribute tested in the tree).

To construct a decision tree from the linear chromosome representation, a requirement for the genetic algorithm, Kennedy et al. proceed in the following depth-first manner:

- Start with the first caltrop (string of three integers) of the chromosome as the root of the decision tree. Construct a partial decision tree with the root and two daughter nodes.
- Build the tree in a depth-first manner, investigating left daughter nodes before right daughter nodes.
- The investigation of a node in the tree is recursive, and replaces the node by the caltrop corresponding to the attribute of the node. This is accomplished by finding the *first* caltrop in the chromosome whose root is the attribute being searched for.

Figure 3 provides an example showing how Kennedy et al. construct a tree from the linear representation. As can be seen from the above description, the Kennedy et al. representation requires that when an attribute is encountered at different locations within the tree, the course of action following the attribute is *identical* at all locations.

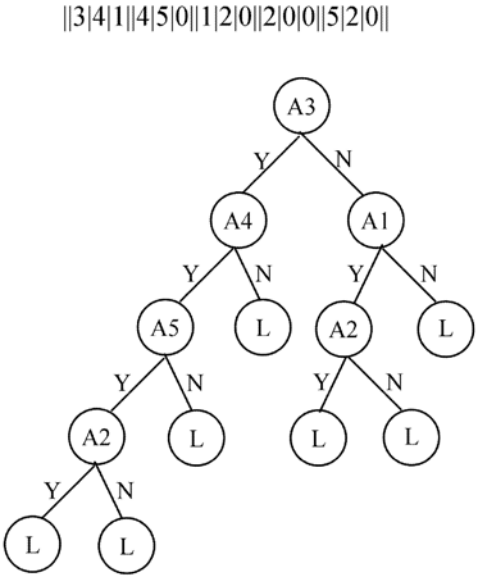

Figure 2 A Classification Tree and the Kennedy et al. Representation
Note. *Ai* represents binary attribute *i*; *Y* (yes) and *N* (no) represent whether the binary attribute is satisfied; and *L* represents a leaf node (terminating condition). Double vertical lines (||) represent boundaries between caltrops in the linear representation.

Using a caltrop as an indivisible genetic alphabet, Kennedy et al. apply a standard crossover procedure for linear chromosomes (by splitting the two parent chromosomes, and combining them to obtain two daughter chromosomes). This can cause some ambiguity when (i) there is no caltrop for an attribute that occurs within the chromosome, (ii) the chromosome contains multiple caltrops with the same attribute (at the root of the caltrop). To rectify these two problems, Kennedy et al. (i) prune the tree making the attribute node a leaf node when there is no caltrop for an attribute within the chromosome, and (ii) only consider the first occurrence when there are multiple caltrops for the same attribute.

Kennedy et al. focus on correctly and completely classifying small artificially constructed data sets (i.e., they construct trees with 100% classification accuracy). Further, they evaluate trees with respect to the

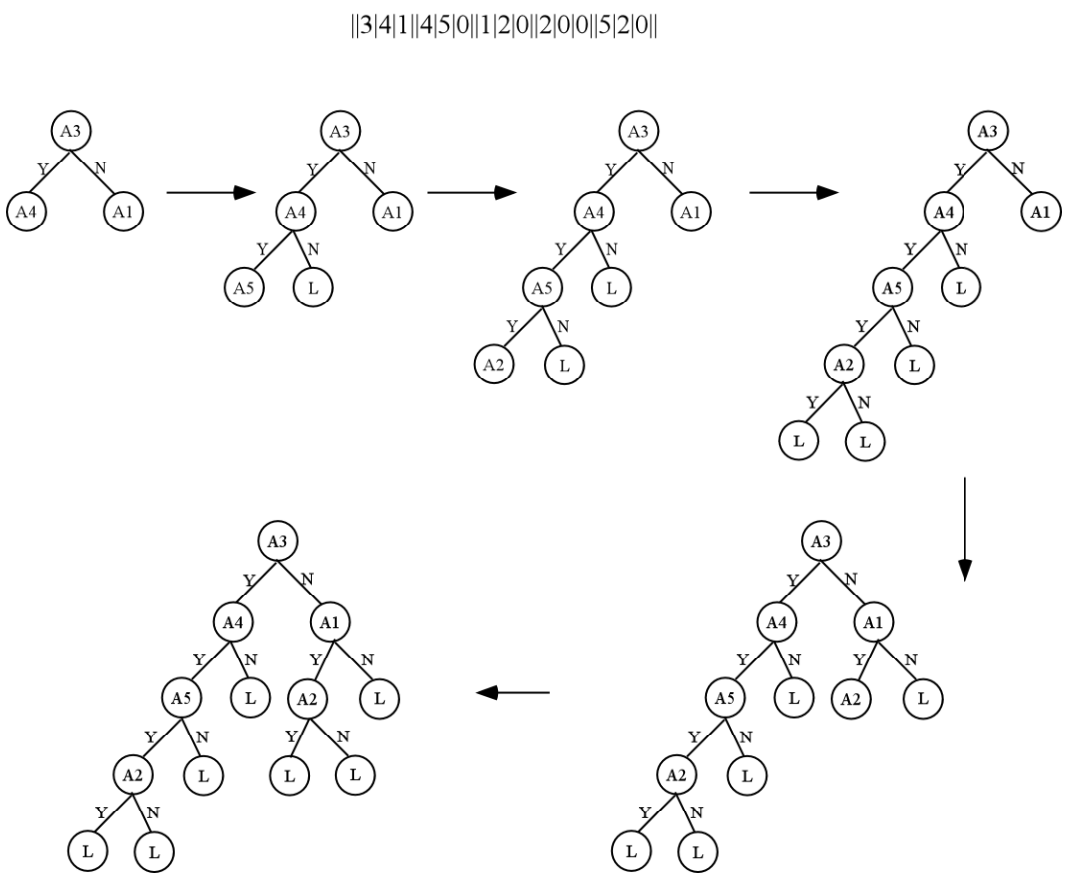

Figure 3 Procedure of Kennedy et al. to Construct a Classification Tree from a Linear Chromosome Representation

number of decisions that need to be made to classify the trees correctly. In their comparisons to ID3 (the precursor to C4.5), using five artificially constructed data sets, they find that CALTROP outperforms ID3 once, is tied twice, and is worse twice. Kennedy et al. do not report the running times of their computational experiments.

Typically, when decision trees are constructed, two data sets are involved: a training data set is used to build the decision tree and a test data set is used to evaluate how the tree classifies records upon which it has not been trained. Focusing solely on the classification accuracy for the training set results in overfitting. Such trees tend to be very large and contain rules that may specifically pertain to the training data set. These trees, typically, classify the test data set poorly. Since Kennedy et al. build trees to classify the training data set accurately, their approach is likely to do poorly on the test data set.

1.3. Organization of Paper

In this paper, we propose the following three-step approach for generating trees. First, extract many subsets of points from the original data set for analysis. Second, using C4.5, construct a decision tree on each subset. Third, use a genetic algorithm that allows the trees to crossover and mutate in order to generate trees of better quality. We call our approach GAIT—*Genetic Algorithm approach for generating Intelligent Trees*. In this study, we test the efficacy of GAIT in quickly generating trees that have high classification accuracy.

We note that the last decade has seen the development of several ensemble methods like boosting and bagging, for combining a set of classifiers (such

as trees) in order to arrive at a more accurate classifier (see Bauer and Kohavi 1999 for an overview of such methods). Boosting algorithms (e.g., AdaBoost by Freund and Schapire 1997 and Arc-x4 by Breiman 1998) create an initial set of trees by adaptively changing the distribution of the training set. These trees are then combined according to a system of weights. In bagging (Breiman 1996), the initial set of trees is produced from bootstrapped samples obtained from a fixed training set. These trees are combined using a majority voting rule. Our proposed method, GAIT, resembles bagging in the creation of the initial set of trees. However, instead of creating bootstrapped samples from the training set, we partition the training set into sub-samples. Of course, GAIT's strategy for combining the resulting initial set of trees is very different from the majority voting rule of bagging. A simple advantage of GAIT over ensemble methods such as boosting and bagging is that the end product of GAIT is a single decision tree. In boosting and bagging, the entire ensemble of initial trees has to be retained in order to classify any future observation. Furthermore, as we demonstrate in this paper, GAIT also improves upon the accuracy that is obtained from a majority voting rule system such as bagging.

We point out that a comparable alternative to our approach is to use an adaptive resampling method such as GLOWER (Dhar et al. 2000). Instead of using a complete decision tree as the chromosomal unit as we do, GLOWER takes a chromosome to be simply a rule (in the context of trees, a rule is any complete path from the root node to a leaf node). A rule is then implemented as a database query.

In Section 2, we describe GAIT. In Section 3, we conduct several computational experiments that are designed to test the accuracy and speed of GAIT on a real-life marketing data set of 440,000 points and on a second data set of observations on 48,842 adult individuals. In Section 4, we present and discuss a scalability experiment. In Section 5, we give our conclusions and provide possible directions for future research with GAIT.

2. GAIT

In GAIT, we develop an intelligent search technique using a genetic algorithm. First, we select a simple random sample of points from the original data set to serve as our training set. Second, we partition this training set into several subsets of points. Each subset is then used to generate a decision tree using C4.5. These decision trees are taken as inputs (i.e., the initial population of trees) to our genetic algorithm.

There are three key issues involved in genetic algorithms: the initial population of trees, the genetic operations, and the evaluation of a tree. We point out that, in contrast to Kennedy et al., we work directly with the binary tree representation. Our crossover and mutation operations are designed specifically for a binary tree. Consequently, we can represent any binary decision tree, even those where different courses of action are taken when the binary attribute is encountered at different locations within the tree. Further, our crossover and mutation operations always generate complete decision trees (i.e., no missing attributes as in Kennedy et al.) In our exposition of the genetic algorithm we will describe them using the binary tree data structure. Later, we will describe how the same may be accomplished with a linear representation (if desired).

Initial Population of Trees. Many different methods can be used to generate the initial population of decision trees. It is important that the initial population of trees contains a wide variety of structures. In GAIT, the initial population is created from the trees generated by C4.5 on the subsets of the training set.

Operations. Selection, crossover, and mutation are the three major operations that we use in our algorithm. Selection is the process of choosing trees for the next generation from the trees in the current generation. Crossover performs exchanges of subtrees between trees. Mutation, in our application, is an operation that exchanges subtrees within a given tree. Trees are randomly selected for the crossover and mutation operations. The probability of being selected for one of these operations is proportional to the fitness score of the tree. The fitness score is described in the following paragraph on tree evaluation. In Figure 4, we show two types of crossover operations in our program, a subtree-to-subtree crossover and a subtree-to-leaf crossover (a leaf-to-leaf crossover is also possible). Crossover is performed by exchanging

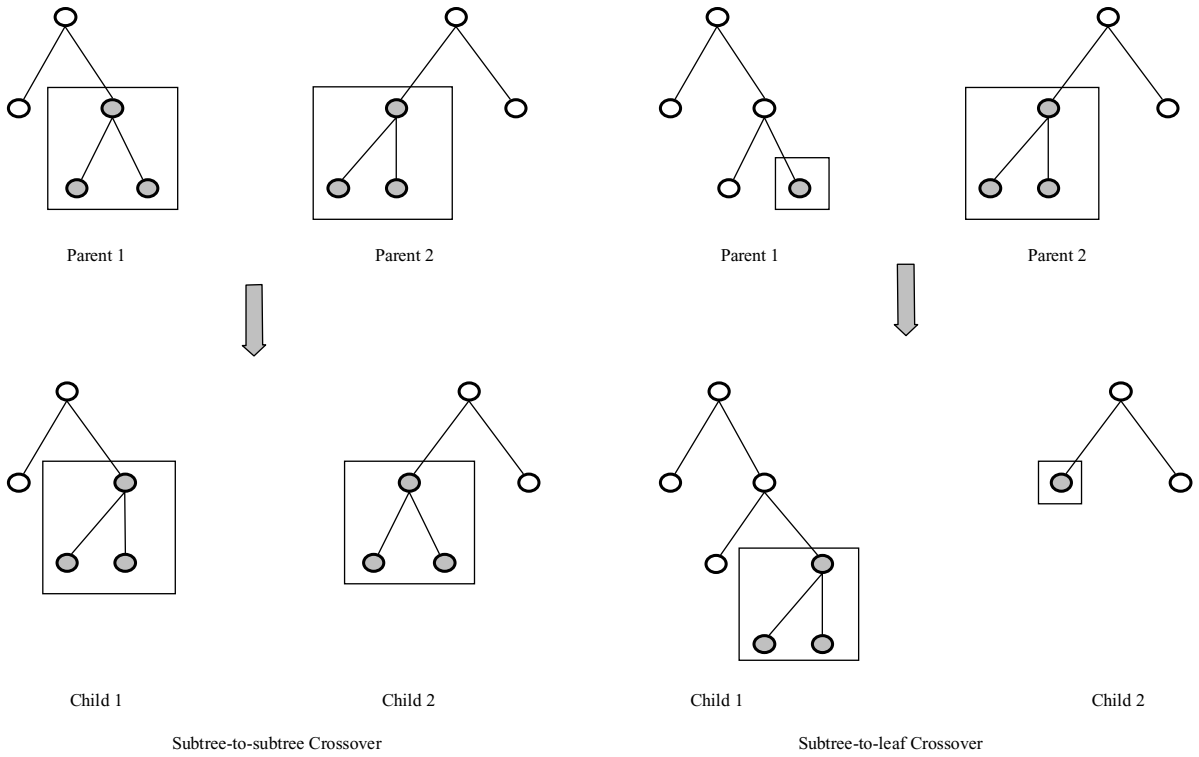

Figure 4 Crossover Operations

a subtree or a leaf from one tree with a subtree or a leaf of another tree at any level. We determine the subtrees for crossover by random selection. This can be done by choosing a node randomly in the tree and selecting the subtree rooted at that node as the subtree for crossover. In Figure 5, two types of mutation are performed within a tree by occasionally randomly exchanging a subtree with another subtree or a leaf (leaf-to-leaf mutation is also possible).

Notice that the crossover and mutation operations do not result in trees with binary attributes at leaf nodes. Thus, we do not have the two problems that Kennedy et al. encounter when they apply crossover and mutation.

Evaluation. A central issue in successfully applying a genetic algorithm to any problem is to define the fitness criterion appropriately. In GAIT, we evaluate the fitness of the decision trees by calculating the percentage of correctly classified observations in a scoring data set. The scoring set is a random sample from the original data set. The training set and the scoring set are mutually disjoint.

During evolution, some of the decision trees might have logic violations after crossover and mutation, so that we need to perform a logic check (we call this a feasibility check). The feasibility check can be carried out immediately after each crossover operation and each mutation operation or it can be delayed to the end of the final generation. The computational time for the feasibility check depends mainly on the size of the generated decision trees and the size of the scoring data set. Preliminary computational testing reveals that the feasibility check does not consume a significant amount of running time. We remark that it is possible to design more complex and sophisticated crossover and mutation operations. This would probably lead to an increase in computational expense, but might also result in superior solutions. Our choice, in this paper, is to keep the crossover and mutation operations simple. The associated computational expense is relatively small and the genetic algorithm is effective (as evidenced by our computational results in Sections 3 and 4).

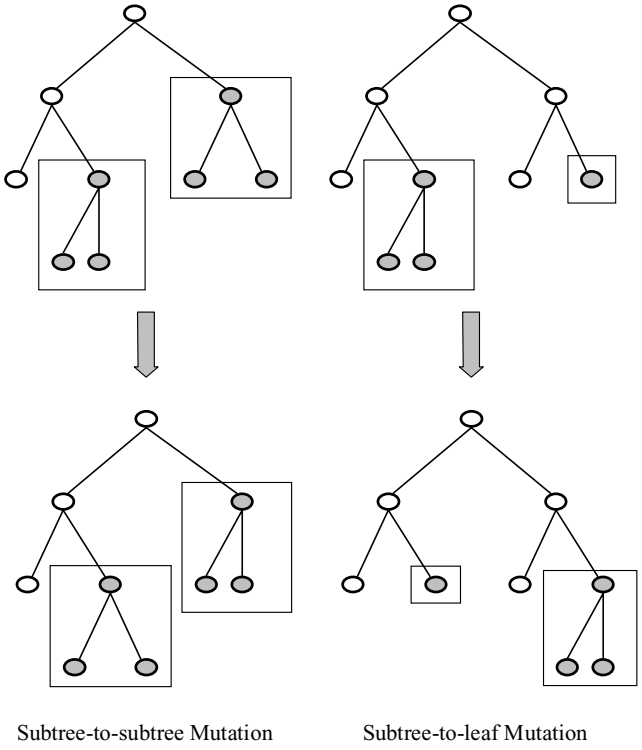

Figure 5 Mutation Operations

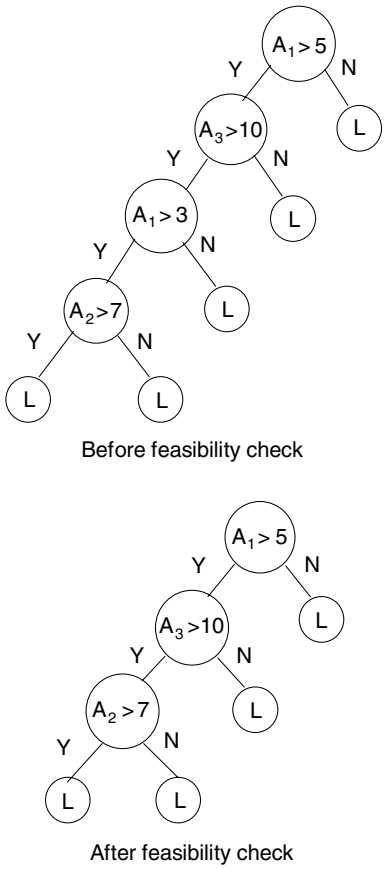

Figure 6 Feasibility Check

Our feasibility check involves the elimination of all logic violations. We illustrate the feasibility check in Figure 6. In Figure 6, there is a logic violation with $A_1 > 5$ and $A_1 \leq 3$ in the tree on the top. Consequently, the subtree associated with $A_1 \leq 3$ would be eliminated and the subtree associated with $A_1 > 3$ would be concatenated since $A_1 > 5$ implies $A_1 > 3$.

After the tree has been concatenated to ensure that there are no logic violations, it is pruned (see Figure 7) to increase its ability to classify other data sets (e.g., the test set). Pruning starts from the bottom of the tree and examines each nonleaf subtree. Each subtree is replaced with a leaf if the replacement results in an increase in the error rate that is smaller than a prespecified upper bound (e.g., 0.5%).

Although we work directly with the binary tree representation, for completeness, we now briefly describe how the same may be accomplished via a linear representation. In this discussion, we assume familiarity with depth-first search (DFS) algorithms at the level of Cormen et al. (1990). A linear representation can be obtained via a depth-first ordering of all the attribute nodes in the decision tree. For example, a depth-first ordering of all attribute nodes in the binary decision tree shown in Figure 2 is 3-4-5-2-0-0-0-0-1-2-0-0-0. The positive integers represent the binary attribute and 0 represents a leaf node. The convention is that the yes branch is to the left and the no branch is to the right. With this ordering, and because 0 signifies a leaf node, it is straightforward to see that (i) the representation can be constructed from the tree in linear time (since DFS takes linear time), and (ii) a binary decision tree can be constructed from this linear representation in linear time. Depth-first ordering has a unique and particularly useful ordering property. Each subtree is a contiguous string in the linear representation. Further, the subtree may be constructed by using the substring corresponding to it in the linear representation in a similar manner to which the entire tree is constructed. Thus, by keep-

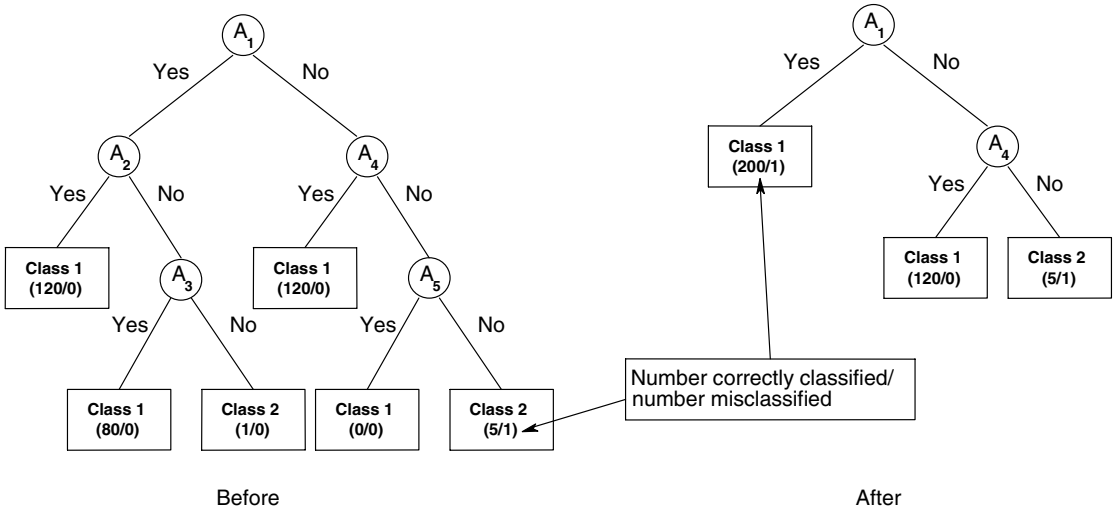

Figure 7 Pruning

ing track of the position of the end node in the substring for each node in the linear representation, it is easy to see that the crossover and mutation operations in our genetic algorithm may also be done using a linear representation (if desired). For example, subtree-to-subtree crossover may be accomplished by switching two substrings, corresponding to the subtrees, between parent nodes.

In this paper, we have restricted our attention to binary decision trees. However, any poly-ary tree may be converted to a binary tree as shown in Figure 8. The procedure to convert a poly-ary decision tree to a binary decision tree is a top-down procedure on the decision tree. Whenever a non-binary internal node is encountered in the poly-ary tree the procedure replaces it by a binary decision tree structure as shown in Figure 8. Notice that the number of leaf nodes in the binary decision tree and the original decision tree are identical. Further, the number of operations required to perform this transformation is proportional to the number of nodes in the binary decision tree. It is well known (e.g., see Deo 1974) that a binary tree with L leaves has $2L$-1 nodes. Consequently, the transformation takes linear time.

As a result, our approach can be generalized to poly-ary decision trees by first converting the initial set of trees to binary trees, and then applying our genetic algorithm. Observing that the number of nodes in the binary tree representation of a poly-ary tree is at most twice the number of nodes in the poly-ary tree, it follows that the running time of the genetic algorithm (from a worst-case computational complexity standpoint) as a function of the number of nodes in the decision trees remains unchanged (since the depth of a d-ary tree with n nodes is $O(\log_d n)$, we consider the computational complexity in terms of the number of nodes in the tree).

Alternatively, if one works directly with the poly-ary tree representation (instead of the binary tree representation), it is easy to extend the genetic algorithm, as the crossover and mutation operations easily generalize. The running time of the genetic algorithm procedure applied to poly-ary trees as a function of the number of nodes in the poly-ary trees must be similar to the genetic algorithm procedure applied to binary trees with the same number of nodes (because a binary tree is a special case of a poly-ary tree). Noting that the number of nodes in the poly-ary trees is at most a constant times the number of nodes in the binary representation of a poly-ary tree, the running time of the two approaches should be equivalent. However, from a practical standpoint, using a poly-ary tree structure may run faster since the size of the poly-ary trees would be more compact than their binary representation.

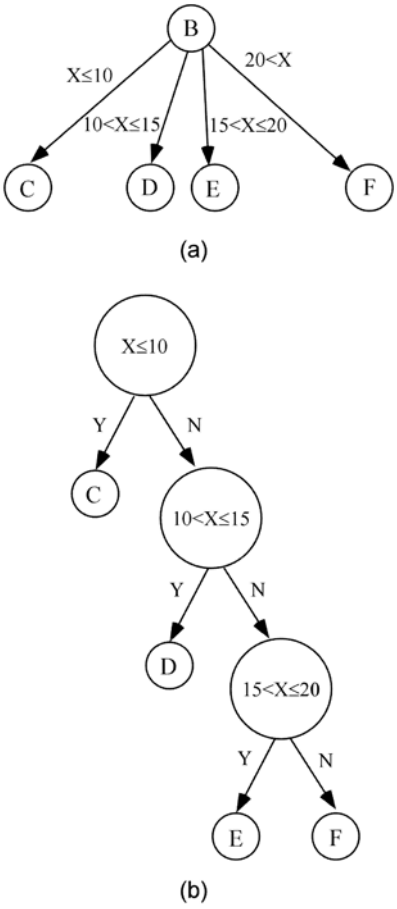

Figure 8 Transformation to a Binary Decision Tree
Note. (a) A poly-ary node with four daughters and (b) its binary counterpart.

3. Computational Experiments

In this section, we describe the series of computational experiments that we conducted to test the performance of GAIT with respect to accuracy and speed. The tests were conducted on a real-life marketing data set obtained from a firm in the transportation services industry. These experiments build upon and substantially extend earlier work by Fu et al. (2000). The data set contains information on approximately 440,000 customers. Each customer record contains demographic and usage information. Furthermore, each customer is identified on the basis of whether or not the customer reuses the firm's services in a certain window of time following a marketing promotion. It is of considerable interest to the firm to identify patterns among repeat customers. Recognition of such patterns is beneficial not only for developing a focused marketing strategy, but also for evaluating existing marketing efforts.

By way of preprocessing the data, we reduced the number of variables in the original data set from 33 to 11. This was necessary in order to protect the confidentiality of the data source. The aim of the decision tree procedure is to identify, on the basis of these 11 variables, whether or not a customer will reuse the firm's services in the specified window of time. Of the 440,000 customers in the data set, 21.63% are observed to be repeat customers, while the remaining 78.37% are not repeat customers. Classification errors for each class are treated equally, i.e., the primary performance measure is the *overall* classification accuracy, hereafter referred to as simply the *classification accuracy*. The Bayes risk (see Ripley 1996) for this data set is not known. We coded GAIT in Microsoft Visual C++ 5.0 and ran our experiments on a Windows 95 PC with a 400 MHz Pentium II processor and 128 MB of RAM.

3.1. Experimental Design

From the original data set of 440,000 points, we randomly selected 1,000 points and set them aside to form a test set. Note that the test set is not available in the development of any of the classifiers discussed. Next, we obtained a simple random sample (without replacement) of 3,000 points to serve as the training set. An additional 1,500 points were randomly selected from the remaining 436,000 data points to serve as the scoring set. Thus, the combined size of the training and scoring sets was approximately 1% of the original data set. In order to test the accuracy and speed of GAIT, we designed three different experiments.

In Experiment 1, the 3,000 points in the training set were randomly partitioned into 50 subsets of 60 points each. Each subset was used to create a decision tree using C4.5. The 50 trees so obtained served as the first generation for GAIT. At each step in GAIT, the fitness score of a tree was computed by calculating the classification accuracy of the tree on the scoring set. The genetic algorithm was allowed to evolve for ten generations using an elitist strategy, with the crossover and mutation probabilities at each generation set at one and 0.01, respectively. After ten generations, the tree with the highest fitness score was

designated as the best tree generated by GAIT. Then, we computed the accuracy of the best GAIT tree on the test set.

In Experiment 2, we used the training set of 3,000 to obtain ten random samples of 60 points each using sampling without replacement. Each sample was used to create a decision tree as in Experiment 1, which then served as a member of the first generation for GAIT. Thus, Experiment 2 is distinguished from Experiment 1 in having a smaller initial population for the genetic algorithm. Again, we computed the accuracy of the best generated decision tree on the test set.

In Experiment 3, we seek to investigate the behavior of GAIT when the first generation of trees is of particularly poor quality, i.e., when the initial population consists of trees of low classification accuracies. Such trees were obtained as follows. As in Experiment 1, the training set of 3,000 points was randomly partitioned into 50 subsets of 60 points each, leading to 50 decision trees. Each tree was then scored on the scoring set on the basis of its classification accuracy. The ten lowest-scoring trees were retained as the first generation for the genetic algorithm.

For each experiment, we obtained two performance measures—classification accuracy on the test set and total computing time for training and scoring. These measures were obtained in each experiment for four different decision trees (described below). An alternative approach for predicting reuse is Logistic Regression (see Menard 1995) for details), a standard statistical method available in SAS. We include it in this paper for benchmarking purposes. A detailed discussion of our implementation of Logistic Regression may be found in Fu (2000).

With respect to decision trees, first, we obtained the performance measures for the decision tree obtained from using the entire training set prior to partitioning. We denote this by *Whole-Training*. In Experiment 1, the Whole-Training set consists of all 3,000 points, while in Experiments 2 and 3 it consists of the set of 600 points that is formed by the union of the ten subsets that create the ten first-generation trees. We note that, in Experiment 2, the Whole-Training set of 600 points is a simple random sample from the original set of 3,000 points, while in Experiment 3 the Whole-Training set is not a simple random sample from the original set of 3,000 points. Second, we obtained performance measures for the *Best-Initial* tree, which we define as the best tree (as measured by its classification accuracy on the scoring set) in the first generation of each experiment. Third, we aggregated the first generation of trees using a majority voting rule to create a "bagged" classifier. (Strictly speaking, this is not bagging: the trees are not generated from bootstrapped samples from the given training set, but rather from sub-sampling without replacement from the training set.) We denote this classifier by *Aggregate-Initial*. Next, we obtained performance measures for the best tree generated by that variant of GAIT where each tree generated through each genetic operation is checked for feasibility. Recall from Section 2 that the feasibility check consists of searching a tree for logic violations and redundancies, and then pruning the tree to eliminate these conditions. We denote this procedure by *GAIT-w-FC* (GAIT with feasibility check). Next, we ran GAIT without the feasibility check after each operation (we denote this by *GAIT-w/o-FC*) and recorded the performance measures of the best tree so obtained. To ensure clarity, we point out that, in GAIT-w-FC, we perform a feasibility check after each operation, whereas, in GAIT-w/o-FC, we perform a feasibility check only at the end of the final generation.

For each experiment, we set the size of the scoring set at three different levels in turn: 500 points, 1,000 points, and 1,500 points. The smaller scoring sets are proper subsets of the larger scoring sets. In order to enhance and assess the reliability of the performance measures, each experiment/scoring-size combination was replicated ten times. The mean and standard error of each performance measure were computed for each experiment/scoring-size combination using the test set of 1,000 points.

3.2. Results

In Tables 1 and 2, we give the average performance measures, with the standard errors listed in parentheses. In Table 1, we present the classification accuracy and, in Table 2, we present the computing time. The total number of data points in the training and

Table 1 Classification Accuracy on the Test Set

Size	Method	Experiment 1	Experiment 2	Experiment 3
500	Logistic Regression	0.7563 (0.0091)	0.7412 (0.0067)	0.7228 (0.0069)
	Whole-Training	0.7612 (0.0087)	0.7491 (0.0064)	0.7226 (0.0071)
	Best-Initial	0.7525 (0.0069)	0.7392 (0.0059)	0.6767 (0.0057)
	Aggregate-Initial	0.7853 (0.0005)	0.7784 (0.0007)	0.7687 (0.0007)
	GAIT-w/o-FC	0.7882 (0.0060)	0.7833 (0.0057)	0.7721 (0.0062)
	GAIT-w-FC	0.7903 (0.0058)	0.7854 (0.0058)	0.7787 (0.0059)
1000	Logistic Regression	0.7627 (0.0077)	0.7432 (0.0063)	0.7317 (0.0059)
	Whole-Training	0.7595 (0.0080)	0.7457 (0.0066)	0.7316 (0.0063)
	Best-Initial	0.7557 (0.0069)	0.7394 (0.0065)	0.6778 (0.0061)
	Aggregate-Initial	0.7849 (0.0007)	0.7775 (0.0007)	0.7661 (0.0008)
	GAIT-w/o-FC	0.7894 (0.0056)	0.7839 (0.0057)	0.7762 (0.0060)
	GAIT-w-FC	0.7910 (0.0058)	0.7853 (0.0059)	0.7784 (0.0062)
1500	Logistic Regression	0.7598 (0.0072)	0.7478 (0.0060)	0.7305 (0.0055)
	Whole-Training	0.7603 (0.0077)	0.7495 (0.0061)	0.7312 (0.0056)
	Best-Initial	0.7527 (0.0064)	0.7398 (0.0065)	0.6791 (0.0054)
	Aggregate-Initial	0.7830 (0.0005)	0.7790 (0.0005)	0.7691 (0.0005)
	GAIT-w/o-FC	0.7886 (0.0055)	0.7818 (0.0056)	0.7723 (0.0056)
	GAIT-w-FC	0.7898 (0.0056)	0.7844 (0.0057)	0.7756 (0.0057)

Note. Average accuracy from ten replications: number in parentheses is the standard error. The left-most column gives the size of the scoring set.

Table 2 Computing Time (in Seconds) for Training and Scoring

Size	Method	Experiment 1	Experiment 2	Experiment 3
500	Logistic Regression	0.94 (0.0391)	0.53 (0.0163)	0.57 (0.0153)
	Whole-Training	2.07 (0.1001)	1.23 (0.0434)	1.35 (0.1025)
	Best-Initial	1.34 (0.0165)	0.54 (0.0130)	0.59 (0.0154)
	Aggregate-Initial	2.17 (0.0042)	1.33 (0.0034)	1.35 (0.0029)
	GAIT-w/o-FC	16.58 (0.0443)	8.03 (0.0222)	8.01 (0.0228)
	GAIT-w-FC	16.70 (0.0443)	8.15 (0.0216)	8.14 (0.0214)
1000	Logistic Regression	0.95 (0.0420)	0.55 (0.0179)	0.59 (0.0135)
	Whole-Training	2.12 (0.1086)	1.29 (0.0299)	1.40 (0.1104)
	Best-Initial	1.39 (0.0161)	0.57 (0.0117)	0.63 (0.0124)
	Aggregate-Initial	2.17 (0.0060)	1.19 (0.0049)	1.44 (0.0040)
	GAIT-w/o-FC	31.04 (0.0495)	14.22 (0.0331)	14.28 (0.0330)
	GAIT-w-FC	31.26 (0.0470)	14.38 (0.0318)	14.40 (0.0313)
1500	Logistic Regression	1.02 (0.0512)	0.54 (0.0132)	0.61 (0.0109)
	Whole-Training	2.14 (0.0120)	1.37 (0.0329)	1.45 (0.1086)
	Best-Initial	1.44 (0.1125)	0.59 (0.0112)	0.68 (0.0116)
	Aggregate-Initial	2.07 (0.0022)	1.28 (0.0033)	1.51 (0.0026)
	GAIT-w/o-FC	45.27 (0.0699)	20.61 (0.0562)	20.59 (0.0563)
	GAIT-w-FC	45.70 (0.0790)	20.77 (0.0547)	20.74 (0.0552)

Note. Average time from ten replications: number in parentheses is the standard error. The left-most column gives the size of the scoring set.

scoring sets is 4,500. Since we can randomly generate training sets and scoring sets repeatedly from these 4,500 points, we replicate each experiment ten times, as reflected in Tables 1 and 2.

From Table 1, we see that the size of the scoring set has a negligible effect on the classification accuracy of any particular method. Indeed, there is no discernable pattern in the accuracies as the size of the scoring set is increased. Since we observe from Table 2 that GAIT computing times appear to increase almost linearly in the size of the scoring set, we recommend using the smallest-size scoring set from among the sizes we considered.

In assessing the effect of conducting a feasibility check after each genetic operation, we see that the classification accuracy and the computing time of GAIT-w-FC are marginally higher than that of GAIT-w/o-FC. On balance, we recommend using GAIT-w-FC after each genetic operation.

Next, we observe that in terms of classification accuracy (on the test set), the Best-Initial tree performs worst in all experiment/scoring-size combinations. It indicates that while Best-Initial may be good for the scoring set, it fails to generalize well enough on the test set. In Experiment 3, the classification accuracy of Best-Initial on the test set is especially bad. This is not surprising. Recall that, in Experiments 1 and 2, the Best-Initial tree is that tree whose classification accuracy, with respect to the scoring set, is highest among all trees generated from randomly selected subsets of the Whole-Training set. In Experiment 3, the Best-Initial tree is generated from a data set that is artificially constructed by aggregating the points of the ten worst-performing trees. Since the data set is not a simple random sample, it is not surprising that the accuracy of the Best-Initial tree on the previously unseen test set is poor.

As a benchmark, we calculated the classification accuracy produced by Logistic Regression. In Experiment 1 and Experiment 2, the performance of Logistic Regression, with respect to accuracy, falls between that of Best-Initial and both GAIT procedures.

In Experiment 3, we observe that the accuracy of the Best-Initial tree on the test set is significantly worse than the accuracy of the other four approaches. Again, this is explained by the fact that the ten initial subsets in Experiment 3 are not a simple random sample from the Whole-Training set (unlike Experiments 1 and 2), but are chosen in order to create a first generation of trees that is of exceptionally low quality. Aggregating these low-quality subsets increases their information content and, consequently, enhances the quality of the resulting Whole-Training tree. As a benchmark, for Experiment 3, we calculated the classification accuracy on the test set produced by Logistic Regression. As compared to the other accuracies reported in Table 1, we see that the classification accuracies on the test set generated by Logistic Regression for all three experiments are roughly the same as those generated by Whole-Training.

Across all experiment/scoring-size combinations, we observe that the classification accuracies of GAIT-w/o-FC and GAIT-w-FC are significantly higher than the classification accuracy of Logistic Regression. For example, in the case where the size of the scoring set is 500, the percentage improvements in accuracy of GAIT-w-FC over that of Logistic Regression are 4.49, 5.96, and 7.73 in Experiments 1, 2, and 3, respectively.

We also see that both GAIT procedures improve upon each of the non-GAIT approaches with respect to accuracy. Let us focus on GAIT-w-FC. Since the advantage of GAIT-w-FC in accuracy over Logistic Regression, Whole-Training, and Best-Initial increases as we move from Experiment 1 to Experiment 2 to Experiment 3, we draw several inferences. First, since the improvement by GAIT-w-FC is stronger when the size of the first generation is smaller, we conclude that it is more advantageous to use GAIT-w-FC when one is restricted in the number of subset trees that can be constructed. Second, since the percent improvement by GAIT-w-FC is larger in Experiment 3 than in Experiment 2, we conclude that it is more advantageous to use GAIT-w-FC when the quality of the first generation of trees is especially suspect. Third, the accuracies of Logistic Regression, Whole-Training, and Best-Initial vary widely across the three experiments. The GAIT-w-FC results are more consistent. From this, we conclude that, regardless of the number of first-generation trees formed and regardless of the quality of the first-generation trees, GAIT-w-FC (in sharp contrast to the non-GAIT approaches) has

the useful property of reliably producing uniformly high-quality trees. We again note that Logistic Regression generated classification accuracies comparable to Whole-Training on the test set for all three experiments.

We notice that aggregating the initial generation of trees (Aggregate-Initial) produces a significant gain in accuracy over Logistic Regression, Whole-Training, and Best-Initial. However, we can improve upon Aggregate-Initial by employing GAIT, with or without the feasibility check.

In order to test whether all these improvements in accuracy are statistically significant, we conducted paired-difference t-tests across each pair of methods. In Table 3, we show the p-values for the various pairwise t-tests for Experiment 1, scoring size = 1,000. For each cell in this table, the alternative hypothesis is that the mean accuracy of the row method is greater than the mean accuracy of the column method. We found that the two GAIT methods are indeed significantly better than the non-GAIT methods. Similar p-value tables were obtained for other settings of the scoring size and for Experiments 2 and 3.

We notice that, in the three experiments, all classifiers, except the Whole-Training tree, are obtained from the training and scoring data sets and are then evaluated on the test set. The Whole-Training tree is obtained from the training set without use of the scoring set. This suggests the following question: If both the training set and the scoring set data are fed to Whole-Training, would the resulting tree outperform the GAIT-w-FC tree? We try to answer this question by referring to Experiment 1. New Whole-Training trees are generated from training sets of 3,500, 4,000, and 4,500 data points. When we compare the new Whole-Training trees and the GAIT-w-FC trees with respect to classification accuracy on the test set, we obtain accuracies of 0.7710 vs. 0.7903 (for size = 500), 0.7750 vs. 0.7910 (for size = 1,000), and 0.7720 vs. 0.7898 (for size = 1,500). From this experiment, we conclude that, even if one generates the decision tree using all the available data points, GAIT-w-FC *still* outperforms Whole-Training.

3.3. Second Data Set

A second data set, called *adult*, was obtained from the UCI Machine Learning Repository (see Blake and Merz 1998). This data set consists of observations on 48,842 adult individuals along 14 demographic variables. The objective is to predict one of two income categories for an individual based on the given demographic variables. The two income categories are low (for individuals earning less than $50,000) and high (for individuals earning greater than $50,000). The proportion of the majority class (low income) in the data set is 75.22%. As in the marketing data set, classification errors for each class are treated equally and the primary performance measure is the overall classification accuracy.

We conducted the same set of experiments on this data set as those described in the previous section on the marketing data set. The training, scoring, and test set sizes are the same as before. For this data set, the overall accuracies are reported in Table 4 and the computing times in Table 5. Analogous to Table 3, Table 6 shows the p-values for pairwise testing of improvements in accuracy. We find that the results on the adult data set are almost identical to those observed on the marketing data set.

Table 3 Marketing Data: p-Values for Experiment 1

	GAIT-w-FC	GAIT-w/o-FC	Aggregate-Initial	Whole-Training	Logistic Regression	Best-Initial
GAIT-w-FC		0.0000	0.0000	0.0000	0.0000	0.0000
GAIT-w/o-FC			0.0000	0.0000	0.0000	0.0000
Aggregate-Initial				0.0000	0.0000	0.0000
Whole-Training						0.0038
Logistic Regression				0.0292		0.0001
Best-Initial						

Note. Scoring size = 1,000: p-value for testing improvement of row method over column method.

Table 4 Classification Accuracy on the Test Set for Adult Data

Size	Method	Experiment 1	Experiment 2	Experiment 3
	Logistic Regression	0.7899 (0.0053)	0.7773 (0.0072)	0.7514 (0.0050)
	Whole-Training	0.7945 (0.0068)	0.7805 (0.0058)	0.7493 (0.0069)
500	Best-Initial	0.7907 (0.0129)	0.7767 (0.0103)	0.7273 (0.0130)
	Aggregate-Initial	0.8225 (0.0025)	0.8019 (0.0032)	0.7873 (0.0025)
	GAIT-w/o-FC	0.8270 (0.0063)	0.8063 (0.0056)	0.7996 (0.0057)
	GAIT-w-FC	0.8349 (0.0074)	0.8245 (0.0060)	0.8128 (0.0060)
	Logistic Regression	0.7928 (0.0063)	0.7789 (0.0030)	0.7514 (0.0072)
	Whole-Training	0.7937 (0.0048)	0.7808 (0.0043)	0.7534 (0.0055)
1000	Best-Initial	0.7910 (0.0134)	0.7754 (0.0095)	0.7296 (0.0110)
	Aggregate-Initial	0.8240 (0.0031)	0.8032 (0.0025)	0.7869 (0.0028)
	GAIT-w/o-FC	0.8244 (0.0054)	0.8052 (0.0051)	0.7971 (0.0035)
	GAIT-w-FC	0.8344 (0.0065)	0.8310 (0.0062)	0.8200 (0.0056)
	Logistic Regression	0.7923 (0.0050)	0.7770 (0.0068)	0.7512 (0.0034)
	Whole-Training	0.8328 (0.0055)	0.7788 (0.0047)	0.7529 (0.0061)
1500	Best-Initial	0.7946 (0.0077)	0.7755 (0.0128)	0.7306 (0.0132)
	Aggregate-Initial	0.8374 (0.0014)	0.8091 (0.0028)	0.7904 (0.0030)
	GAIT-w/o-FC	0.8389 (0.0006)	0.8169 (0.0046)	0.7931 (0.0055)
	GAIT-w-FC	0.8398 (0.0004)	0.8277 (0.0063)	0.8168 (0.0052)

Note. Average accuracy from ten replications: number in parentheses is the standard error. The left-most column gives the size of the scoring set.

Table 5 Computing Time (in Seconds) for Training and Scoring for Adult Data

Size	Method	Experiment 1	Experiment 2	Experiment 3
	Logistic Regression	1.04 (0.0380)	0.63 (0.0146)	0.66 (0.0145)
	Whole-Training	2.27 (0.0700)	1.32 (0.0543)	1.40 (0.0625)
500	Best-Initial	1.30 (0.0156)	0.55 (0.0163)	0.58 (0.0158)
	Aggregate-Initial	2.70 (0.0040)	1.43 (0.0031)	1.39 (0.0036)
	GAIT-w/o-FC	17.82 (0.0478)	8.40 (0.0262)	7.88 (0.0280)
	GAIT-w-FC	17.80 (0.0436)	8.75 (0.0242)	8.64 (0.0240)
	Logistic Regression	0.99 (0.0402)	0.58 (0.0179)	0.61 (0.0150)
	Whole-Training	2.18 (0.0985)	1.30 (0.0299)	1.34 (0.0610)
1000	Best-Initial	1.69 (0.0166)	0.59 (0.0117)	0.63 (0.0140)
	Aggregate-Initial	2.56 (0.0067)	1.19 (0.0049)	1.33 (0.0040)
	GAIT-w/o-FC	33.00 (0.0480)	15.02 (0.0423)	14.80 (0.0283)
	GAIT-w-FC	32.60 (0.0467)	15.38 (0.0368)	14.54 (0.0300)
	Logistic Regression	1.45 (0.0438)	0.64 (0.0120)	0.57 (0.0286)
	Whole-Training	3.01 (0.0130)	1.48 (0.0323)	1.35 (0.0600)
1500	Best-Initial	1.64 (0.1022)	0.66 (0.0120)	0.63 (0.0115)
	Aggregate-Initial	2.73 (0.0025)	1.40 (0.0039)	1.35 (0.0039)
	GAIT-w/o-FC	39.30 (0.0497)	19.61 (0.0525)	17.36 (0.0363)
	GAIT-w-FC	41.36 (0.0560)	21.59 (0.0570)	20.40 (0.0520)

Note. Average time from ten replications: number in parentheses is the standard error. The left-most column gives the size of the scoring set.

Table 6 Adult Data: *p*-Values Experiment 1

	GAIT-w-FC	GAIT-w/o-FC	Aggregate-Initial	Whole-Training	Logistic Regression	Best-Initial
GAIT-w-FC		0.0000	0.0000	0.0000	0.0000	0.0000
GAIT-w/o-FC			0.3893	0.0000	0.0000	0.0000
Aggregate-Initial				0.0000	0.0000	0.0000
Whole-Training						0.2231
Logistic Regression				0.2624		0.2763
Best-Initial						

Note. Scoring size = 1,000: *p*-value for testing improvement of row method over column method.

4. Scalability Experiments with GAIT

In this section, we increase the size of the training set and the size of the scoring set for each of five algorithms in order to investigate the scalability of GAIT. As before, all accuracy measures are obtained from the test set, which is not accessible in the development of any of the classifiers.

4.1. Experimental Design

In the scaling experiments, we use the marketing data set of approximately 440,000 data points. From the original data set of 440,000 points, we randomly select 4,000 points and set them aside to form a test set. We then generate a series of training/scoring-size combinations to test the scalability of GAIT. Specifically, we split the original data set into a training set of 310,000 points and a scoring set of 124,000 (this is approximately 99% of the data). This training/scoring-size combination (310,000/124,000) is the largest combination used in our experiments to test GAIT's scalability. In Table 7, we show the training/scoring- size combinations in this as well as the other experiments, and the corresponding percentages of the original data set. For each of the training/scoring-size combinations (x/y) shown in Table 7, we select a simple random sample (without replacement) of x points from the original data (excluding test data) to serve as a training set. An additional y points are selected from the remaining data points to serve as the scoring set.

Table 7 The Training/Scoring Set Combinations and Corresponding Percentages of the Data in Scaling Experiments

	Size	
Percent (%)	Training Set	Scoring Set
99	310,000	124,000
72	240,000	96,000
51	160,000	64,000
25	80,000	32,000
3	10,000	4,000
1	3,000	1,500

For each scaling experiment on each training/scoring-size combination, we record classification accuracy on the test set and computing time on the training and scoring sets. We also include results from Logistic Regression for benchmarking purposes. We define Best-Initial as the best tree (as measured by its classification accuracy on the scoring set) in the first generation of each experiment. We also obtain performance measures for the best trees generated by GAIT-w-FC and GAIT-w/o-FC. With respect to Whole-Training, we again use the training set *and* scoring set data as input and compute the accuracy on the test set. Each experiment is replicated ten times and the mean and standard error of each performance measure are computed on the test set of 4,000 points.

4.2. Computational Results and Analysis

The average results for classification accuracy and computing time based on ten replications are given in Table 8 and Table 9, respectively.

In Table 8, we see that the average accuracy of each algorithm always increases as the size of the training set and the size of the scoring set are increased. We see that, in terms of classification accuracy, the Best-Initial tree performs worst on all training/scoring-size combinations. It appears that Best-Initial can build a good tree, but the generated tree fails to generalize well on the test set. In Table 8, we also notice that,

Table 8 Classification Accuracy on the Test Set for Five Training/Scoring-Size Combinations

	Training	Scoring %	Training	Scoring %	Training	Scoring %	Training	Scoring %	Training	Scoring %
	10,000	4,000 3	80,000	32,000 25	160,000	64,000 51	240,000	96,000 72	310,000	124,000 99
Method	Accuracy	Standard Error	Accuracy	Standard Error	Accuracy	Standard Error	Accuracy	Standard Error	Accuracy	Standard Error
Logistic Regression	0.7610	0.0009	0.7709	0.0010	0.7790	0.0009	0.7811	0.0008	0.7825	0.0009
Whole-Training	0.7642	0.0013	0.7765	0.0006	0.7832	0.0008	0.7843	0.0009	0.7856	0.0008
Best-Initial	0.7555	0.0015	0.7632	0.0016	0.7708	0.0011	0.7755	0.0013	0.7766	0.0013
GAIT-w/o-FC	0.7953	0.0012	0.8070	0.0012	0.8085	0.0010	0.8096	0.0010	0.8102	0.0010
GAIT-w-FC	0.7987	0.0011	0.8104	0.0009	0.8124	0.0010	0.8133	0.0011	0.8139	0.0009

Note. Average accuracy for ten replications.

on all training/scoring-size combinations, the classification accuracy of Logistic Regression falls between that of Best-Initial and Whole-Training, while GAIT-w/o-FC and GAIT-w-FC have the best average classification accuracies. For each of the five algorithms, we conducted paired-difference t-tests (with respect to accuracy) at different percentages of the data. The p-values of the paired-difference t-tests indicate that for any specific percentage of the data, GAIT-w-FC outperforms GAIT-w/o-FC, GAIT-w/o-FC outperforms Whole-Training, and Logistic Regression outperforms Best-Initial. All of these improvements are statistically significant at the 1% level. Whole-Training outperforms Logistic Regression at the 5% level of significance.

In Table 9, we see that the computing time increases when the data size increases for all five algorithms. Except for the smallest combination, GAIT-w-FC, GAIT-w/o-FC, and Whole-Training take more computing time than Logistic Regression and Best-Initial.

In Figure 9, we plot the classification accuracy of each algorithm against the size of the training/scoring-size combination as a percentage of the size of the data. In Figure 9, we see that GAIT-w-FC and GAIT-w/o-FC outperform Whole-Training, Logistic Regression, and Best-Initial at each of the six percentages. For GAIT-w-FC and GAIT-w/o-FC, the classification accuracy jumps dramatically when the training/scoring size increases from 1% to 3% to 25%, and then levels off. The difference in classification accuracy between GAIT-w-FC and GAIT-w/o-FC is small at 1%, and becomes larger when the size increases from 1% to 3% to 25%, and then levels off. In Figure 9, we see that when the size of the training/scoring set is less than a certain percentage of the data, e.g., 25%, an increase of the data size can

Table 9 Computing Time (in Minutes) for Five Training/Scoring-Size Combinations

	Training	Scoring %	Training	Scoring %	Training	Scoring %	Training	Scoring %	Training	Scoring %
	10,000	4,000 3	80,000	32,000 25	160,000	64,000 51	240,000	96,000 72	310,000	124,000 99
Method	Time	Standard Error	Time	Standard Error	Time	Standard Error	Time	Standard Error	Time	Standard Error
Logistic Regression	0.52	0.0108	3.55	0.0507	6.03	0.0617	14.79	0.0845	25.89	0.0588
Whole-Training	0.42	0.0104	23.62	0.1137	71.43	0.2879	137.29	0.6485	358.15	0.6184
Best-Initial	1.81	0.0940	16.76	0.2441	28.48	0.2251	45.77	0.2673	56.46	0.2739
GAIT-w/o-FC	10.38	0.1238	71.07	0.3855	162.58	0.7074	261.00	1.0609	348.72	1.1860
GAIT-w-FC	10.82	0.1026	74.53	0.2695	170.83	0.5135	275.92	1.0873	371.38	1.1035

Note. Average time for ten replications.

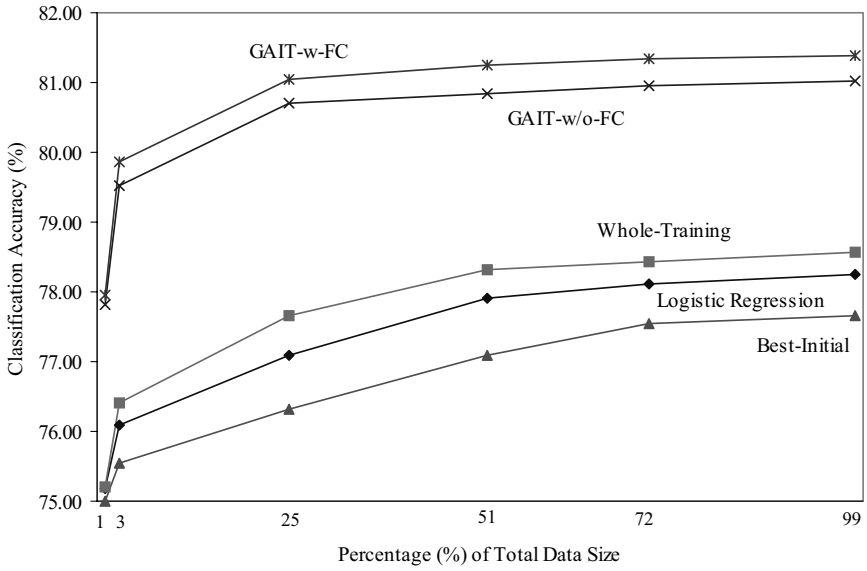

Figure 9 Classification Accuracy Against Training/Scoring Set Size
Note. Results are on the test set.

improve the classification accuracy significantly; however, once the size is large enough, a further increase in size does not significantly improve the accuracy.

In Figure 9, we also see that each of the accuracies of Whole-Training, Logistic Regression, and Best-Initial are lower than those of GAIT-w-FC and GAIT-w/o-FC at any sampling percentage. The accuracy of each method increases dramatically when the data size increases from 1% to 3%, and then the accuracy improves steadily from 3% to 25% to 51% (to 72% for Best-Initial).

Next, we conduct independent sample t-tests on each of the algorithms. The p-values of independent sample t-tests are given in Table 10. In Table 10, we see that the accuracy improvements for Logistic Regression, Whole-Training, and Best-Initial at 3% over 1%, 25% over 3%, and 51% over 25% of the data are statistically significant. The improvements for GAIT-w-FC and GAIT-w/o-FC are statistically significant at 3% over 1% and 25% over 3% of the data. When the size of the training set and scoring set taken together is relatively small, say 3% of the total data size, an increase in the size of the training/scoring set can significantly increase the classification accuracy on the test set; however, once the size of the training/scoring set is large enough, say 50%, a further increase in size cannot significantly increase the accuracy.

In Figure 10, we show the computing times for the five methods. For the most part, we see that Logistic Regression and Best-Initial take the least amount of computing time in training and scoring. GAIT-w-FC and GAIT-w/o-FC take roughly the same computing time; however, as the size of the training/scoring set

Table 10 The p-Values for Independent Sample t-Tests of Accuracy Improvement for Different Percentages of the Total Data Size

Method	3% over 1%	25% over 3%	51% over 25%	72% over 51%	99% over 72%
Logistic Regression	**0.0000**	**0.0000**	**0.0000**	0.0529	0.1360
Whole-Training	**0.0000**	**0.0000**	**0.0000**	0.1894	0.1491
Best-Initial	**0.0001**	**0.0011**	**0.0005**	**0.0062**	0.2755
GAIT-w/o-FC	**0.0000**	**0.0000**	0.1809	0.2214	0.3329
GAIT-w-FC	**0.0000**	**0.0000**	0.0819	0.2750	0.3386

Note. Statistically significant improvements (at the 1% level) are indicated in boldface type.

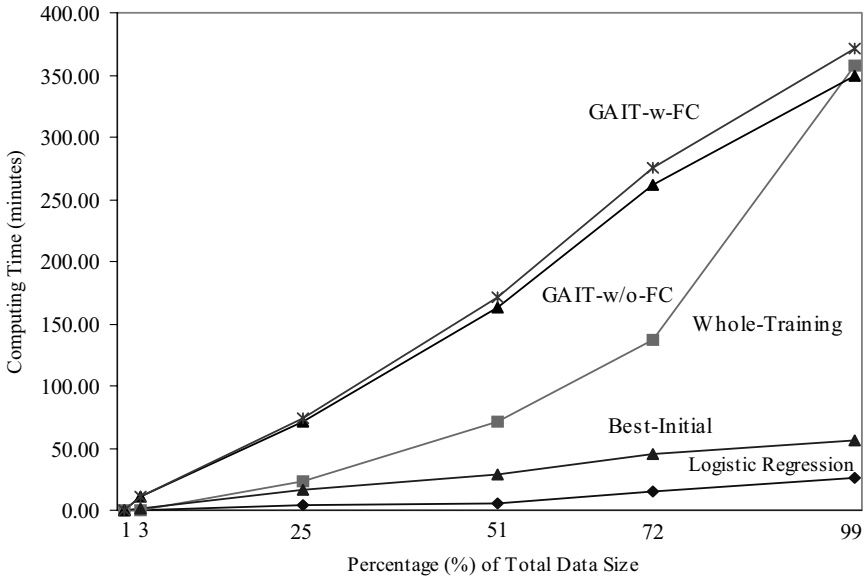

Figure 10 Computing Time (in Minutes) for Training and Scoring on Different Percentages of the Data

increases, GAIT-w-FC tends to take slightly more time than GAIT-w/o-FC. The computing time of Whole-Training is in the same range as the computing times of Best-Initial and Logistic Regression when the size of the training/scoring set is small (1%, 3%, 25%, and even 51%), but its computing time increases greatly thereafter. For example, when 99% of the total data set is used, Whole-Training takes more time than even GAIT-w/o-FC.

We make several observations from these experiments. First, GAIT-w-FC and GAIT-w/o-FC generate much more accurate trees than Logistic Regression, Best-Initial, and Whole-Training, and their computing times increase linearly as the size of the training/scoring set increases. More importantly, although GAIT-w-FC and GAIT-w/o-FC require more computing time than the other three methods, for the most part, both methods generate much more accurate decision trees using only a small percentage of the data. It is important to point out that GAIT-w-FC and GAIT-w/o-FC (using only 3% of the data) outperform Logistic Regression, Best-Initial, and Whole-Training (using 99% of the data) *and* both methods take less computing time. In summary, GAIT-w-FC and GAIT-w/o-FC are quick to find much better solutions than those found by Logistic Regression, Best-Initial, and Whole-Training.

Based on the computing time results in Figure 10 (also in Tables 2 and 5), we see that GAIT-w-FC and GAIT-w/o-FC take more computing time than the other methods when the size of the training/scoring set is small. We attribute this to the fact that the evolution process in GAIT takes a substantial proportion of its total computing time. As the size of the training/scoring set increases, the amount of time to construct a decision tree (by Whole-Training) increases significantly (see Figure 10). In contrast, GAIT handles a larger data set by splitting it into many small-size subsets (the diversity found in the overall data set is inherited by the smaller subsets). Smaller and simpler trees are then constructed on these subsets and the genetic operations of GAIT are more efficient and less time consuming using these trees. Thus, GAIT achieves its efficiency by using smaller size trees in its evolution process.

5. Conclusions

In building decision trees for very large data sets, it is quite natural to estimate the parameters of the tree from a sample of data points. However, the quality of the resulting decision tree is then subject to the errors that accompany all sampling-based procedures.

In order to avoid poor-quality decision trees, we propose a method that improves upon the sampled trees and uniformly raises their quality. Our method uses a genetic algorithm to monotonically increase the classification accuracy (with respect to the scoring set) at each generation of decision trees. All resulting trees are evaluated using the same test set. Our computational experiments on two data sets demonstrate the robustness of the resulting decision tree with respect to the quality as well as the quantity of the initial samples. In addition, we demonstrate that the genetic algorithm outperforms the common-sense approach of building a single decision tree using all the available data. With respect to scalability, we have demonstrated that GAIT-w-FC and GAIT-w/o-FC can be used effectively on very large data sets. The key to the success of our approach seems to be the combined use of sampling, genetic algorithms, and C4.5, which is a very fast decision-tree package.

Our work can be extended in several ways. First, we would like to extend the fitness function that is used in our genetic algorithm to evaluate a given decision tree. In the current study, we use the overall classification accuracy as the sole criterion for assessing the quality of a decision tree. The fitness function of a tree can be modified to account for the consistency of classification accuracy, as well as the simplicity of the classification rules (e.g., as measured by the depth of the decision tree), in addition to the overall classification accuracy. We hope to explore this extension in future work. Second, we expect to apply our genetic algorithm-based methods to a variety of new data sets. Third, we envision aggregating the final generation of trees to improve classification accuracy further; preliminary results indicate that some marginal improvement may be obtained. Fourth, we might develop and apply more sophisticated crossover and mutation operations, which would replace those mentioned in Section 2.

As noted in Section 1, an adaptive resampling method like GLOWER could also be used. The advantage of such a rule-based approach is that, when properly implemented, it can be extremely thorough in searching the space of rules, especially if the data suffer from weak structure and nonlinearities. The disadvantage is that this thoroughness is computationally intensive and, consequently, requires extensive running times. In our approach, we forsake the thoroughness of the search for rules by starting with quickly generated decision trees (from C4.5). We show that by evolving such complete trees, we can significantly improve their accuracy. It remains for future work to compare comprehensively the tree-based search that we advocate here with the rule-based (or query-based) adaptive resampling approaches.

References

Banzhaf, W., P. Nordin, R. Keller, F. Francone. 1998. *Genetic Programming: An Introduction*. Morgan Kaufmann, San Francisco, CA.

Bauer, E., R. Kohavi. 1999. An empirical comparison of voting classification algorithms: bagging, boosting, and variants. *Machine Learning* **36** 105–139.

Blake, C., C. Merz. 1998. UCI repository of machine learning databases. Department of Information and Computer Science, University of California, Irvine, CA. http://www.ics.uci.edu/~mlearn/MLRepository.html

Bojarczuk, C., H. Lopes, A. Freitas. 2000. Genetic programming for knowledge discovery in chest pain diagnosis. *IEEE Engineering in Medicine and Biology* **19** 38–44.

Bot, M., W. Langdon. 2000. Application of genetic programming to induction of linear classification trees. R. Poli, W. Banzhaf, W. Langdon, J. Miller, P. Nordin, T. Fogarty, eds. *Proceedings of the Third European Conference on Genetic Programming*. Springer-Verlag, Edinburgh, Scotland, U.K., 247–258.

Breiman, L. 1996. Bagging predictors. *Machine Learning* **24** 123–140.

Breiman, L. 1998. Arcing classifiers. *Annals of Statistics* **26** 801–849.

Breiman, L., J. H. Friedman, R. A. Olshen, C. J. Stone. 1984. *Classification and Regression Trees*. Wadsworth and Brooks/Cole, Monterey, CA.

Cormen, T., C. Leiserson, R. Rivest. 1990. *Introduction to Algorithms*. MIT Press, Cambridge, MA.

Deo, N. 1974. *Graph Theory with Applications to Engineering and Computer Science*. Prentice-Hall, Englewood Cliffs, NJ.

Dhar, V., D. Chou, 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.

Fayyad, U., G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy, eds. 1996. *Advances in Knowledge Discovery and Data Mining*. MIT Press, Cambridge, MA.

Folino, G., C. Pizzuti, G. Spezzano. 2000. Genetic programming and simulated annealing: a hybrid method to evolve decision trees. R. Poli, W. Banzhaf, W. Langdon, J. Miller, P. Nordin, T. Fogarty, eds. *Proceedings of the Third European Conference on Genetic Programming*. Springer-Verlag, Edinburgh, Scotland, U.K., 294–303.

Freitas, A. 2002a. Evolutionary computation. W. Klösgen, J. Zytkow, eds. *Handbook of Data Mining and Knowledge Discovery*. Oxford University Press, New York, 698–706.

Freitas, A. 2002b. A survey of evolutionary algorithms for data mining and knowledge discovery. A. Ghosh, S. Tsutsui, eds. Forthcoming in *Advances in Evolutionary Computing*. Springer-Verlag, Heidelberg, Germany.

Freund, Y., R. Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences* **55** 119–139.

Fu, Z. 2000. Using genetic algorithms to develop intelligent decision trees. Ph.D. dissertation, University of Maryland, College Park, MD.

Fu, Z., B. Golden, S. Lele, S. Raghavan, E. Wasil. 2000. Building a high-quality decision tree with a genetic algorithm: a computational study. M. Laguna, J. L González-Velarde, eds. *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*. Kluwer Academic Publishers, Boston, MA, 25–38.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

Kennedy, H., C. Chinniah, P. Bradbeer, L. Morss. 1997. The construction and evaluation of decision trees: a comparison of evolutionary and concept learning methods. D. Corne, J. Shapiro, eds. *Evolutionary Computing, Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 147–161.

Koza, J. 1991. Concept formation and decision tree induction using the genetic programming paradigm. H-P. Schwefel, R. Maenner, eds. *Parallel Problem Solving from Nature*. Springer-Verlag, Berlin, Germany, 124–128.

Koza, J. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA.

Marmelstein, R., G. Lamont. 1998. Pattern classification using a hybrid genetic program-decision tree approach. J. Koza, ed. *Proceedings of the Third Annual Conference on Genetic Programming*. Morgan Kaufmann, San Francisco, CA.

Menard, S. 1995. *Applied Logistic Regression Analysis*. Sage, Thousand Oaks, CA.

Michalewicz, Z. 1996. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, New York.

Nikolaev, N., V. Slavov. 1998. Inductive genetic programming with decision trees. *Intelligent Data Analysis* **2**. http://www-east.elsevier.com/ida

Quinlan, J. 1986. Induction of decision trees. *Machine Learning* **1** 81–106.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.

Ripley, B. D. 1996. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, UK.

Ryan, M., V. Rayward-Smith. 1998. The evolution of decision trees. J. Koza, ed. *Proceedings of the Third Annual Conference on Genetic Programming*. Morgan Kaufmann, San Francisco, CA, 350–358.

Accepted by Amit Basu; received June 2000; revised July 2001, April 2002, May 2002; accepted October 2002.