

Inducing Oblique Decision Trees With Evolutionary Algorithms

Erick Cantú-Paz, *Member, IEEE*, and Chandrika Kamath, *Senior Member, IEEE*

Abstract—This paper illustrates the application of evolutionary algorithms (EAs) to the problem of oblique decision-tree (DT) induction. The objectives are to demonstrate that EAs can find classifiers whose accuracy is competitive with other oblique tree construction methods, and that, at least in some cases, this can be accomplished in a shorter time. We performed experiments with a $(1 + 1)$ evolution strategy and a simple genetic algorithm on public domain and artificial data sets, and compared the results with three other oblique and one axis-parallel DT algorithms. The empirical results suggest that the EAs quickly find competitive classifiers, and that EAs scale up better than traditional methods to the dimensionality of the domain and the number of instances used in training. In addition, we show that the classification accuracy improves when the trees obtained with the EAs are combined in ensembles, and that sometimes it is possible to build the ensemble of evolutionary trees in less time than a single traditional oblique tree.

Index Terms—Classification, decision trees, ensembles, machine learning, sampling.

I. INTRODUCTION

DECISION trees (DTs) are popular classification methods, and there are numerous algorithms to induce a tree classifier from a data set [32]. Most of the tree inducing algorithms create tests at each node that involve a single attribute of the data. These tests are equivalent to hyperplanes that are parallel to one of the axes in the attribute space; therefore, the resulting trees are called axis-parallel. These simple univariate tests are convenient because a domain expert can interpret them easily, but they may result in complicated and inaccurate trees if the data is more suitably partitioned by hyperplanes that are not axis-parallel. Oblique decision trees use multivariate tests that are not necessarily parallel to an axis, and, in some domains, may result in much smaller and more accurate trees. However, oblique trees are not as popular as the axis-parallel trees because the tests are more difficult to interpret, and the oblique inducers require greater computational resources than the axis-parallel algorithms.

Evolutionary algorithms (EAs) are stochastic search methods based on the mechanics of natural selection and genetics. The purpose of this paper is to illustrate the application of EAs to the task of oblique DT induction. The objectives are to show that evolutionary optimization may result in classifiers whose accu-

racy is competitive with other oblique tree construction methods and that, at least in some cases, this can be accomplished in shorter time. The results of our experimental study suggest that the EA-augmented inducers can find competitive classifiers, and that they scale up better than traditional oblique DT inducers to the size of the training sets and to the number of attributes that describe each instance. In addition, we present experiments that demonstrate how combining multiple oblique trees in ensembles improves the classification accuracy, and that sometimes the ensembles can be built faster than a single oblique tree with existing algorithms.

This paper extends our previous work [12] in several ways. We use additional data sets and perform additional experiments to evaluate the scalability of the algorithms (both on the size and dimensionality of the data); we extend significantly our review of previous work; we compare the results against an additional oblique DT algorithm based on simulated annealing that is our variation of an existing algorithm (our version seems to perform better than the original); and we introduce new ways to create ensembles by randomizing the instances used at each node and by using the EAs to find the splits.

The paper is organized as follows. The next section provides a brief background on oblique decision trees and a review of relevant previous work. Section III describes some of the advantages of using EAs to find splits in oblique DTs and our approach to this problem. Section IV describes the algorithms and experimental methods in detail. Section V presents experimental results that illustrate the advantages of the evolutionary approach using public domain and artificial data sets. Finally, Section VI provides a brief summary and the conclusions of the paper.

II. OBLIQUE DECISION TREES

The task of any DT inducer is to use the information contained in a training set of labeled instances to create a model that predicts the class of unseen instances. In this paper, we consider that the instances take the form $(x_1, x_2, \dots, x_d, c_j)$, where the x_i are real-valued attributes, d is the number of attributes, and c_j is a discrete value that represents the class label of the instance. As mentioned in the introduction, most tree inducers consider tests of the form $x_i > k$ that are equivalent to axis-parallel hyperplanes in the attribute space. In this case, the task of the inducer is to find the best values for i and k . Most tree-building algorithms evaluate the candidate hyperplanes using the impurity of the split defined by the hyperplane. Many impurity measures have been proposed, such as the information gain [37], the Gini index, or the twoing rule [8]. Each impurity measure has its own

Manuscript received January 31, 2002; revised May 28, 2002. This work performed under the auspices of the U.S. Department of Energy by the University of California Lawrence Livermore National Laboratory under Contract W-7405-Eng-48. (UCRL-JC143718.)

The authors are with the Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94551 USA (e-mail: cantupaz; kamath2@llnl.gov).

Digital Object Identifier 10.1109/TEVC.2002.806857

advantages and disadvantages [32], and each defines a different optimization problem.

In this paper, we consider more general tests of the form

$$\sum_{i=1}^d a_i x_i + a_{d+1} > 0 \quad (1)$$

where the a_i are real-valued coefficients. In this case, the task of the inducer is much harder than before, because it involves searching in a $(d + 1)$ -dimensional space. Finding the best oblique tree is an NP-complete problem [23], and therefore existing oblique DT inducers use a heuristic to find appropriate values for the coefficients of the hyperplane.

Several early attempts to build oblique decision trees used some form of feature construction. The first work in this area appears to be that of Henrichon and Fu [24]. They proposed an axis-parallel algorithm that may be complemented at the user's discretion by a feature construction algorithm that, at each tree node, adds linear combinations of the existing features. Henrichon and Fu proposed several methods to construct the new features, such as using a linear discriminant function or projecting the instances onto the direction of the eigenvector associated with the largest eigenvalue of the covariance matrix (i.e., the first principal component). Friedman introduced a similar feature construction algorithm [17]. One of the methods he proposed was to add a feature by projecting the examples under consideration on the hyperplane given by Fisher's linear discriminant. One difference with the previous system is that the new features in Friedman's system become available for the remainder of the induction process. Gama independently implemented an almost identical procedure to Friedman's [18]. Iyengar introduced an iterative procedure for feature construction that attempts to identify promising oblique directions on which to partition the data [26]. On each iteration, an entire (axis-parallel) tree is built and the training set is extended with new oblique features, which are based on the vectors that connect the centroids of promising pairs of leaf nodes. In all of the systems described above, the axis-parallel tree inducer remains unchanged, but new oblique features are added.

Breiman *et al.* [8] introduced CART with linear combinations (CART-LC) as an option in their popular DT algorithm CART. At each node of the tree, CART-LC iteratively finds locally optimal values for each of the a_i coefficients. Hyperplanes are generated and tested until the marginal benefits become smaller than a specified constant. Recognizing that the oblique splits are harder to interpret than a simpler univariate split, Breiman *et al.* used a backward-deletion procedure to simplify the structure of the split by weeding out variables that contribute little to the effectiveness of the split. The iterative simplification procedure deletes one variable at a time until no further variables can be deleted, and the coefficient optimization algorithm is executed anew on the remaining variables.

Loh and Vanichsetakul used a modified linear discriminant function on the principal components of the data [29]. Their method avoids singular covariance matrices in the subsets produced after the splits (the covariance matrix is necessary to determine the discriminant function). In addition, they proposed to use polar coordinate splits if spherical symmetry was detected on the data.

Finding the splitting hyperplane is an optimization problem, and researchers can use their favorite optimization algorithm. For example, Heath *et al.* [23] used simulated annealing (SA) to find the hyperplane at each node. They showed that SA can produce small and highly accurate trees in some domains. However, Heath *et al.* perturbed only one coefficient at a time and used SA for a large number of iterations (their stopping criterion varied between 3000 and 30 000 iterations without improvement), which may be inadequate for large data sets. We show in this paper that it is not necessary to run SA for a long time to find acceptable solutions. Another example of using a traditional optimization algorithm to induce oblique trees is the work of Brown *et al.* [10], who used linear programming repeatedly at each node of the tree. Their approach is to find the hyperplane that best separates one of the classes from all the others (i.e., solve a two-class problem), compare these solutions with each other and with the univariate splits, and select the best split to be used in the tree node. Brown *et al.* showed that this method produces more accurate and smaller trees than CART-LC on parity problems.

Murthy *et al.* [33] introduced OC1, which uses an *ad-hoc* combination of hillclimbing and randomization. As in CART-LC, the hillclimber finds locally optimal values for one coefficient at a time, although OC1 offers several options to choose the order in which the coefficients are optimized. The randomization component takes two forms: OC1 uses multiple random restarts, and when hillclimbing reaches a local minimum, the hyperplane is perturbed in a random direction. Murthy *et al.* present OC1 as an alternative algorithm that overcomes some of the limitations of CART-LC. In particular, they claim that the deterministic nature of CART-LC may cause it to get trapped in local minima, and that using randomization may improve the quality of the DTs. In addition, OC1 produces multiple trees using the same data, and unlike CART-LC, the time used at each node in the tree is bounded. Murthy *et al.* present experimental results that demonstrate that OC1 outperforms CART-LC in several domains. Their algorithm has been used successfully in several applications [38], [39].

Other related work in this area includes the linear machine decision trees (LMDT) system [41], [9]. The LMDT algorithm is very different from the other systems. Instead of using a test similar to Equation (1) at each node, the LMDT has a set of R linear discriminant functions $g_i(\mathbf{X}) = \mathbf{W}_i^T \mathbf{X}$ and assigns class $i \in \{1, \dots, R\}$ to the instance described by $\mathbf{X} = (x_1, \dots, x_d)$ if $\forall i, i \neq j, g_i(\mathbf{X}) > g_j(\mathbf{X})$. The training algorithm changes the weight vectors \mathbf{W}_i according to a specific correction rule, and the tree is built recursively until all the instances in a node belong to the same class. Although, EAs could also be used in combination with LMDT to adapt the weight vectors, we do not address LMDT in this paper.

Some authors have proposed to build oblique DTs optimizing criteria other than the impurity of the splits. For example, Shah and Sastry [40] choose the hyperplane that maximizes the separability of the subsets created by the split, and Bennett *et al.* [3] maximize the margin of splitting hyperplane with respect to the training data in a manner analogous to support vector machines.

EAs have been used to induce decision trees, although most efforts have been directed toward axis-parallel DTs. Koza [27]

used a slightly modified version of his basic genetic programming (GP) system [28] to evolve DTs. GP does not use the normal top-down recursive partitioning method of other algorithms. Instead, GP works with a set of trees and performs operations inspired by sexual recombination and gene mutations to alter the structure of the trees. GP uses the accuracy of each tree on the training set to evaluate their fitness, and those trees with higher fitness are selected for future processing.

Folino *et al.* [15] also used GP to induce axis-parallel DTs, but used a spatially distributed population model that can be easily parallelized. They later extended their algorithm with a selection method that sometimes accepts less fit individuals in a manner analogous to simulated annealing [16]. Nicolaev and Slavov proposed a different fitness evaluation procedure that attempts to reduce the difficulty of the search and guarantee the parsimony of the solutions [34].

Closer to our interests, Bot and Langdon used genetic programming to induce oblique DTs [6]. Their experiments with four small public-domain data sets showed that sometimes GP can find oblique trees that are as accurate as those found by OC1 and C5.0, a popular commercial axis-parallel DT package. When care was taken to restrict the size of the trees (a common problem in GP), the resulting trees were five to ten times smaller than those of C5.0 and OC1.

GP appears well suited to the DT induction task because it operates directly on entire trees, but the major disadvantage of GP is its long execution time. Although it is possible to improve GP's performance [5], the approach that we follow in this paper is to use fixed-length EAs to optimize the split at each node.

III. EVOLUTIONARY OBLIQUE DECISION TREES

At the heart of all traditional DT inducing algorithms there is an optimization task. In traditional "top-down" induction of decision trees [37], at each node of the tree the algorithm attempts to find the best way to divide the data to minimize some measure of impurity. Then the data are partitioned into subsets, and the algorithm is applied recursively to each subset. As we saw in the previous section, several algorithms have been used to solve this optimization problem for both axis-parallel and oblique trees. We propose to use EAs to optimize the splitting criteria. We believe that EAs are a promising technique to build oblique decision trees for several reasons.

More sophisticated optimizers. CART-LC and OC1 use greedy optimizers that try to optimize the impurity measure at each node by modifying one coefficient at a time and leaving the other coefficients unaltered. This unidimensional greedy search may get trapped in local optima. Recognizing this problem, Murthy *et al.* included an *ad-hoc* randomization procedure in OC1 to try to avoid this problem. In contrast, EAs can consider more than one coefficient at a time, and we conjecture that EAs may not get trapped in local optima as easily as the simple greedy hillclimbers that are currently in use.

Acceptable solutions are found quickly. Finding the best split at each node does not guarantee that the best tree will be found. After all, the algorithms attempt to optimize a heuristic measure of the impurity of the split, and in each

tree node beyond the root, the algorithm considers only a subset of all the data available. Therefore, there is no need to run the EAs (or any other optimizer, for that matter) until they find the most accurate discrimination possible. It is well known that EAs often improve quickly on the initial solutions, and so we may use the best hyperplanes found after just a few iterations.

Scalability to high-dimensional spaces. The dimension of the search space is defined by the number of attributes that describe each instance. In practice this can be a large number, and the execution time of some existing DT algorithms may not scale up well. In contrast, EAs have been shown to have good scalability to the dimensionality of the problem [31], [22].

Use of problem-specific knowledge. There are numerous opportunities to incorporate knowledge about the DT-inducing problem into the EAs. For instance, real-valued encodings and operators seem natural to represent and manipulate hyperplanes. In addition, it is well known that seeding the initial population of the EA with known "good" solutions can enhance the quality of the search and shorten the execution time. For example, we could initialize the EAs with axis-parallel hyperplanes or oblique solutions based on linear discriminant functions.

Hybridization. Hybridizing an EA with a local optimizer is a well-known method to boost performance [36], [25], [19]. Most DT algorithms use a local optimizer that is well tuned to the tree induction task, and combining this or other local optimizer with the EA may improve performance significantly.

Tolerance to noise. More efficient EA-based DT inducers may be obtained by approximating the fitness of a hyperplane by using a small random sample of instances to evaluate the split. This approximation would assign different fitness values to the same hyperplane every time that it is evaluated, but EAs are tolerant to such noisy fitness evaluations [20], [30].

Parallel implementations. When the evaluation of fitness is expected to take a long time (as is the case with large data sets), parallel implementations of EAs may reduce considerably the overall execution time of the algorithm [11]. In addition, it is straightforward to implement EAs on parallel computers, in contrast with the difficulty of parallelizing existing oblique tree algorithms.

Amenability to create ensembles. Since EAs are stochastic algorithms, they likely produce a different tree every time that they are run on the same data set. These trees can be easily combined into ensembles where the classification of an example is determined by the (possibly weighted) vote of all the trees. Traditional methods of creating ensembles, such as bagging or boosting, usually result in a lower error rate than single classifiers [2], [35], [14]. Our experiments show that combining multiple oblique trees obtained with EAs also improves the classification accuracy.

This paper does not consider all of the possibilities mentioned above. In particular, we do not use hybrids or parallel implementations, but we use knowledge about the problem in our choice

To find a split of a set of examples S with dimension d :
 Find H_0 , the best axis-parallel split of S . $I = \text{impurity}(H_0)$
 If $(|S| < 2d)$ then exit and output H_0
 Repeat R times:
 Choose a random hyperplane H
 (For the first iteration, initialize H to the best axis-parallel split)
 Step 1: Until the impurity measure does not improve, do:
 Perturb each of the coefficients of H in sequence
 Step 2: Repeat at most J times:
 Choose a random direction and perturb H in that direction
 If this reduces the impurity of H , go to Step 1.
 $I_1 = \text{impurity}(H)$
 If $(I_1 < I)$ then $I = I_1$
 Output the split corresponding to I

Fig. 1. Overview of the OC1 algorithm for a single node of a decision tree. Adapted from [33].

of floating-point encoding and to seed the initial population. The experiments described below confirm that the EAs reach acceptable solutions when run for a fixed number of iterations that, in many cases, are not enough for the EA to converge to a unique solution or to find the best possible hyperplane that it could. The experiments also demonstrate that EAs scale up better than traditional oblique DT algorithms to the dimensionality of the problem. In addition, we performed experiments to explore the sensitivity of EAs to random sampling, and to demonstrate the construction of ensembles with better accuracy than single trees.

IV. ALGORITHMS AND EXPERIMENTAL METHODS

We extended the publicly- available source code of OC1 to find oblique partitions using two different EAs and our version of simulated annealing. The experiments compare the performance of six algorithms.

- 1) OC1 limited to axis-parallel tests, which we call OC1-AP.¹
- 2) OC1 with its default parameters.
- 3) CART-LC as implemented by Murthy *et al.* in the OC1 system [33].
- 4) OC1-ES, our extension to OC1 using evolution strategies.
- 5) OC1-GA, our extension to OC1 using genetic algorithms.
- 6) OC1-SA, our extension to OC1 using simulated annealing.

Fig. 1 outlines the original OC1 algorithm to find a split at every node of a decision tree. To be at least as powerful as standard DT methods, OC1 first finds the best axis-parallel split that minimizes a user-selected impurity measure [33]. The original OC1 and our modifications use oblique splits only when they can find an oblique split with lower impurity than the best axis-parallel. In addition, the oblique algorithms will not seek to improve the axis-parallel splits at tree nodes where the number of examples available is less than twice the dimensionality of the problem (d). OC1 has an option to start the search from a random oblique hyperplane, but Murthy *et al.*'s and our own

¹We decided to use OC1-AP in the comparisons instead of other popular axis-parallel DT inducers such as C4.5, because the axis-parallel splits found by OC1-AP are used as the initial splits in all the oblique algorithms considered in this paper. We felt that doing this would better illustrate the possible advantages of oblique splits. We could, of course, initialize the oblique algorithms with axis-parallel splits produced by C4.5 or any other AP algorithm.

To find a split of a set of examples S with dimension d :
 Find H_0 , the best axis-parallel split of S . $I = \text{impurity}(H_0)$
 If $(|S| < 2d)$ then exit and output H_0
 For t in $[1, 1000]$:
 For i in $[1, d+1]$:
 $\nu = N(0, 1)$
 $\sigma_i^{t+1} = \sigma_i^t \exp(\tau' \nu + \tau N(0, 1))$
 $a_i^{t+1} = a_i^t + \sigma_i^{t+1} N(0, 1)$
 $I_1 = \text{impurity}(H)$
 If $(I_1 < I)$ then $I = I_1$
 Output the split corresponding to I

Fig. 2. Overview of the OC1-ES algorithm for a single node of a decision tree.

experiments show that random initialization results in less accurate trees.

OC1 uses a two-step process to find oblique splits. The first step finds locally optimal values for one coefficient at a time. The sequence in which coefficients are examined can be determined by the user or can be set randomly (the default is that coefficients are examined sequentially). After no improvements are made, the second step of the OC1 algorithm perturbs the oblique hyperplane in a random direction. The default parameter values for the algorithm in Fig. 1 are $R = 20$ restarts and $J = 5$ random perturbations (jumps).

OC1-ES, our first extension to OC1, replaces the hillclimbing with multiple restarts of the OC1 algorithm with a $(1 + 1)$ evolution strategy with self-adaptive mutations. The algorithm is outlined in Fig. 2. The candidate hyperplane is represented as a vector of real-valued coefficients, a_1, \dots, a_{d+1} . The initial hyperplane is the best axis-parallel split. For each hyperplane coefficient, there is a corresponding mutation coefficient σ_i , $i = 1, \dots, d + 1$, which is initially set to 1. At each iteration, t , the mutation coefficients are updated and a new hyperplane is obtained according to the following rule:

$$\begin{aligned} \nu &= N(0, 1) \\ \sigma_i^{t+1} &= \sigma_i^t \exp(\tau' \nu + \tau N(0, 1)) \\ a_i^{t+1} &= a_i^t + \sigma_i^{t+1} N(0, 1) \end{aligned} \quad (2)$$

where $N(0, 1)$ indicates a realization of a unit normal variate, $\tau = (\sqrt{2} \sqrt{d})^{-1}$, and $\tau' = (\sqrt{2d})^{-1}$. The ES was stopped after 1000 iterations. We recognize that using a plus selection strategy $(1 + 1)$ with self-adaptation may cause misadapted strategy parameters to survive for a long time, and this may result in subpar solutions [1]. However, for this application, we found that removing the self-adaptation consistently resulted in less accurate classifiers. Using larger populations usually results in solutions of better quality and is one of our future research directions.

Our second extension to OC1 uses a simple generational GA with real-valued genes (OC1-GA). The algorithm is outlined in Fig. 3. For all the experiments, the GA used pairwise tournament selection without replacement, uniform crossover with probability 1.0, and no mutation. Preliminary experiments (not reported) did not show any benefit of adding mutation to our GA for inducing oblique trees. The population size was set to $20 \sqrt{d}$, using a population-sizing theory that asserts that the population size required to reach a solution of a particular quality is $O(\sqrt{d})$ [22]. The best axis-parallel hyperplane was copied to 10% of the

To find a split of a set of examples S with dimension d :
Find H_0 , the best axis-parallel split of S . $I = \text{impurity}(H_0)$
If $(|S| < 2d)$ then exit and output H_0
Copy H_0 to 10% of the initial population P_1 , initialize rest randomly
Evaluate(P_1)
 $H = \text{best hyperplane in } P_1$
For t in $[1, 24]$:
 $P' = \text{tournament-selection}(P_t)$
 $P_{t+1} = \text{uniform-crossover}(P')$
 Evaluate(P_{t+1})
 $H = \text{best hyperplane in } P_{t+1}$
Output H

Fig. 3. Overview of the OC1-GA algorithm for a single node of a decision tree.

To find a split of a set of examples S with dimension d :
Find H_0 , the best axis-parallel split of S . $I = \text{impurity}(H_0)$
If $(|S| < 2d)$ then exit and output H_0
 $T = 1$
For t in $[1, 20]$:
 successes = 0
 For k in $[1, 50 d]$:
 For i in $[1, d + 1]$:
 $a_i^{t+1} = a_i^t + N(0, 1)$
 $I_1 = \text{impurity}(H)$
 $\delta = I_1 - I$
 If $(\delta < 0$ or with $\text{Prob}(\exp(-\delta/T)))$
 $I = I_1$
 successes++
 If (successes > 10 d) then break
 $T = T/2$
Output the split corresponding to I

Fig. 4. Overview of the OC1-SA algorithm for a single node of a decision tree.

initial population, and the remainder of the population was initialized randomly with coefficients $a_i \in [-200, 200]$. The GA was stopped after 25 generations because, in preliminary experiments, we observed that the quality did not improve much after 15–20 generations.

Our simulated annealing algorithm (OC1-SA) is different from the one used by Heath *et al.* [23]. Their algorithm modified only one coefficient at a time and they let the algorithm run for a large number of iterations to try to reach a global optimum. Since in our preliminary studies [12], we had success with EAs that can change all the coefficients simultaneously, we decided to try a simple SA algorithm with this characteristic.

Our SA algorithm is outlined in Fig. 4. It begins with a temperature of 1, and the initial hyperplane is the best axis-parallel solution found by OC1. The inner loop of the algorithm consists of perturbing the hyperplane by adding independent unit normal variates to each of the coefficients. The new hyperplane is evaluated and accepted as the current solution if it is better than the previous one or if it is worse with a probability $\exp(-\delta/T)$, where δ is the difference between the qualities of the hyperplane before and after it was perturbed and T is the temperature. This inner loop is repeated for $50d$ iterations or until $10d$ hyperplanes are accepted. The temperature is then reduced by half and the process is repeated 20 times.

The algorithms used the parameter settings specified above for all the data sets. The parameters were calibrated with a few runs using two data sets (Diabetes and LS10, which will be described later) that were chosen mainly because they are not too large, and because one of them (LS-10) is completely separable by oblique hyperplanes. We did not spend much time adjusting the parameters of the algorithms, and it is probable that we could have obtained higher accuracies or smaller trees in shorter times by tuning the parameters to each data set. Note, however, that in contrast with OC1 and CART-LC, the algorithms that we introduced (OC1-SA, OC1-GA, and OC1-ES) consider the dimensionality of the problem to set their control parameters or to generate new candidate solutions.

All experiments measure the impurity of a split at each tree node using the twoing rule [8], which is the default in OC1

$$\text{impurity} = \frac{N_L}{N} \frac{N_R}{N} \left(\sum_{i=1}^k \frac{L_i}{N_L} - \frac{R_i}{N_R} \right)^2 \quad (3)$$

where N_L and N_R are the number of examples on the left and right of split, N is the total number of examples under consideration at a node, and L_i and R_i are the number of examples of category i on the left and right of the split, respectively. For the EAs, the impurity was used without modification as the fitness of the hyperplanes.

For comparison purposes, we followed Murthy *et al.* [33] and used crossvalidation experiments. Crossvalidation is a well-known technique to estimate the generalization of classifiers (i.e., their ability to correctly classify unseen data). In a k -fold crossvalidation experiment, the data D is randomly divided into k nonoverlapping sets D_1, D_2, \dots, D_k . At each iteration i (from 1 to k), the tree is trained with $D \setminus D_i$ and tested on D_i . The result of a crossvalidation experiment is the average of the k trials.

All our results are the average of ten five-fold crossvalidation experiments (i.e., 50 trees in total were trained and tested for each data set). We report the accuracy measured as the percentage of instances classified correctly, the size of the tree measured by the number of leaves, and the execution time of the program measured in seconds, along with 95% confidence intervals for each result. We apply a t test with a 0.05 level of significance to determine if the performance of two algorithms is different, but we recognize that t tests based on cross-validation experiments have an elevated type I error [13] (i.e., the test

TABLE I
DESCRIPTIONS OF THE SMALL PUBLIC-DOMAIN DATA SETS USED IN THE EXPERIMENTS

| Name | Task Description | Attributes | No. of Instances |
|----------|---------------------------------------------|------------|------------------|
| Cancer | Diagnose a tumor as benign or malignant | 9 | 683 |
| Diabetes | Detect presence of diabetes | 8 | 768 |
| Glass | Identify type of glass | 10 | 214 |
| Housing | Predict housing values in suburbs of Boston | 12 | 506 |
| Iris | Classify type of iris | 4 | 150 |
| Vehicle | Identify vehicle silhouettes | 18 | 846 |
| Vowel | Identify spoken vowels | 10 | 990 |

will incorrectly indicate a significant difference more often than the expected 5% of the time).

The execution times were measured on a 500-MHz Pentium III PC with 128 Mb of RAM running NT 4.0. The programs were compiled with the ecgs compiler version 2.91 using -O optimizations.

V. RESULTS

To demonstrate the feasibility of using EAs to search for oblique splits, we conducted four sets of experiments. In the first set, we used public-domain data sets from the UCI machine learning repository [4]. We included the four data sets that Murthy *et al.* [33] used to evaluate OC1 to compare our results with theirs. Next, we used artificial data with known properties, and we performed experiments to study the scalability of the different algorithms to the dimensionality of the domain. We also experimented with larger data sets to illustrate how sampling may help to scale up the evolutionary approach to more realistic situations. Finally, we present experiments with ensembles.

A. Small Data Sets

The first round of experiments used small public domain data sets, which are available at the UCI machine learning repository. The data sets are briefly described in Table I, and have been used in numerous studies of machine learning and data mining algorithms.

The results are summarized in Fig. 5. The data used to produce the graphs is presented in the Appendix. For most data sets, the differences in the accuracy of the algorithms was small, although in most cases the AP trees were significantly less accurate than the best oblique trees. For six data sets, OC1-SA and OC1-ES found trees that are the most accurate or statistically indistinguishable from the most accurate, followed by OC1 and OC1-GA with five top-performing trees each. In terms of tree size, OC1 found most of the smallest trees. The average size of the trees found by the GA and ES-augmented inducers was close to the axis-parallel algorithm. The most noticeable differences were in execution times: the EAs were, on average, approximately three times faster than OC1 and about five times faster than OC1-SA, but much slower than OC1-AP and CART-LC. On average, CART-LC was the fastest oblique tree algorithm on these data sets, but it found top performing trees less often than the other oblique algorithms.

Our results on Cancer, Iris, Housing, and Diabetes are similar to those obtained by Murthy *et al.* [33]. The small differences are not statistically significant and may be due to differences in

the hardware, operating system, and compiler that were used. Using simulated annealing, Heath *et al.* reported accuracies of 94.9% and 94.7% on the Cancer and Iris data sets, respectively [23], while our OC1-SA found trees with accuracies of 93.5% and 96.3%. We presume that our version of SA is faster than theirs because it evaluates far fewer hyperplanes per node, but we cannot make precise comparisons since they did not report execution times (and our hardware is probably very different from theirs).

B. Artificial Data

The next set of experiments used three artificial data sets. The purpose of these experiments is to ensure that the target concept matches the bias of the algorithms—the classes are separable by oblique hyperplanes, so we expect the AP trees to perform poorly on these data sets. In addition, we performed experiments to explore the scalability of the algorithms as the number of attributes was varied. Similar data sets were also used by Murthy *et al.* in their evaluation of OC1, but we used these data sets to study different properties of the algorithms.

The first artificial data set has 2000 instances divided into two classes. Each instance has d attributes whose values are uniformly distributed in $[0,1]$. The data are separable by the hyperplane $x_1 + \dots + x_{d/2} < x_{d/2+1} + \dots + x_d$, where $d \in \{10, 20, 50, 100\}$. These data sets are labeled LS10, LS20, LS50, and LS100 according to their dimensionality.

The results are summarized in Table II. In the table, the results highlighted in bold are the most accurate and those that are not significantly different (with 95% confidence) from the most accurate.² In this case, OC1-AP consistently found the least accurate and largest trees. As expected, OC1-AP was the fastest algorithm, but its accuracy was too low to consider AP trees competitive (consider that random guessing would result in a 50% accuracy and the accuracy of OC1-AP on LS100 is 56%). Our OC1-SA produced accurate and very small trees for LS10 and LS20, but in higher dimensions its performance dropped below the EA-augmented inducers, and it took the longest time at dimensions higher than 20. Murthy *et al.* [33] reported that Heath’s SA algorithm [23] is 95.2% accurate on LS10 (and LMDT obtained 95.2%). OC1-GA performed well at low dimensions and became the top performing algorithm at high dimensions. However, its execution time increased faster than OC1-ES, which appears to scale well to the increase in dimensionality, although it never found a top performing tree.

²Our results with the LS10 data are different from Murthy *et al.*’s because we used OC1’s default pruning option (using 10% of the data), but they did not prune the resulting trees.

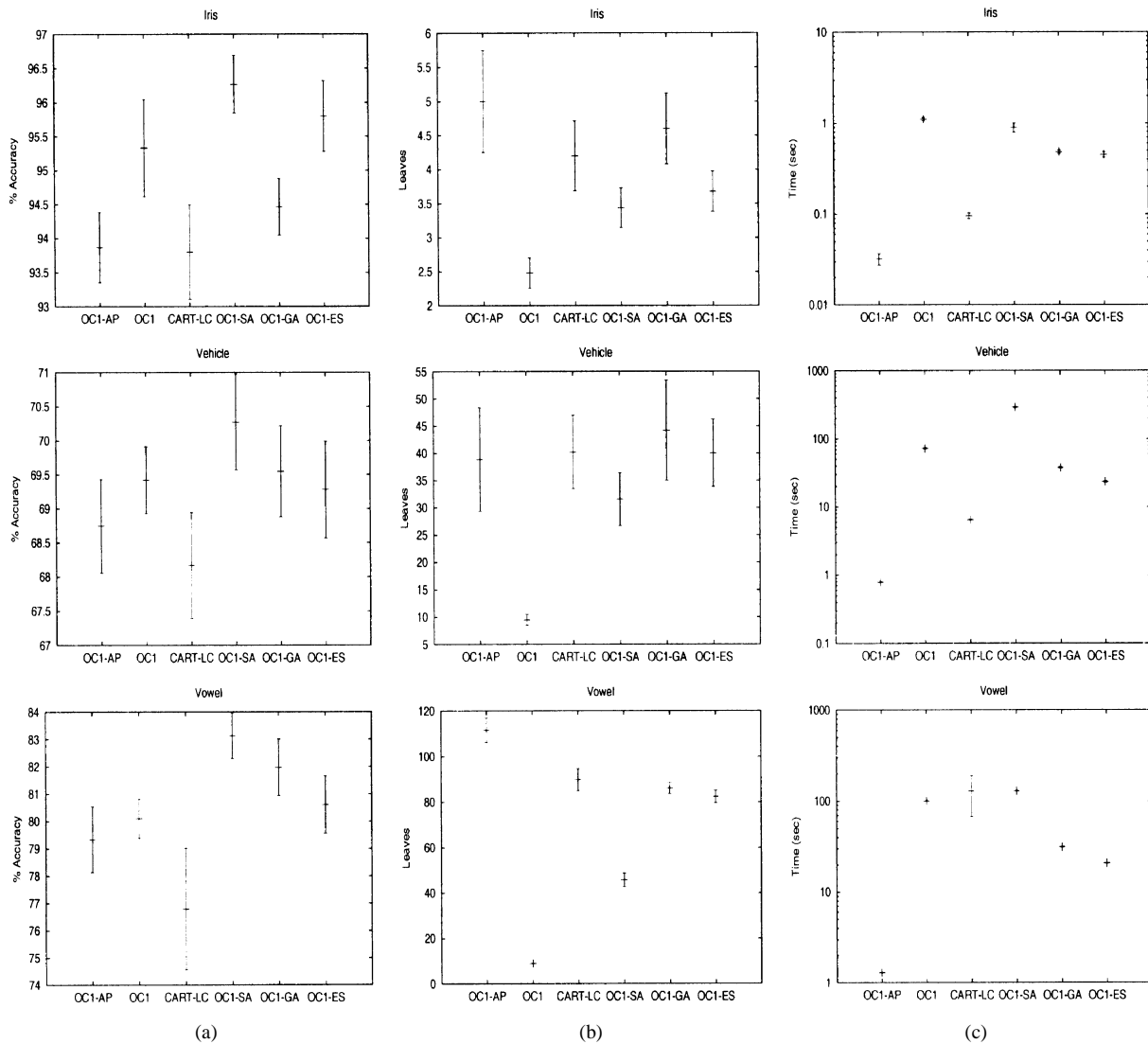


Fig. 5. Results comparing algorithms on small public-domain data sets. For each of the four data sets (Cancer, Diabetes, Glass, Housing), there are three graphs that present the (a) accuracy, (b) number of leaves in the tree, and (c) time measured in seconds. The error bars represent 95% confidence intervals.

The size of the trees found by OC1 and CART-LC increases with the number of dimensions, but those of OC1-GA, OC1-ES, and OC1-SA remained relatively small. However, consider that the ideal tree for this domain has two leaves and all the algorithms found much larger trees (except for OC1-SA on LS10 and LS20).

The second and third artificial data sets, POL2 and RCB2, are defined in two dimensions ($x_1, x_2 \in [0, 1]$), and depicted in Fig. 6. The concept represented by the POL2 data is a set of four parallel oblique lines (hence its name), it contains 2000 instances divided into two classes. The “rotated checker board” (RCB2) data also has 2000 instances, but in this case they are divided into eight classes.

The results are summarized in Table III. In these two domains, OC1 produced the most accurate and smallest trees. The smallest trees for POL2 and RCB2 have five and eight leaves, respectively, and OC1 consistently found trees of those sizes. As expected, the axis-parallel trees are the largest and least accurate, but OC1-GA found only slightly more accurate and smaller trees. We hypothesize that the population size may have been too

small for the GA to reach good solutions. Doubling the population size to $40\sqrt{2}$ resulted in an accuracy of 98.65% (± 1.2) and trees with 14.5 leaves for POL2, and 95.5% (± 1.6) and 47 leaves for RCB2. The fastest oblique DT algorithm was CART-LC, but its accuracy is lower than OC1 and OC1-ES. Both of the EA inducers were approximately eight times faster than OC1, but in these two problems the overall performance of OC1-ES was much better than OC1-GA.

Murthy *et al.* reported that LMDT and Heath’s SA algorithm obtained accuracies of 89.6% and 99.3% in POL2, and 95.7 and 97.9% on RCB2 [33].

C. Larger Data Sets and Sampling

To study the problem of scalability to larger data sets, we experimented with three larger data sets, which are also available at the UCI repository. With these data sets, we illustrate a more realistic application of EAs to the problem of oblique DT induction. The larger size of the training set could cause fitness evaluations to be prohibitively expensive, and therefore we seek to

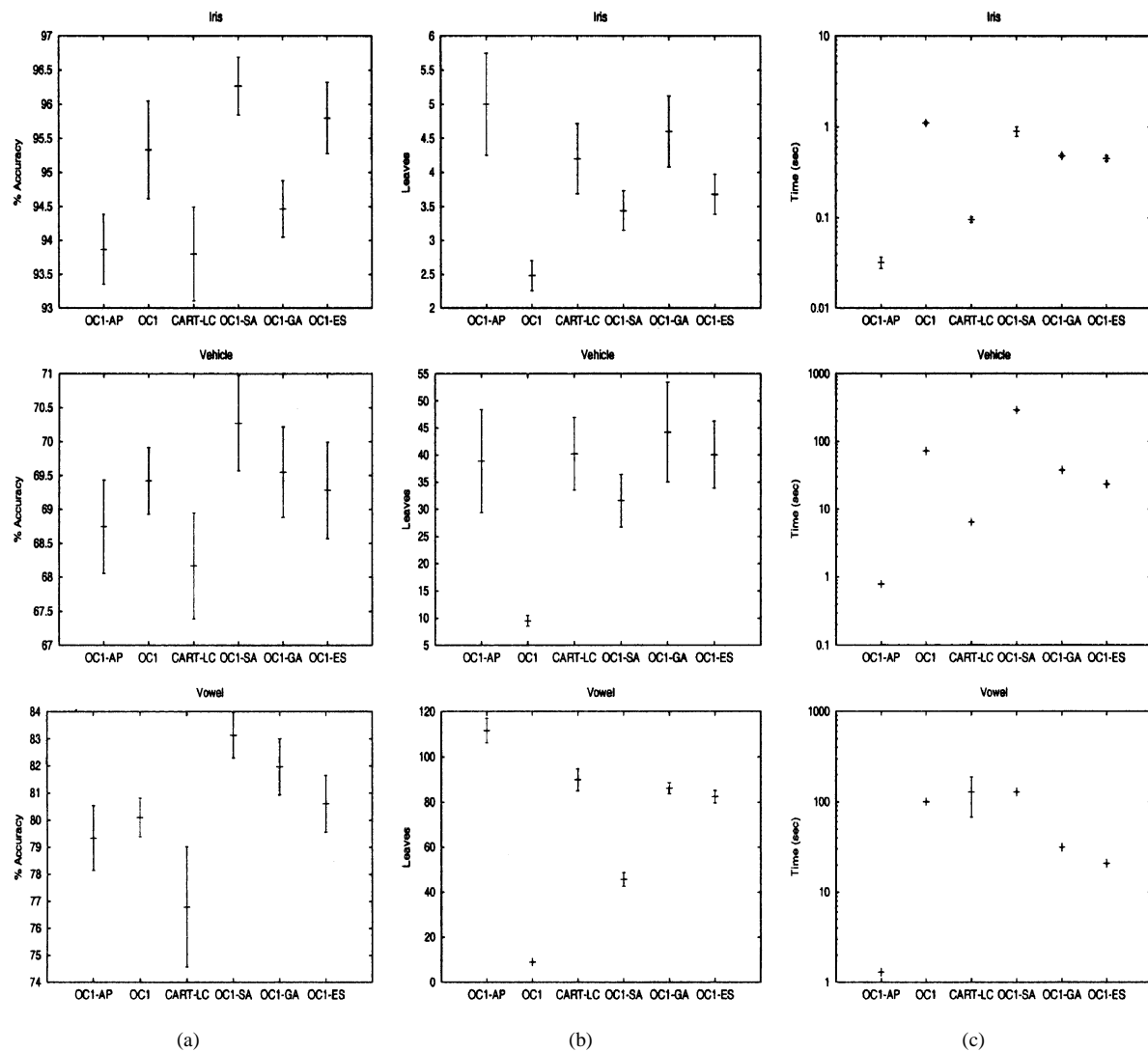


Fig. 5. (Continued.) Results comparing algorithms on small public-domain data sets (cont.). For each of the three data sets (Iris, Vehicle, Vowel), there are three graphs that present the (a) accuracy, (b) number of leaves in the tree, and (c) time measured in seconds. The error bars represent 95% confidence intervals.

TABLE II
COMPARISON OF DIFFERENT ALGORITHMS ON THE LS ARTIFICIAL DATA SETS

| Dim. | | OC1-AP | OC1 | CART-LC | OC1-SA | OC1-GA | OC1-ES |
|------|----------|-----------|------------|-----------|-----------------|-----------------|------------|
| 10 | Accuracy | 73.0±0.9 | 97.1±0.2 | 96.0±0.9 | 99.4±0.1 | 95.4±0.4 | 93.7±0.5 |
| | Leaves | 86.7±9.7 | 5.3±1.3 | 5.9±2.1 | 2.0±0 | 8.8±2.3 | 9.9±1.6 |
| | Time | 1.6±0.0 | 170.9±7.0 | 16.8±0.7 | 28.0±1.9 | 36.3±2.3 | 29.8±1.4 |
| 20 | Accuracy | 64.4±0.5 | 88.5±0.7 | 87.3±1.1 | 96.3±0.9 | 92.0±0.4 | 87.0±0.6 |
| | Leaves | 71.5±17.0 | 5.9±1.6 | 9.3±2.1 | 3.3±0.8 | 9.8±3.5 | 14.4±3.3 |
| | Time | 3.5±0.1 | 391.5±9.8 | 54.9±2.1 | 154.7±15.1 | 101.5±2.8 | 65.1±2.0 |
| 50 | Accuracy | 58.6±0.6 | 72.5±0.8 | 66.3±0.6 | 79.8±1.3 | 85.2±0.6 | 78.5±0.9 |
| | Leaves | 58.0±12.2 | 10.0±2.1 | 25.0±10.4 | 15.6±2.9 | 9.5±3.3 | 16.3±5.5 |
| | Time | 11.7±0.3 | 608.7±19.3 | 113.9±2.1 | 1278.6±51.2 | 333.3±13.0 | 163.9±8.7 |
| 100 | Accuracy | 56.1±0.5 | 61.8±0.6 | 58.5±0.7 | 70.6±0.7 | 75.9±0.6 | 70.1±0.9 |
| | Leaves | 37.7±9.1 | 28.5±7.4 | 36.2±12.3 | 17.6±4.8 | 13.9±3.8 | 13.8±2.8 |
| | Time | 30.9±0.3 | 802.6±12.2 | 156.3±6.7 | 5020.4±187.9 | 900.1±7.9 | 296.9±17.4 |

obtain faster approximate evaluations by sampling the training set.

We consider two ways of sampling. The first is a preprocessing step in which the training set is sampled once at the beginning of an experiment. The instances sampled are input to the tree-induction algorithm, which is not modified. This static

sampling ignores all the instances that were not selected originally, possibly wasting valuable information. However, static sampling is valuable because it simulates a situation when not much data are available for training, which is often the case in scientific domains as the training sets have to be labeled manually. The second way of sampling is to modify the tree induc-

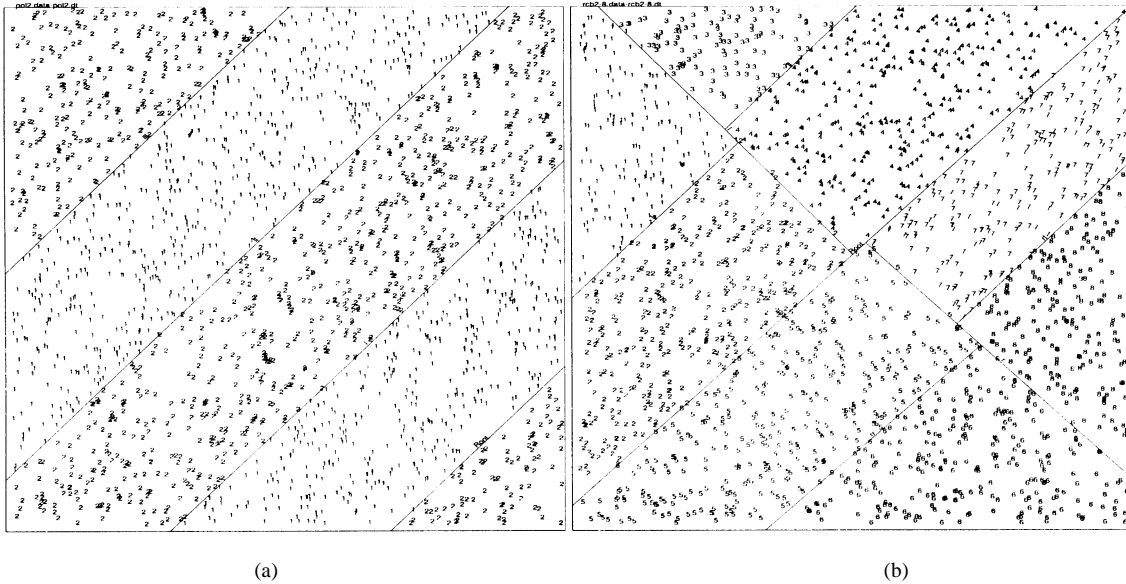


Fig. 6. (a) POL2 and (b) RCB2 artificial data sets.

TABLE III
COMPARISON OF DIFFERENT ALGORITHMS ON THE 2-D ARTIFICIAL DATA SETS

| Data set | | OC1-AP | OC1 | CART-LC | OC1-SA | OC1-GA | OC1-ES |
|----------|----------|----------|------------------|----------|----------|----------|-----------------|
| POL2 | Accuracy | 94.2±0.3 | 99.6±0.07 | 97.6±0.3 | 99.0±0.2 | 95.3±0.2 | 99.4±0.2 |
| | Leaves | 77.7±6.1 | 5.0±0 | 14.4±1.7 | 10.3±1.5 | 57.5±6.1 | 6.3±0.6 |
| | Time | 0.3±0.01 | 36±1.3 | 2.7±0.1 | 8.1±0.5 | 4.7±0.2 | 4.5±0.2 |
| RCB2 | Accuracy | 92.8±0.2 | 99.0±0.07 | 94.4±0.2 | 97.9±0.2 | 93.8±0.4 | 98.1±0.2 |
| | Leaves | 85.7±4 | 8.4±0.2 | 50.6±4.2 | 17.5±2.4 | 64.6±5.7 | 10.9±1.1 |
| | Time | 0.4±0.01 | 44.8±0.9 | 3.4±0.1 | 10.0±0.3 | 5.0±0.1 | 6.0±0.2 |

tion algorithms to choose a random fraction of the data available at a tree node each time a hyperplane is evaluated. This dynamic sampling method is more expensive than sampling statically once per tree induction. For example, using OC1-GA with a population of 20 individuals and running for 25 generations would require 500 hyperplane evaluations and, therefore, 500 samplings. However, sampling is an inexpensive operation, and the experiments demonstrate that sampling dynamically is especially advantageous when the sample sizes are small.

Evaluating the hyperplanes with dynamic samples also means that every time a particular hyperplane is evaluated, its fitness estimate is different, because it is evaluated on a different random sample. Repeated evaluations of the same hyperplane would enable us to better estimate its true fitness (e.g., by taking the average of multiple evaluations), and some recent theory can be used to determine the optimal number of repetitive evaluations that would minimize the execution time of the GA [30]. As a first cut, however, we decided to use a single sample as a crude—but fast—estimate of fitness.

The experiments in this section used two handwritten digit-recognition data sets. The objective is to identify each instance as one of ten digits. The first data set is the optical digit-recognition data set, which has 3823 instances in a training set and 1797 in a test set; each instance is described by 64 numeric attributes. The second data set is the pen-based set that has 7494 training cases and 3498 testing cases; each instance is described by 16 numeric attributes. For these experiments, we report the

average of ten experiments, with training and testing using the partition of the instances as in the UCI repository. The samples are drawn without replacement.

The results on the optical recognition set with dynamic sampling are summarized in Fig. 7. The data used to generate the plots is in Table IV. Sampling decreased the execution time as desired, but it also affected the accuracy. For all the sample sizes, OC1-GA found the smallest and most accurate classifiers, and in most cases it was faster than the original oblique OC1. OC1-ES was the fastest of the oblique classifiers, and in most cases its accuracy was better than OC1, CART-LC, and OC1-SA, but not as good as OC1-GA. Note, however, that the axis-parallel OC1 was the fastest algorithm, and that its accuracy was similar to OC1-ES. In fact, using OC1-AP with the entire data set was faster and more accurate than OC1-GA on 10% samples, so if the end user does not care about the relatively small differences in accuracy, axis-parallel DTs would be a good choice in this domain. If accuracy or tree size is a premium, then OC1-GA would be the best option.

In separate experiments, we found that dynamic sampling gave more accurate results than sampling statically at the beginning of the experiments. For static samples of 25% or more of the training set, the accuracy was only slightly lower than with dynamic sampling (approximately 4% to 5%), but for smaller static samples, the accuracy was between 6 to 22% lower. The general trends were the same as with repetitive sampling, so we omit those results.

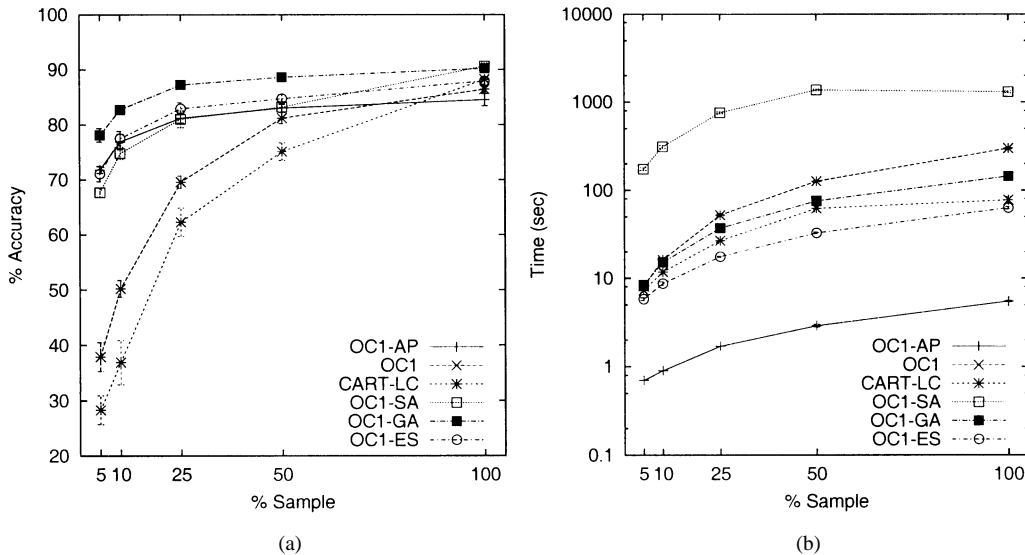


Fig. 7. Comparison of different algorithms on the optical digit-recognition data sampling dynamically (5%–100% of the data at each node) every time that a hyperplane is evaluated.

TABLE IV

COMPARISON OF DIFFERENT ALGORITHMS ON THE OPTICAL DIGIT RECOGNITION DATA SAMPLING DYNAMICALLY (5%–100% OF THE DATA AT EACH NODE) EVERY TIME THAT A HYPERPLANE WAS EVALUATED. THE RESULTS HIGHLIGHTED IN BOLD ARE THE MOST ACCURATE AND THOSE THAT ARE NOT SIGNIFICANTLY DIFFERENT FROM THE MOST ACCURATE ACCORDING TO THE t TEST AT A 0.05 LEVEL OF SIGNIFICANCE

| % Sample | | OC1-AP | OC1 | CART-LC | OC1-SA | OC1-GA | OC1-ES |
|----------|----------|------------|------------|------------|-----------------|-----------------|-----------|
| 5 | Accuracy | 71.8±0.6 | 37.9±2.6 | 28.3±2.6 | 67.7±0.8 | 78.1±1.2 | 71.1±1.4 |
| | Leaves | 32.0±3.3 | 72.3±11.5 | 61.1±33.3 | 17.3±2.3 | 14.7±2.4 | 19.1±2.3 |
| | Time | 0.7±0.0 | 8.1±0.2 | 7.3±0.4 | 173.8±6.2 | 8.4±0.3 | 5.8±0.2 |
| 10 | Accuracy | 76.9±1.4 | 50.2±1.5 | 36.9±4.0 | 74.8±1.1 | 82.7±0.8 | 77.5±1.3 |
| | Leaves | 49.0±3.8 | 101.8±19.9 | 158.2±37.8 | 34.9±4.2 | 20.0±3.2 | 26.6±4.3 |
| | Time | 0.9±0.0 | 16.2±0.3 | 11.8±0.7 | 310.9±11.8 | 15.2±0.3 | 8.7±0.4 |
| 25 | Accuracy | 81.2±0.9 | 69.6±1.1 | 62.3±2.6 | 81.0±1.5 | 87.2±0.5 | 82.9±1.1 |
| | Leaves | 77.6±6.7 | 155.4±52.9 | 179.6±68.2 | 66.9±16.3 | 33.0±5.4 | 43.4±7.7 |
| | Time | 1.7±0.0 | 52.1±1.7 | 26.7±1.9 | 755.0±19.9 | 37.1±0.5 | 17.6±0.2 |
| 50 | Accuracy | 83.0±0.7 | 81.2±1.0 | 75.1±1.6 | 83.2±1.0 | 88.6±0.7 | 84.7±0.8 |
| | Leaves | 112.1±23.8 | 182.1±48.9 | 163.0±56.6 | 64.1±99.0 | 31.9±7.1 | 84.0±16.7 |
| | Time | 2.9±0.1 | 126.6±2.4 | 62.1±5.3 | 1374.2±33.7 | 75.6±1.7 | 32.7±0.8 |
| 100 | Accuracy | 84.5±1.1 | 86.4±0.5 | 88.2±0.4 | 90.6±0.6 | 90.2±0.6 | 87.9±0.6 |
| | Leaves | 125.8±28.3 | 53.7±18.0 | 60.6±15.2 | 38.2±10.0 | 52.3±20.3 | 84.0±22.1 |
| | Time | 5.5±0.1 | 298.6±6.5 | 77.4±6.3 | 1305.0±33.9 | 144.2±2.6 | 63.0±1.9 |

Fig. 8 summarizes the results with the pen-based data. The data used to generate the plots is in Table V. Again, OC1-GA found top trees in all of the sample sizes. OC1-ES also performed well, finding top trees in four cases and in less time than all the other oblique algorithms.

D. Ensembles

An ensemble is a set of classifiers whose individual decisions are combined in some way to classify new examples [14]. In this section, we show experimental results with ensembles that combine the decisions of the oblique trees using simple plurality voting (i.e., the class of the new example is the class that receives the largest number of votes). Ensembles will have better accuracy than single trees as long as the accuracy of each tree in the ensemble is better than random guessing and the trees are different among themselves [21]. Numerous theoretical and empirical studies confirm that ensembles perform better than single classifiers [2], [35], [14].

The experiments with ensembles used the optical and pen digit-recognition data sets described in the previous section. We performed several sets of experiments using each data set. First, we used all of the data available to build the individual trees, expecting that this would produce the most accurate ensembles. However, since the cost of building the ensemble is a multiple of the cost of building individual classifiers, we also expected that this option would be the most expensive. Since the last section showed that sampling has the potential of reducing the execution time, we also built ensembles by sampling at each node of the tree. Sampling is an additional source of randomization that permits us to build ensembles using the deterministic axis-parallel and CART-LC algorithms. Note that this is a different source of randomization than the bootstrap samples used in bagging [7], where the samples are drawn once from the training set before building each tree.

The experiments considered ensembles of ten trees, and the training and testing sets were the same as in the previous section. The OC1 code was easily modified to handle ensembles.

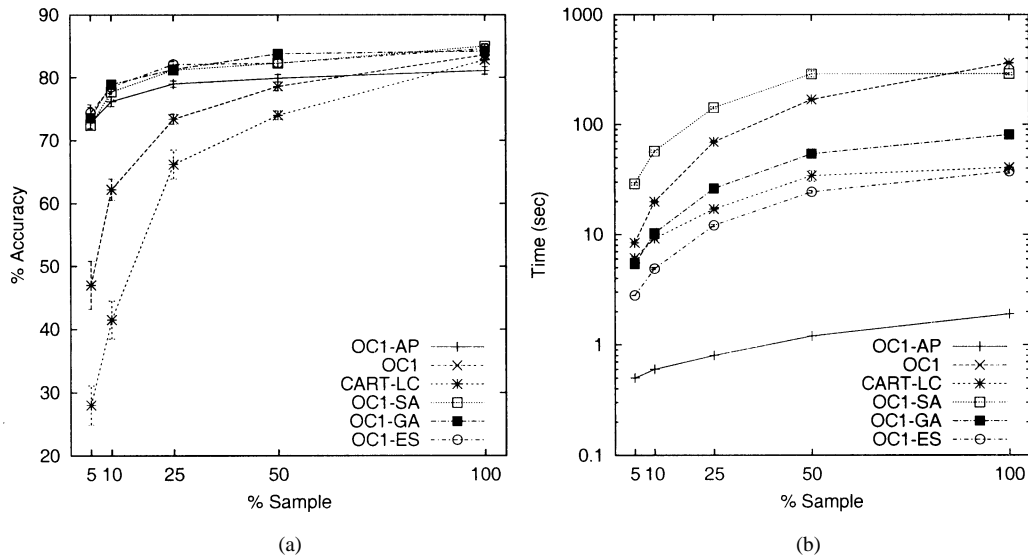


Fig. 8. Comparison of different algorithms on the pen digit-recognition data sampling dynamically (5%–100% of the data at each node) every time a hyperplane is evaluated.

TABLE V

ACCURACY OF DIFFERENT ALGORITHMS ON THE PEN-BASED DIGIT RECOGNITION DATA SAMPLING DYNAMICALLY EVERY TIME THAT A HYPERPLANE IS EVALUATED. THE RESULTS HIGHLIGHTED IN BOLD ARE THE MOST ACCURATE AND THOSE THAT ARE NOT SIGNIFICANTLY DIFFERENT FROM THE MOST ACCURATE ACCORDING TO THE t TEST AT A 0.05 LEVEL OF SIGNIFICANCE

| % Sample | | OC1-AP | OC1 | CART-LC | OC1-SA | OC1-GA | OC1-ES |
|----------|----------|-----------------|------------|------------|-----------------|-----------------|-----------------|
| 5 | Accuracy | 72.8±1.1 | 47.0±3.8 | 28.0±3.1 | 72.4±0.7 | 73.6±1.7 | 74.4±1.3 |
| | Leaves | 53.2±9.6 | 111.3±18.7 | 270.7±15 | 20.3 1.2 | 20.0±2.6 | 21.8±2.0 |
| | Time | 0.5±0.001 | 8.4±0.2 | 6.1±0.2 | 28.9±0.8 | 5.4±0.2 | 2.8±0.0 |
| 10 | Accuracy | 76.2±0.8 | 62.2±1.7 | 41.5±3.0 | 77.7±1.2 | 78.9±0.7 | 78.5±1.0 |
| | Leaves | 75.2±13.0 | 161.9±27.5 | 430.8±17.2 | 27.9±2.1 | 25.4±4.1 | 29.4±2.4 |
| | Time | 0.6±0.01 | 19.9±0.5 | 9.2±0.3 | 57.0±1.5 | 10.3±0.3 | 4.9±0.1 |
| 25 | Accuracy | 79.0±0.5 | 73.4±0.8 | 66.2±2.3 | 81.2±0.6 | 81.3±0.6 | 82.1±0.5 |
| | Leaves | 123.7±26.8 | 374.9±95.0 | 329.4±12.4 | 34.9±3.2 | 52.3±6.8 | 48.1±7.3 |
| | Time | 0.8±0.02 | 69.2±1.5 | 17.0±0.6 | 141.6±3.2 | 26.2±0.6 | 12.1±0.3 |
| 50 | Accuracy | 79.9±0.6 | 78.6±0.7 | 74.0±0.7 | 82.3±0.6 | 83.8±0.5 | 82.3±0.6 |
| | Leaves | 136.6±19.6 | 495.7±53.4 | 443.1±14 | 49.9±49.0 | 62.7±9.5 | 68.3±11.1 |
| | Time | 1.2±0.02 | 168.2±3.3 | 34.3±3.8 | 287.6±18.8 | 54.1±0.8 | 24.4±0.7 |
| 100 | Accuracy | 81.1±0.6 | 83.6±0.4 | 82.8±0.5 | 85.0±0.4 | 84.2±0.5 | 84.6±0.3 |
| | Leaves | 197.0±29.3 | 77.9±11.6 | 106.7±16.0 | 50.7±7.0 | 97.1±20.9 | 91.4±11.3 |
| | Time | 1.9±0.02 | 362.8±5.6 | 40.7±2.0 | 288.4±7.4 | 80.6±2.0 | 37.5±1.5 |

TABLE VI

RESULTS USING ENSEMBLES OF TEN TREES ON THE OPTICAL DIGIT RECOGNITION DATA. FIRST LINE: ACCURACY, SECOND LINE: TIME

| % Sample | | OC1-AP | OC1 | CART-LC | OC1-SA | OC1-GA | OC1-ES |
|----------|----------|----------|-------------|------------|-----------------|-----------------|-----------------|
| 5 | Accuracy | 86.6±0.6 | 59.9±2.3 | 38.6±1.8 | 87.9±0.6 | 89.6±0.5 | 89.5±0.3 |
| | Time | 7.7±0.6 | 78.5±1.5 | 7.9±0.7 | 1766.5±10.2 | 83.0±1.7 | 55.2±1.5 |
| 10 | Accuracy | 88.6±0.4 | 76.2±1.2 | 63.2±2.1 | 90.8±0.2 | 92.3±0.4 | 91.6±0.6 |
| | Time | 8.7±0.6 | 154.2±4.9 | 11.1±0.8 | 3125.0±15.0 | 151.7±2.8 | 78.5±1.9 |
| 100 | Accuracy | 84.5±1.1 | 93.9±0.2 | 91.1±0.6 | 95.8±0.3 | 95.6±0.2 | 94.9±0.3 |
| | Time | 50.6±1.1 | 2806.6±21.3 | 708.0±34.3 | 13101±108 | 2272.4±23.5 | 622.1±7.7 |

The results presented in Tables VI and VII are the average of ten trials with each configuration. Note that when all of the data was used, the deterministic algorithms (OC1-AP and CART-LC) produced ensembles of ten identical trees that have exactly the same performance as the individual trees; their results are included here to facilitate comparisons.

As expected, the ensembles created with all the data exhibit better accuracy than the individual classifiers (compare Tables IV and V with Tables VI and VII, respectively). Perhaps it is more interesting to note that some of the ensembles created

by sampling also had higher accuracy than the most accurate individual trees, and that sometimes the ensembles were created in shorter time. For example, on the optical digit data, building an ensemble with OC1-ES and sampling at 10% gave a higher accuracy (91.6%) than any single classifier built from the entire data, and it was faster to build the ensemble than the most accurate single tree. Actually, the ensembles generated by OC1-GA and OC1-ES on 5% samples were more accurate (89.6% and 89.5%, respectively) and faster to generate (83 and 55.2 s, respectively) than single trees found by the existing

TABLE VII
RESULTS USING ENSEMBLES OF TEN TREES ON THE PEN DIGIT RECOGNITION DATA. FIRST LINE: ACCURACY. SECOND LINE: TIME

| % Sample | | OC1-AP | OC1 | CART-LC | OC1-SA | OC1-GA | OC1-ES |
|----------|----------|----------|-------------|------------|-----------------|-----------------|-----------------|
| 5 | Accuracy | 79.9±0.5 | 66.1±1.8 | 50.2±3.4 | 80.0±0.3 | 80.4±0.5 | 81.0±0.6 |
| | Time | 5.1±0.2 | 79.2±2.1 | 59.5±0.9 | 290.1±1.9 | 54.3±0.8 | 26.9±0.7 |
| 10 | Accuracy | 81.0±0.3 | 74.6±0.5 | 62.0±3.1 | 84.1±0.3 | 83.5±0.4 | 84.2±0.2 |
| | Time | 6.5±0.5 | 188.5±3.3 | 90.3±1.2 | 567.9±3.8 | 101.8±1.5 | 46.7±1.1 |
| 100 | Accuracy | 81.1±0.6 | 86.8±0.1 | 82.8±0.4 | 87.4±0.1 | 87.1±0.1 | 87.3±0.1 |
| | Time | 17.1±0.4 | 3677.8±34.8 | 403.1±20.2 | 2921.0±35.5 | 796.4±9.1 | 367.0±3.6 |

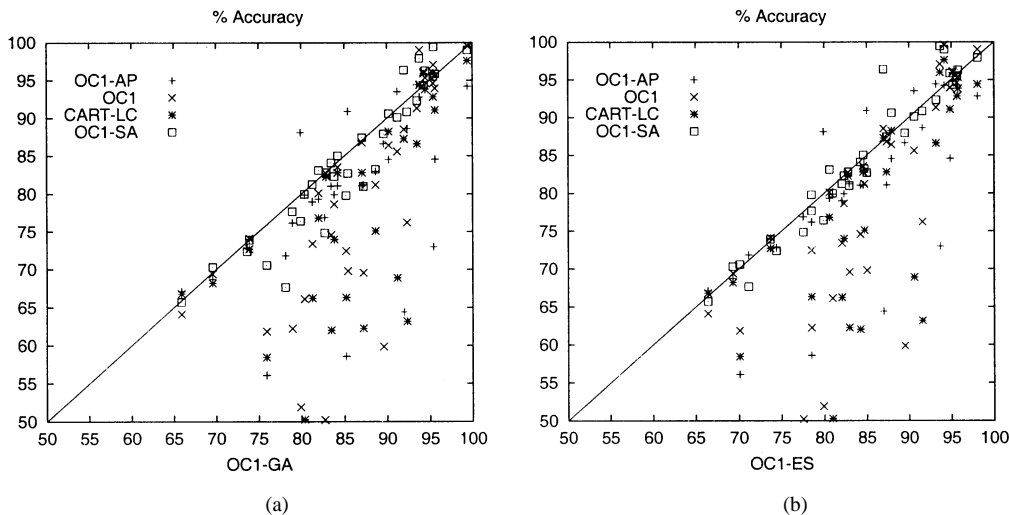


Fig. 9. Comparison of accuracy rates of the (a) GA and (b) ES against the other algorithms. Most points are below the diagonal indicating that in most cases the EAs found more accurate solutions than the other methods.

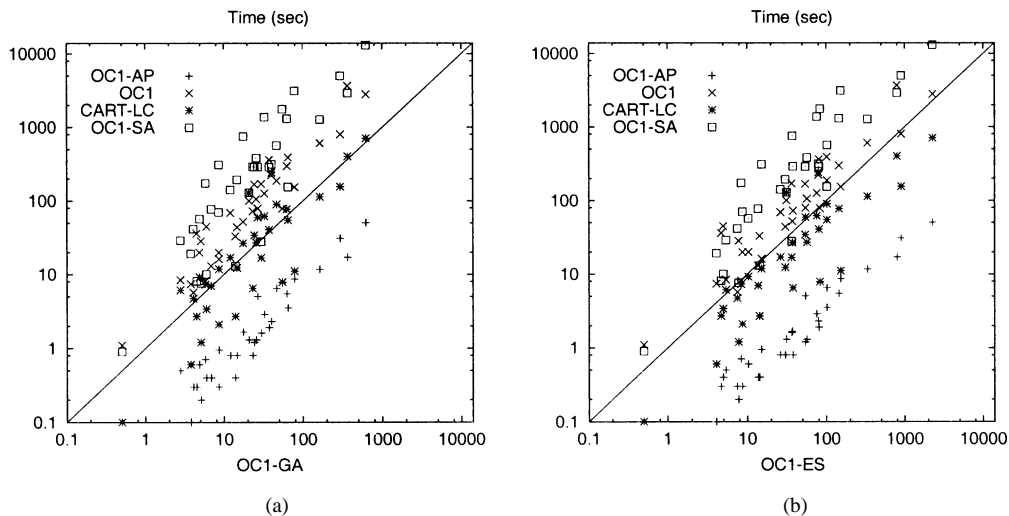


Fig. 10. Comparison of the execution times of the (a) GA and (b) ES against the other algorithms. The points below the diagonal mostly correspond to AP and CART-LC trees. AP trees were always faster than the EAs, but sometimes CART-LC was slower than the EAs. OC1 and OC1-SA were generally slower.

OC1 and CART-LC algorithms on the entire data (86.4% and 88.2%; 298 and 77 s). The results with the pen-based data are not as impressive, but still the ensembles created by ES with 10% samples outperform the single trees found by OC1 and CART-LC. As in the previous section, the degradation in accuracy as the sample fraction is reduced is smaller for OC1-SA, OC1-GA, and OC1-ES than for OC1 and CART-LC.

E. Summary of Results

Fig. 9 shows scatter plots that summarize the accuracy of the trees found by OC1-GA and OC1-ES against the other algorithms on all the data sets used in the previous sections. For each

point (x, y) , x represents the accuracy obtained by OC1-GA or the OC1-ES and y represents the accuracy of the other method. The points above the diagonal indicate that the other method outperforms the EAs. The plots show that the other methods (mainly OC1-SA and CART-LC) found more accurate trees than the EAs in only a few cases.

Figs. 10 and 11 show similar comparisons for the execution time and the number of leaves. In these graphs, points below the diagonal indicate that the other method outperformed the EAs. We can see that OC1-AP and CART-LC were generally faster than the EAs, and that OC1-SA and OC1 tended to be slower but found smaller trees than the EAs.

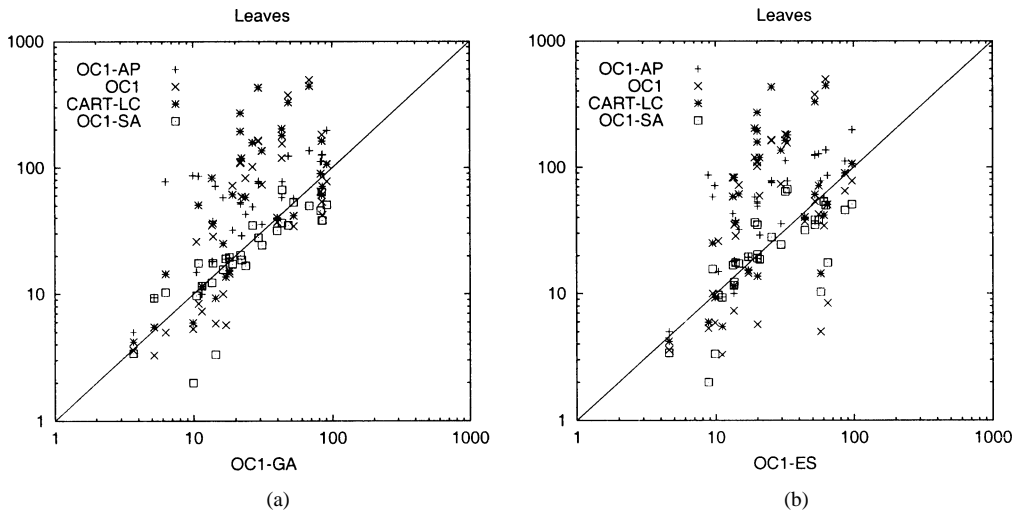


Fig. 11. Comparison of the tree sizes (given by the number of leaves) found by the (a) GA and (b) ES against the other algorithms.

TABLE VIII

COMPARISON OF DIFFERENT ALGORITHMS ON THE SMALL PUBLIC DOMAIN DATA SETS. THE RESULTS HIGHLIGHTED IN BOLD ARE THE MOST ACCURATE AND THOSE THAT ARE NOT SIGNIFICANTLY DIFFERENT FROM THE MOST ACCURATE ACCORDING TO THE t TEST AT A 0.05 LEVEL OF SIGNIFICANCE

| Data set | | OC1-AP | OC1 | CART-LC | OC1-SA | OC1-GA | OC1-ES |
|----------|----------|-----------------|------------------|------------------|-----------------|------------------|-----------------|
| Cancer | Accuracy | 94.7±0.4 | 96.1±0.6 | 95.9±0.3 | 93.5±0.5 | 94.3±0.4 | 95.2±0.5 |
| | Leaves | 9.4±1.5 | 3.3±0.6 | 5.5±1.4 | 11.5±4.2 | 11.1±2.2 | 5.2±2.1 |
| | Time | 0.2±.002 | 28.4±5.13 | 1.2±0.1 | 12.5±0.5 | 7.8±0.5 | 5.1±0.2 |
| Diabetes | Accuracy | 74.0±0.6 | 74.1±1.16 | 72.7±1.06 | 73.9±0.7 | 73.9±0.95 | 73.7±0.8 |
| | Leaves | 18.9±5.1 | 15.7±1.24 | 3.7±4.58 | 19.0±6.7 | 20.0±6.13 | 17.1±2.9 |
| | Time | 0.4±0.01 | 33.0±0.84 | 2.7±0.06 | 13.0±0.1 | 14.3±0.27 | 14.0±0.2 |
| Glass | Accuracy | 67.1±0.9 | 64.1±1.8 | 66.8±1.3 | 65.7±1.8 | 65.9±1.8 | 66.4±2.1 |
| | Leaves | 19.3±4.1 | 14.5±3.6 | 15.2±3.1 | 19.5±4.6 | 17.2±3.3 | 18.1±3.4 |
| | Time | 0.1±.008 | 7.4±0.2 | 0.6±0.02 | 19.1±0.5 | 4.1±0.1 | 3.8±0.1 |
| Housing | Accuracy | 82.2±0.6 | 82.7±1.1 | 82.3±0.8 | 82.8±0.7 | 82.9±0.5 | 82.8±0.6 |
| | Leaves | 10.0±3.8 | 7.3±1.5 | 11.5±1.6 | 11.6±3.3 | 13.5±3.2 | 11.5±3.3 |
| | Time | 0.3±0.01 | 19.8±0.7 | 2.1±.08 | 70.2±2.9 | 8.7±0.2 | 8.6±0.2 |
| Iris | Accuracy | 93.9±0.5 | 95.3±0.7 | 93.8±0.7 | 96.3±0.4 | 94.5±0.4 | 95.8±0.5 |
| | Leaves | 5.0±0.7 | 3.6±0.2 | 4.2±0.5 | 3.4±0.3 | 4.6±0.5 | 3.7±0.03 |
| | Time | 0.03±0.004 | 1.1±0.4 | 0.1±.006 | 0.9±0.1 | 0.5±0.02 | 0.5±0.03 |
| Vehicle | Accuracy | 68.7±0.7 | 69.4±0.5 | 68.2±0.8 | 70.3±0.7 | 69.6±0.7 | 69.3±0.7 |
| | Leaves | 38.9±9.5 | 37.1±7.9 | 40.2±6.7 | 31.6±4.8 | 44.2±9.2 | 40.0±6.1 |
| | Time | 0.8±0.01 | 72.5±1.7 | 6.5±0.1 | 290±7.6 | 37.7±0.7 | 23.3±0.5 |
| Vowel | Accuracy | 79.2±1.2 | 80.1±0.7 | 76.8±2.2 | 83.1±0.8 | 82±1.03 | 80.6±1.05 |
| | Leaves | 111±5.3 | 64.8±2.6 | 89.8±4.8 | 45.7±3 | 86.2±2.5 | 82.4±2.8 |
| | Time | 1.3±0.01 | 100.4±0.6 | 128.4±60 | 129±1.9 | 31.3±0.5 | 20.8±0.3 |

VI. SUMMARY AND CONCLUSIONS

Traditional oblique DT inducers use some form of heuristic greedy search to find appropriate splits. In this paper, we substituted the greedy search with two EAs: a $(1 + 1)$ evolution strategy and a simple GA. We experimented with public domain and artificial data sets with different characteristics to evaluate the performance of the EA-based tree inducers, and we compared the results against an axis-parallel and three other oblique algorithms. We evaluated the use of sampling to further reduce the execution time of the inducers. As expected, sampling resulted in faster training times, but also in a loss of accuracy, which was more pronounced in the existing oblique algorithms. In addition, we introduced new methods to generate ensembles of trees.

Our results suggest that in many cases, the EAs are capable of finding oblique trees with similar or higher accuracy than

existing algorithms, and that this can be done at a competitive cost. The experiments also suggest that the EAs scale up better than traditional methods to the dimensionality of the data. In our experiments, the EAs seem to perform better than existing methods when the available data are sampled. This is important because these algorithms can be used confidently on large data sets where small samples may be required to reach a solution quickly. In addition, creating ensembles with the EAs results in higher accuracy than single trees produced by existing methods. In some cases, the cost of generating an ensemble with EAs using sampling may be lower than generating a single tree of similar accuracy with a traditional algorithm.

This paper is only a first step in the application of EAs to oblique DT induction, and there are multiple opportunities to expand our work. In particular, we plan to continue the study of scalability using larger data sets (both artificial and “real-world”), and experiment with other algorithms such as

(μ, λ) -ES. Also, we plan to exploit the knowledge about the tree induction problem by designing specialized operators and by combining EAs with local hillclimbers.

APPENDIX

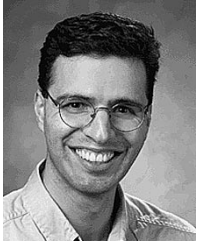
Table VIII shows the results of experiments summarized in the paper as graphs in Fig. 5.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers who provided them with detailed comments and useful suggestions.

REFERENCES

- [1] T. Bäck, *Evolutionary Algorithms in Theory and Practice*. New York: Oxford Univ. Press, 1996.
- [2] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Mach. Learn.*, vol. 36, no. 1/2, pp. 105–139, 1999.
- [3] K. P. Bennett, N. Cristianini, J. Shawe-Taylor, and D. Wu, "Enlarging the margins in perceptron decision trees," *Mach. Learn.*, vol. 41, no. 3, pp. 295–313, 2000.
- [4] C. L. Blake and C. J. Merz. (1998) UCI repository of machine learning databases. [Online]. Available: <http://www.ics.uci.edu/~mllearn/ML-Repository.html>.
- [5] M. C. J. Bot, "Improving induction of linear classification trees with genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference 2000*, D. Whitley, D. E. Goldberg, E. Cantú-Paz, L. Spector, L. Parmee, and H.-G. Beyer, Eds. San Francisco, CA: Morgan Kaufmann, 2000, pp. 403–410.
- [6] M. C. J. Bot and W. B. Langdon, "Application of genetic programming to induction of linear classification trees," in *Genetic Programming: Third European Conference*, R. Poli, W. Banzhaf, W. B. Langdon, J. Miller, P. Nordin, and T. C. Fogarty, Eds. Berlin, Germany: Springer-Verlag, 2000, pp. 247–258.
- [7] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 26, no. 2, pp. 123–140, 1996.
- [8] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Pacific Grove, CA: Wadsworth and Brooks, 1984.
- [9] C. E. Brodley and P. E. Utgoff, "Multivariate decision trees," *Mach. Learn.*, vol. 19, pp. 45–77, 1995.
- [10] D. E. Brown, C. L. Pittard, and H. Park, "Classification trees with optimal multivariate decision nodes," *Pattern Recognit. Lett.*, vol. 17, pp. 699–703, 1996.
- [11] E. Cantú-Paz, *Efficient and Accurate Parallel Genetic Algorithms*. Boston, MA: Kluwer, 2000.
- [12] E. Cantú-Paz and C. Kamath, "Using evolutionary algorithms to induce oblique decision trees," in *Proceedings of the Genetic and Evolutionary Computation Conference 2000*, D. Whitley, D. E. Goldberg, E. Cantú-Paz, L. Spector, L. Parmee, and H.-G. Beyer, Eds. San Francisco, CA: Morgan Kaufmann, 2000, pp. 1053–1060.
- [13] T. G. Dietterich, "Approximate statistical tests for comparing supervised classification learning algorithms," *Neur. Computat.*, vol. 10, no. 7, pp. 1895–1924, 1998.
- [14] —, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," *Mach. Learn.*, vol. 40, no. 2, pp. 139–158, 2000.
- [15] G. Folino, C. Pizzuti, and G. Spezzano, "A cellular genetic programming approach to classification," in *Proceedings of the Genetic and Evolutionary Computation Conference 1999: Volume 2*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds. San Francisco, CA: Morgan Kaufmann, 1999, pp. 1015–1020.
- [16] —, "Genetic programming and simulated annealing: A hybrid method to evolve decision trees," in *Genetic Programming: Third European Conference*, R. Poli, W. Banzhaf, W. B. Langdon, J. Miller, P. Nordin, and T. C. Fogarty, Eds. Berlin, Germany: Springer-Verlag, 2000, pp. 294–303.
- [17] J. H. Friedman, "A recursive partitioning decision rule for nonparametric classification," *IEEE Trans. Comput.*, vol. C-26, pp. 404–408, 1977.
- [18] J. Gama, "Oblique linear tree," in *Second International Symposium on Advances in Intelligent Data Analysis*, X. Liu and P. Cohen, Eds. Berlin, Germany: Springer-Verlag, 1997, pp. 187–198.
- [19] D. E. Goldberg and S. Voessner, "Optimizing global-local search hybrids," in *Proceedings of the Genetic and Evolutionary Computation Conference 1999: Volume 1*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds. San Francisco, CA: Morgan Kaufmann, 1999, pp. 220–228.
- [20] J. J. Grefenstette and J. M. Fitzpatrick, "Genetic search with approximate function evaluations," in *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, J. J. Grefenstette, Ed. Hillsdale, NJ: Lawrence Erlbaum, 1985, pp. 112–120.
- [21] L. Hansen and P. Salamon, "Neural network ensembles," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, pp. 993–1001, 1990.
- [22] G. Harik, E. Cantú-Paz, D. E. Goldberg, and B. L. Miller, "The gambler's ruin problem, genetic algorithms, and the sizing of populations," *Evolut. Computat.*, vol. 7, no. 3, pp. 231–253, 1999.
- [23] D. Heath, S. Kasif, and S. Salzberg, "Induction of oblique decision trees," in *Proceedings of the 13th International Joint Conference on Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann, 1993, pp. 1002–1007.
- [24] E. G. Henrichon and K. Fu, "A nonparametric partitioning procedure for pattern classification," *IEEE Trans. Comput.*, vol. C-18, pp. 614–624, 1969.
- [25] G. E. Hinton and S. J. Nowlan, "How learning can guide evolution," *Complex Syst.*, vol. 1, pp. 495–502, 1987.
- [26] V. S. Iyengar, "HOT: Heuristics for oblique trees," in *Eleventh International Conference on Tools with Artificial Intelligence*. New York: IEEE Press, 1999, pp. 91–98.
- [27] J. R. Koza, "Concept formation and decision tree induction using the genetic programming paradigm," in *Parallel Problem Solving from Nature*, H.-P. Schwefel and R. Männer, Eds. Berlin, Germany: Springer-Verlag, 1991, pp. 124–128.
- [28] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: The MIT Press, 1992.
- [29] W.-Y. Loh and N. Vanichsetakul, "Tree-structured classification via generalized discriminant analysis," *J. Amer. Statist. Assoc.*, vol. 83, no. 403, pp. 715–728, 1988.
- [30] B. L. Miller and D. E. Goldberg, "Genetic algorithms, selection schemes, and the varying effects of noise," *Evolut. Computat.*, vol. 4, no. 2, pp. 113–131, 1996.
- [31] H. Mühlenbein and D. Schlierkamp-Voosen, "Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization," *Evolutionary Computation*, vol. 1, no. 1, pp. 25–49, 1993.
- [32] S. K. Murthy, "On growing better decision trees from data," Ph.D. dissertation, Univ. of Maryland, College Park, 1997.
- [33] S. K. Murthy, S. Kasif, and S. Salzberg, "A system for induction of oblique decision trees," *J. Artif. Intell. Res.*, vol. 2, no. 1, pp. 1–32, 1994.
- [34] N. I. Nikolaev and V. Slavov, "Inductive genetic programming with decision trees," *Intell. Data Anal.*, vol. 2, no. 1, pp. 31–44, 1998.
- [35] D. Opitz and R. Maclin, "Popular ensemble methods: An empirical study," *J. Artif. Intell. Res.*, vol. 11, pp. 169–198, 1999.
- [36] D. J. Powell, S. S. Tong, and M. M. Skolnick, "EnGENEous domain independent, machine learning for design optimization," in *Proceedings of the Third International Conference on Genetic Algorithms*, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 151–159.
- [37] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, pp. 81–106, 1986.
- [38] S. Salzberg, "Locating protein coding regions in human DNA using a decision tree algorithm," *J. Computat. Biol.*, vol. 2, no. 3, pp. 473–485, 1995.
- [39] S. Salzberg, R. Chandar, H. Ford, S. K. Murthy, and R. White, "Decision trees for automatic classification of cosmic rays in Hubble Space Telescope images," *Publicat. Astronom. Soc. Pacific*, vol. 107, pp. 1–10, 1995.
- [40] S. Shah and P. S. Sastry, "New algorithms for learning and pruning oblique decision trees," *IEEE Trans. Syst., Man, Cybern. C*, vol. 29, no. 4, pp. 494–505, 1999.
- [41] P. E. Utgoff and C. E. Brodley, "Linear machine decision trees," Department of Computer Science, Univ. of Massachusetts, Amherst, MA, Tech. Rep. COINS 91-10, Jan. 1991.



Erick Cantú-Paz (S'99–M'99) received the B.S. degree in computer engineering from the Instituto Tecnológico Autónomo de México in 1994 and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 1999.

He is a Computer Scientist at Lawrence Livermore National Laboratory, Livermore, CA, where he conducts research on scalable data mining of scientific data. He is author of the book *Efficient and Accurate Parallel Genetic Algorithms* (Boston, MA: Kluwer, 2000) and over 30 peer-reviewed publications. He is

an Associate Editor for the *Journal of Heuristics* and a member of the Editorial Board of *Computational Optimization and Applications*. His research interests include theoretical foundations and practical applications of evolutionary algorithms, machine learning, and data mining.

Dr. Cantú-Paz served as Chairman of the Genetic and Evolutionary Computation Conference (GECCO) in 2002 and as Editor-in-Chief in 2003. He is a Member of ACM and of the International Society of Genetic and Evolutionary Computation.



Chandrika Kamath (S'84–M'03) received the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 1986.

She is a Computer Scientist at the Center for Applied Scientific Computing, Lawrence Livermore National Laboratory (LLNL), Livermore, CA. Prior to joining LLNL in 1997, she was a Consulting Software Engineer at Digital Equipment Corporation, developing high-performance mathematical software. Since January 1998, she has been the project lead and an individual contributor for

Sapphire, a project in large-scale data mining at LLNL. Her research interests are in large-scale data mining and pattern recognition, including image processing, feature extraction, dimension reduction, and classification and clustering algorithms. She is also interested in the practical application of these techniques.