Accepted for publication in the Genetic and Evolutionary Computation Conference (GECCO-2000)

# Using Evolutionary Algorithms to Induce Oblique Decision Trees

**Erick Cantú-Paz and Chandrika Kamath**
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94550
cantupaz,kamath2@llnl.gov

## Abstract

This paper illustrates the application of evolutionary algorithms (EAs) to the problem of oblique decision tree 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. Experiments were performed with a (1+1) evolution strategy and a simple genetic algorithm on public domain and artificial data sets. 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.

## 1 INTRODUCTION

Decision trees (DTs) are popular classification methods, and there are numerous algorithms to induce a tree classifier from a given set of data (Murthy, 1997). 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, and 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 accurate trees. However, these trees are not as popular as the axis-parallel trees because the tests are harder to interpret, and the problem of finding oblique hyperplanes is more difficult than finding axis-parallel partitions, requiring greater computational effort.

The purpose of this paper is to illustrate the application of EAs to the task of oblique decision tree induction. The objectives are to show that, in some domains, evolutionary optimization may result in classifiers whose accuracy is competitive with other oblique tree construction methods, and that this can be accomplished in shorter time, at least in some cases. The results of our experimental study suggest that the EA-augmented inducers can quickly 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.

The paper is organized as follows. The next section provides a brief background on oblique decision trees and a brief review of relevant previous work. Section 3 describes some of the advantages of using EAs to find splits in oblique DTs, and describes our approach to this problem. Section 4 has experimental results that illustrate the advantages of the evolutionary approach using public domain and artificial data sets. Finally, section 5 has a brief summary and the conclusions of the paper.

## 2 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 $(x_1, x_2, ..., x_d, c_j)$, where the $x_i$ are real-valued attributes, and the $c_j$ is a discrete value that represents the class label of the instance. As was 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. The task of the

inducer is to find appropriate values for $i$ and $k$. 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. It has been shown that finding the best oblique tree is NP-complete (Heath, Kasif, & Salzberg, 1993), and therefore existing oblique DT inducers use some sort of greedy search to find values for the coefficients. To evaluate the candidate hyperplanes, these search algorithms use a heuristic measure of the impurity of the split. Many impurity measures have been proposed (Murthy, 1997) such as Quinlan's (1986) information gain, the Gini index, or the twoing rule (Breiman et al., 1984). Each impurity measure has its own advantages and disadvantages, and each defines a different optimization problem.

Breiman, Friedman, Olshen, and Stone (1984) introduced the first algorithm to induce oblique DTs: CART with linear combinations. At each node of the tree, the algorithm 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 constant.

Murthy, Kasif, and Salzberg (1994) introduced OC1, which uses an ad-hoc combination of hillclimbing and randomization. As in CART, the hillclimber finds locally optimal values for one coefficient at a time, although OC1 offers several variants 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 extension of CART with linear combinations that overcomes some of its limitations. In particular, they claim that CART's deterministic nature 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, the time used at each node in the tree is bounded. They present experimental results that suggest that OC1 outperforms CART in several domains.

Heath, Kasif, and Salzberg (1993) used simulated annealing to perturb the hyperplane's coefficients. Simulated annealing is a more sophisticated optimizer than those used in CART and OC1, and in some domains it can produce small and highly accurate trees. However, simulated annealing converges very slowly, and the DT inducer has to examine a large number of hyperplanes, making it inadequate for large data sets.

Other related work in this area includes the Linear Machine Decision Trees (LMDT) system (Utgoff & Brodley, 1991; Brodley & Utgoff, 1995). 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 $i \in \{1, ..., R\}$ to the instance described by $\mathbf{X} = (x_1, ..., x_d)$ 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. There is no need to use a heuristic to measure the impurity of each split or to decide when to stop. EAs could also be used in combination with LMDT to adapt the weight vectors, but we do not address LMDT in this paper.

## 3 EVOLUTIONARY OBLIQUE DTs

At the heart of all DT inducing algorithms there is an optimization task: to minimize the impurity of the split defined by a hyperplane. This task is performed at each node of the tree, then the data is partitioned into subsets, and the algorithm is applied recursively to each subset. Quinlan (1986) described this procedure as 'top-down' induction of decision trees. Our proposal is to use EAs as the optimization algorithm.

Evolutionary algorithms are a promising alternative to existing oblique tree algorithms for several reasons:

**More sophisticated optimizers.** EAs are not limited to considering one coefficient at a time (unlike CART and OC1), and it is likely that EAs find better splits than the simple greedy hillclimbers that are currently in use.

**No need for optimal splits.** Finding the best split at each node does not guarantee that the best tree will be found. Therefore, there is no need to run the EAs (or any other optimizer, for that matter) until they find the best solution that they can. It is well known that EAs quickly improve 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 properties (Mühlenbein & Schlierkamp-Voosen, 1993; Harik et al., 1999).

**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 hyperplanes. The positive experiences with existing DT inducers suggest that new hyperplanes that are only slight variations of the originals may work well. This can be accomplished by restricting recombination between similar hyperplanes or by using small mutation steps, for example. In addition, the execution time may be reduced using known 'good' solutions to seed the initial population.

**Hybridization.** Most DT algorithms use a local optimizer that is well tuned to the tree induction task, and interfacing it to the EA could boost 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 (Grefenstette & Fitzpatrick, 1985; Miller & Goldberg, 1996a).

**Parallel implementations.** It is straightforward to implement EAs on parallel computers (see e.g., (Cantú-Paz, 1998)), and the expected performance improvements are very promising.

This paper does not consider hybrids or parallel implementations, but we use knowledge about the problem in our choice of encoding and operators and to seed the initial population. The EAs were run for a fixed number of iterations that, in many cases, were not enough for the EA to converge to a unique solution or to find the best hyperplane that it could, but that were sufficient to reach acceptable solutions. In addition, we performed experiments to explore the scalability of EAs and their sensitivity to sampling.

## 4 EXPERIMENTS

To demonstrate the feasibility of using EAs to search for oblique partitions, we conducted three sets of experiments. In the first set, we used the same four public-domain data sets from the UCI repository that Murthy et al. (1994) used to evaluate OC1. 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. Finally, we present experiments with a larger database to illustrate how sampling may help to scale up the evolutionary approach to more realistic situations.

The experiments compare the performance of three baseline DT inducers against two inducers that use EAs. The first baseline DT system is OC1 with its default parameters; the second is OC1 limited to axis-parallel partitions, which we call OC1-AP; and the third is Murthy et al.'s implementation of CART-LC, which we call OC1-CART.

The first DT system with an EA is an extension of OC1 that uses a (1+1) evolution strategy with self-adaptive mutations. We call this OC1-ES. The candidate hyperplane is represented as a vector of real-valued coefficients, $a_1, ..., a_{d+1}$. The initial hyperplane is the best axis-parallel split found by OC1. For each hyperplane coefficient there is a corresponding mutation coefficient $\sigma_1, ..., \sigma_{d+1}$, which are initially set to 1. At each iteration, the mutation coefficients are updated and a new hyperplane is obtained according to the following rule:

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

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.

The second extension of OC1 with an EA uses a simple generational GA with real-valued genes, and is called OC1-GA. For the experiments, the GA used pairwise tournament selection without replacement, uniform crossover with probability 1.0, and no mutation. The population size was set to $20\sqrt{d}$, along the lines of a population-sizing theory that proposes that the population size required to reach a solution of a particular quality is $O(\sqrt{d})$ (Harik et al., 1999). The best axis-parallel hyperplane was copied to 10% of the 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.

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.

All experiments measure the impurity of a split at each tree node using the twoing rule (Breiman et al., 1984),

| Name | Task Description | Attributes | No. of Instances |
|---|---|---|---|
| Cancer | Diagnose a tumor as benign or malignant | 9 | 683 |
| Diabetes | Detect presence of diabetes | 8 | 768 |
| Housing | Predict housing values in suburbs of Boston | 12 | 506 |
| Iris | Classify type of iris | 4 | 150 |

Table 1: Descriptions of the small public domain data sets used in the experiments.

| Algorithm | Parameter | Cancer | Diabetes | Housing | Iris |
|---|---|---|---|---|---|
| OC1 | Accuracy | 96.2 (1.0) | 74.1 (2.0) | 82.8 (2.0) | 95.5 (1.8) |
| | Leaves | 3.3 (1.1) | 5.7 (2.1) | 7.3 (2.6) | 3.5 (0.2) |
| | Time | 28.4 (8.7) | 33.0 (1.4) | 19.8 (1.3) | 1.2 (0.1) |
| OC1-AP | Accuracy | 94.7 (0.7) | 74.0 (1.0) | 82.2 (1.0) | 92.8 (2.6) |
| | Leaves | 9.4 (2.6) | 18.9 (8.7) | 10.0 (6.6) | 5.2 (1.4) |
| | Time | 0.2 (0.0) | 0.4 (0.0) | 0.3 (0.0) | 0.1 (0.0) |
| OC1-CART | Accuracy | 95.9 (0.5) | 72.7 (1.8) | 82.3 (1.5) | 94.2 (1.8) |
| | Leaves | 5.5 (2.5) | 13.7 (7.8) | 11.5 (2.8) | 4.2 (0.7) |
| | Time | 1.2 (0.2) | 2.7 (0.1) | 2.1 (0.1) | 0.1 (0.1) |
| OC1-ES | Accuracy | 95.2 (0.9) | 73.7 (1.4) | 82.8 (1.2) | 96.3 (1.5) |
| | Leaves | 5.2 (2.2) | 17.1 (5.0) | 11.5 (5.7) | 3.5 (0.4) |
| | Time | 5.1 (0.4) | 14.0 (0.3) | 8.6 (0.3) | 0.9 (.1) |
| OC1-GA | Accuracy | 94.3 (0.5) | 73.9 (1.3) | 82.4 (1.1) | 93.6 (1.3) |
| | Leaves | 9.6 (2.1) | 19.0 (11.6) | 12.5 (5.2) | 4.3 (1.4) |
| | Time | 7.7 (0.4) | 13.0 (0.3) | 8.5 (0.4) | 0.37 (0.04) |

Table 2: Comparison of different algorithms on the small public domain data sets.

which is the default in OC1:

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

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 each node; $L_i$ and $R_i$ are the number of examples of category $i$ on the left and right of the split. The impurity was used without modification as the fitness of the hyperplanes.

### 4.1 SMALL DATA SETS

The first round of experiments use small public domain data sets, which are available at UCI's machine learning repository (Blake & Merz, 1998). These are briefly described in Table 1, and have been used in numerous studies of machine learning and data mining algorithms. For our comparison we follow the experimental procedure that Murthy et al. (1994) used to compare OC1 to other DT inducers: we use the standard parameters of OC1, and the results presented (in Table 2) are the average of ten five-fold cross-validation experiments (50 trees total). We report 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 their standard deviations (in parenthesis).

From the table it is clear that for a given dataset the differences in the accuracy of the algorithms is very small. There are statistically significant differences (at least at the 0.05 confidence level) on the cancer and iris data, but the magnitude of the differences is still small. For the four data sets, OC1 found the smallest trees, but in three cases (cancer, housing, and iris) OC1-ES and OC1-CART found trees comparable to OC1. The average size of the trees found by the GA-augmented inducer was close to the axis-parallel algorithm. The largest differences are in execution times; the EAs being on average approximately 3 times faster than OC1, but much slower than OC1-AP and OC1-CART.

### 4.2 ARTIFICIAL DATA

The next set of experiments used three artificial data sets. The purpose of these experiments is to ensure that the concept to be learned matches the bias of the algorithms—the classes are separable by oblique hyperplanes. In addition, we performed experiments to explore the scalability of the algorithms as the number of attributes varies. The three data sets were also used by Murthy et al. (1994) in their evaluation of OC1, but we used them 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 is separable by the hyperplane $x_1 + ... + x_{d/2} < x_{d/2+1} + ... + x_d$, where $d \in \{10, 20, 50\}$. These data sets are labeled LS10, LS20, and LS50 according to their dimensionality.

We followed the same experimental procedure as in the previous experiments, and the results are summarized in Table 3[1]. In this case, OC1-AP consistently found the least accurate and largest trees. Of course, it was the fastest algorithm, but its accuracy is too low to consider AP trees competitive (consider that random guessing would result in a 50% accuracy and the accuracy of OC1-AP on LS50 is 58%). OC1 produces the most accurate trees for LS10, but as the number of dimensions increases its performance seems to drop below the EA-augmented inducers. OC1-CART does a little worse. OC1-GA maintains the highest accuracy, but its execution time seems to increase faster than OC1-ES. In any case, both of the EA inducers are faster than OC1 (approximately between 2x and 6x), and appear to be more robust to the increase in dimensionality. The size of the trees found by OC1, OC1-CART, and OC1-ES increases with the number of dimensions, but those of OC1-GA seem to remain of a constant size. However, consider that the ideal tree for this domain has two leaves, and all the algorithms find much larger trees.

The second and third artificial data sets, POL2 and RCB2, represent concepts that are supposed to be more difficult to learn than the LS problems. POL2 and RCB2 are defined in 2 dimensions ($x_1, x_2 \in [0, 1]$), and depicted in Figure 1. 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)

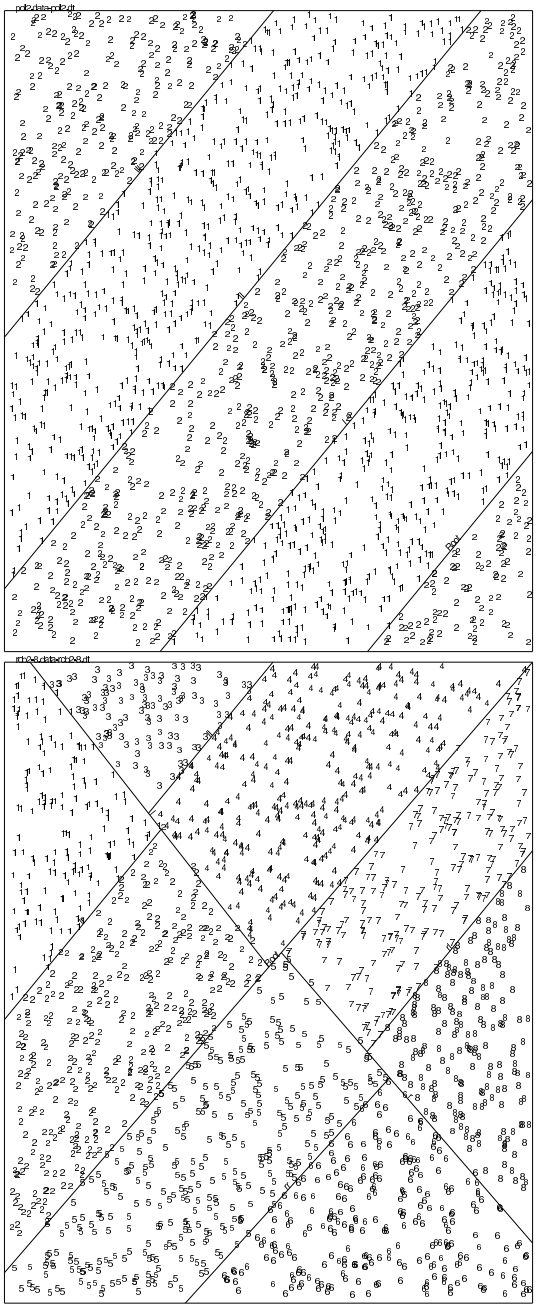

Figure 1: The POL2 and RCB2 data sets.

[1]Our 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.

| Algorithm | Parameter | LS10 | LS20 | LS50 | POL2 | RCB2 |
|---|---|---|---|---|---|---|
| OC1 | Accuracy | 97.1 (0.4) | 88.5 (1.1) | 72.5 (1.3) | 99.6 (0.1) | 99.0 (0.1) |
| | Leaves | 5.3 (2.2) | 5.9 (2.7) | 10.0 (3.6) | 5.0 (0.0) | 8.4 (0.3) |
| | Time | 170.9 (12) | 391.5 (16.6) | 608.7 (32.8) | 36.0 (2.3) | 44.8 (1.5) |
| OC1-AP | Accuracy | 73.0 (1.5) | 64.6 (0.8) | 58.6 (1.0) | 94.2 (0.6) | 92.8 (0.4) |
| | Leaves | 86.7 (16.5) | 71.5 (29.0) | 58.0 (20.8) | 77.7 (10.4) | 85.9 (6.8) |
| | Time | 1.6 (0.0) | 3.5 (0.1) | 11.7 (0.6) | 0.3 (0.0) | 0.4 (0.0) |
| OC1-CART | Accuracy | 96.0 (1.5) | 87.3 (1.9) | 66.3 (1.0) | 97.6 (0.5) | 94.4 (0.3) |
| | Leaves | 5.9 (3.5) | 9.3 (3.6) | 25.0 (17.7) | 14.4 (2.9) | 50.6 (7.1) |
| | Time | 16.8 (1.3) | 54.9 (3.6) | 113.9 (3.6) | 2.7 (0.2) | 3.4 (0.1) |
| OC1-ES | Accuracy | 93.7 (0.8) | 87.0 (1.0) | 78.5 (1.6) | 99.4 (0.3) | 98.1 (0.3) |
| | Leaves | 9.9 (2.8) | 14.4 (5.6) | 16.3 (9.4) | 6.3 (1.2) | 10.9 (1.9) |
| | Time | 29.8 (2.4) | 65.1 (3.3) | 163.9 (14.9) | 4.5 (0.4) | 6.0 (0.4) |
| OC1-GA | Accuracy | 95.4 (0.6) | 92.0 (0.7) | 85.2 (1.0) | 95.3 (0.4) | 93.8 (0.7) |
| | Leaves | 8.8 (3.8) | 9.8 (5.9) | 9.5 (5.6) | 57.5 (10.5) | 64.6 (9.7) |
| | Time | 36.3 (3.8) | 101.5 (4.8) | 333.3 (22.2) | 4.7 (0.3) | 5.0 (0.2) |

Table 3: Comparison of different algorithms on the artificial data sets.

data also has 2000 instances, but in this case they are divided into eight classes. We used the same experimental setup as before, and the results are in Table 3.

In these two domains, OC1 and OC1-ES 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 AP trees are the largest and least accurate, but OC1-GA found only slightly more accurate and smaller trees. The fastest oblique DT algorithm was OC1-CART, 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 the ES was much better than the GA.

### 4.3 OPTICAL DIGIT RECOGNITION DATA

To study the problem of scalability to larger data sets, we experimented with the optical digit recognition data set, which is also available at UCI's ML repository. This data set has 3823 instances in a training set and 1797 in a testing set; each instance is described by 64 numeric attributes. The objective is to identify the instances as one of 10 digits.

With this domain, 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 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. 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 is available for training, which is often the case in the scientific domains that interest us. The second way of sampling is to choose a fraction of the training instances every time that a hyperplane is evaluated. This dynamic sampling method is slightly more expensive than sampling statically once per experiment, but it may be advantageous especially when samples are small, because numerous hyperplanes are evaluated in every tree node and the sampling will eventually consider all the available labeled instances.

Evaluating the hyperplanes with dynamic samples also means that every time that a particular hyperplane is evaluated its fitness estimate is different. 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 could be used to determine the optimal number of repetitive evaluations that would minimize the execution time of the GA (Miller & Goldberg, 1996b). As a first cut, however, we decided to use a single evaluation as a crude (but fast) estimate of fitness.

The results with dynamic sampling are reported in Table 4. In this case, we report the average of 10 experiments, and training and testing used the partition of the instances as in the UCI repository. The algo-

| Algorithm | Parameter | 5% | 10% | 25% | 50% | 100% |
|---|---|---|---|---|---|---|
| OC1 | Accuracy | 37.9 (4.4) | 50.2 (2.6) | 69.6 (2.0) | 81.2 (1.7) | 86.4 (0.9) |
| | Leaves | 72.3 (19.5) | 101.8 (22.8) | 155.4 (90) | 182.1 (83.1) | 53.7 (30.6) |
| | Time | 8.1 (0.3) | 16.2 (0.5) | 52.1 (2.9) | 126.6 (4.0) | 298.6 (11.1) |
| OC1-AP | Accuracy | 71.8 (1.1) | 76.9 (2.3) | 81.2 (1.6) | 83.0 (1.2) | 84.5 (1.9) |
| | Leaves | 32.0 (5.6) | 49.0 (6.4) | 77.6 (11.5) | 112.1 (40.7) | 125.8 (48.2) |
| | Time | 0.7 (0.0) | 0.9 (0.0) | 1.7 (0.0) | 2.9 (0.1) | 5.5 (0.2) |
| OC1-CART | Accuracy | 28.3 (4.4) | 36.9 (6.9) | 62.3 (4.5) | 75.1 (2.7) | 88.2 (0.7) |
| | Leaves | 61 (56.6) | 158 (64.3) | 179 (116) | 163 (96.8) | 60.6 (25.8) |
| | Time | 7.3 (0.6) | 11.8 (1.1) | 26.7 (3.2) | 62.1 (9.0) | 77.4 (10.7) |
| OC1-ES | Accuracy | 71.1 (2.4) | 77.5 (2.3) | 82.9 (1.9) | 84.7 (1.3) | 87.9 (1.0) |
| | Leaves | 19.1 (3.8) | 26.6 (7.2) | 43.4 (13.0) | 84.0 (28.4) | 84.0 (37.6) |
| | Time | 5.8 (0.4) | 8.7 (0.6) | 17.6 (0.4) | 32.7 (1.4) | 63.0 (3.2) |
| OC1-GA | Accuracy | 78.1 (2.0) | 82.7 (1.4) | 87.2 (0.9) | 88.6 (1.1) | 90.2 (1.1) |
| | Leaves | 14.7 (4.2) | 20.0 (5.5) | 33.0 (9.2) | 31.9 (12.0) | 52.3 (34.6) |
| | Time | 8.4 (0.5) | 15.2 (0.5) | 37.1 (0.9) | 75.6 (2.9) | 144.2 (4.5) |

Table 4: Comparison of different algorithms on the digit recognition data sampling dynamically (5%-100% of the training set) every time that a hyperplane was evaluated.

rithms use the same parameters as before. Sampling decreases the execution time as desired, but it also affects the accuracy. For each sample size, the GA finds the smallest and most accurate classifiers, and in most cases it is faster than the original oblique OC1. The ES is the fastest of the oblique classifiers, and its accuracy is better than OC1 and CART, but not as good as the GA. Note, however, that the axis-parallel OC1 is the fastest algorithm, and that its accuracy is similar to OC1-ES. In fact, using OC1-AP with the entire data set is faster and more accurate than the GA on 5% samples, so if the end user does not care about the relatively small differences in accuracy, in this domain axis-parallel DTs would be a good choice. If accuracy or tree size is a premium, then OC1-GA would be the best option.

In separate experiments we found that dynamic sampling gives 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 ($\approx 4-5\%$), 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.

## 5 SUMMARY AND CONCLUSIONS

Traditional DT inducers use some form of heuristic greedy search to find appropriate splits. In this paper, we substitute the greedy search with two evolutionary algorithms: a (1+1) evolution strategy and a simple GA. We performed experiments on public domain and artificial data sets with different characteristics to evaluate the performance of the EA-based tree inducers. The results suggest that EAs are capable of finding oblique trees with similar accuracy to OC1, 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.

We evaluated the use of sampling to further reduce the execution time of the inducers. Sampling resulted in faster training times, but also in a loss of accuracy, which was more pronounced when the training set was sampled statically.

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 should continue the study of scalability using larger data sets (both artificial and 'real-world'), and incorporate other algorithms such as $(\mu, \lambda)$-ES. Also, the knowledge about the tree induction problem should be exploited by designing specialized operators and by combining EAs with local hillclimbers.

## Acknowledgments

UCRL-JC-137202. This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract no. W-7405-Eng-48.

## References


Blake, C., & Merz, C. (1998). UCI repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees*. Pacific Grove, CA: Wadsworth & Brooks Advanced Books and Software.

Brodley, C. E., & Utgoff, P. E. (1995). Multivariate decision trees. *Machine Learning*, *19*, 45–77.

Cantú-Paz, E. (1998). A survey of parallel genetic algorithms. *Calculateurs Parallèles, Reseaux et Systems Repartis*, *10*(2), 141–171.

Grefenstette, J. J., & Fitzpatrick, J. M. (1985). Genetic search with approximate function evaluations. In Grefenstette, J. J. (Ed.), *Proceedings of an International Conference on Genetic Algorithms and Their Applications* (pp. 112–120). Hillsdale, NJ: Lawrence Erlbaum Associates.

Harik, G., Cantú-Paz, E., Goldberg, D. E., & Miller, B. L. (1999). The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation, 7*(3), 231–254.

Heath, D., Kasif, S., & Salzberg, S. (1993). Induction of oblique decision trees. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence* (pp. 1002–1007). San Mateo, CA: Morgan Kaufmann.

Miller, B. L., & Goldberg, D. E. (1996a). Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, *4*(2), 113–131.

Miller, B. L., & Goldberg, D. E. (1996b). Optimal sampling for genetic algorithms. In Dagli, C. H., Akay, M., Chen, C. L. P., Fernández, B. R., & Ghosh, J. (Eds.), *Proceedings of the Artificial Neural Networks in Engineering (ANNIE '96) conference*, Volume 6 (pp. 291–297). New York: ASME Press.

Mühlenbein, H., & Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization. *Evolutionary Computation*, *1*(1), 25–49.

Murthy, S. K. (1997). *On growing better decision trees from data*. Doctoral dissertation, University of Maryland.

Murthy, S. K., Kasif, S., & Salzberg, S. (1994). A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, *2*(1), 1–32.

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

Utgoff, P. E., & Brodley, C. E. (1991, January). *Linear machine decision trees* (Technical Report COINS 91-10). Amherst, MA: Department of Computer Science, University of Massachusetts.