Improving Induction of Linear Classification Trees with Genetic Programming

Martijn C.J. Bot
Vrije Universiteit
De Boelelaan 1081
1081 HV Amsterdam
+31 20-444790
mbot@cs.vu.nl

William B. Langdon
CWI
Kruislaan 413
1098 SJ Amsterdam
W.B.Langdon@cwi.nl

Abstract

A common problem in datamining is to find accurate classifiers for a dataset. For this purpose, genetic programming (GP) is applied to a set of benchmark classification problems. Using GP, we are able to induce decision trees with a linear combination of variables in each function node. A new representation of decision trees using Strong Typing in GP was introduced in [Bot and Langdon, 2000]. The effects of techniques such as limited error fitness, fitness sharing Pareto scoring and domination Pareto scoring are evaluated. Results indicate that GP can be applied successfully to classification problems. Comparisons with current state-of-the-art algorithms in machine learning are presented and areas of future research are identified.

1 INTRODUCTION

Classification problems form an important area in datamining. For example, a bank may want to classify its clients in good and bad credit risks or a doctor may want to classify his patients as having diabetes or not. Classifiers may take the form of decision trees [Murthy, 1998] (see Figure 1). In each node, a test is made in which one or more variables is used. Depending on the outcome of the test, the tree is traversed to the left or the right subtree (see Section 2.1). In our decision trees, the tests are linear combinations of some of the variables. This allows classification of continuous and integer valued datasets with an (unknown) inherent linear structure. An optimal tree is one which makes as few misclassifications as possible on the validation set.

In [Bot and Langdon, 2000], a new representation for decision trees in GP using Strong Typing was introduced. In this paper, three modifications to that system are evaluated. Limited Error Fitness (LEF) is a technique for maintaining more population flux over evolution and reducing runtime (see Section 2.3). Pareto scoring with fitness sharing and Pareto scoring with domination based ranking are techniques for creating and maintaining a diverse population (see Section 2.5). In our system, a population diverse in tree size and accuracy is created. If a population isn't very diverse, then most trees are similar. A disadvantage of this is that really new trees can only be found my mutation, which is a quite slow process. The reason for this is that mutations are random, so most mutations won't be beneficiary.

The classification accuracy of the GP is compared to that achieved by several other decision tree classification techniques, such as the OC1-algorithm, C5.0 and the M5' algorithm (Section 5.2).

In Section 2 the theoretical background behind the system is given. A short introduction to decision trees is given and the representation of decision trees in GP that was used in [Bot and Langdon, 2000], LEF and Pareto scoring are described. Section 3 explains the experimental setup for the experiments in Section 4. In Section 5 an analysis is made of the performance of the GP-system and a comparison is made to other decision tree algorithms. Section 6 contains our conclusions.

2 BACKGROUND

2.1 DECISION TREES

Decision trees [Murthy, 1998] are a well known technique for representing the underlying structure of a dataset.

An **axis-parallel** decision tree is one in which each

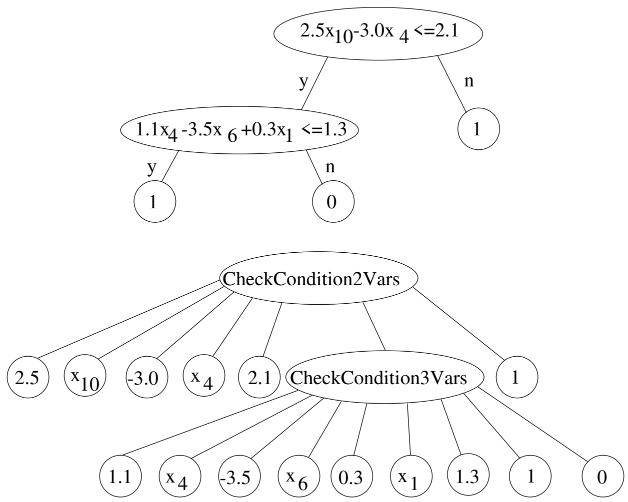

Figure 1: Example decision tree and its representation in the GP. x_{10} means the tenth variable from the dataset. Each function node's first children are the weights and variables for the linear combination. The last two children are other function nodes or classifications. When evaluating the CheckCondition2Vars node on a certain case, if $2.5\mathrm{x}_{10} + -3.0\mathrm{x}_4 \leq 2.1$, the CheckCondition3Vars node is evaluated; otherwise the final classification is 1 and the evaluation of the decision tree on this particular case is finished.

node only contains one variable. All hyperplanes are parallel to the axes, hence the name. See Figure 2 for an example. A decision tree is **oblique** when the nodes contain one or more variables. Now the hyperplanes are not necessarily parallel to the axes, but can have any orientation in the attribute space. See Figure 3 for an example. Note that axis-parallel trees are special cases of oblique trees. Clearly, oblique trees are more general than axis-parallel trees. There exist many domains in which oblique hyperplanes will form the best classification model. For example, any domain in which the (weighed) sum of two variables is crucial for correct classification needs an oblique hyperplane.

Because of this larger generality however, the search space is also much larger: there are many more possible oblique models than axis-parallel models.

When evaluating a decision tree on an individual case from the database, in each non-leaf node, a linear combination is made of some of the available variables. Each function node has a tuple ($\{c_i, x_i\}$, threshold, ifTrue, ifFalse) as its children with c_i and x_i the ith constant and the ith variable. The node is evaluated as follows:

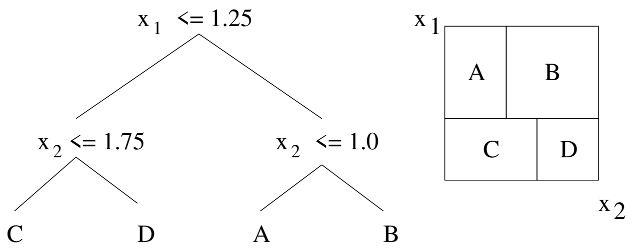

Figure 2: The left side shows a simple axis-parallel decision tree. The right side shows the partitioning that this tree creates in the two-dimensional attribute space.

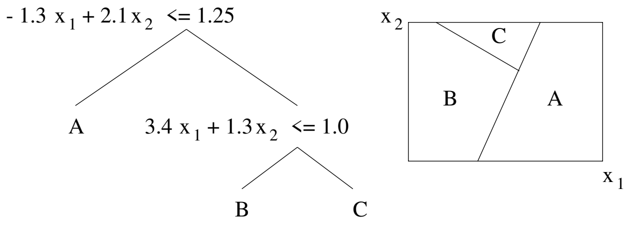

Figure 3: The left side shows an oblique decision tree. The right side shows the partitioning that this tree creates in the two-dimensional attribute space.

if $\sum_i c_i x_i$ <=threshold	evaluate ifTrue branch
else	evaluate ifFalse branch

When finally arrived at a leaf node, a classification is returned for the case.

The two rightmost children are either direct classifications or other function nodes.

2.2 STRONG TYPING

In order to ensure that only valid individuals are constructed in the GP, strong typing was used [Montana, 1995, Bot and Langdon, 2000]. This is a technique that allows several datatypes to be used in one tree. For each function, it can be specified of which datatype each parameter must be and which datatype it returns itself. When generating a random tree, only nodes of the correct datatype are inserted at each child node. Our strong typing system contains three datatypes:

1. `Variable`

   Terminals A terminal of the `Variable` type is an integer which ranges between 0 and the number of variables in the database $-$ 1. It represents the number of a variable in the database. When evaluated, it looks up the value in the database and returns it as a double.

Functions There are no functions of this type.

2. `Constant`

   Terminals A terminal of the type `Constant` is a double within a certain range. In the experiments, the range is [-10, 10].

   Functions There are no functions of this type.

3. `Classification`

   Terminals A terminal of this type is an integer which ranges between 0 and the number of possible classifications − 1. `MAX_CLASSIFICATIONS` is the number of different classifications that are allowed.

   Functions All functions have this return type.

2.3 LIMITED ERROR FITNESS

Limited Error Fitness (LEF) [Gathercole, 1998] is a modification to the standard supervised learning in GP, where an error limit is introduced. Each individual is evaluated on a number of training cases. The predicted classification of the individual is compared to the real classification of each training case. If the number of errors it has made so far is higher than the error limit, all remaining cases are counted as errors. The fitness score is the total number of errors. This method achieves that poor individuals will not be evaluated on the entire training set, saving CPU time. Good individuals, making fewer errors than the error limit, will be evaluated on the entire training set.

The error limit is modified over time. If no improvement has been made on the best individual in the population for some generations, the error limit is changed. It is raised if the best individual scores more errors than the error limit and lowered if it scores fewer errors than the error limit. Also, the easiest training case (the case that most individuals classify correctly) is moved to the end of the training set. The most difficult cases are emphasized at the cost of the easier ones, because the easier ones move towards the back of the training set.

Gathercole's suggested parameter values are used in our experiments. Since the initial order of the training cases matters when LEF is used, the training cases are shuffled before each run.

2.4 BLOAT

In GP, individuals tend to become larger over time [Koza, 1992, Blickle and Thiele, 1994, Nordin and Banzhaf, 1995, McPhee and Miller, 1995, Soule *et al.*, 1996, Soule, 1998, Langdon *et al.*, 1999]. This phenomenon is known as bloat. Disadvantages include longer execution time for evaluating individuals and lower understandibility of the trees for humans.

In [Bot and Langdon, 2000] several measures are compared to avoid this problem. In these experiments, a nodes penalty of 0.5 and a depth penalty of 2.0 were found to be best. Here, another way to avoid bloat is used which is called Pareto scoring (see the next section).

2.5 PARETO SCORING

Pareto scoring [Goldberg, 1989] is a way of combining multiple goals in the fitness function. Formally, a multi-objective optimization problem is defined as follows:

$$\min_{x \in F} f_1(x), f_2(x), \ldots, f_M(x)$$

where $f_1(x), f_2(x), \ldots, f_M(x)$ are the objective functions to be minimized simultaneously, x is the decision variable and F is the feasible region (the ranges of values allowed for each variable).

One way of doing this is to optimize a linear combination of the different goals. A common problem with this is finding good weights for the respective goals [Michalewicz, 1996].

Pareto scoring avoids this problem by allowing different individuals to "specialize" in different goals.

If an individual does not perform worse than another individual on every goal, and better in at least one, this individual is said to dominate the other. Individuals can be scored on the number of other individuals they dominate: the more, the better. Those individuals that aren't dominated by any other individual are located on the Pareto optimal front. Solution x_P is on the Pareto optimal front if there exists no $x \in F$ which satisfies

$$\forall m \in 1, \ldots, M : f_m(x) < f_m(x_P)$$

In other words: the closer to the Pareto front, the better the individual.

2.5.1 Fitness Sharing.

Using Pareto scoring, the population tends to converge to a few (possibly suboptimal) solutions [Goldberg, 1989]. Fitness sharing (also described in [Goldberg, 1989]) ensures that more different local minima are found. It is a so-called "crowding technique". To avoid premature convergence, a penalty is given to individuals that have many "neighbours": other individuals that are nearly the same.

Between two individuals, a distance measure $d(i, j)$ is calculated, based either on genotype or phenotype. The more similar they are, the lower the distance measure will be.

For each individual i on the Pareto front of the tournament, a sharing function $s(i)$ is calculated. All individuals in the population (or, for reasons of efficiency, a sample of the population) contribute to the sharing function $s(i) = \sum_j s(i, j)$ with $s(i, j)$ the contribution of individual j. $s(i, j)$ is a triangular function with value 1 for individuals similar in genotype or phenotype and value 0 for individuals less similar:

$$s(i,j) = \begin{cases} 1 - \frac{d(i,j)}{max} & \text{if } d(i,j) < max \\ 0 & \text{otherwise} \end{cases}$$

with max the maximum distance at which individual j contributes to $s(i)$

The individual with the lowest $s(i)$ is the best among them and will be allowed to reproduce. This way, the population itself is used for controlling the evolution process, because the fitness of each individual is not only dependant on the individual itself, but also on other individuals.

2.5.2 Application of Pareto Scoring and Fitness Sharing to minimize size and errors.

Pareto scoring will be applied with two dimensions: number of errors (the smaller, the better) and number of nodes in the tree (the smaller, the better). [Langdon, 1998] By using fitness sharing, a diverse population with both small trees and good scoring trees can evolve. In the end of the run, the individual which scores fewest errors is returned. Hopefully, this will also be a small tree.

In [Horn and Nafpliotis, 1993], genotype-based Pareto scoring and phenotype-based Pareto scoring are distinguished. Genotype-based Pareto scoring compares individuals on the basis of their "genes", i.e. the internal representation. Phenotype-based Pareto scoring compares the solutions produced by the system by various dimensions of the solutions. Our system is a combination of the two, as one dimension is genotypic (the number of nodes) and the other is phenotypic (the number of errors).

2.5.3 Domination based Fitness Sharing.

An alternative way to provide fitness sharing [Horn and Nafpliotis, 1993] is by counting the number of individuals in the population (or a sample of it) that dominate or are equal in size and errors to each of the non-dominated individuals in the tournament. The individual that is dominated by or equal to the smallest number of individuals wins the tournament and is allowed to reproduce. This is a form of niching, since if there are many individuals which score the same on all dimensions, they will get a bad score. This amounts to a diversivity pressure on the population, spreading out the population over the Pareto front.

3 EXPERIMENTAL SETUP

We used the GP system by Qureshi (GPSys) [Qureshi, 1997], which is written in Java. GPSys is a steady state, elitist system with tournament selection.

In Table 1, a Koza-style tableau is given with the standard settings for the experiments.

Table 1: Standard settings of the GP

Objective	Classify training cases correctly
Terminal set	Variable, Constant Classification
Function set	CheckCondition1Var, CheckCondition2Vars, CheckCondition3Vars
Fitness Cases	Various databases (see Section 3.1)
Selection	Tournament of size 7
Hits	not used
Wrapper	not used
Parameters	Population size = 250 No of runs = 30 No of generations=1000 Steady state, elitist 10-fold crossvalidation Mutation rate 50% Pareto sample size 50
Initial population	RAMPED_HALF_AND_HALF
Termination	All cases correctly classified

The more generations allowed, the more accurate individuals are, so the number of generations was set quite high. Ten-fold crossvalidation [Mitchell, 1997] was used, with 3 runs in each split. After each run, the best individual is reported.

In [Bot and Langdon, 2000], different settings for the tournament size, mutation and crossover rates and bloating-penalties were compared. These settings result from those experiments. A penalty of 0.5 times the number of nodes or 2 times the depth proved to be the best penalties. Those are compared to the fitness sharing techniques.

3.1 MACHINE LEARNING REPOSITORY DATABASES

Four databases from the Machine Learning Repository [MLR, 1999] were used in the experiments:

- The Glass database contains 214 instances of 9 continuous variables each plus a classification (the type of glass). There are 7 classes (1...7), one of which (4) isn't used.
- The Ionosphere database contains 351 instances of 34 continuous variables plus a class attribute.
- The Pima database contains 768 instances of 8 continuous variables plus a class attribute. Classification is binary, either the Pima-Indians are positive of negative for diabetes.
- The Segmentation database comes in two parts. The training database consists of 210 and the validation database of 2100 cases. For crossvalidation, these were added together and crossvalidation took place on all 2310 cases. There are 19 attributes plus a class variable. There are 7 different classes.

4 EXPERIMENTS

4.1 PARETO SCORING WITH FITNESS SHARING

As explained in Section 2.5, Pareto scoring and fitness sharing are combined in order to find both accurate and small decision trees.

The distance between 2 individuals is based on the shape of the two trees. This is done to create a population diverse in tree shapes and sizes. The Pareto front will range from small, poorly scoring individuals to larger, better scoring individuals. Since a smaller individual would dominate a larger individual with the same number of errors, bloat will be reduced.

The distance between 2 individuals i and j is calculated as follows:

$$d_{i,j} = |\#nodes_i - \#nodes_j| + |depth_i - depth_j|$$

where $\#nodes_i$ is the number of nodes of individual i and $depth_i$ is the depth of individual i. The emphasis in this distance measure is on tree size (calculated in number of nodes), since that is what we're trying to reduce.

The fitness share of individual j to individual i is

$$s(i,j) = \begin{cases} 1 - \frac{d(i,j}{10} & \text{if} \quad d(i,j) <= 10 \\ 0 & \text{if} \quad d(i,j) > 10. \end{cases}$$

The value 10 seemed reasonable since one extra function node adds 5, 7 or 9 extra nodes to the tree (and sometimes increases depth by 1). Note that in the distance measure the number of errors is not included. This is because we don't want to force the system in producing some bad individuals. If the number of nodes is included in the distance measure, the diversity pressure of the fitness sharing allows both well scoring and badly scoring individuals to emerge. For a given tree size, we want the trees to be as accurate as possible.

4.2 RESULTS

The following tables summarize the results from the experiments. The mean validation accuracy and tree size of the best individuals from the 30 runs is reported, plus standard deviation.

4.2.1 LEF

First the performance of LEF is compared to that without LEF. In Tables 2 until 5 accuracies and tree sizes are given for the nodes-penalty of 0.5 and the depth-penalty of 2, with and without LEF. There are no significant differences (according to the one-sided t-test). Since the use of LEF results in a reduction in run time of approximately 40% and makes no difference in accuracy or tree size, it will be used in the subsequent experiments.

Table 2: GP without LEF, nodes penalty = 0.5. Mean validation accuracy and tree size of the best individual of the runs plus standard deviation

Problem	with LEF	
	accuracy in %	tree size
Glass	64.1 ± 9.1	17.6 ± 4.5
Ionosphere	90.2 ± 5.5	19.7 ± 4.1
Pima	71.1 ± 5.0	14.7 ± 7.3
Segmentation	72.0 ± 6.4	53.8 ±14.3

Table 3: GP with LEF, nodes penalty = 0.5. Mean validation accuracy adn tree size of the best individual of the runs plus standard deviation

Problem	with LEF	
	accuracy in %	tree size
Glass	57.9 ± 15.2	19.4 ± 4.0
Ionosphere	91.9 ± 4.7	20.9 ± 3.9
Pima	75.1 ± 6.8	11.6 ± 4.8
Segmentation	75.6 ± 6.5	56.6 ±19.4

Table 4: GP without LEF, depth penalty = 2. Mean validation accuracy and tree size of the best individual of the runs plus standard deviation

Problem	without LEF	
	accuracy in %	tree size
Glass	63.0 ± 10.8	43.2 ±22.5
Ionosphere	92.0 ± 5.9	40.5 ±21.2
Pima	69.8 ± 5.8	43.8 ±23.6
Segmentation	78.2 ± 5.3	191.9 ±90.9

Table 5: GP with LEF, depth penalty = 2. Mean validation accuracy and tree size of the best individual of the runs plus standard deviation

Problem	with LEF	
	accuracy in %	tree size
Glass	61.2 ± 11.1	43.3 ±19.0
Ionosphere	90.8 ± 3.5	44.9 ±17.9
Pima	72.5 ± 6.5	64.1 ±53.9
Segmentation	75.5 ± 7.2	175.8 ±88.1

4.2.2 Fitness Sharing Pareto and Domination Pareto

Next, experiments were performed to compare the performance of fitness sharing Pareto and domination Pareto. In Tables 6 and 7 mean accuracy and tree size of the best individuals from each of the 30 runs is given with fitness sharing Pareto and with domination Pareto. One-sided t-tests were performed to determine if one method produces significantly better or worse validation accuracies.

Fitness sharing Pareto generally creates both larger trees and more or equally accurate trees than when a size-penalty is applied. Domination Pareto creates smaller but less or equally accurate trees than fitness sharing Pareto.

Table 6: Fitness sharing Pareto. Mean validation accuracy of the best individual of the runs plus standard deviation. The asterisks mark the significantly higher validation accuracies (by a t-test) and significantly larger trees.

Problem	fitness sharing Pareto	
	Accuracy	Tree size
Glass	58.9 ±11.4	30.6 ±15.6
Ionosphere	92.4 ± 5.1	25.9 ± 6.8 *
Pima	73.6 ± 6.6	24.2 ±16.7
Segmentation	83.3 ± 4.6 *	115.1 ±32.9 *

Table 7: Domination Pareto scoring. Mean validation accuracy of the best individual of the runs plus standard deviation.

Problem	domination Pareto	
	Accuracy	Tree size
Glass	62.7 ±12.3	22.8 ± 7.1
Ionosphere	90.2 ± 5.4	20.9 ± 5.4
Pima	72.8 ± 5.6	19.2 ±12.3
Segmentation	74.3 ± 5.0	62.9 ±25.1

5 ANALYSIS

5.1 PARETO FRONTS

In Figures 4 and 5 the evolution over time of the Pareto fronts in the population is drawn, with fitness sharing and domination Pareto scoring. With fitness sharing Pareto, improvement can be seen over time over the whole Pareto front. With domination Pareto, the fronts stay more or less the same over time. The only improvements in later generations are made at the very end of the graph (at the highest accuracies).

It would seem that in later generations improvement comes from fortunate mutations, looking at the long periods of time in which nothing changes.

Because LEF is used in these runs, it can be seen that for example the performance of the smallest individual (which is bad of course) decreases over time. This can be attributed to the error limit, which lowers over time (as better individuals are found), causing the smallest individual to perform worse than before the error limit was lowered.

The zeroth generation contains only few individuals on the Pareto front with the most accurate individual being very large (see the spikes at generation 0). After only a few generations, a much smaller and more accurate individual is found.

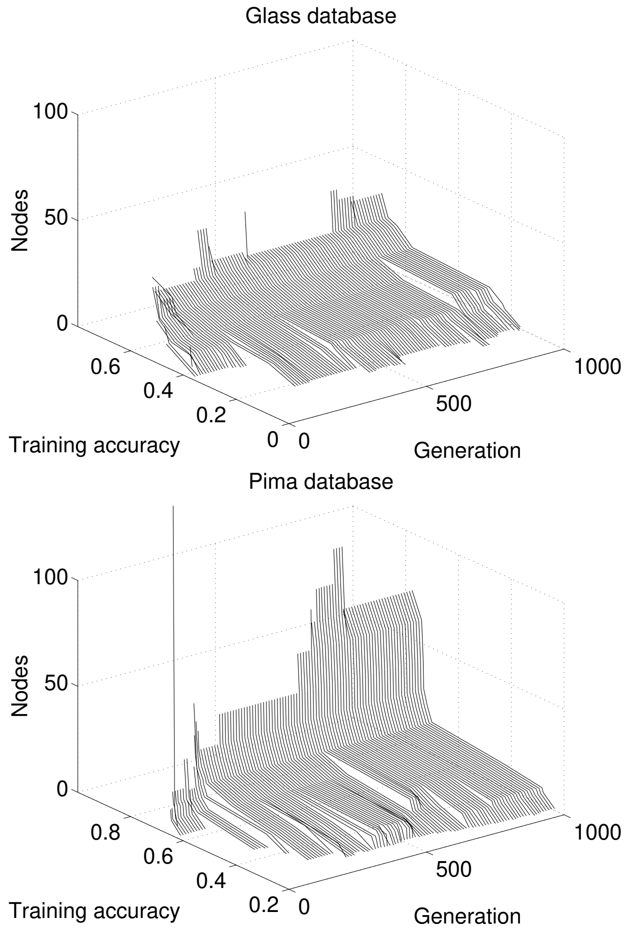

Figure 4: Evolution of Pareto fronts in a run with fitness sharing Pareto on two of the machine learning databases (Glass and Pima).

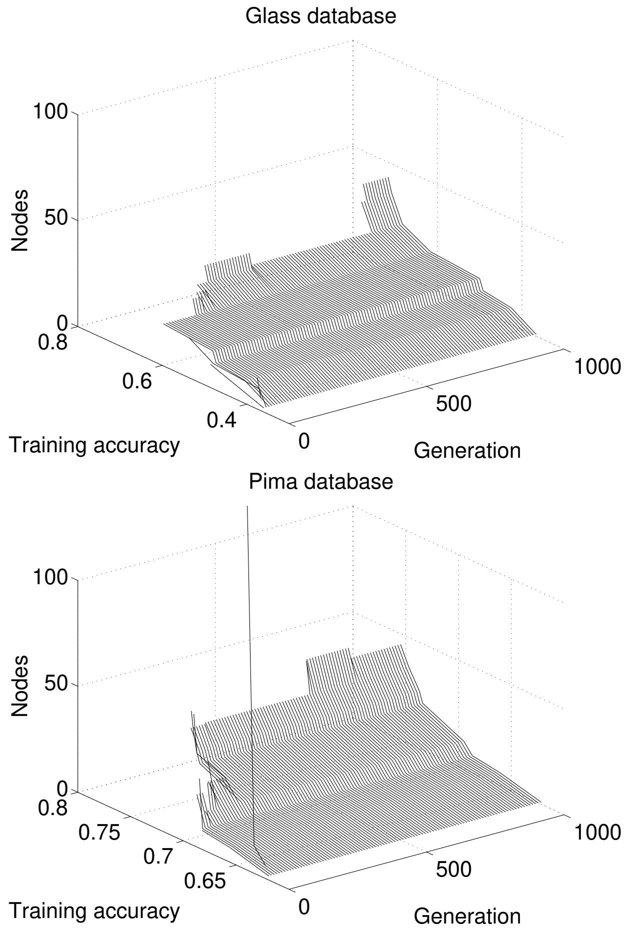

Figure 5: Evolution of Pareto fronts in a run with domination Pareto of two of the machine learning databases (Glass and Pima)

5.2 COMPARISON TO MACHINE LEARNING ALGORITHMS

In Tables 8 and 8 the performance of the GP is compared to that of three other decision tree classification algorithms, namely OC1 [Murthy *et al.*, 1993], C5.0 [Quinlan, 1993] and M5' [Frank *et al.*, 1998]. Ten-fold crossvalidation and standard parameter settings are used in the other algorithms.

The GP performs as well as or better than reported decision tree algorithms (OC1, C5.0 and M5') on two datasets (Ionosphere and Pima), but worse on Glass and Segmentation. Determining characteristics of databases on which the GP does well or poorly is one of the subjects for future research.

The GP is slower than the other techniques. One run on a dataset of 700 cases on an Pentium III 450 takes approximately 5-8 minutes. A run of the other techniques (which are written in C or C++) typically takes about one or two minutes.

Table 8: GP and OC1 algorithm. The GP is with fitness sharing Pareto and LEF. Mean training and validation accuracy of the best individual of the runs plus standard deviation. Asterisks mark significantly better accuracy compared to the GP or better accuracy of the GP.

Problem	GP	OC1
Glass	58.9 ± 11.4	62.3 ± 13.4
Ionosphere	92.4 ± 5.1 *	90.0 ± 5.8
Pima	73.6 ± 6.6	74.1 ± 6.1
Segmentation	83.3 ± 4.6	95.4 ± 1.5 *

Table 9: C5.0 and M5' algorithm. Mean training and validation accuracy of the best individual of the runs plus standard deviation. Asterisks mark significantly better accuracy compared to the GP or better accuracy of the GP.

Problem	C5.0	M5'
Glass	67.5 ± 2.6	70.5 ± 2.8 *
Ionosphere	88.9 ± 1.2	89.7 ± 1.2
Pima	74.5 ± 1.2	76.2 ± 0.8
Segmentation	96.8 ± 0.2 *	97.0 ± 0.2 *

6 CONCLUSIONS

On some datasets, the GP works as well as or better than reported accuracies for decision tree algorithms, but on others, the GP does worse (see Tables 8 and 9). More research is needed to find characteristics of datasets on which the GP does well or poorly.

Genetic programming is a promising approach to classification problems. A disadvantage of this approach is its long run time. Future research will aim at improving execution speed and accuracy (for example by dynamic sampling of training cases (DSS [Gathercole, 1998]) or by seeding the population with trees constructed by standard decision tree algorithms such as C5.0).

As shown in Table 6, fitness sharing Pareto is a promising approach for controlling bloat. No artificial size-penalty is needed. Fitness sharing Pareto scoring is shown to perform better than or equally good as domination Pareto scoring (Tables 6 and 7).

References

[Blickle and Thiele, 1994] Tobias Blickle and Lothar Thiele. Genetic programming and redundancy. In J. Hopf, editor, *Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94, Saarbrücken)*, pages 33–38, Im Stadtwald, Building 44, D-66123 Saarbrücken, Germany, 1994. Max-Planck-Institut für Informatik (MPI-I-94-241).

[Bot and Langdon, 2000] M.C.J. Bot and W.B. Langdon. Application of genetic programming to induction of linear classification trees. In *Proceedings of the Third European Conference on Genetic Programming*, 2000.

[Frank *et al.*, 1998] E. Frank, Y. Wang, S. Inglis, G. Holmes, and I.H. Witten. Using model trees for classification. *Machine Learning*, 32:63–76, 1998.

[Gathercole, 1998] Chris Gathercole. *An Investigation of Supervised Learning in Genetic Programming*. PhD thesis, University of Edinburgh, 1998.

[Goldberg, 1989] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Inc., January 1989.

[Horn and Nafpliotis, 1993] J. Horn and N. Nafpliotis. Multiobjective optimization using the niched pareto genetic algorithm. Technical Report 93005, University of Illinois, July 1993.

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

[Langdon *et al.*, 1999] W.B. Langdon, T. Soule, R. Poli, and J.A. Foster. The evolution of size and shape. In L. Spector, W.B. Langdon, U. O'Reilly, and P.J. Angeline, editors, *Advances in Genetic Programming 3*, chapter 8, pages 163–190. MIT Press, Cambridge, MA, USA, May 1999. Forthcoming.

[Langdon, 1998] William B. Langdon. *Data Structures and Genetic Programming: Genetic Programming + Data Structures = Automatic Programming!* Kluwer, Boston, 24 April 1998.

[McPhee and Miller, 1995] Nicholas Freitag McPhee and Justin Darwin Miller. Accurate replication in genetic programming. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 303–309, Pittsburgh, PA, USA, 15-19 July 1995. Morgan Kaufmann.

[Michalewicz, 1996] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996.

[Mitchell, 1997] T. Mitchell. *Machine Learning*. WCB/McGraw-Hill, 1997.

[MLR, 1999] Machine learning repository. `http://www.ics.uci.edu/~mlearn/MLRepository.html`, 1999.

[Montana, 1995] D.J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.

[Murthy *et al.*, 1993] S. Murthy, S. Kasif, S. Salzberg, and R. Beigel. Oc1: Randomized induction of oblique decision trees. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 322–327. AAAI, MIT Press, 1993.

[Murthy, 1998] S. K. Murthy. Automatic construction of decision trees from data: a multi-disciplinary survey. In *Data Mining and Knowledge Discovery*, number 2, pages 345–389, 1998.

[Nordin and Banzhaf, 1995] Peter Nordin and Wolfgang Banzhaf. Complexity compression and evolution. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 310–317, Pittsburgh, PA, USA, 15-19 July 1995. Morgan Kaufmann.

[Quinlan, 1993] J.R. Quinlan. *C4.5 Programs for Machine Learning.* Morgan Kaufmann, 1993.

[Qureshi, 1997] A. Qureshi. Gpsys 1.1. `http://www.cs.ucl.ac.uk/staff/A.Qureshi/gpsys.html`, June 1997.

[Soule *et al.*, 1996] Terence Soule, James A. Foster, and John Dickinson. Code growth in genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 215–223, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[Soule, 1998] Terence Soule. *Code Growth in Genetic Programming.* PhD thesis, University of Idaho, Moscow, Idaho, USA, 15 May 1998.