Application of Genetic Programming to Induction of Linear Classification Trees

Martijn C.J. Bot[1] and William B. Langdon[1]

CWI, Kruislaan 413, 1098 SJ Amsterdam
mbot@cs.vu.nl 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 is introduced. With this representation it is now possible to let the GP classify into any number of classes. 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 [11] (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.

Well known decision tree algorithms such as ID3, CART, OC1 and C4.5 are greedy local search algorithms which construct trees top-down [11]. In this paper, genetic programming (GP) [5] is used as a global stochastic search technique for finding accurate decision trees.

Previous work on evolving decision trees with GP was done in [5] and [12]. The standard representation of GP was used in these experiments.

In this paper, a new representation of decision trees in GP using Strong Typing is introduced. 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. Section 3 explains the experimental setup for the experiments. The results are given 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.

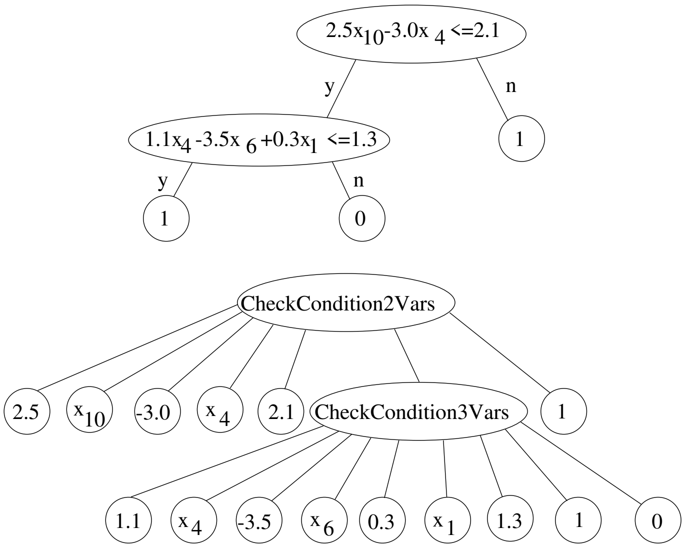

Fig. 1. Example decision tree and its representation in the GP. x_{10} means the tenth variable from the dataset. Each function node's first children are the weights and variables for the linear combination. The last two children are other function nodes or classifications. When evaluating the CheckCondition2Vars 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.

2 Background

2.1 Decision Trees

Decision trees[11] are a well known technique for representing the underlying structure of a dataset.

An **axis-parallel** decision tree is one in which each node contains only 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 (weighted) 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.

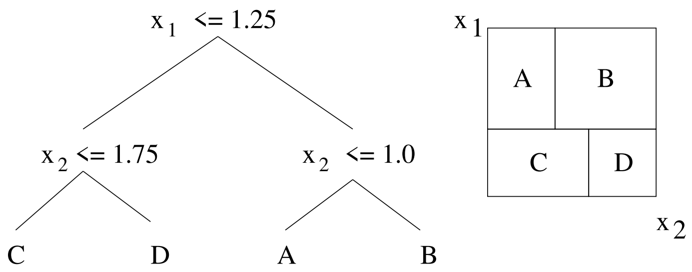

Fig. 2. The left side shows a simple axis-parallel decision tree. The right side shows the partitioning that this tree creates in the two-dimensional attribute space.

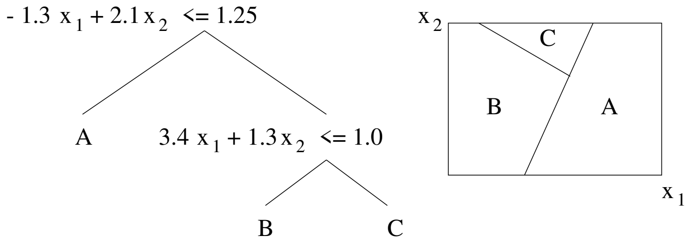

Fig. 3. The left side shows an oblique decision tree. The right side shows the partitioning that this tree creates in the two-dimensional attribute space.

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:

if $\sum_i c_i x_i$ <=threshold then	return value of ifTrue branch
else	return value of 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. Terminals are either constants (doubles), variables (integers) or classifications (integers).

2.2 Strong Typing

In order to ensure that only valid individuals are constructed in the GP, strong typing was used [9]. 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 Bloat

In GP, individuals tend to become larger over time [5] [2] [13] [7] [17] [16][6]. This phenomenon is known as bloat. Disadvantages include overtraining, longer execution time for evaluating individuals and lower understandibility of the trees for humans.

In our experiments, several measures are compared to avoid this problem. One way is to give penalties to larger individuals [16]. This may be done by lowering the fitness value by some factor times the number of nodes in the tree or its depth.

3 Experimental Setup

We used the GP system by Qureshi (GPSys) [15], which is written in Java. GPSys is a steady state, elitist system with tournament selection.

Table 1. Standard settings of the GP

Objective	Classify all 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% Initial population creation RAMPED_HALF_AND_HALF Pareto sample size 50
Termination criterium	All cases correctly classified

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

The more generations allowed, the more accurate individuals are, so the number of generations was set quite high. Ten-fold crossvalidation [8] was used, with 3 runs in each split. After each run, the best individual is reported. The focus could be on accuracy, speed, generalization and simplicity of the trees. The best individual in our system is the most accurate one. Only after the runs are completed we look at speed, generalization and simplicity.

Some initial runs were performed to compare settings for the tournament size, mutation and crossover rates and bloating-penalties. In Section 4 the effects of different mutation rates are examined. We found that the exact amount of mutation and crossover was of no high importance, as long the mutation rate is larger than 0%. Thus, the mutation rate was set to 0.5 and the crossover rate also at 0.5. De GROW-population creation method was also used, but this made no significant difference to the RAMPED method.

Different bloating penalties were applied, on the number of nodes and on the depth of the tree. 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 [1] 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 Results

4.1 Mutation rate

In Figure 4 the performance of the GP on the Pima dataset with different mutation rates is shown. There's no significant difference between most mutation rates when a one-sided t-test is performed. This can also be seen in the figure, since most confidence intervals overlap. Only a rate of 0% is significantly worse than the other settings. Since at 50% there's a peak in validation accuracy, this setting will be used in the other experiments.

4.2 Bloating penalties

In Figure 5 the effects of different nodes penalties on the validation accuracy of the GP is shown. In Figure 6 depth penalties are examined. Again, most differences aren't significant. The nodes penalty that will be used in the other experiments is 0.5. The depth penalty is 2.0. Like the setting for mutation rate, these values are taken because there are peaks at those values.

4.3 Results on all datasets

In Table 2 the performance of the GP with a nodes penalty of 0.5 and that of the GP with a depth penalty of 2.0 are compared. The mean validation accuracy and tree size of the best individuals from the 30 runs is reported, plus standard deviation.

5 Analysis

5.1 Accuracy

In most runs, there is some overtraining, since the training accuracy is higher than the validation accuracy.

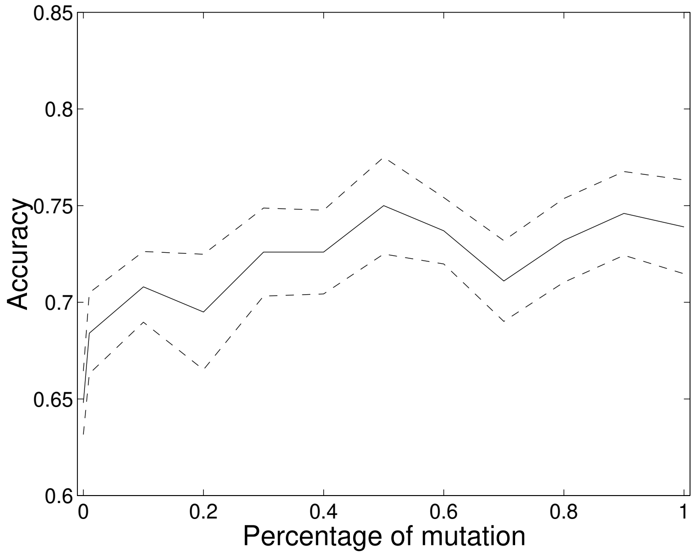

Fig. 4. Average validation accuracy and 95% confidence interval of the GP on the Pima dataset with different mutation rates

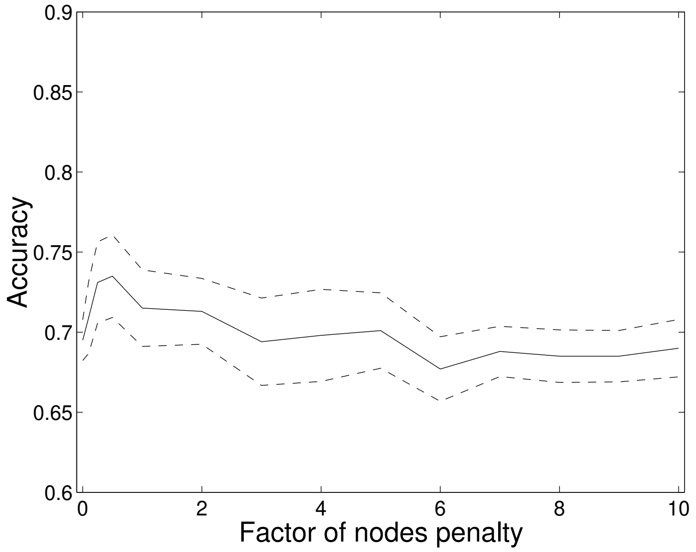

Fig. 5. Average validation accuracy and 95% confidence interval of the GP on the Pima dataset with different nodes penalties

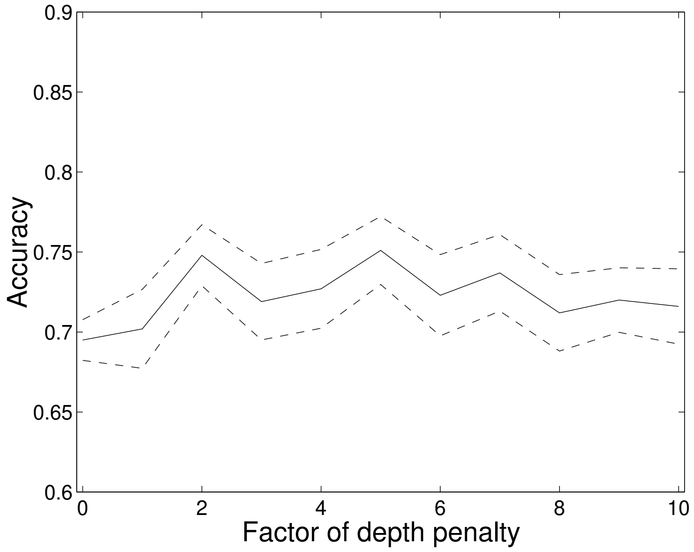

Fig. 6. Average validation accuracy and 95% confidence interval of the GP on the Pima dataset with different depth penalties

Table 2. Comparison between nodes penalty = 0.5 and depth penalty = 2.0. Mean training and validation accuracy and tree size of the best individual of the runs plus standard deviation

Problem	Nodes penalty		Depth penalty	
	accuracy in %	tree size	accuracy in %	tree size
Glass	63.2 ± 3.1	±	66.5 ± 4.2	±
Validation	64.1 ± 9.1	17.6 ± 4.5	63.0 ± 10.8	43.2 ±22.5
Ionosphere	93.7 ± 1.7	±	94.6 ± 1.3	±
Validation	90.2 ± 5.5	19.7 ± 4.1	92.0 ± 5.9	40.5 ±21.2
Pima	75.3 ± 1.4	±	75.4 ± 2.0	±
Validation	71.1 ± 5.0	14.7 ± 7.3	69.8 ± 5.8	43.8 ±23.6
Segmentation	73.1 ± 5.2	±	78.6 ± 5.8	±
Validation	72.0 ± 6.4	53.8 ±14.3	78.2 ± 5.3	191.9 ±90.9

In Figures 7 and 8 plots are drawn of the average number of validation errors over time on the four datasets.

Clearly, the performance of the GP is different per database. On the ionosphere database, the GP improves for a few generations after which there's no more improvement. On the Pima and Glass database, the validation accuracy slowly improves for approximately 500 generations. After that, it stays more or less the same. The Segmentation keeps improving for 1000 generations and possibly goes on even after that. The number of generations that is needed to find a good decision tree depends on the database.

The extra generations are neither beneficial nor harmful. They don't cause extra overtraining. Confidence intervals are very regular, but fairly wide (note the different scales in the figures). This suggests that the performance of the GP is stable in the different runs.

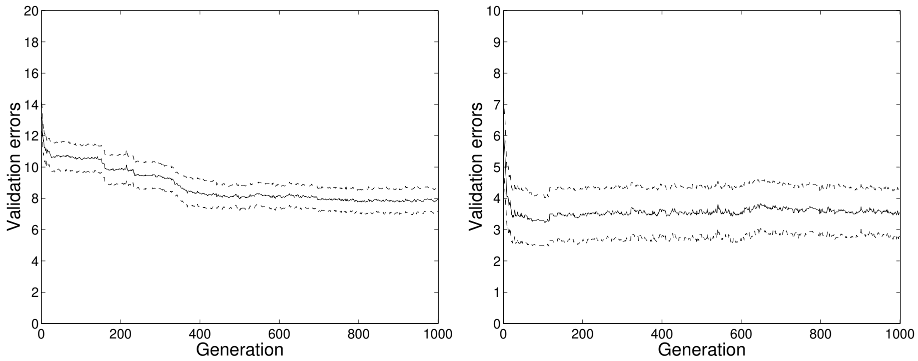

Fig. 7. Average validation errors and 95% confidence interval over time on the Glass dataset (left figure) and the Ionosphere dataset (right figure)

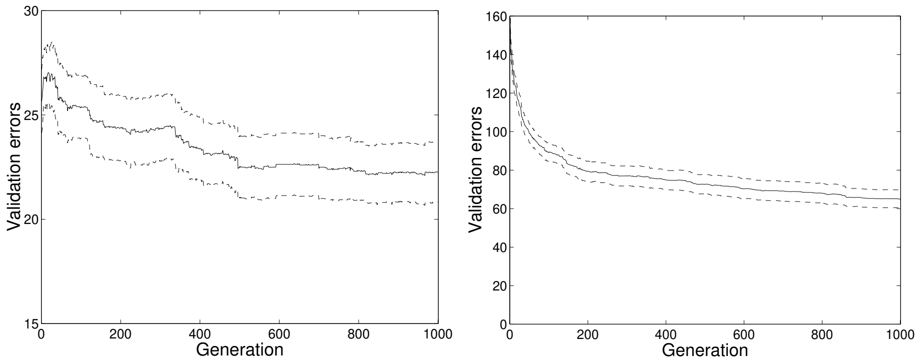

Fig. 8. Average validation errors and 95% confidence interval over time on the Pima dataset (left figure) and the Segmentation dataset (right figure)

5.2 Comparison to Machine Learning Algorithms

In Table 3 the performance of the GP is compared to that of three other decision tree classification algorithms, namely OC1 [10], C5.0 [14] and M5' [3]. Ten-fold crossvalidation and standard parameter settings are used in the other algorithms.

Table 3. Comparison between the GP and the OC1, C5.0 and M5' 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. Stars mark significantly better accuracy compared to the GP or better accuracy of the GP.

Problem	GP	OC1	C5.0	M5'
Glass	64.1 ± 9.1	62.3 ± 13.4	67.5 ± 2.6	70.5 ± 2.8 *
Ionosphere	92.0 ± 5.9 *	90.0 ± 5.8	88.9 ± 1.2	89.7 ± 1.2
Pima	71.1 ± 5.0	74.1 ± 6.1	74.5 ± 1.2	76.2 ± 0.8
Segmentation	78.2 ± 5.3	95.4 ± 1.5 *	96.8 ± 0.2 *	97.0 ± 0.2 *

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. Tree sizes of the GP are usually 5−10 times smaller than those from OC1 and C4.5 (from M5' no data was reported on tree sizes). However, if we let the GP run without any size restrictions, the produced trees are even larger than those from OC1 and C4.5 and overtraining increases much (training accuracy goes up and validation accuracy goes down). 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.

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 Table 3). 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 [4]) or by seeding the population with trees constructed by standard decision tree algorithms such as C5.0).

References

1. Machine learning repository. http://www.ics.uci.edu/~mlearn/MLRepository.html.
2. 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).
3. E. Frank, Y. Wang, S. Inglis, G. Holmes, and I.H. Witten. Using model trees for classification. *Machine Learning*, 32:63–76, 1998.
4. Chris Gathercole. *An Investigation of Supervised Learning in Genetic Programming*. PhD thesis, University of Edinburgh, 1998.
5. J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
6. 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.
7. 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.
8. T. Mitchell. *Machine Learning*. WCB/McGraw-Hill, 1997.
9. D.J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.
10. 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.
11. 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.
12. Nikolay I. Nikolaev and Vanio Slavov. Inductive genetic programming with decision trees. In *9th European Conference on Machine Learning*, Prague, Czech Republic, 3-26 April 1997.
13. 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.
14. J.R. Quinlan. *C4.5 Programs for Machine Learning*. Morgan Kaufmann, 1993.
15. A. Qureshi. Gpsys 1.1. http://www.cs.ucl.ac.uk/staff/A.Qureshi/gpsys.html, June 1997.
16. Terence Soule. *Code Growth in Genetic Programming*. PhD thesis, University of Idaho, Moscow, Idaho, USA, 15 May 1998.
17. 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.