A comparison of genetic programming variants for data classification

J. Eggermont, A.E. Eiben, and J.I. van Hemert

Leiden University
P.O. Box 9512
2300 CA Leiden
The Netherlands
{jeggermo,gusz,jvhemert}@cs.leidenuniv.nl

Abstract. In this paper we report the results of a comparative study on different variations of genetic programming applied on binary data classification problems. The first genetic programming variant is weighting data records for calculating the classification error and modifying the weights during the run. Hereby the algorithm is defining its own fitness function in an on-line fashion giving higher weights to 'hard' records. Another novel feature we study is the atomic representation, where 'Booleanization' of data is not performed at the root, but at the leafs of the trees and only Boolean functions are used in the trees' body. As a third aspect we look at generational and steady-state models in combination of both features.

1 Introduction

Binary data classification problems (with exactly two disjoint classes) form an important application area of machine learning techniques, in particular *genetic programming* (GP) [7, 1]. In this paper we compare a number of different variants for a GP applied to such problems. Rather than simply tuning on traditional GP parameters, we investigate the effect of two significant changes in a fixed GP setup (closely matching the setups in [7]) in combination with a generational, respectively steady-state model.

The first modification we consider amounts to using a fitness function based on weighting data records when calculating the total classification error and modifying the weights (thus the fitness function) on-line, during the run. This feature, called Stepwise Adaptation of Weights (SAW) has been first used in penalty functions for constraint satisfaction problems (see e.g. [3] and applied in one particular setup within a GP [2]. Here we conduct a systematic study of combinations of SAW-ing with a new representation in GP.

A specific representation forms the second line of investigation. In standard GP a function set of real valued operators is used and a special operator in the root of the tree 'Booleanizes' the outcome resulting in a (binary) classification of a given data record [7]. In our atomic representation 'Booleanization' of data is not performed at the root, but immediately at the leafs of a tree and only Boolean functions are used in the tree's body. By such a relatively simple function set the flexibility to create different models, i.e. trees, is lower than in the standard representation, but the models become more transparent, i.e. better readable for humans. In practical applications this property is often more important than a lower classification error. The first experiments with this representation are reported in [6]

The third aspect we consider concerns steady-state and generational GP models. The standard GP approach is based on the generational population model and although some authors do use a steady-state model [2, 8], to our knowledge there is no experimental comparison available on the relative advantage of either model.

All the experiments were done using the *Library for Evolutionary Algorithm Programming* (LEAP)[1] system for the construction of the GPs, [6]. This library is currently under development using C++ and Design Patterns [5], and is aiming to be a framework that makes it easy to test out different methods within the field of evolutionary computation. Also it provides an easy way for users to incorporate their own techniques, representations and problems, thus assuring a fast and easy way of dealing with testing evolutionary algorithms in general.

2 Data sets and experiment setup

For comparing different algorithm variants we use four different data sets from the Statlog collection[2]: the Australian Credit, the German Credit, the Heart Disease, and the Pima Indians Diabetes data set [9]. Each algorithm is evaluated using n-fold cross validation and the performance measure for different algorithms is the average classification error over the n-folds. Statlog uses a cost matrix for the reported results on the Heart Disease and German Credit data set. This influences the measured classification error as different penalties are given for each value of an attribute. For instance, misclassifying a patient with a heart disease receives a higher penalty than misclassifying a patient without a heart disease. We do not consider the Statlog cost matrix. In Table 1 we give the number of records for each data set and the number of folds done in the cross validation tests.

Besides the two tested features the other components and parameters of our GP are kept close to the experiments as described in [7], although we do use mutation in all cases (depending on the representation). The parameters that are common in each tested algorithm variants are summarized in Table 2.

3 SAW-ing

The rationale behind the Stepwise Adaptation of Weights (SAW) mechanism is to let the algorithm define the weights itself. The application domain of this

[1] Available on WWW at `http://www.wi.leidenuniv.nl/~jvhemert/leap`
[2] Available on WWW at `http://www.ncc.up.pt/liacc/ML/statlog`

Table 1. Test data sets

data set	number of records	cross validation
Australian Credit	690	10-fold
German Credit	1000	10-fold
Heart Disease	270	9-fold
Pima Indians Diabetes	768	12-fold

Table 2. Main GP parameters shared by all algorithm variants

Parameter	Value
Initial max. tree depth	5
Max. number of nodes	200
Initialization	ramped half-and-half
Population size	1000
Parent selection	linear ranking
Bias for linear ranked selection [10]	1.5
Replacement strategy	replace worst in population
Stop condition	perfect classification or 40000 evaluations
Mutation (atom/subtree) probability	0.1
Xover probability	0.9
Xover type	swap subtrees
Xover functions:atoms or functions:terminals ratio	4:1

mechanism is not restricted to GP and to data analysis, it can be used for any EA where the fitness function is composed in a manner represented by Equation 1. In fact, SAW-ing has been introduced and first applied in the context of constraint satisfaction [3, 4].

$$f(x) = \sum_{r \in D} w_r \cdot error(x, r) \tag{1}$$

where w_r is a weight assigned to the record r and $error(x, r)$ is a measure of misclassification. In the most simple case

$$error(x, r) = \begin{cases} 1 \text{ if } x \text{ classifies } r \text{ incorrectly} \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

In a SAW-ing evolutionary algorithm the weights are initially all set at the same value and these weights are repeatedly increased with a certain step size Δw at predefined moments during the run. The general mechanism is presented in Figure 1.

Note that in a classification problem the overall quality of a candidate solution (the accuracy of a model on the whole data set) is determined by local scores on data records. In other words, the evaluation function (fitness function)

On-line weight update mechanism
set initial weights (thus fitness function f)
while not EA is finished **do**
for the next T_p fitness evaluations **do**
let the EA run with this f
end for
redefine weights in f and recalculate fitness of individuals
end while

Fig. 1. Stepwise adaptation of weights (SAW)

measuring the total classification error is composed by the errors on particular data records. An evolutionary algorithm searching for a good model classifying the given records in a data set D could use the fitness function (to be minimized) defined as follows.

It is clear that a GP will primarily 'concentrate' on classifying those records correctly that carry the highest weights. Therefore, the weights need to be determined in accordance with the hardness of the records. Nevertheless, to determine how weights should be assigned to records appropriately, may require substantial insight into the problem, which may not be available, or only at substantial costs.

In the present SAW-ing GP implementation the weights are initially set as $w_r = 1$. Redefining the fitness function happens by adding Δw to the weights of those records that are misclassified by the best individual at the end of each period of T_p fitness evaluations. Obviously, the general rationale behind SAW-ing applies to our situation: records that are hard to classify correctly should have a higher weight, and by the on-line f adjust the weights according to its own experience.

The parameters of the SAW-ing mechanism as used through the present study are displayed in Table 3.

Table 3. SAW-ing parameters

Parameters	Value
Initial weights w_r	1
Δw	1
T_p steady-state	200 evaluations
T_p generational	1000 evaluations

As stated earlier, SAW-ing has been tested on various problems in the field of constraint satisfaction. The results obtained in these experiments are very promising. We therefore have extended the study of the SAW-ing mechanism to data classification using genetic programming. In Figure 2 we show typical runs of the SAW-ing mechanism on different binary CSPs [4]. Figures from other problems share a common feature; at first the fitness of the best individual is rising and then it suddenly drops to either a local optimum or eventually a global optimum. Also every fitness curve of a SAW-ing evolutionary algorithm shows a saw-shaped graph, due to redefinition of the weights every T_p fitness evaluations.

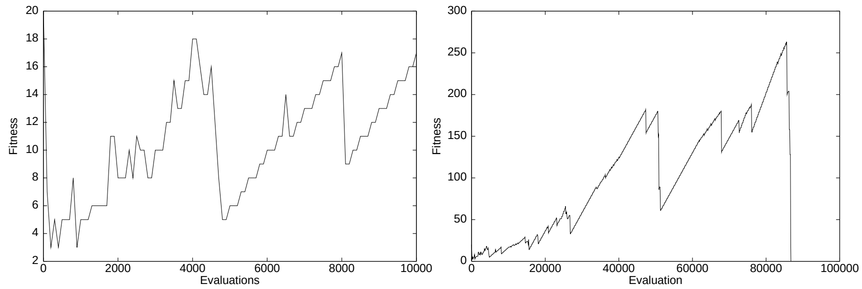

Fig. 2. Fitness curves for the SAW-ing EA on binary CSP. Figure on the left is a zoom in of the figure on the right for the first 10000 evaluations.

4 Atomic representation

The motivation behind our so-called atomic representation comes from practical applications, where the selection of the best model to classify data is not only depending on classification accuracy. In practice, before adopting a certain model an intuitive verification by the users common sense takes place. To this end, it is crucial that models (trees in a GP based problem solver) are transparent, that is easy to read and understand for humans. Often, the size of the trees is limited to achieve a satisfactory level of transparency. Here we investigate another option. Rather than using a function set of numerical operators and a special operator in the root of the tree that 'Booleanizes' the outcome (resulting in a binary classification of a given data record), we process numerical information at the leafs of a tree, transform it into Boolean statements, and apply only Boolean functions in the body of the tree (see Figure 3).

Formally, an atom is syntactically a predicate of the form *operator*(*var*, *const*), built up from a variable indicating a field in the data set, a constant between 0 and 1, and a comparing operator, denoted by $A_<$ and $A_>$. In this representation the conditional part of a classification rule could look like:

$$(A_>(r_1, 0.3) \texttt{ nor } A_<(r_0, 0.6)) \texttt{ or } A_>(r_1, 0.2)$$

In this representation we use subatomic mutation. Every time an individual is selected for a mutation, we first choose a node in the tree to work on. If this node

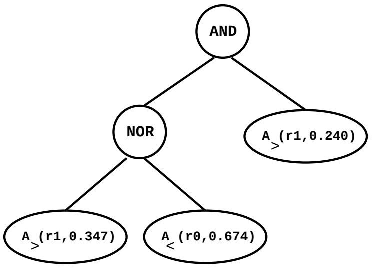

Fig. 3. Representation of a classification rule as a tree.

is part of the function set, a subtree mutation will be performed. If this node is a leaf (an atom), we choose with equal chance if this will be a subtree mutation or a subatomic mutation. A subatomic mutation works by first selecting, with equal chance, if the operation will be performed on the variable or on the constant. In case of a variable we randomly select a new variable. In case of the constant c a small number Δc $(-d < \Delta c < d)$ is generated which is then added to the constant as show in Equation 3. The values for all records are between 0 and 1 in the data sets we consider.

$$c' = \begin{cases} 0, & \text{if } c + \Delta c < 0, \\ 1, & \text{if } c + \Delta c > 1, \\ c + \Delta c, & \text{otherwise.} \end{cases} \tag{3}$$

With this representation we use the parameter setting displayed in Table 4 and with the standard GP representation we use the parameter setting displayed in Table 5.

Table 4. Parameters of the GP with an atomic representation.

Parameter	Value
Function set	{`and`, `or`, `nand`, `xnor`}
Atom set	attribute greater or less than a constant
Mutation type	1. subtree replacement 2. subatomic mutation
Subatomic parameter d	0.1

5 Steady-state vs. generational model

Here we will look at steady-state and generational GP models. The standard GP approach is based on the generational population model [7] and although

Table 5. Parameters of the standard GP.

Parameter	Value
Function set	$\{+, -, \times, \%\}$
Terminal set	$\{x_1, \ldots, x_n\} \cup [0, 1]$
Mutation type	1. subtree replacement
	2. point mutation

some authors do use a steady-state model [2, 8], to our knowledge there is no experimental comparison available on the relative advantages of these models.

In the steady-state variant we update the population after creating two offspring, thus after one cycle of parent selection, crossover and mutation, and use $T_p = 200$ (recalculating the weights after 100 cycles). In the generational case, however, maintaining the same setup is not possible. Namely, for a generational GP the update interval T_p must be a multiple of the population size, otherwise a weight revision would be necessary in the middle of creating a new generation. This imposes a restriction on the combination of SAW-ing and the generational model, leaving less freedom for the user to determine an algorithm setup. In order to allow the maximal number of weight recalculations, while maintaining the population size 1000 we use $T_p = 1000$ in the experiments with the generational model. Notice that this implies a handicap for SAW-ing as the number of weight adjustments is reduced by a factor of five.

6 Results

We are presenting the results of the experiments arranged around the four data sets used in this study. For each algorithm variant we give the average classification error percentage as performance measure. The best result in each table is presented in italics.

Table 6. Average classification error on the Australian Credit data set.

	steady state		generational	
algorithm	atom	standard	atom	standard
SAW	0.278	0.241	0.242	0.242
no-SAW	0.246	0.241	0.243	*0.232*

Looking at the tables 6 through 9 we can make a number of observations. Under the steady-state model the SAW-ing variant has or shares the first place in five of the eight tests. Looking from a different perspective, the standard GP seems slightly better than the atomic representation. Having a better performance five times out of eight.

Table 7. Average classification error on the German Credit data set.

	steady state		generational	
algorithm	atom	standard	atom	standard
SAW	*0.281*	0.303	0.295	0.292
no-SAW	*0.281*	0.284	0.300	*0.281*

Table 8. Average classification error on the Heart Disease data set.

	steady state		generational	
algorithm	atom	standard	atom	standard
SAW	0.200	0.211	0.222	0.181
no-SAW	0.211	0.200	0.230	*0.170*

To give some insight of what influence the SAW-ing mechanism has on the behavior on the fitness function we give some figures of a typical run of the GP. In Figure 4 we show plots of the steady-state GP, with and without the SAW-ing mechanism. The GP without SAW-ing shows the typical decrease of the fitness[3]. However, the plot of the GP with SAW-ing shows much fluctuation, in the range of 200. It is therefore crucial to stop the algorithm at the right time, i.e. when the fitness value is on its lowest point.

7 Comparison with other techniques

In order to compare our GP to other data mining techniques found in [9], we have chosen three algorithms outside the field of evolutionary computation. The algorithms we have chosen are LogDisc (a statistical method), C4.5 (a well-known decision trees algorithm) and Back-propagation (the standard algorithm for training multi-layered feed-forward networks). Since the results reported in Statlog have used a cost matrix (Heart Disease and German Credit [9, 2]) which we have not, we only compare our results to the Australian Credit and Pima Indians Diabetes data sets. The results can be found in Table 10. As representative for GP we use the standard (generational) GP which has the best overall performance.

[3] Recall that we are trying to minimize the fitness function

Table 9. Average classification error on the Pima Indian Diabetes data set.

	steady state		generational	
algorithm	atom	standard	atom	standard
SAW-ing	0.263	0.257	0.267	0.257
no-SAW-ing	0.283	0.266	*0.250*	0.253

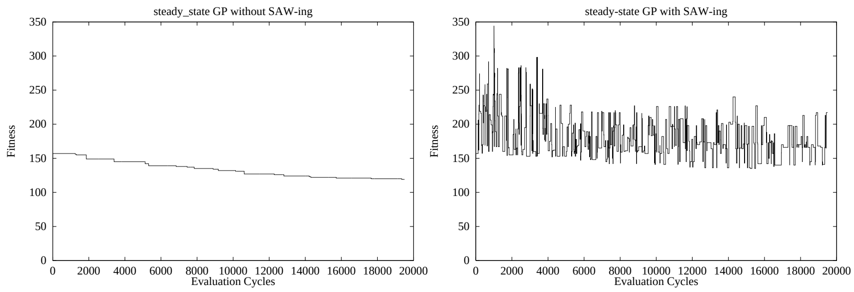

Fig. 4. Typical fitness curve for the steady-state GP without SAW-ing (left) and with SAW-ing (right) on the Australian Credit data set (first fold).

Table 10. Results on the Australian Credit and Pima Indians Diabetes Datasets.

data set	Australian Credit	Pima Indians Diabetes
BackProp	0.154	0.248
C4.5	0.155	0.270
standard GP	0.232	0.253
LogDisc	0.141	0.223

As we can clearly see in Table 10 the standard GP performs poorly on the Australian Credit data set when compared to the other 3 techniques. The performance of the standard GP on the Pima Indians data set not too bad. It is close to back-propagation and much better than C4.5.

8 Conclusions and future research

The comparison of the atomic and the standard representation indicates a hard trade-off situation. Giving up the flexibility of numerical operators for the sake of transparency achieved by using Boolean functions in the bodies of the trees comes at costs of performance. It seems that there is no generally advisable option, the choice between the two representations has to be made on a case-by-case basis, depending on the priorities in the given problem context.

Looking at the differences between the effects of the SAW-ing mechanism we should distinguish between the steady-state and the generational models. Using the steady-state model the SAW-ing variant helping the GP to a better performance three times out of eight, is not able to provide a constant improvement. This is in contrast to the results found in constraint satisfaction [3, 4] and it needs further investigation to pinpoint the exact reason of this observation. Although, one observation shows that the GP version with SAW-ing show much fluctuation in the fitness, causing a performance decrease when the algorithm is stopped at the wrong time, i.e. when the best solution in the population does not perform well.

These experiments are just the beginning of an extensive study of using genetic programming for data mining purposes. Research in the near future will focus on the comparison of techniques in this paper and the standard GP using commercial data as a benchmark.

References

1. Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications.* Morgan Kaufmann, dpunkt.verlag, January 1998.
2. J. Eggermont, A.E. Eiben, and J.I. van Hemert. Adapting the fitness function in GP for data mining. In P. Nordin and R. Poli, editors, *Proceedings of Second European Workshop on Genetic Programming*, LNCS. Springer, Berlin, 1999. in press.
3. A.E. Eiben, J.K. van der Hauw, and J.I. van Hemert. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics*, 4(1):25–46, 1998.
4. A.E. Eiben, J.I. van Hemert, E. Marchiori, and A.G. Steenbeek. Solving binary constraint satisfaction problems using evolutionary algorithms with an adaptive fitness function. In A.E. Eiben, Th. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of the 5th Conference on Parallel Problem Solving from Nature*, number 1498 in LNCS, pages 196–205, Berlin, 1998. Springer.
5. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: elements of reusable object-oriented software.* Addison-Wesley, 1994.
6. J.I. van Hemert. Applying adaptive evolutionary algorithms to hard problems. Master's thesis, Leiden University, 1998. Also available as `http://www.wi.leidenuniv.nl/~jvhemert/publications/IR-98-19.ps.gz`.
7. J.R. Koza. *Genetic Programming.* MIT Press, 1992.
8. W.B. Langdon. *Genetic Programming + Data Structures = Automatic Programming!* Kluwer, 1998.
9. D. Michie, D.J. Spiegelhalter, and C.C. Taylor, editors. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, February 1994.
10. D. Whitley. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms (ICGA'89)*, pages 116–123, San Mateo, California, 1989. Morgan Kaufmann Publishers, Inc.