EuroGP'2000 Third European Conference on Genetic Programming, Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian F. Miller, Peter Nordin and Terence C. Fogarty editors, Edingburgh, 15-16 April, Springer-Verlag, LNCS 1802.

Seeding Genetic Programming Populations

W.B. Langdon[1] and J.P. Nordin[2]

[1] Centrum voor Wiskunde en Informatica, Kruislaan 413, NL-1098 SJ, Amsterdam
bill@cwi.nl http://www.cwi.nl/~bill
Tel: +31 20 592 4093, Fax: +31 20 592 4199

[2] Chalmers University of Technology, S-41296, Göteborg, Sweden
nordin@fy.chalmers.se http://fy.chalmers.se/~nordin
Tel: +49 31 772 3159, Fax: +46 31 772 3150

Abstract. We show genetic programming (GP) populations can evolve under the influence of a Pareto multi-objective fitness and program size selection scheme, from "perfect" programs which match the training material to general solutions. The technique is demonstrated with programmatic image compression, two machine learning benchmark problems (Pima Diabetes and Wisconsin Breast Cancer) and an insurance customer profiling task (Benelearn99 data mining).

1 Introduction

In every day speech we regard learning and remembering as somewhat similar. However *machine learning* means something very different from memorising. In machine learning we are not interesting in letting the computer memorise a number of facts, we know this is something it does very well. Instead we want our algorithm to find patterns in data, particularly patterns that enables generalisation on so far *unseen* data. We would like the computer find the "essence" of the information that we present to it.

Here we try and turn genetic programming (GP) on its head. Instead of asking GP to find a function which matches some training data and then seeing how well the evolved function generalises, we construct such a function before hand and give it to GP. We then run GP and see if it can evolve the function to be more general. That is GP starts from a solution rather than a random starting point. This is done by *seeding* the initial population with perfect individuals that can already solve the fitness cases. We thus skip the memorisation part and go right to the generalisation part.

We use simple deterministic algorithms to produce perfect individuals from the fitness cases used for training. There are many possible ways of doing this. In this paper (rather than creating initial programs at random) we assemble them from a large number of *if-then-else clauses* that either find a combination of input variables for each training case or we produce something like a decision tree where the desired output is narrowed down through interval tests on all of its input variables. The evolution is started with a *Pareto fitness function* [11] where multiple objectives are sought concurrently. In our case we want to

optimise (keep) a good performance while reducing the size of the individuals. In GP there is strong evidence of the link between small size and good generalisation capabilities [15,17]. So we need some kind of parsimony pressure. This is achieved by a Pareto tournament where individuals can win either by being good or being short. Our hope is that the parsimony pressure will replace the bulky if-then-else clauses with more elegant, short and general expressions.

Seeding of initial populations is not a very well studied technique in GP. It has been used in some financial applications [2], game playing [5], information retrieval [6,9], speeding evolution [11, 6.9], scheduling [11, C.9.4], image compression [14] and planning [1]. There has also been research in genetic algorithms for instance [3,4,18]. Here seeding is used not to find a good starting point for learning but to give a perfect individual and a good starting point for generalisation.

In Sect. 2 we describe our experiments on the programmatic image compression problem [7,14], where the objective is to evolve a program that reproduces a bitmap image when run. This is an extremely hard problem and although our method is possible, we had to abandon these experiments in favour of problems which need less computational power. We chose (Sect. 3) the North American Pima Indians diabetes problem. Despite starting with a very specific individual, GP was able to quickly boil the genetic soup down to very simple expressions. In one instance, to a program with only one input, see Fig. 1. This is a remarkable data mining and data selection feat. We suggest GP has an inherent capability to perform *variable selection* where by non-relevant variables are discarded during evolution. Finally we also report results from the Wisconsin Breast Cancer Database (9 attributes, Sect. 4) and from a real-world customer profiling problem with 85 attributes (Sect. 5).

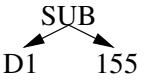


Fig. 1. Short program evolved by the second seeded Pima Diabetes run. It means *if Plasma glucose concentration at two hours in an oral glucose tolerance test exceeds 155.5 then the Diabetes test will be positive.* (All values in the training and verification sets are actually integers). This very simple program scores approximately as well as sophisticated machine learning techniques [12].

2 Programmatic Compression

We can view any system which creates programs which create data as a compression system if the programs are smaller than the data they create. In GP terms, the data to be compressed are fitness cases for symbolic regression. GP tries to evolve an individual program that outputs the uncompressed data, to a certain degree of precision. If the evolved program solution can be expressed by fewer bits than the target data, then we have achieved a compression. Here the data are the individual pixels of a picture (Fig. 2).

Fig. 2. Target image (256 × 167). Note variety of repeating patterns.

After 50 generations in a GP run (using crossover only) with Pareto selection and a population of 100, the seed program proved to be extremely robust. On average across the whole population each pixel was less than 1 grey level out, with 99.41% being exactly correct. However on average the seed had only shrunk from 275,083 to 273,259 i.e. by 0.7%. (Mean depth has increased from 18 to 19.9).

3 Pima Diabetes

This real-world medical classification problem is taken from the UCI Machine Learning repository `http://www.ics.uci.edu/~mlearn/MLSummary.html`. The target is to predict whether the patient shows signs of diabetes. Five sets of runs where conducted (with five independent runs for each experiment, details in Table 1). Three experiments used three different seeds and in another two the initial population was created at random.

All the seeds were created deterministically by creating a function composed of `IF` statements, one per training case (less one). Each `IF` tests all attributes and if they all match returns this training case's class. If not the else branch contains an `IF` relating to the next training case. And so on. If none of the `IF`'s match, the program returns the class of the last training case.

Terminals containing each attribute's value are compared with constant expressions using the APPROX primitive. APPROX returns true if its two arguments are within 10% of each other. The outputs of the eight APPROX functions per training case are combined using 7 AND functions. The seed generation program chooses the closest available constant to the actual attribute value in the training record. If none are within 10% then an expression combining two constants with ADD, SUB, MUL or DIV is used instead. Typically such expressions will occur more than once in the seed but no effort is made to avoid creation of duplicate code. As we expect these expressions are large, asymmetric, have high fitness on the training cases but fail to generalise.

Table 1. GP parameters for Pima Indians Diabetes (defaults are as [8, page 655])

Objective:	Find a program that predicts Diabetes.
Terminal set:	One terminal per data attribute (8). For each attribute, create unique random constants between its minimum and maximum value (in the training set). Use integers where the attribute has integer only values (total 255 primitives).
Function set:	IF IFLTE MUL ADD DIV SUB AND OR NAND NOR XOR EQ APPROX NOT
Fitness cases:	Training 576 (194 positive). 192 (74 positive) for verification only.
Fitness:	number correct (hits).
Selection:	2 objectives, hits and size, combined in Pareto tournament group size 7. Non-elitist, generational. Fitness sharing (comparison set 81) [11].
Wrapper:	Value ≥ 0.5 taken as true
Pop Size:	500
Max prog. size:	no limit
Initial pop:	100% seeded or created using "ramped half-and-half" [7, pages 92–93] (no duplicate checking).
Parameters:	90% one child crossover, 2.5% function point mutation (rate 100/1024), 2.5% constant mutation (rate 100/1024), 2.5% shrink mutation, 2.5% size fair mutation (subtree size$\leq$30) [10].
Termination:	Maximum number of generations G = 50

The first and smallest seed was generated using the first ten negative training case and the first ten positive. These twenty cases alternate, starting with the first negative training case. The second seed was generated using the first half, i.e. 288 cases, of the training set. While the last was generated from all 576 training cases.

Table 2 summarises the verification performance of the five approaches to the Pima problem. For each approach there are two rows. The top relates to the mean of the highest verification score found in each of the five runs, while the lower relates to the first occurrence of programs which score 137 or more. (We chose 137 to indicate satisfactory performance since it is one standard deviation below the best reported results on this problem). For each we report its actual performance "hits", its size (reporting the smallest where multiple programs have the same performance in the same generation) and the first generation when it evolved. In each category: the mean (of five runs), the standard deviation (in brackets) and the minimum and maximum are given.

GP starting from 500 random programs (i.e. no seed) and using a Pareto fitness measure did not perform well. Possibly because it converged too readily to small programs. When we removed the small size objective and just used scalar fitness, 40% of runs achieved reasonable performance. However 100% of runs starting with seeded populations did. It appears there is little actual information in the problem data as a seed constructed from only 20 training records contains enough information to start the GP process off in the right direction. The solutions found when using the two larger seeds are very much bigger. While we anticipate further evolution over more generations would eventually lead to a

Table 2. Highest verification score in run. 5 Pima runs with each seed

Seed size	Hits	Size	Generation	% runs ≥ 137
none	121.0 (4.0) 119–129	25.4 (23.8) 3–56	8.0 (6.6) 1–17	
(Pareto)				0
none	131.4 (7.6) 122–143	30.6 (20.8) 6–64	9.6 (4.8) 4–18	
(scalar)	138.5 (1.5) 137–140	4.5 (1.5) 3–6	17.0 (3.0) 14–20	40
20	143.4 (0.5) 143–144	34.2 (30.2) 3–91	35.4 (6.0) 28–43	
	139.4 (2.9) 137–143	3.0 (0.0) 3–3	26.0 (12.3) 14–42	100
288	144.0 (1.3) 142–145	1603.0 (2462.9) 21–6489	39.8 (6.9) 29–47	
	137.4 (0.8) 137–139	4.0 (2.0) 3–8	22.4 (6.5) 17–34	100
576	141.8 (2.4) 139–146	6369.4 (4603.7) 294–13479	40.6 (6.5) 31–50	
	137.0 (0.0) 137–137	62.4 (118.8) 3–300	32.8 (13.2) 18–50	100

reduction in size, we use GPQUICK where run time is O(|programs| $\times$ |test set|) [11, 8.8]. Thus runs starting from larger seeds are much slower than runs using shorter seeds or starting from random populations.

Figures 3, 4 and 5 plot the evolution of the first run with each of these three seeds. Figure 6 refers to a run starting from random. The graphs on the left hand side refer to training set performance. The two smaller seeds do not generalise to the remainder of the training set and consequently initially fitness is low (cf. gen 0), while the largest seed matches the whole training set and so scores 100%. All three fail to generalise and so have low scores on the verification set (right hand graphs).

The top graphs refer to the Pareto front, i.e. those individuals in the population which are not dominated by others. (Note we have two objectives, training set hits and reducing size). The Pareto front always spreads rapidly away from the initial seed as shorter programs are produced. With the two smaller seeds, there is also a rapid initial increase in fitness in the first generation but slow change after that. With the small seed (20) the population shows less signs of over training and the programs in it shrink. By the end of the run they are on average less than a tenth of the size of the initial seed. With the two bigger seeds, the Pareto front does not collapse in this way but instead the population continues to retain large programs which score well on the training set.

In the case of the two smaller seeds the best validation score in the whole population remains near (but slightly above) the best validation score of the individuals reported on the Pareto front. In the case of the biggest seed, these individuals are large and actually perform much worse on the validations tests than the best program in the population. (The size of these is plotted in the x-y plane of the top right hand graphs of Figs. 3, 4 and 5).

In each case within a few generations programs which score above 70% on the verification set are found. Slow improvements continue so that, by the end of the run, scores (75%) comparable with the best models produced by many machine learning techniques on this problem have been found.

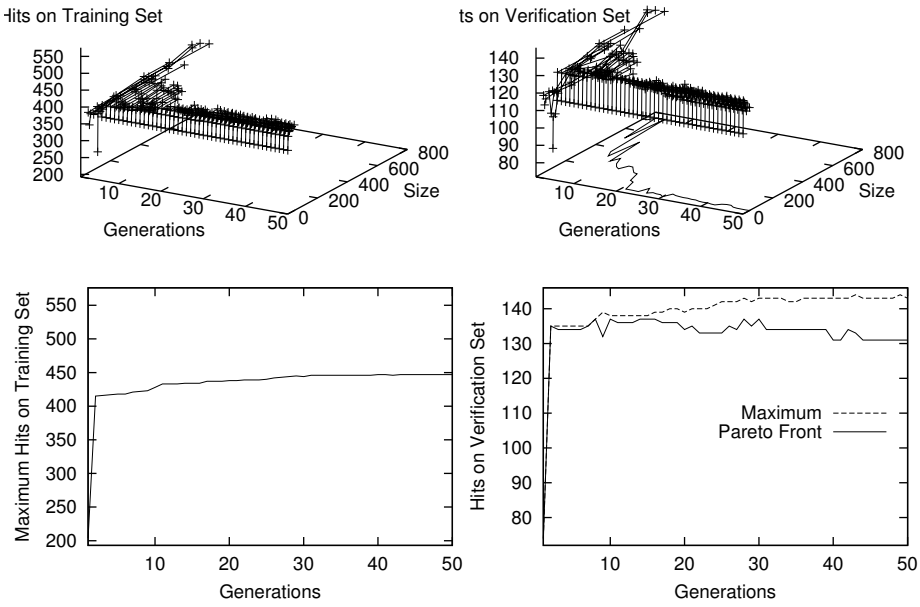


Fig. 3. Evolution of Pareto front in first seeded (20) Pima run

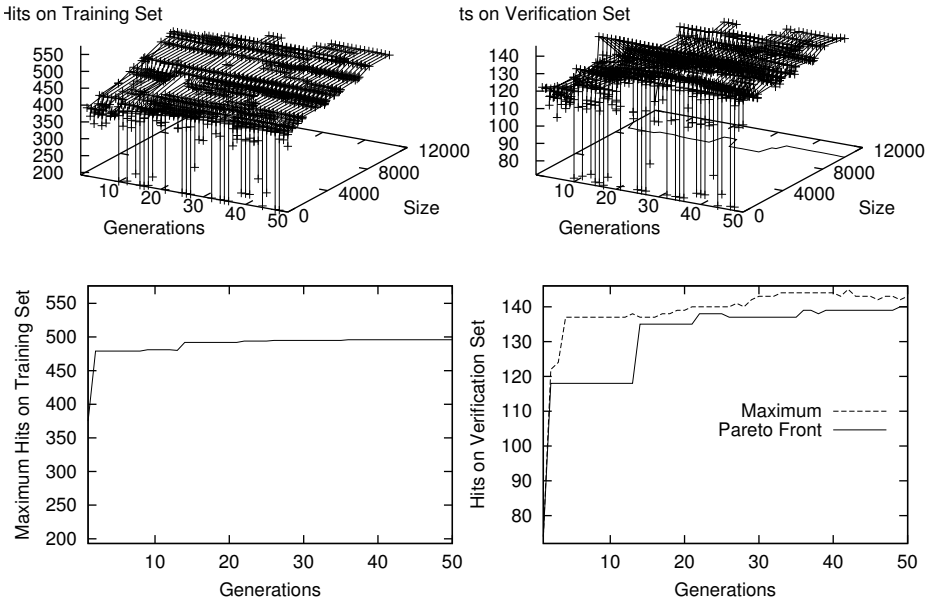


Fig. 4. Evolution of Pareto front in first seeded (288) Pima run

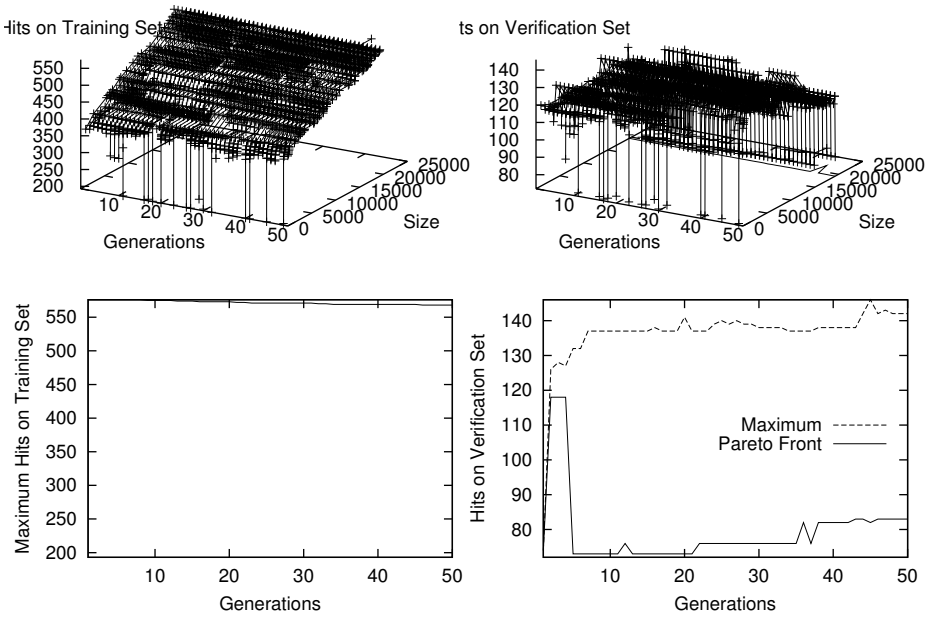
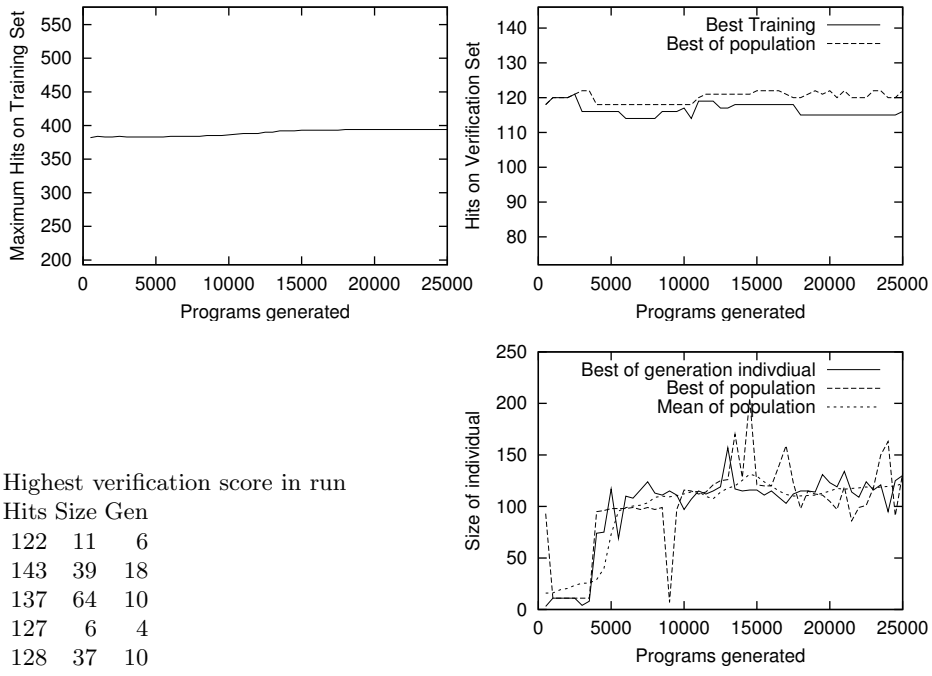


Fig. 5. Evolution of Pareto front in first seeded (576) Pima run



Highest verification score in run

Hits	Size	Gen
122	11	6
143	39	18
137	64	10
127	6	4
128	37	10

Fig. 6. Evolution of Pima population in non-seeded run with scalar fitness function

Table 3. Wisconsin Breast Cancer (as Table 1 except were given)

Objective:	Find a program that predicts Breast Cancer.
Terminal set:	One terminal per data attribute (9). Constants -1 to 10
Fitness cases:	351 (130 positive). 348 (111 positive) verification tests.

Table 4. Cancer. Highest verification score on Pareto front (5 runs on each seed)

Seed size	Hits	Size	Generation	% runs ≥ 326
none	333.2 (4.4) 326–338	22.6 (21.7) 5–65	28.4 (17.0) 3–50	
(Pareto)	328.2 (2.4) 326–332	6.2 (1.6) 5–9	14.2 (11.1) 3–35	100
20	329.8 (1.2) 329–332	101.4 (122.2) 19–342	34.0 (10.1) 20–49	
	327.0 (1.3) 326–329	65.0 (95.8) 5–256	32.0 (10.9) 16–47	100
128	327.0 (2.9) 323–332	46.4 (38.1) 20–122	38.8 (8.1) 29–47	
	327.2 (1.1) 326–329	42.0 (29.1) 20–92	38.0 (9.2) 29–50	80
351	327.4 (2.4) 323–330	6422.0 (2610.8) 3703–11257	41.8 (2.7) 37–45	
	326.5 (0.5) 326–327	2621.8 (2197.1) 871–6390	46.5 (6.1) 36–50	80

4 Wisconsin Breast Cancer

Like the Pima Diabetes, Wisconsin Breast Cancer is a real-world binary medical classification problem often used as a machine learning benchmark. Again three seeds were constructed in the same way as in Sect. 3. Details of the GP parameters are given in Table 3, while Table 4 summarises the verification results of five independent runs starting with each seed. We choose a score of 326 out of 348 on the verification set as indicating satisfactory performance. (State of the art ML performance is about 96% on this problem, so 326 is 2σ below it). Table 4 indicates GP can obtain satisfactory performance without seeding. Indeed the large size of programs in the initial population created from the two bigger seeds appears to hamper GP in this problem.

5 Insurance Customer Profiling

Given 85 attributes relating to a customer, such as age, number of children, number of cars, income, other insurance policies they hold, predict if they want caravan insurance. 5922 records (of which 343 are positive and the rest negative) are available as training data from `http://www.swi.psy.uva.nl/benelearn99/comppage.html` as part of Benelearn'99. The task is to find 800 records of a further 4000 records which contain as many positive examples as possible.

We conducted two experiments: 1) GP starting from a random population and 2) the initial population was filled with copies of a seed created by C4.5 release 8 [16]. (The unpruned C4.5 trees produced using default parameter settings were used). The 5922 records were randomly split in half. One half (with 179 positive examples) was used as the training set and the other half was used as a verification set. The details are given in Table 5.

Table 5. Insurance Customer Profiling (as Table 1 except were given)

Objective:	Find a program that predicts the most likely $1/5^{th}$ of people to become caravan insurance customers.
Terminal set:	One terminal per data attribute, 0..41 and 110 different random numbers uniformly selected from 0..10 (total 255 primitives)
Function set:	IF IFLTE MUL ADD DIV SUB AND OR NAND NOR XOR EQ APPROX GTEQ LTEQ GT LT NOT
Fitness cases:	2911 (179 positive)
Hits:	number of positive cases predicted
Fitness:	At end of each generation each positive fitness case given a weight equal to the reciprocal of the number of individuals which correctly predicted it. Fitness given by sum of weights of positive cases predicted.
Wrapper:	2911 values sorted, top 1/5 (583) treated as positive predicted
Pop Size:	100, 1000, 5000, 20000
Max prog. size:	no limit
Initial pop:	"ramped half-and-half" depth 5–9 Or 100% seeded with C4.5 decision IFLTE tree
Parameters:	90% one child crossover, 5% point mutation (rate 10/1024), 5% size fair mutation
Termination:	Maximum number of generations G = 50

Figure 7 shows the distribution of performance on both the training and verification sets in the final population. The population changed dramatically from the initial seed. As expect the initial seed performs reasonably well on the training set (from which was created) but only slightly better than random guessing on the verification set. By the end of the run the population is widely spread. On the training set, most of the population lies close to the Pareto front (solid line). Since GP has discovered programs that are shorter and/or fitter than the seed, the front is well to the left of it. After 50 generations the population has bloated as is indicated by the cluster of points at the top left. These programs are long and have high training scores but they do not score markedly more than shorter programs in earlier generations. In fact while almost all programs longer than the seed (681) score better than it on the training case, they are worse than it on the verification set, scarcely better than random guessing. That is the population contains long programs which are heavily over trained.

The performance on the verification set of the best members on the training set is also plotted (lower solid line). Above size ≈ 200 many programs perform above the line on the verification set. I.e. many programs do better on the verification set than the best (on the training set) programs.

The lower dotted line shows the best 1% of the population on the verification set. This line is almost flat, Showing that bigger programs give little if any real performance advantage. Indeed in this population programs as short as five appear to be the best. The upper dotted line shows the performance on the training set of the same 50 programs. Not surprisingly it is more erratic than the lower curve and climbs with programs size, again indicating over training.

It is clear that GP has been able to generalise the initial seed program and thereby considerably improve its performance. However there are also clear signs of over training and this increases with size of programs in the population.

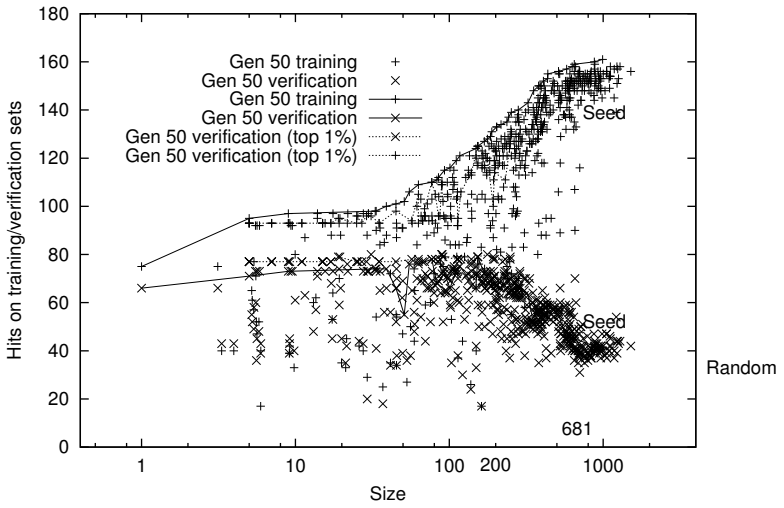


Fig. 7. Distribution of training (+) and verification (×) performance in generation 50 of the first seeded run of the customer profiling problem. Solid lines indicate individuals on the Pareto (training, size) front, while dotted lines indicate the best 1% of the population on the verification set. Note log scale.

Figure 8 show the final population of an unseeded GP run. The best performance on the training set is much lower than for the seeded run. However the population shows only a little sign of over training, and there is less variation with size. The verification performance is approximately the same as the seeded run, Fig. 7.

6 Conclusions

We have seen that it is indeed possible to start GP with non-random populations constructed from perfect individuals and use evolution under parsimony as a way to find good generalisers.

However constructing programs which memorise the training data (as used in Sects. 2, 3, 4 and 5) gives rise to large and unwieldy seeds if the whole training set is used. In our approach GP took many generations to reduce these to general solutions. Sections 3 and 4 show it is not necessary to use the whole training set and sometimes a very small sample of it will do.

GP is unique compared to other soft-computing technique in that it relies on symbolic structures in individuals and population. It is this property that enables

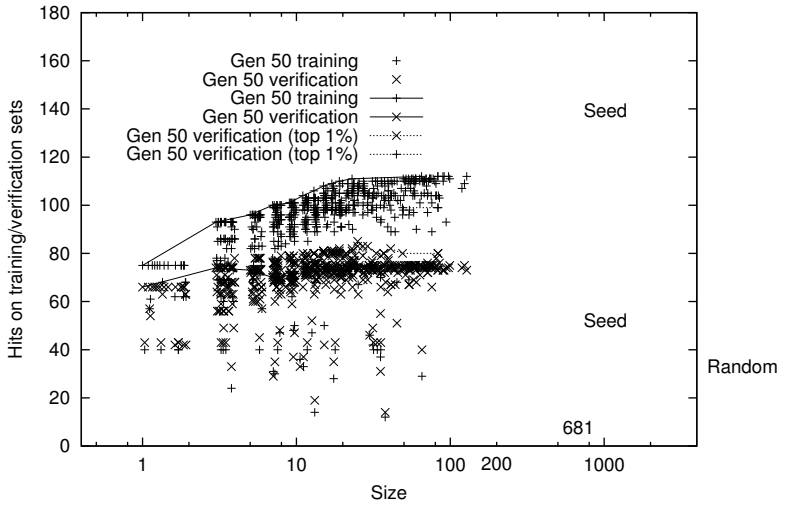


Fig. 8. Distribution of training (+) and verification (×) performance in the final generation of a non-seeded run of the customer profiling problem. Solid lines indicate individuals on the Pareto (training, size) front, while dotted lines indicate the best 1% of the population on the verification set. Note log scale.

seeding of perfect or near perfect starting points. As we saw in Sect. 5, the symbolic nature of GP-individuals also makes the integration of results from other techniques possible and opens possibilities for all sorts of hybrid approaches.

Our studies of populations evolved using Pareto multi-objective size versus performance selection clearly shows the often assumed relationship between long programs and poor generalisation.

In the light of the large amount of previous work trying to make GP memorise examples from Boolean functions or complex multidimensional data, it might seem a bit provocative to have a perfect individual from the start, but the approach turns out to be feasible. We think that this is because GP has a built in ability to generalise [11,13,15].

Acknowledgments

Funded by in part by the Wennergren foundation, NUTEK and TFR.

References

1. Ricardo Aler, Daniel Borrajo, and Pedro Isasi. Genetic programming and deductive-inductive learning: A multistrategy approach. In Jude Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning, ICML'98*, pages 10–18, Madison, Wisconsin, USA, July 1998. Morgan Kaufmann.

2. Shu-Heng Chen and Chia-Hsuan Yeh. Using genetic programming to model volatility in financial time series: The case of nikkei 225 and S&P 500. In *Proceedings of the 4th JAFEE International Conference on Investments and Derivatives (JIC'97)*, pages 288–306, Aoyoma Gakuin University, Tokyo, Japan, July 29-31 1997.
3. E. William East. Infrastructure work order planning using genetic algorithms. In Wolfgang Banzhaf *et. al.*, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1510–1516, 1999. Morgan Kaufmann.
4. Jacob Eisenstein. Genetic algorithms and incremental learning. In John R. Koza, editor, *Genetic Algorithms and Genetic Programming at Stanford 1997*, pages 47–56. Stanford Bookstore, Stanford, California, 94305-3079 USA, 17 March 1997.
5. Gabriel J. Ferrer and Worthy N. Martin. Using genetic programming to evolve board evaluation functions for a boardgame. In *1995 IEEE Conference on Evolutionary Computation*, volume 2, page 747, Perth, Australia. IEEE Press.
6. Paul Holmes. The odin genetic programming system. Tech Report RR-95-3, Computer Studies, Napier University, Craiglockhart, Edinburgh, EH14 1DJ, UK, 1995.
7. John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.
8. John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.
9. D. H. Kraft, F. E. Petry, W. P. Buckles, and T. Sadasivan. The use of genetic programming to build queries for information retrieval. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, pages 468–473, 1994.
10. W. B. Langdon. The evolution of size in variable length representations. In *1998 IEEE International Conference on Evolutionary Computation*, pages 633–638.
11. William B. Langdon. *Data Structures and Genetic Programming: Genetic Programming + Data Structures = Automatic Programming!* Kluwer, 1998.
12. Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 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, 1995. Morgan Kaufmann.
14. Peter Nordin and Wolfgang Banzhaf. Programmatic compression of images and sound. In John R. Koza *et. al.*, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 345–350, 1996. MIT Press.
15. Peter Nordin, Wolfgang Banzhaf, and Frank D. Francone. Compression of effective size in genetic programming. In Thomas Haynes *et. al.*, editors, *Foundations of Genetic Programming*, GECCO'99 workshop, 13 July 1999.
16. J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.
17. Justinian Rosca. Generality versus size in genetic programming. In John R. Koza *et. al.*, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 381–387, 1996. MIT Press.
18. Amir M. Sharif and Anthony N. Barrett. Seeding a genetic population for mesh optimisation and evaluation. In John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1998 Conference*, 1998. Stanford University Bookstore.