2nd International Conference on Autonomous Robots and Agents
December 13-15, 2004 Palmerston North, New Zealand

Comparing the performance of incremental evolution to direct evolution

Matthew Walker
Institute of Information and Mathematical Sciences,
Massey University, Auckland, New Zealand
M.G.Walker@massey.ac.nz

Abstract

The performance of incremental evolution is compared to that of direct evolution. A multi-variable symbolic regression problem is used. Incremental evolution is not found to outperform direct evolution when the full computational cost is considered.

Keywords: incremental evolution, direct evolution, seeding, symbolic regression

1 Introduction

1.1 Direct Evolution

Direct evolution is the canonical form of evolution in genetic programming (GP) [1]. First, a randomly generated population is created. This population is evolved over a number of generations to become better at a given (goal) problem.

1.2 Incremental Evolution

Incremental evolution [2] is the iterative use of an evolutionary search. Given a final (difficult) goal, a series of stages is laid out that starts from an easy problem, goes through intermediate problems, and ends at the goal. Direct evolution is applied to the first (easy) problem. The resulting population is used to seed the initial population of the next stage. This seeding continues through the stages until, finally, a solution is found to the goal problem.

Incremental evolution was first applied to the evolution of a robotic controller [2]. That problem domain is still by far the technique's most common use [3–7].

It is commonly accepted that incremental evolution is beneficial when direct evolution fails to find a solution to the goal problem. However, it is uncommon that the cost of the intermediate evolutionary runs is considered when incremental evolution is compared to direct evolution. An example of such unfair comparison is in the work by Barlow *et. al* [3].

This research looks to compare the performance of incremental evolution with direct evolution. The full computational cost of each technique is considered. One problem domain is used, symbolic regression.

1.3 Symbolic Regression

Symbolic regression is the discovery of the underlying formula to a collection of observed values. It was well covered in Koza's first book on genetic programming [1].

2 Hypothesis/Aim

The aim of this work was to investigate incremental evolution when applied to symbolic regression. The hypothesis was that incremental evolution has a higher probability of success than direct evolution.

Previously, it has been uncommon for the full computational cost of incremental evolution to be assessed. In his doctoral thesis, Harvey commented on the full cost of incrementally evolving a solution to the travelling salesman problem (TSP) [8]. He said no gain was obvious from the use of incremental evolution as against direct evolution, but that incremental evolution had found solutions to the problem in every stage, while direct evolution only worked on the goal problem. Thus, incremental evolution solved more problems, and so had done more work, for the same computational cost.

It was expected that incremental evolution would provide the greatest performance benefit across the "cliff" where direct evolution starts to fall (see figure 1). The incremental runs that started where direct evolution was successful and continued over the cliff were expected to outperform direct evolution.

3 Symbolic Regression Problem

For these experiments a multi-variable symbolic regression problem was used.

The solution was $\sum_{i=1}^{n} c_i x_i$,

where $c_i \in [-5,5)$ and $x_i \in [-1,1)$. The values of c_i were randomly set before the start of each run, except that the intermediate and goal incremental stages used the values from the stage before them.

3.1 Training Setup

The problem for the evolutionary system was to learn to solutions structure and the values of c_i. In order to assess an individual's fitness, sets of x_i were used as training cases. $10n$ sets were used during training. All the values of x_i were randomly assigned at the start of the evolutionary run.

3.2 Verification Setup

An evolutionary run was considered a success if at least one individual in its population was a program that sufficiently modelled the solution.

To test if a population produced an acceptable program, the top three individuals in the final population (as ranked during training) were selected for verification. Each individual was subjected to $200n$ verification tests.

Each test compared the difference between the known solution and the value produced by the given individual. The mean and sample variance of these tests was calculated. If the upper value of the 95% confidence interval was less than 1 for any of the three individuals, then the program was considered sufficiently accurate and the evolutionary run considered a success.

If the 95% confidence interval included 1, then the individual in question was retested with $2000n$ new tests.

If the mean of the $2000n$ tests was below 1 then the individual was considered sufficiently accurate and the evolutionary run considered a success.

If all the upper values of the 95% confidence intervals were above 1, or the means of the $2000n$ tests were all above 1, then the top three individuals were considered not to sufficiently model the solution, and so the evolutionary run was considered a failure.

4 Method

Two sets of experiments were run: those that used direct evolution and those that used incremental evolution.

4.1 Common Details

Genetic Programming (GP) [1] was the used as the evolutionary system.

Common to both setups were the available functions and terminals. Arithmetic functions were made available (that is, add, subtract, multiply, and protected division [1]). The available terminals were a set of real-valued ephemeral random constants (which were kept the same through each of the incremental stages) as well as x_i where $i \in [1,n]$ which held the values for each test case.

Open Beagle [9,10] was used as a GP kernel. Its default settings (as at version 2.1.3) were kept unless otherwise stated. A population size of 1000 was used throughout this research.

As already mentioned, $10n$ tests were executed. (This was decided by a series of experiments that are not discussed here.)

Thus, for a direct evolution run at 15 variables, each individual would complete 150 tests per fitness evaluation. Over 200 generations that would mean 30 million tests.

For an incremental run that started at 11 variables with a goal of 15 variables, 5 stages would be completed. At 40 generations per stage, 200 cumulative generations would have been completed by the termination of the goal stage. However, only 26 million tests would have been completed.

This variation in the number of tests was included firstly to reduce the computational burden, but secondly it was an attempt to ensure a fair comparison. It is known that a more accurate fitness test improves the probability of finding a solution. Scaling the number of fitness tests by the number of variables in the current problem was an attempt to remove any potential bias.

4.2 Direct Evolution

The experiments in direct evolution ranged from 5 to 35 variables (n). Each run went for 1000 generations. On average, 32 runs were completed for each value of n that was sampled.

4.3 Incremental Evolution

31 experiments were run with incremental evolution.

In every experiment the number of generations per stage was fixed. After the allocated number of generations had passed, the *entire* population was used as a seed for the next stage, even if the population failed the verification test.

Eight values were used for the number of generations per stage: 10, 20, 30, 40, 50, 100, 150, and 200. The number was fixed to allow for an easier comparison with direct evolution.

Setup	Start	Goal	Step
1	5	15	1
2	10	20	1
3	15	25	1
4	20	30	1
5	5	25	2
6	10	30	2

Table 1: Setup details

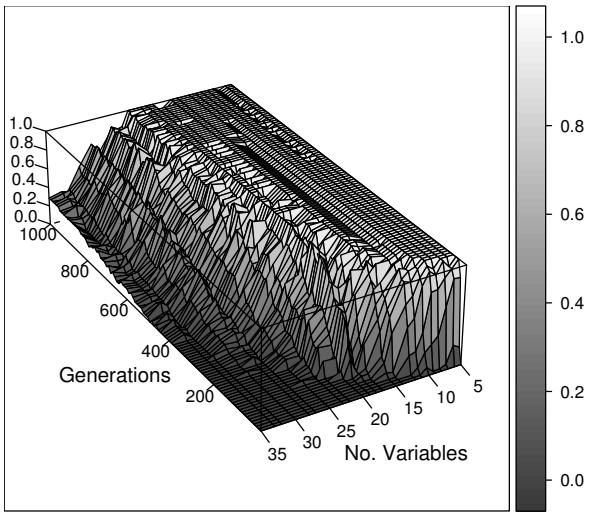

Figure 1: Direct evolution: Proportion of successful runs (vertical axis) against number of variables in problem (n) and number of generations.

Six experimental setups were used. Each setup had ten intermediate stages before the goal stage. In the majority of experiments, successive stages has just one variable added (that is, n increased by 1 per stage). In four experiments successive stages had two variables added. Table 1 lists the details of each of the experimental setups including, the number of variables that the first stage had ("Start"), the number of variables in the goal stage ("Goal"), and the number of variables that were added for each new stage ("Step").

5 Results

5.1 Direct Evolution

The results of the direct evolution runs are shown in figure 1. Experiments with $n \leq 15$ always found a solution within 200 generations. Experiments with $n > 22$ did not always find a solution, even after 1000 generations.

5.2 Incremental Evolution

The full computational cost of incremental evolution is the total number of fitness evaluations performed. To calculate this, the fitness evaluations performed for each stage must be summed. Because the number of fitness evaluations per generation is constant in these experiments, the cumulative number of generations may be used as an equivalent comparison. Thus, it is fair to compare the performance of direct evolution that has completed 100 generations against an incremental system that has performed 10 stages with 10 generations per stage.

Figure 2 compares the results obtained with incremental evolution (black) against the prediction of direct evolution's performance (red/grey) for six experimental setups and up to eight variations in the number of generations per stage.

In each frame, the black line plots the performance of incremental evolution for one experimental setup at the given number of generations per stage. For every incremental evolution experiment, 11 stages were completed, and each sampled up to 50 times, the data point for a given stage represents the proportion of samples that produced a successful population (as defined in section 3.2).

Also plotted in each frame is the success proportion of direct evolution for the equivalent number of generations. This allows for a fair comparison of the two strategies.

Many of the experiments were sampled 50 times. For those that were not, the mean number of runs that were performed is displayed in the lower right corner of its frame.

6 Analysis

It was expected that incremental evolution would be obviously beneficial just on the cusp of direct evolution's failure. For 200 generations, direct evolution's success proportion is 1.0 at 15 variables. But at 20 variables that value is down to 0.5. So it was expected that an incremental system would outperform direct evolution if its goal was 20 variables and it attained that goal within 200 cumulative generations.

At the other end of the spectrum, direction evolution with 200 generations has no difficulty in almost guaranteeing a solution if the number of variables is below fifteen.

So, it was expected that an incremental run that started at 10 variables and went to 20 would be more successful than direct evolution. Setup 2 with 20 generations per stage (hereafter specified as $(2,20)$) demonstrates this scenario. However, as can be seen in figure 2, the performance of incremental evolution dropped significantly below that of direct evolution for the same computational cost.

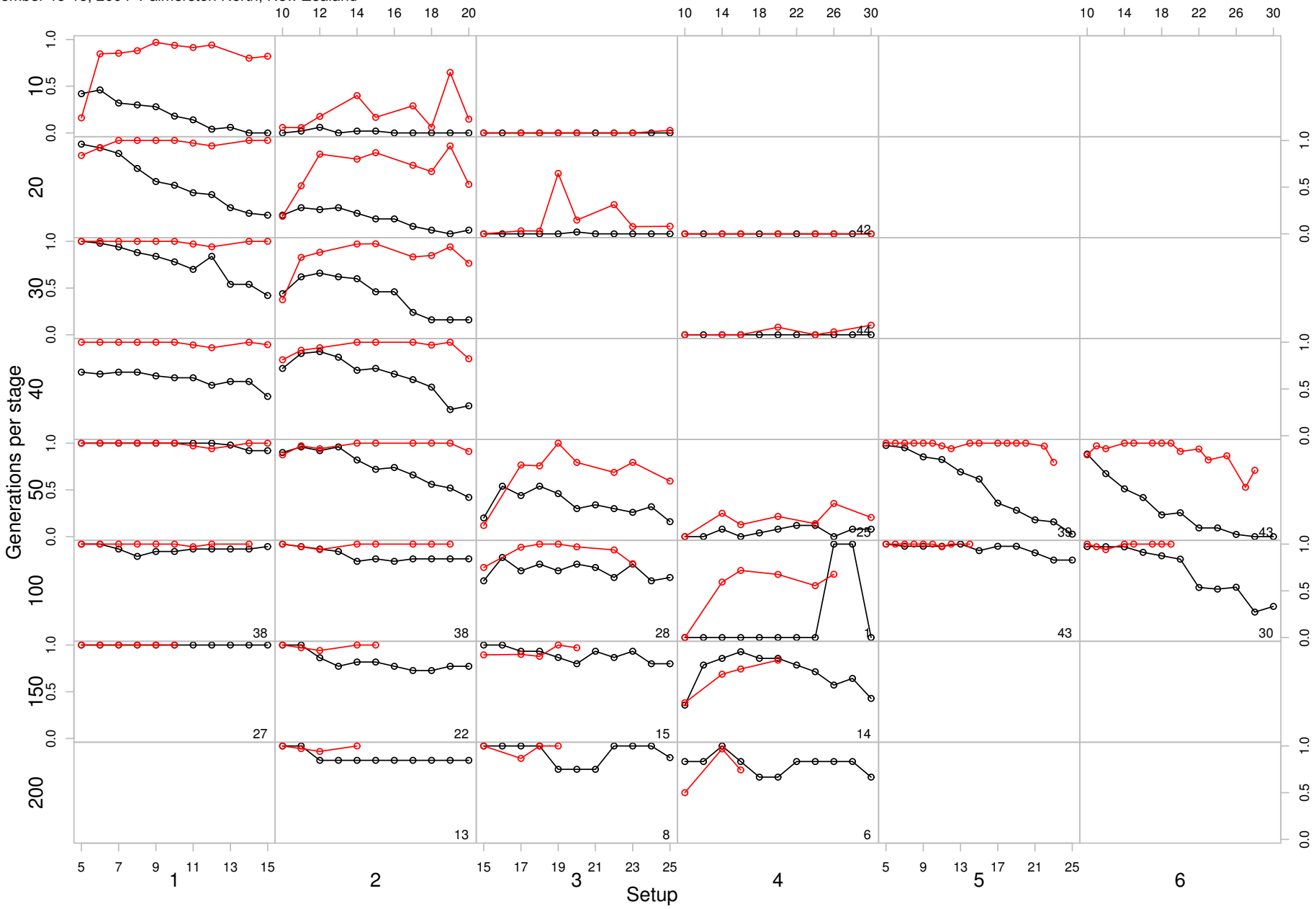

Figure 2: Comparison of incremental evolution (black) with direct evolution (red/grey). Each frame plots success proportion against number of variables.

It could be argued that this is not a fair comparison. The incremental runs were allowed hardly any time on the simple problems and quickly headed into the difficult problems without having attained sufficient ability in the easy problems.

(2,200) and (2,150) both consider this line of argument. Although the difference is not statistically significant, incremental evolution's performance doesn't match direct evolution's.

(2,50) shows a scenario somewhere between; that is, 50 generations per stage is between 20 and 150. Here, direct evolution outperforms incremental evolution.

Direct evolution with 200 generations fails completely around 25 variables. Experiment (4,50) compares direct evolution at 23 variables for 200 generations, but no significant difference can be seen. The concern was that starting at 20 variables was too difficult.

(5,50) starts the incremental run down at 5 variables, but the task of learning two variables at a time must be too difficult, as the performance decreases from the second stage.

(4,150) also demonstrates a typical curve seen throughout many of these incremental evolution experiments. The first few stages almost always see a continued performance increase that later dies. This is an unexpected trend given that the amount to learn when transitioning from a problem with m variables to $m+1$ variables (that is, one stage to the next) does not increase as m increases. However, the curve does show that even stages that failed the verification tests proved to be of some benefit as seeds to the later stages.

From the experiments conducted in this research it would be impossible to conclude that, for equivalent computational costs, incremental evolution gives a higher probability of finding a solution. Therefore the hypothesis from section 2 remains unproven.

7 Discussion

Although this research provides evidence that significantly rejects the hypothesis, it has a number of limitations.

The first is that this research considers only one problem domain. To really conclude that incremental evolution offers no computational benefit, it would have to be demonstrated over a number of problem domains. Such research is planned.

The second limitation is that it could be claimed better performance would be found if the number of generations per stage were allowed to vary. That is, on reaching a solution, the stage could terminate without regard to some fixed number of generations. In this research the number of generations per stage was fixed.

There is a counter-argument however. Time spent honing a solution to one stage need not be wasted. For a population to be considered acceptable, it need not have a perfect individual. The extra time could be spent further reducing the error, thus making the next stage easier.

The argument against that is that a population may converge and the individuals may become too similar to one another. This lack of diversity was discussed in the work of Barlow *et. al* [3]. However, the work they indirectly cite [5] does not seem to support their concerns. Reduced diversity could however be a cause of the poor performance of some of the incremental runs. Further analysis would be required to assess this.

A third limitation is that no attempt was made to vary the proportion of the population that made up the seed for the next stage. Other research has considered the benefits of the introduction of a proportion of randomly generated individuals, but with mixed conclusions [11–13].

Alternative incremental topologies provide further interest as, currently, incremental evolution is defined only as a linear method. The possibilities of a hierarchical method and the addition of concurrency offer new research opportunities.

References

[1] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[2] Inman Harvey, Phil Husbands, and Dave Cliff. Seeing the light: Artificial evolution, real vision. In J. A. Meyer D. Cliff, P. Husbands and S. Wilson, editors, *From Animals to Animats 3, Proceedings of the Third International Conference on Simulation of Adaptive Behavior (SAB94)*, pages 392–401, Brighton, England, 8–12 August 1994. MIT Press. Available at www.cogs.susx.ac.uk/users/inmanh/ (accessed 16 Jan 2004).

[3] Gregory J. Barlow, Choong K. Oh, and Edward Grant. Incremental evolution of autonomous controllers for unmanned aerial vehicles using multi-objective genetic programming. In *Late Breaking Papers of the Genetic and Evolutionary Computation Conference (GECCO); Seattle*, 2004. Available on CD only.

[4] William H Hsu, Scott J Harmon, Edwin Rodríguez, and Christopher Zhong. Empirical comparison of incremental reuse strategies in genetic programming for keep-away soccer. In *Late Breaking Papers of the Genetic and Evolutionary Computation Conference (GECCO); Seattle*, 2004. Available on CD only.

[5] Faustino Gomez and Risto Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behaviour*, 5(3/4):317–342, 1997. Available at citeseer.ist.psu.edu/gomez97incremental (accessed 24 July 2004).

[6] D. Filliat, J. Kodjabachian, and J.-A. Meyer. Incremental evolution of neural controllers for navigation in a 6-legged robot. In *Proceedings of the Fourth International Symposium on Artificial Life and Robotics (AROB99)*, Oita, Japan, 19–22 January 1999. Oita University Press. Available at citeseer.nj.nec.com/filliat99incremental and animatlab.lip6.fr/Filliat/index.en.html (accessed 1 Mar 2004).

[7] J Chavas, Christophe Corne, P Horvai, Jérôme Kodjabachian, and Jean-Arcady Meyer. Incremental evolution of neural controllers for robust obstacle-avoidance in khepera. In *Proceedings of the First European Workshop on Evolutionary Robotics*, Lecture Notes in Computer Science, pages 227–247. Springer-Verlag, 1998. Available on citeseer and ACM.

[8] Inman Harvey. *The Artificial Evolution of Adaptive Behaviour*. PhD thesis, University of Sussex, 1995. Available from www.cogs.susx.ac.uk/users/inmanh/.

[9] Christian Gagné and Marc Parizeau. Open BEAGLE: A new versatile C++ framework for evolutionary computation. In *Late-Breaking Papers of the Genetic and Evolutionary Computation Conference (GECCO); New York City*, pages 161–168, 2002. Available at www.gel.ulaval.ca/~beagle/ (accessed 3 Dec 2003).

[10] Open BEAGLE: A software library for evolutionary computation. Available at www.gel.ulaval.ca/~beagle/ and beagle.sourceforge.net (accessed 13 Jan 2004).

[11] John J. Grefenstette. Incorporating problem specific knowledge into genetic algorithms. In Lawrence Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 42–60. Morgan Kaufmann, 1987.

[12] P. J. Darwen and X. Yao. On evolving robust strategies for iterated prisoner's dilemma. In Xin Yao, editor, *Progress in Evolutionary Computation*, volume 956 of *Lecture Notes in Computer Science*, pages 276–292. Springer-Verlag, Heidelberg, Germany, 1995. Available at citeseer.nj.nec.com/darwen95evolving.html (accessed 18 Jan 2004).

[13] René Thomsen, Gary B. Fogel, and Thiemo Krink. Improvement of clustal-derived sequence alignments with evolutionary algorithms. In *Proceedings of the 2003 Congress on Evolutionary Computation*, volume 1, pages 312–319, Canberra, Australia, 8–12 December 2003. IEEE Press.