Population Implosion in Genetic Programming

Sean Luke and Gabriel Catalin Balan

Department of Computer Science, George Mason University
4400 University Drive MSN 4A5, Fairfax, VA 22030, USA
{sean, gbalan}@cs.gmu.edu
http://www.cs.gmu.edu/∼eclab

Abstract. With the exception of a small body of adaptive-parameter literature, evolutionary computation has traditionally favored keeping the population size constant through the course of the run. Unfortunately, genetic programming has an aging problem: for various reasons, late in the run the technique become less effective at optimization. Given a fixed number of evaluations, allocating many of them late in the run may thus not be a good strategy. In this paper we experiment with gradually decreasing the population size throughout a genetic programming run, in order to reallocate more evaluations to early generations. Our results show that over four problem domains and three different numbers of evaluations, decreasing the population size is always as good as, and frequently better than, various fixed-sized population strategies.

1 Introduction

In generational EC, the choice of how to allocate evaluations may be cast as a tradeoff between exploration and exploitation. A maximally large population, run for a single generation, is an extreme in exploration, approximating random search. On the other hand, depending on the locality of the modification operator, a minimal (2 person) population exploits local features to the point of hill-climbing.

Choosing a good population size vs. runlength tradeoff is a long-explored topic in evolutionary computation. This choice has traditionally hinged on two factors: theoretical justifications for a given population size; and practical necessity for capping population size, primarily because of memory capacity. Recent advances in computer technology have obviated much of the second factor: genetic programming has occasionally seen population sizes around a million individuals divided among multiple machines in Beowulf clusters [1, 2]. Contrast this to early $1 + 1$ evolution strategies techniques which used a population size of 2!

The most common runlength for a genetic programming problem is a very short 50 generations, and the standard population sizes have ranged from 500 to 2000. The reason for these layouts is mostly tradition: they were the layouts popularized by Koza in [3]. Koza established these layouts through trial and error, and his decisions seem to be justified for GP problems. As was discussed in [4], it appears that many of the canonical test problem domains for GP have a surprisingly short maximum useful runlength. In the Symbolic Regression domain, for example, beyond 32 generations it becomes advantageous to split the run into two or more shorter runs.

1.1 Population Implosion

This paper examines the effects of decreasing the population size towards zero during the course of a genetic programming run. Early in the run the system would evolve with a large population of individuals, but late in the run the system would be effectively reduced to hill-climbing. This idea was inspired by simulated annealing, which also explores early and exploits (hill-climbs) late in the run, albeit in a very different fashion. We imagined that this "population implosion" would be effective because it would reallocate evaluations away from late generations where the population would have mostly stagnated anyway.

Decreasing the population size has another unforseen advantage in genetic programming: it counters the effects of bloating on memory consumption. The total memory consumed by the system is a function largely of the size of the average individual times the number of individuals in the population. As the first factor goes up, a drop in the other factor may maintain current memory consumption levels. However, we do not consider this issue in the experiments in this paper.

We define a *layout* as a choice of population size and runlength in order to allocate N evaluations for a run. In generational evolutionary computation, layouts are almost always rectangular, with the minor exception of the $\mu + \lambda$ evolution strategy, which may have a small initial generation. In this paper we have chosen to decrease the population using a *diagonal* layout. The diagonal layout starts with a large population, then linearly decreases the population size each generation until it reaches 0.

2 Previous Work

Setting the population size has been a challenging issue in the EC domain for a very long time. Initial work concentrated on deciding the optimal number of individuals in fixed-sized populations. Interesting research has recently suggested methods for adapting the size of the population during the search process, depending on various online parameters like fitness improvement or size of individuals.

In [5], De Jong investigated the influence of population size, mutation, and crossover rates on the efficiency of the search process. He also suggested parameter values that showed good performance in his experiments. Later, Grefenstette used a meta-GA to find good parameters for the search process and recommended a smaller population size than the one suggested by De Jong [6]. Grefenstette's results were later supported by theoretical investigations [7]. [8] provides a theoretical and empirical investigation on the relation between population size and crossover probability, while [9] presents a theoretical analysis meant to determine how to set the population size in order to promote the selection of correct building blocks. More recently, [10] describes another theoretical investigation of the impact of the population size on the performance of the GA algorithms in the OneMax problem domain, and [11] finds statistical significant differences when using different population sizes and classifier lengths in an SCS/LCS system.

Some studies have also raised the possibility of modifying the population size during the search process. [12] introduces the GAVaPS algorithm, which assigns fitness-dependent lifetimes to individuals in a steady-state EC system. Individuals are removed

from the population only when their lifetime is exceeded. Another interesting approach is presented in [13], where an adaptive mechanism adjusts the population size as a way to control selective pressure.

Population size has been studied in the GP domain as well. Koza ([3]) advocated using large population sizes, but small populations have also been espoused [14]. An interesting approach is reported in [15], where the number of *tree nodes*, not individuals, in the population is kept constant in order to prevent bloating. This approach is reported to give similar results to the standard method, however it reduces the use of computational resources.

3 First Experiment

In our first experiment, we compared four layouts against three different numbers of evaluations and four different genetic programming problem domains. The problem domains were Symbolic Regression, Artificial Ant, 5-bit Parity, and 11-bit Boolean Multiplexer. We chose to include three choices of number of evaluations: 26600, 52224, and 102400 evaluations (hereafter referred to as, inaccurately, 25K , 50K, and 100K).[1] For each number of evaluations, we picked rectangular layouts with population sizes of 1024, 1448, and 2048. We compared these against a diagonal layout starting at a population size of 2048, and decreasing linearly towards zero.

1448 was included because we were concerned that the diagonal might outperform other choices simply because it struck a middle-ground between large population sizes and long runlengths. Therefore we included our own rectangular middle-ground. Note however that the 1448 rectangular layout cannot have *exactly* the same number of evaluations as the others: we chose to err on the side of very slightly fewer evaluations. For the 50K evaluations runs, for example, the 1448 layout had 520 fewer evaluations than the others did.

Figure 1 illustrates the four layouts used.

3.1 Other Parameters

Our evolutionary computation system was ECJ [16]. Symbolic Regression used no ephemeral constants and a function of $x^4 + x^3 + x^2 + x$. Artificial Ant used the Santa Fe trail. We used 7-tournament selection. All other initialization, modification, and representation parameters are the same as those used in [3].

We compared mean best fitness of run among experiments, and performed 200 runs for each experiment setup. Our difference of means test was an ANOVA at 95% confidence, plus a Tukey post-hoc comparison.

3.2 Results

We show the mean fitness results in Tables 1 through 4. Layouts are ordered left-to-right in worsening fitness. Horizontal bars above and connecting different layouts indicate no statistically significant difference between them.

[1] These slightly odd evaluation sizes are due to our decision to run for 51 generations.

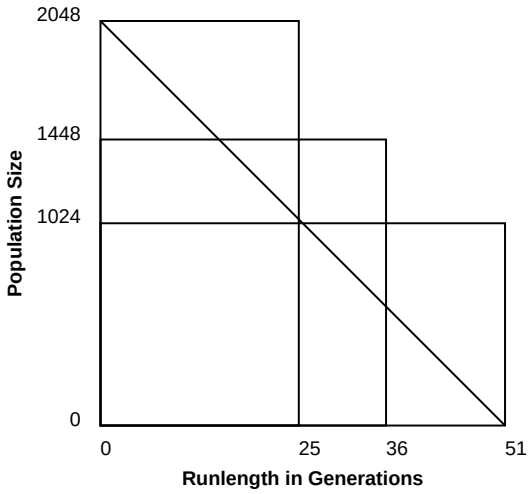

Fig. 1. Four layouts for allocating ∼50K (52224) evaluations: 2048x26; 1448x36; 1024x51; and Diagonal 2048x51

The four problems vary significantly in problem difficulty; some prefer much larger populations, others prefer longer runlengths. The diagonal layout often outperformed all the other layouts, though not always by a statistically significant margin. Nonetheless, what was interesting was that the diagonal layout consistently appeared in the best class of layouts for every single problem domain and every single number of evaluations attempted; and it was the only layout to do so. The diagonal layout did particularly well with smaller numbers of evaluations; for two of the domains (Multiplexer and Ant) it had no peer.

There is an interesting trend among the rectangular layouts. In all four problems, for smaller numbers of evaluations, smaller-population layouts are preferred; and for larger numbers of evaluations, larger-population layouts are preferred. This suggests that there is a optimal useful runlength for each of the problems. As the number of evaluations increases, the smaller-population layouts are extended beyond this runlength and are essentially wasting time. Similarly, for small numbers of evaluations, the large population layouts do not run for enough generations to produce competitive results.

Why did the diagonal perform well? We include Figure 2, showing results in the Artificial Ant domain, to illustrate the general trend during the course of evolution. We think that the diagonal layout is effectively taking advantage of the maximal-runlength phenomenon by providing *both* the runlength of the longer layouts *and* the emphasis on population size of the larger layouts. We note that in both cases, up until 10,000 evaluations, the diagonal layout closely resembles the 2048-population layout in performance, and then in the 20,000-30,000 range the diagonal layout “speeds up” and surpasses the smaller 1024-population layout, which has mostly converged.

Problem and Evals	Layouts			
Regression 25K	Diagonal	1024	1448	2048
Mean Best Fitness	0.09131543	0.103177731	0.145182262	0.169305242
Regression 50K	Diagonal	2048	1448	1024
Mean Best Fitness	0.05270353	0.057168302	0.075613984	0.079054647
Regression 100K	Diagonal	2048	1448	1024
Mean Best Fitness	0.037023631	0.038560673	0.039290056	0.060902268

Table 1. Statistical significance groupings for mean best fitness of run for the Symbolic Regression domain (First Experiment).

Problem and Evals	Layouts			
Ant 25K	Diagonal	1024	1448	2048
Mean Best Fitness	22.325	24.63	24.79	25.99
Ant 50K	Diagonal	2048	1024	1448
Mean Best Fitness	17.71	18.505	20.015	21.87
Ant 100K	2048	Diagonal	1024	1448
Mean Best Fitness	15.865	16.18	18.16	18.775

Table 2. Statistical significance groupings for mean best fitness of run for the Artificial Ant domain (First Experiment).

4 Second Experiment

Our second experiment added additional layouts and repeated the 50K evaluation runs for each problem domain, to see if the diagonal was outperforming other layouts simply because it started at 2048K or was able to eke out long numbers of generations in its tail. The additional layouts included ones with both larger and smaller population sizes (and correspondingly shorter and longer runlengths).

Our additional layouts brought the total rectangular layouts to 4096, 2048, 1448, 1024, 501, 251, and 125 population sizes. The diagonal layout again started at 2048 and decreased to 0.

We again compared mean best fitness of run among experiments, and performed 100 runs for each experiment setup. Our difference of means test was an ANOVA at 95%, plus a Tukey post-hoc comparison.

4.1 Results

In the second experiment we changed the random seeds from the first experiment; hence the means for the original layouts are slightly different. We show the mean fitness results in Table 5. Layouts are again ordered left-to-right in worsening fitness, and

Problem and Evals	Layouts			
5-bit Parity 25K	1024	Diagonal	1448	2048
Mean Best Fitness	6.68	6.885	7.785	8.85
5-bit Parity 50K	1024	Diagonal	1448	2048
Mean Best Fitness	4.77	4.835	4.835	5.825
5-bit Parity 100K	Diagonal	2048	1448	1024
Mean Best Fitness	3.35	3.44	3.46	3.55

Table 3. Statistical significance groupings for mean best fitness of run for the 5-bit Parity domain (First Experiment).

Problem and Evals	Layouts			
11-bit Multiplexer 25K	Diagonal	1024	1448	2048
Mean Best Fitness	197.16	230.475	274.695	352.72
11-bit Multiplexer 50K	Diagonal	1448	2048	1024
Mean Best Fitness	92.38	119.33	122.43	128.32
11-bit Multiplexer 100K	2048	Diagonal	1448	1024
Mean Best Fitness	46.59	53.32	70.43	90.41

Table 4. Statistical significance groupings for mean best fitness of run for the 11-bit Multiplexer domain (First Experiment).

horizontal bars above and connecting different layouts indicate no statistically significant difference between them.

Keep in mind that due to outlier biases from the worst layout results, the conservative Tukey test now cannot discern statistically significant differences among the original four layouts. This is as expected: the goal of the second experiment was only to see if the original four layouts were good choices, and it appears that they were. The new rectangular layouts were usually poor performers, and the diagonal layout usually outperformed all of them by a statistically significant margin.

5 Third Experiment

We also performed a similar experiment using two well-known GA problem domains, the Rastrigin and Rosenbrock problems. Each GA individual's representation was a vector of 100 floating-point genes each ranging from -5.12 to 5.12. We used one-point GA crossover and a gene-independent mutation probability of 0.01, where mutation consisted of gene randomization. We applied tournament selection of size 2, plus one-individual elitism.

Both functions are minimization functions. The Rosenbrock problem [5] computes fitness over a genome of size n using the function

Problem and Evals	Layouts							
Regression 50K	Diagonal	2048	1024	1448	4096	251	502	125
Mean Best Fitness	0.051	0.063	0.078	0.084	0.102	0.198	0.232	0.373
Ant 50K	Diagonal	2048	1448	4096	1024	251	502	125
Mean Best Fitness	17.7	20.23	21.25	21.31	22.49	26.63	28.45	29.68
Parity 50K	Diagonal	1448	1024	251	2048	125	502	4096
Mean Best Fitness	4.67	4.93	4.95	5.32	5.78	6.01	6.16	8.48
Multiplexer 50K	Diagonal	2048	1448	1024	251	4096	502	125
Mean Best Fitness	54.68	54.72	69.64	81.38	247.26	296.98	301.84	363.03

Table 5. Statistical significance groupings for mean best fitness of run for the Symbolic Regression, Artificial Ant, 5-bit Parity, and 11-bit Multiplexer domains (Second Experiment).

Problem and Evals	Layouts			
Rosenbrock	32	Diagonal	45	64
Mean Best Fitness	2256.585	2415.601	3281.273	5131.429
Rastrigin	32	Diagonal	45	64
Mean Best Fitness	136.0726	142.2995	155.8251	184.689

Table 6. Statistical significance groupings for mean best fitness of run for the Rosenbrock and Rastrigin domains (Third Experiment).

$$\text{Rosenbrock}(x_1,\ldots,x_n) = \sum_{i=1}^{n} 100({x_i}^2 - x_{i+1})^2 + (1-{x_i}^2)$$

Similarly, the difficult Rastrigin problem [17] computes fitness using the function

$$\text{Rastrigin}(x_1\ldots.x_n) = \sum_{i=1}^{n} {x_i}^2 + a(1-\cos(2\pi x_i))$$

We adjusted the size of layouts to make them more appropriate to the GA realm: 32x1024, 45x724, 64x512, and a diagonal layout with initial population size of 64, running for 1024 generations. We performed 100 independent trials for each problem. As shown in Table 6, for both problem domains the ordering was the same, namely: 32x1024 outperformed diagonal, which in turn outperformed 45x724, which in turn outperformed 64x512. However, on the Rastrigin domain the difference between the 32x1024 and the diagonal layout was not statistically significant.

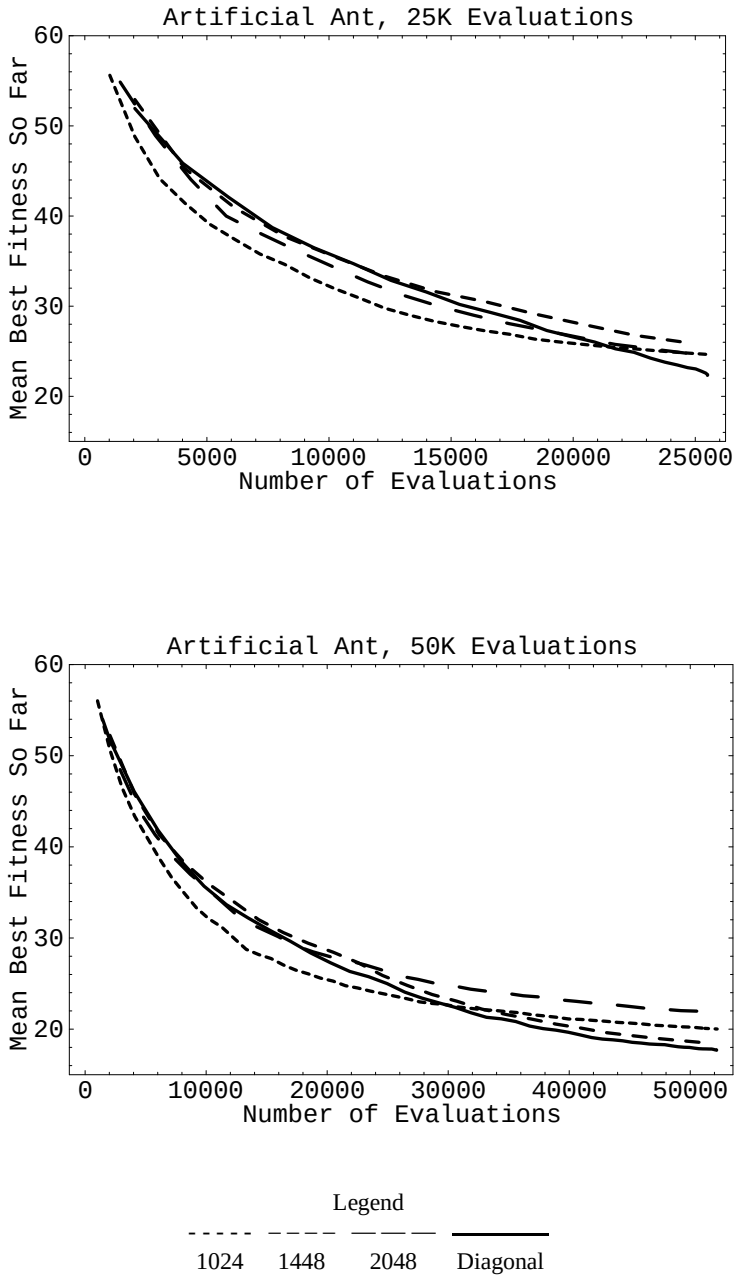

Fig. 2. Best-So-Far Curves of Four Layouts, Artificial Ant Problem, at 25K and 50K Evaluations

This is a mixed result. The result is similar to that of the 5-bit Parity GP domain, namely that longer runs are consistently preferred over shorter ones. But for the first time in this paper, diagonal has come in second in one of the experiments (Rastrigin). We note that the number of evaluations (32K, a typically size for GAs) is nonetheless similar to the smaller GP experiments. Further runs may suggest that GAs too have "diminishing returns" late in the run: but this is not yet borne out from the evidence here.

6 Conclusions and Future Work

In this paper we examined the possibility of imploding the population — gradually decreasing it towards zero as the run progressed — in the context of genetic programming. A linear decrease in population size proved effective regardless of the number of evaluations used. A diagonal layout was consistently in the top tier in every GP experiment, and usually gave the best results in the experiments. In initial GA experiments however, the results were mixed.

From this we can draw two conclusions: the primary conclusion is of course that non-rectangular layouts may yield better results than rectangular ones in environments like GP. But the second more troubling conclusion is that our experiments add to existing evidence that GP may have an aging problem. Whether due to premature convergence, bloat, or other factors, GP does not appear to use large-population resources effectively late in the run. While methods like diagonal layouts may serve to work around the issue, there is need for a closer examination as to *why* GP has diminishing returns, and how the representation or breeding methods may be changed to alleviate the problem.

References

1. Streeter, M.J., Keane, M.A., Koza, J.R.: Iterative refinement of computaitonal circuits using genetic programming. In Langdon, W.B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M.A., Schultz, A.C., Miller, J.F., Burke, E., Jonoska, N., eds.: GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, Morgan Kaufmann Publishers (2002) 877–884
2. Koza, J.R., Keane, M.A., Yu, J., Mydlowec, W.: Automatic synthesis of electrical circuits containing a free variable using genetic programming. In Whitley, D., Goldberg, D., Cantu-Paz, E., Spector, L., Parmee, I., Beyer, H.G., eds.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), Las Vegas, Nevada, USA, Morgan Kaufmann (2000) 477–484
3. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
4. Luke, S.: When short runs beat long runs. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 2001, Morgan Kaufmann Publishers (2001) 74–80
5. De Jong, K.: An Analysis of the Behavior of a Class of Genetic Adaptive Systems. PhD thesis, University of Michigan, Ann Arbor, MI (1975)
6. Grefenstette, J.J.: Optimization of control parameters for genetic algorithms. IEEE Transactions on Systems Man and Cybernetics **16** (1986) 122–128

7. Schaffer, J.D., Caruana, R.A., Eshelman, L.J., Das, R.: A study of control parameters affecting online performance of genetic algorithms for function optimization. In: Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc. (1989) 51–60
8. De Jong, K., Spears, W.M.: An analysis of the interacting roles of population size and crossover in genetic algorithms. In Schwefel, H.P., Männer, R., eds.: Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1. Volume 496., Dortmund, Germany, Springer-Verlag, Berlin, Germany (1991) 38–47
9. Goldberg, D.E., Deb, K., Clark, J.H.: Genetic algorithms, noise, and the sizing of populations. Complex Systems **6** (1992) 333–362
10. Giguère, P., Goldberg, D.E.: Population sizing for optimum sampling with genetic algorithms: A case study of the onemax problem. In Koza, J.R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M.H., Goldberg, D.E., Iba, H., Riolo, R., eds.: Genetic Programming 1998: Proceedings of the Third Annual Conference, University of Wisconsin, Madison, Wisconsin, USA, Morgan Kaufmann (1998) 496–503
11. Federman, F., Dorchak, S.F.: A study of classifier length and population size. In Koza, J.R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M.H., Goldberg, D.E., Iba, H., Riolo, R., eds.: Genetic Programming 1998: Proceedings of the Third Annual Conference, University of Wisconsin, Madison, Wisconsin, USA, Morgan Kaufmann (1998) 629–634
12. Arabas, J., Michalewicz, Z., Mulawka, J.J.: GAVaPS - a genetic algorithm with varying population size. In: Proceedings of IEEE Conference on Evolutionary Computation. Volume 1. (1994) 73–78
13. Balazs, M.E., Richter, D.L.: A genetic algorithm with dynamic population: Experimental results. In Brave, S., Wu, A.S., eds.: Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference, Orlando, Florida, USA (1999) 25–30
14. Fuchs, M.: Large populations are not always the best choice in genetic programming. In Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E., eds.: Proceedings of the Genetic and Evolutionary Computation Conference. Volume 2., Orlando, Florida, USA, Morgan Kaufmann (1999) 1033–1038
15. Wagner, N., Michalewicz, Z.: Genetic programming with efficient population control for financial time series prediction. In Goodman, E.D., ed.: 2001 Genetic and Evolutionary Computation Conference Late Breaking Papers, San Francisco, California, USA (2001) 458–462
16. Luke, S. ECJ 9: A Java EC research system. Available at http://www.cs.umd.edu/projects/plus/ec/ecj/ (2002)
17. Cervone, G., Michalski, R., Kaufman, K., Panait, L.: Combining machine learning with evolutionary computation: Recent results on LEM. In: Proceedings of the Fifth International Workshop on Multistrategy Learning. (2000) 41–58