When Short Runs Beat Long Runs

Sean Luke
George Mason University
http://www.cs.gmu.edu/∼sean/

Abstract

What will yield the best results: doing one run n generations long or doing m runs n/m generations long each? This paper presents a technique-independent analysis which answers this question, and has direct applicability to scheduling and restart theory in evolutionary computation and other stochastic methods. The paper then applies this technique to three problem domains in genetic programming. It discovers that in two of these domains there is a maximal number of generations beyond which it is irrational to plan a run; instead it makes more sense to do multiple shorter runs.

1 INTRODUCTION

Research in stochastic search has long struggled to determine how best to allocate precious resources to find the best possible solution. This issue has not gone away with increases in computer power: rather, the difficulty of our optimization problems has more than kept up with our new computational muscle. And the rise of massive parallelism has added an additional constraint to how we may divvy up our total evaluations.

Studies in resource allocation have attacked different aspects of the problem. One popular area of study in genetic algorithms is *online restart determination*. This area asks: while in the midst of a stochastic run and with no *a priori* knowledge, should I restart now and try again? This used to be a critical issue for GAs because of the spectre of premature convergence. Detecting the approach of premature convergence during a run saved valuable cycles otherwise wasted. There has been much work in this area; for a few examples, see [Goldberg, 1989, Collins and Jefferson, 1991, Eshelman and Schaffer, 1991]. This work usually assumes certain heuristics about convergence which may or may not be appropriate. Commonly the work relies on variance within a population or analysis of change in performance over time. These techniques are ad-hoc, but more problematic, they are often domain-specific. For example, they would not work in general on genetic programming.

In some sense, detecting premature convergence is an analysis of time-to-failure. A more cheerful focus in evolutionary computation, *convergence velocity*, is not directly involved in resource allocation but has many important ties. Evolutionary strategies analysis can demonstrate the rates at which specific techniques are expected to move towards the optimum, either in solution space or in fitness space [Bäck, 1996]. Since different population sizes can be considered different techniques, this analysis can shed light on resource allocation issues.

One area which directly tackles resource allocation is *scheduling* [Fukunaga, 1997]. A schedule is a plan to perform n runs each l generations long. The idea is to come up with a schedule which best utilizes available resources, based on past knowledge about the algorithm built up in a database. Typically this knowledge is derived from previous applications of the algorithm to various problem domains different from the present application. [Fukunaga, 1997] argues that previous problem domains are a valid predictor of performance curves in new domains, for genetic algorithms at least.

Outside of evolutionary computation, there is considerable interest in *restart methods* for global optimization. For difficult problems where one expects to perform many runs before obtaining a satisfactory solution, one popular restart method is to perform random restarts [Hu et al., 1997, Ghannadian and Alford, 1996]. If the probability density function of probability of convergence at time t is known then it is also possible to derive the *optimum restart time* such that, as the number of evaluations approaches infinity, the algorithm converges with the most rapid possible rate [Magdon-Ismail and Atiya, 2000].

Lastly, much genetic programming work has assumed that

the optimum can be discovered. A common metric of time-to-optimal-discovery is called *cumulative probability of success* [Koza, 1992]. However, this metric does not directly say anything about the rate of success nor whether or not shorter runs might yield better results.

The analysis presented in this paper takes a slightly different tack. It attempts to answer the question: is it rational to try a single run n generations long? Would it be smarter to instead try m runs each $\frac{n}{m}$ generations long? As it turns out, this question can be answered with a relatively simple procedure derived from a manipulation of order statistics. The procedure is entirely problem-independent; in fact it can be easily applied to any stochastic search method.

Unlike some of the previous methods, this analysis does not attempt to determine how long it takes to discover the optimum, nor the probability of discovering it, nor how fast the system converges either globally or prematurely. It is simply interested in knowing whether one schedule is likely to produce better net results than another schedule.

This paper will first present this analysis and prove it. It will then apply the analysis to three problems in genetic programming, an evolutionary computation approach which is notorious for requiring large populations and short runlengths. It then discusses the results.

2 PRELIMINARIES

We begin with some theorems based on order statistics, which are used to prove the claims in Section 3. These theorems tell us what the expected value is of the highest quality (fitness) found among of some n samples picked with replacement from a population. The first theorem gives the continuous case (where the population is infinite in size). The second theorem gives the discrete case.

Theorem 1 *Let $X_1, ..., X_n$ be n independent random variables representing n selections from a population whose density function is $f(x)$ and whose cumulative density function is $F(x)$. Let X_{max} be the random variable representing the maximum of the X_i. Then the expected value of X_{max} is given by the formula $\int_{-\infty}^{\infty} xnf(x)(F(x))^{n-1}dx$.*

Proof Note that for any given x, $X_{max} \leq x$ if and only if for all i, $X_i \leq x$. Then the cumulative density function $F_{X_{max}}(x)$ of the random variable X_{max} is as follows: $F_{X_{max}}(x) = P(X_{max} \leq x) = P(X_1 \leq x)P(X_2 \leq x)...P(X_n \leq x) = F(x)F(x)...F(x) = (F(x))^n$. The density function $f_{X_{max}}(x)$ for X_{max} is the derivative of this, so $f_{X_{max}}(x) = nf(x)(F(x))^{n-1}$. The expected value of any density function $G(x)$ is defined as $\int_{-\infty}^{\infty} xG(x)dx$, so the expected maximum value of the n random variables is equal to $\int_{-\infty}^{\infty} xf_{X_{max}}dx = \int_{-\infty}^{\infty} xnf(x)(F(x))^{n-1}dx$. ■

Lemma 1 *Given n selections with replacement from the set of numbers $\{1, ..., m\}$, the probability that r is the maximum number selected is given by the formula $\frac{r^n-(r-1)^n}{m^n}$. The sum of probabilities for all such r is 1.*

Proof Consider the set S_r of all possible events for which, among the n numbers selected with replacement, r is the maxiumum number. These events share the two following criteria. First, for each selection x among the n selections, $x \leq r$. Second, there exists a selection y among the n for which $y \geq r$. The complement to this second criterion is that for each selection x among the n selections, $x \leq (r-1)$. Since this complement is a strict subset of the first criterion, then S_r is the set difference between the first criterion and the complement, thus the probability of P_r of an event in S_r occuring is the difference between the probability of the first criterion and the probability of the complement, that is, $P_r = P(\forall x : x \leq r) - P(\forall x : x \leq (r-1))$.

For a single selection with replacement from the set of numbers $\{1, ..., m\}$, the probability that the selection is less than or equal to some value q is simply $\frac{q}{m}$. Thus for n independent such selections, the probability that all are $\leq q$ is $\frac{q^n}{m^n}$. Substituting into the solution above, we get $P_r = \frac{r^n}{m^n} - \frac{(r-1)^n}{m^n} = \frac{r^n-(r-1)^n}{m^n}$. Further, the sum of such probabilities for all r is $\sum_{r=1}^{m} \frac{r^n-(r-1)^n}{m^n} = \frac{1^n-0^n}{m^n} + \frac{2^n-1^n}{m^n} + \cdots + \frac{m^n-(m-1)^n}{m^n} = \frac{m^n-0^n}{m^n} = 1$ ■

Theorem 2 *Consider a discrete distribution of m trials, with each trial r having a quality $Q(r)$, sorted by Q so that trial 1 has the lowest quality and trial m has the highest quality. If we pick n trials with replacement from this distribution, the expected value of the maximum quality among these n trials will be*

$$\sum_{r=1}^{m} Q(r)\frac{r^n - (r-1)^n}{m^n}$$

Proof The *rank* of a trial is its position $1, ..., m$ in the sorted order of the m trials. The expected value of the maximum quality among the n selected trials is simply the sum, over each rank r, of the probability that r will be the highest rank among the selected trials, times the quality of r. This probability is given by Lemma 1. Hence the summation is $\sum_{r=1}^{m} Q(r)\frac{r^n-(r-1)^n}{m^n}$. ■

3 SCHEDULES

These order statistics results make possible the creation of tools that determine which of two techniques A and B is expected to yield the best results. This paper discusses a specific subset of this, namely, determining whether evo-

lutionary technique A run m_1 generations n_1 times (commonly 1 time) is superior the same technique A run m_2 generations n_2 times, where $n_1 m_1 = n_2 m_2$. We begin with some definitions.

Definition 1 *A* **schedule** *S is a tuple $\langle n_S, l_S \rangle$, representing the intent to do n_S independent runs of length l_S each.*

Definition 2 *Let S, T be two schedules. Then S* **reaches** *T if n_S runs of length l_S are expected to yield as good as or higher quality than n_T runs of length l_T. Define the predicate operator $S \succeq T$ to be true if and only if S reaches T.*

The following two theorems assume that higher quality is represented by higher values. In fact, for the genetic programming examples discussed later, the graphs shown have *lower* fitness as higher quality; this is rectified simply by inverting the fitness values.

Theorem 3 *Let $p_t(x)$ be the probability density function and $P_t(x)$ the cumulative probability density function of the population of all possible runs, reflecting their quality at time t (assume higher values mean higher quality). Then $S \succeq T$ if and only if:*

$$\int_{-\infty}^{\infty} x n_S p_{l_S}(x)(P_{l_S}(x))^{n_S - 1} dx \geq \int_{-\infty}^{\infty} x n_T p_{l_T}(x)(P_{l_T}(x))^{n_T - 1} dx$$

Proof Both sides of this inequality are direct results of Theorem 1. ■

The continuous case above is not that useful in reality, since we rarely will have an infinite number of runs to draw from! However, if we perform many runs of a given runlength, we can estimate the expected return from doing n runs at that runlength, and use this to determine if some schedule outperforms another schedule. The estimate makes the assumption that the runs we performed (our sample) is *exactly representative* of the full population of runs of that runlength.

Theorem 4 *Given a schedule $S = \langle n_S, l_S \rangle$, consider a random sample, with replacement, of m_S runs from all possible runs of runlength l_S. Let these runs be sorted by quality and assigned ranks $1, ..., m_S$, where a run's rank represents its order in the sort, and rank 1 is the lowest quality. Further, let $Q_S(r)$ be the quality of the run from the sample whose rank is r; $Q_S(r)$ should return higher values for higher quality. For another schedule T, similarly define m_T and $Q_T(r)$. Then an estimate of reaching is as follows. $S \succeq T$ if and only if:*

$$\sum_{r=1}^{m_S} Q_S(r) \frac{r^{n_S} - (r-1)^{n_S}}{{m_S}^{n_S}} \geq \sum_{r=1}^{m_T} Q_T(r) \frac{r^{n_T} - (r-1)^{n_T}}{{m_T}^{n_T}}$$

Proof Both sides of this inequality are direct results of Theorem 2. ■

These theorems give tools for determining whether one schedule reaches another. We can use this to estimate what schedule is best for a given technique. If we wanted to examine a technique and determine its best schedule, we have two obvious options:

1. Perform runs out to our maximum runlength, and use run-data throughout the runs as estimates of performance at any given time t. The weakness in this approach is that these estimates are not statistically independent.

2. Perform runs out to a variety of runlengths. The weakness in this approach is that it requires $O(n^2)$ evaluations.

A simple compromise adopted in this paper is to do runs out to 1 generation, a separate set of runs out to 2 generations, another set of runs out to 4 generations, etc., up to some maximal number of generations. This is $O(n)$, yet still permits runlength comparisons between statisticatlly independent data sets.

Two statistical problems remain. First, these comparisons do not come with a difference-of-means test (like a t-test or ANOVA). The author is not aware of the existence of any such test which operates over order statistics appropriate to this kind of analysis, but hopes to develop (or discover!) one as future work. This is alleviated somewhat by the fact that the result of interest in this paper is often not the hypothesis but the null hypothesis. Second, the same run data for a schedule is repeatedly compared against a variety of other schedules; this increases the alpha error. To eliminate this problem would necessitate $O(n^3)$ evaluations (!) which is outside the bounds of the computational power available at this time.

4 ANALYSIS OF THREE GENETIC PROGRAMMING DOMAINS

Genetic Programming is an evolutionary computation field with traditionally short runlengths and large population sizes. Some of this may be due to research following in the footsteps of [Koza, 1992, 1994] which used large populations (500 to 1000 individuals) and short runlengths (51 generations). Are such short runlengths appropriate? To

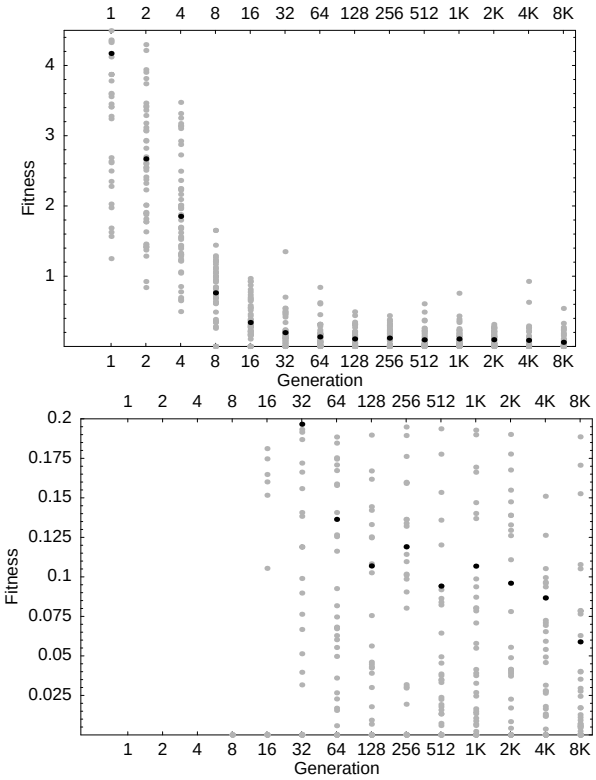

Figure 1: Runlength vs. Fitness, Symbolic Regression Domain (Including Detail)

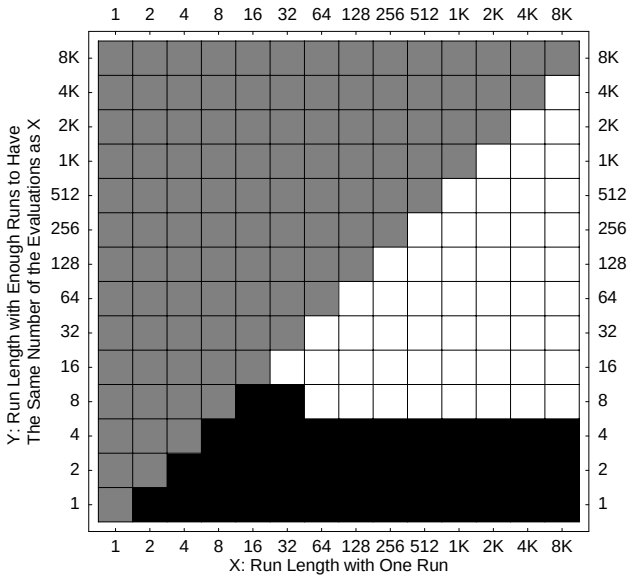

Figure 2: Runlength Analysis of Symbolic Regression Domain. Areas are black where X is a superior strategy to Y and white where Y is as good or better than X. Gray regions are out of bounds.

consider this, I analyzed three GP problem domains: Symbolic Regression, Artificial Ant, and Even 10-Parity. These three domains have very different dynamics.

In all three domains, I performed 50 independent runs for runlengths of 2^i generations ranging from 2^0 to some 2^{max}. Because these domains differ in evaluation time, max varied from domain to domain. For Symbolic Regression, $2^{max} = 8192$. For Artificial Ant, $2^{max} = 2048$. For Even 10-Parity, $2^{max} = 1024$. For all three domains, lower fitness scores represent better results. The GP system used was ECJ [Luke, 2000].

The analysis graphs presented in this paper compare single-run schedules with multiple-run schedules of shorter length. However additional analysis comparing n-run schedules with nm-run schedules of shorter length has yielded very similar results.

4.1 Symbolic Regression

The goal of the Symbolic Regression problem is to find a symbolic expression which best matches a set of randomly-chosen target points from a predefined function. Ideally, Symbolic Regression discovers the function itself. I used the traditional settings for Symbolic Regression as defined in [Koza, 1992], with a population size of 500 and tournament selection with a tournament of size 7. The function to be fitted was $x^4 + x^3 + x^2 + x$.

Unlike the other two problems, Symbolic Regression operates over a continuous fitness space; if cannot find the optimal solution, it will continue to find incrementally smaller improvements. Although Symbolic Regression very occasionally will discover the optimum, usually it tends towards incrementalism. As such, Symbolic Regression fitness values can closely approach 0 without reaching it, so Figure 1 shows both zoomed-out and zoomed-in versions of the same data. Grey dots represent individual best-of-run results for each run; black dots represent means of 50 runs of that runlength.

As can be seen, the mean continues to improve all the way to runlengths of 8192. But is it rational to plan to do a run out to 8192 generations? Figure 2 suggests otherwise.

The runlength analysis graphs can be confusing. On the graph, the point $(X, Y), X > Y$ indicates the result of comparing a schedule $A = \langle 1, X \rangle$ with the schedule $B = \langle \frac{X}{Y}, Y \rangle$, which has the same total number of evaluations. The graph is white if $B \succeq A$, black otherwise. This is a lower-right matrix: gray areas are out-of-domain regions.

Figure 2 shows that the expected quality of a single run of length $\geq$32 is reached by doing some n runs of length 16 which total the same number of evaluations. Another

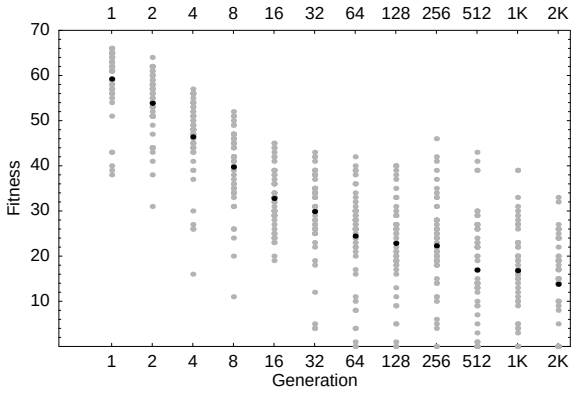

Figure 3: Runlength vs. Fitness, Artificial Ant Domain

interesting feature is that there is a minimum acceptable runlength: under no circumstances could multiple runs less than 8 generations reach a single run of larger size.

What about comparing a schedule $A = \langle c, X \rangle$ with schedules $B = \langle \frac{cX}{Y}, Y \rangle$? Even with values of $c = 2, 4, 8$, the resultant runlength analysis graphs were almost identical.

4.2 Artificial Ant

Artificial Ant moves an ant across a toroidal world, attempting to follow a trail of food pellets and eat as much food as possible in 400 moves. I used the traditional Artificial Ant settings with the Santa Fe trail as defined in [Koza, 1992], with a population size of 500 and tournament selection using a tournament of size 7.

As shown in Figure 3, the mean Artificial Ant best-of-run fitness improved monotonically and steadily with longer runlengths clear out to 2048 generations. But this did not mean that it was rational to plan to do a run out that far. Figure 4 suggests that single runs of runlengths beyond 64 generations were reached by multiple runs with shorter runlengths but the same number of total evaluations.

This is very similar to the Symbolic Regression results. Also similar was the existence of a minimum acceptable runlength: runs less than 4 could not reach a single run of larger size. Lastly, runlength analysis graphs with values of $c = 2, 4,$ or 8 were very similar.

4.3 Even-10 Parity

The last problem analyzed was Even-10 Parity, a very difficult problem for Genetic Programming. Even-10 Parity evolves a symbolic boolean expression which correctly identifies whether or not, in a vector of 10 bits, an even number of them are 1. This is a large and complex function and necessitates a large GP tree. To make the problem even harder, I used a small population (200), but otherwise

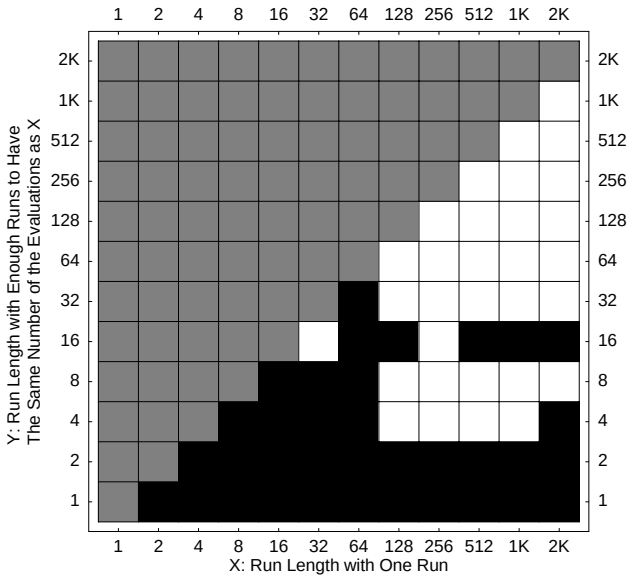

Figure 4: Runlength Analysis of Artificial Ant Domain. Areas are black where X is a superior strategy to Y and white where Y is as good or better than X. Gray regions are out of bounds.

followed the specifications for the Parity problem family as outlined in [Koza, 1992].

Figure 5 shows just how difficult it is for Genetic Programming to solve the Even-10 Parity problem. Even after 1024 generations, no run has reached the optimum; the mean best-of-run fitness has improved by only 25% over random solutions. The curve does not resemble the logistic curve of the other two GP domains.

One might suppose that in a domain where 1024 generations improves little over 1 generation, runlength analysis would argue for the futility of long runs. Yet the results were surprising: a single run of any length was always consistently superior to multiple runs of shorter lengths. Even though Even-10 Parity is very difficult for Genetic Programming to solve, it continues to plug away at it. It is conceivable that, were we to run out far enough, we might see a maximal rational runlength in the Even-10 Parity domain. Nonetheless, it is surprising that even at 1024 generations, Even-10 Parity is still going strong.

5 DISCUSSION

As the Symbolic Regression and Artificial Ant domains have shown, there can be a runlength beyond which it seems irrational to plan to do runs, because more runs of shorter length will do just as well if not better. I call this

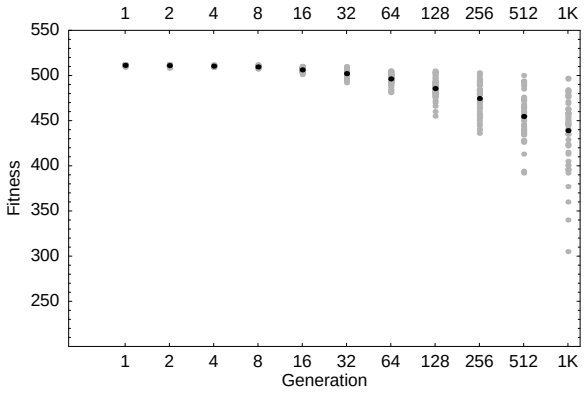

Figure 5: Runlength vs. Fitness, Even-10 Parity Domain

runlength a *critical point*. The location of the critical point suggests interesting things about the ability of the technique to solve the problem at hand. As the critical point approaches 1, the technique becomes less and less of an improvement over blind random search.

Symbolic Regression only occasionally finds the optimum, but if it is lost, around generation 64 it seems to begin to search for incrementally smaller values. One is tempted to suggest that this is why it is irrational to continue beyond about generation 32 or so. However, while the curve flattens out, as the detail shows, it still makes improvements in fitness. The critical feature is that the *variance* among the runs stays high even though the mean improves only slowly. This is what makes it better to do 2 runs of length 32 (or 8 of 8) than 1 run of length 64, for example.

Artificial Ant demonstrates a similar effect. Even though the mean improves steadily, the variance after generation 32 stays approximately the same. As a result, 4 runs of 32 will handily beat out 1 run of 128 despite a significant improvement in the mean between 32 and 128 generations.

The interesting domain is Even 10-Parity. In this domain the mean improves and the variance also continues to increase. As it turns out, the mean improves just enough to counteract the widening variance. Thus even though this is a very difficult problem for genetic programming to solve, it never makes sense to do multiple short runs rather than one long run!

Symbolic Regression and Artificial Ant also suggest that there can exist a *minimum* runlength such that any number of runs with fewer generations are inferior to a single run of this runlength. In some sense it is also irrational to do multiple runs with fewer generations than this minimum runlength instead of (at least) one run at the minimum runlength. Thus there is a window between the minimum and maximum rational runlengths. If one has enough evaluations, it appears to makes most sense to spend them on

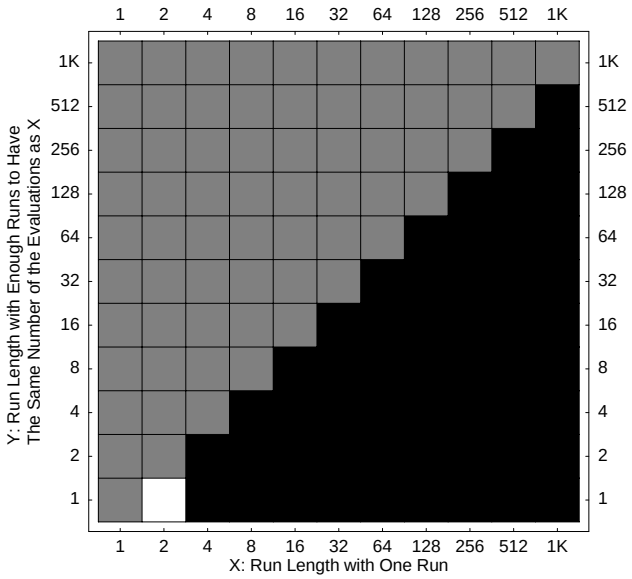

Figure 6: Runlength Analysis of Even-10 Parity Domain. Areas are black where X is a superior strategy to Y and white where Y is as good or better than X. Gray regions are out of bounds.

runs within this window of rationality.

One last item that should be considered is *evaluation time*, which for genetic programming is strongly influenced by the phenomenon of *code bloat* . As a genetic programming run continues, the size of its individuals grows dramatically, and so does the amount of time necessary to breed and particularly to evaluate them. So far we have compared schedules in terms of total number of evaluations; but in the case of genetic programming it might make more sense to compare them in terms of *total runtime*. The likely effect of this would be to make the maximally rational runtime even shorter. In the future the author hopes to further explore this interesting issue.

6 CONCLUSION

Genetic programming has traditionally not done runs longer than 50 generations or so, at least for the common cannonical problems. Instead it prefers larger population sizes. The results of this analysis suggest one reason why this might be: beyond a very small runlength (16 for Symbolic Regression, about 32 or 64 for Artificial Ant) the diminishing returns are such that it makes more sense to divvy up the total evaluations into multiple smaller runs.

But “rapidly diminishing returns” is not the same thing as “difficult problem”. In a hard problem like Even-10 Parity,

it *still* makes sense on average to press forward rather than do many shorter runs.

This paper presented a formal, heuristic-free, domain-independent analysis technique for determining the expected quality of a given schedule, and applied it to three domains in genetic programming, with interesting results. But this analysis is applicable to a wide range of stochastic techniques beyond just GP, and the author hopes to apply it to other techniques in the future.

Acknowledgements

The author wishes to thank Ken DeJong, Paul Wiegand, Liviu Panait, and Jeff Bassett for their considerable help and insight.

References

T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

R. J. Collins and D. R. Jefferson. Selection in massively parallel genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA)*, pages 249–256, 1991.

L. J. Eshelman and J. D. Schaffer. Preventing premature convergence in genetic algorithms by preventing incest. In *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA)*, pages 115–122, 1991.

A. Fukunaga. Restart scheduling for genetic algorithms. In Thomas Back, editor, *Genetic Algorithms: Proceedings of the Seventh International Conference*, 1997.

F. Ghannadian and C. Alford. Application of random restart to genetic algorithms. *Intelligent Systems*, 95:81–102, 1996.

D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, 1989.

X. Hu, R. Shonkwiler, and M. Spruill. Random restart in global optimization. Technical Report 110592-015, Georgia Tech School of Mathematics, 1997.

John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.

Sean Luke. ECJ: A Java-based evolutionary computation and genetic programming system. Available at http://www.cs.umd.edu/projects/plus/ecj/, 2000.

M. Magdon-Ismail and A. Atiya. The early restart algorithm. *Neural Computation*, 12(6):1303–1312, 2000.