Cutting Evaluation Costs: An Investigation into Early Termination in Genetic Programming

Namyong Park
School of Computer Science & Engineering
Seoul National University
Seoul, Korea
Email: park.namyong@gmail.com

Kangil Kim
School of Computer Science & Engineering
Seoul National University
Seoul, Korea
Email: kangil.kim.01@gmail.com

R. I. (Bob) McKay
School of Computer Science & Engineering
Seoul National University
Seoul, Korea
Email: rimsnucse@gmail.com

***Abstract*—Genetic programming is very computationally intensive, particularly in CPU time. A number of approaches to evaluation cost reduction have been proposed, among them early termination of evaluation (applicable in problem domains where estimates of the final fitness value are available during evaluation). Like all cost reduction techniques, early termination balances overall computation cost against the risk of finding worse solutions. We evaluate the influence of various properties of the problem domain – problem class, reliability of fitness estimates, trajectory of fitness estimates, and evolutionary trajectory – to determine whether any is able to predict the effects of early termination. There is little correlation with any of these, with one exception. Boolean problems see little change in running time, and hence only small changes in performance, are distinguished by both problem class, and each of the other metrics.**

I. Introduction

Right from the start, Genetic Programming (GP) has been used to generate solutions to many real-world problems in diverse fields [1]–[3]. Yet there remain many real-world problems which lie outside GP's reach purely because of the computational cost of search, both in CPU time and in memory [3]. While many problems can be solved with relatively modest computational effort, many other potential applications make very high processor demands, with most of the time spent in performing fitness evaluations [4].

In our direct experience, the initial C++ implementation of the fitness function for the water quality model which motivated this work, by a highly experienced and expert programmer, required 5 cpu minutes per evaluation. The project would have been abandoned had this not been reduced. The problem was only finally overcome through a combination of the techniques described here, and a highly complex (and fragile) C++ run-time compilation-and-loading architecture, requiring months of development that would be unacceptable in most industrial environments. Another project was terminated after a pilot study, once it became clear that the evaluation cost for the full problem was likely to be of the order of hours. But even much lower evaluation costs can block an application. In on-line applications, even millisecond evaluation times, that would generally be acceptable off-line, may be intolerable.

The problem becomes more serious with growth in problem size and complexity – both in direct computational cost, and in the cost of manual code optimisation. There are a number of ways to alleviate this problem, which we introduce in section II. One relevant to a wide range of problems – those in which evaluation either is or can be made incremental – is early termination. In this approach, the system learns to estimate the final fitness from early stages of the evaluation, and also how to determine when the estimate is sufficiently accurate. It has the important advantage of being – at least conceptually – orthogonal to the other approaches which have been proposed, suggesting the possibility of combining these approaches, and generating greater savings. This paper represents the first stage of this investigation, examining the effects of early termination on the evaluation cost of GP runs, and the corresponding changes in the solution-finding capacity of GP. It also investigates problem characteristics that might explain and/or predict the effect of early termination on specific problem classes. Finally, it discusses how the method might be combined with other cost-reduction techniques.

The remainder of the paper is organised thus: In section II, we overview methods for accelerating GP running time. We summarise research on noisy fitness evaluation that forms a backdrop to early termination methods, and summarise Grammar Guided Genetic Programming (GGGP), used for one of the problems. In section III, we detail early termination, and its relationship with time-series and machine learning problems. Section IV describes the benchmark problems and the experiment settings. In section V, results are provided and their implications are discussed. We conclude in section VI with a summary and pointers for future work.

II. Background

Evaluation cost reduction techniques divide naturally into two classes: some fully evaluate fitness of all individuals, while others make a cheaper approximation to the fitness, and use that cheaper approximation for some evaluations. The latter depend heavily on the robustness of evolutionary algorithms to noise in fitness evaluation, a topic that has been heavily researched in recent years.

A. Uncertain Evaluation in Evolutionary Algorithms

The literature on noise in evolutionary algorithms has two main branches – theoretical analysis as exemplified by the work of Beyer and Arnold [5], [6], and the practical experimentation detailed in Jin and Branke's survey [7]. To summarise, evolutionary algorithms handle noise better than other algorithms applying the same resources to the same domain. However the effect of noisy evaluation varies. For simpler domains, noise leads to slower convergence to good solutions, and the solutions may not be quite as good [6]. On the other hand, improved performance has been reported on more complex domains [8], perhaps because noise can permit escape from local optima.

B. Reducing the Cost of Fitness Evaluation

A wide range of methods have been used to speed up fitness evaluation. Among those which do so directly, we may include direct machine code evaluation, parallel execution, simplification, caching and more pragmatic solutions such as run-time compilation. In more detail:

1) Machine code evolution [3], [9]: represents solutions as fragments of machine code (embedded in other machine code structures that maintain the semantics). These fragments are the direct targets of evolution, and thus avoid the overheads of compilation – Nordin reported speed-ups of up to two orders of magnitude relative to an interpreted C-language system. More recently, even higher speed-ups have been reported from direct implementations on GPUs [10], and also from GPU-implemented interpreters for GP [11]. This speed comes at a cost in terms of programming flexibility and interpretability.

2) Parallel implementations [4]: have been attempted at a number of granularity levels. The most successful, partly because of simplicity, use coarse-grained parallelism as in the island model, where whole populations are evolved in parallel, with slow exchange of individuals between populations. Precisely speaking, these algorithms reduce not the computational resources, but the elapsed time, since the same (or even greater) resources may be used, but in parallel rather than in sequence. They may gain further advantage from the demic isolation, but this is a benefit from the grid structure, which is often used in sequential systems as well, rather than of parallelism per se.

3) Algebraic simplification [12]: directly reduces the cost of evaluation by simplifying GP trees before they are evaluated. Early approaches retained the simplified individuals in the population (a Lamarckian perspective), leading to inconsistent results from premature convergence. More recent systems generally retain the complex individual in the population, using simplification purely for evaluation.

4) Caching [13]: trades off memory resources against computation time, storing the results of subtree evaluation in case they may be evaluated again. Hashing is used to detect this, the stored result being used in future evaluations. The effectiveness depends on the hit rate of repeated evaluations, which varies from domain to domain, since reductions in evaluation cost must be traded off against the cost of hashing and storage. Caching may be effectively combined with simplification, which improves the hit rate.

5) Pragmatic approaches: apply a combination of software engineering methods (code profiling, structured design) and compiler mechanisms (run-time compilation, just-in-time compilation methods) to speed up code in more generally applicable ways.

These methods all conduct exact fitness evaluation, employing a variety of strategies to speed it up. The remainder rely on the robustness of evolutionary algorithms to noise, providing fitness estimates which, while not exact, are good enough to guide the GP system to an equivalent destination.

6) Fitness approximation [14]: evaluates a surrogate for the fitness. The surrogate is usually built by machine learning, and approximates the value that would have been obtained. The use of learning requires some instances to be fully evaluated. As evolution progresses, the new instances will differ from the training population, so that re-learning must be conducted– determining how often is a key issue.

7) Clustering [15]: uses a heuristic estimate of similarity to cluster individuals. Only one individual is evaluated per cluster, the others being assigned the same fitness. It can be viewed as a subtype of fitness approximation, in which the set of exemplars from the clusters form an instance-based model.

8) Training subset evaluation [16]: estimates the fitness of an individual using only a subset of training data. How to select a subset, and how to use the partial evaluation resulting from it determine the effectiveness of this method. A previous study [17] applied limited error fitness to a classification problem, where the fitness is computed according to the number of cases an individual misclassifies after it exceeds an error limit, at which point the fitness evaluation stops.

Early termination, which is the primary theme of this paper, falls into this category, though it differs from aforementioned training subset evaluation methods in handling cases where the evaluation order of training instances is fixed. We will describe it in detail in section III.

C. Grammar Guided Genetic Programming

Most domains in these experiments were encoded in Koza-style expression-tree GP (denoted as GP). One domain required more sophisticated mechanisms to restrict the individuals generated. We used Grammar Guided GP (GGGP), in which grammars delineate the search space in the form of the Context-Free Grammar (CFG)-based GGGP of [18].

III. Early Termination

Genetic Programming (GP) is computationally intensive: large populations and many generations lead to many fitness evaluations; bloat leads to large individuals so that each evaluation is expensive [3]; for many problems, multiple tree evaluations are required for each individual. In the latter cases, fitness evaluation is incremental, and early stages may give a more or less reliable estimate of the ultimate fitness, depending on the specific problem. One way to reduce the cost of fitness evaluation is to terminate early, and use the estimate as a surrogate, when it is sufficiently reliable.

For such problems that require incremental fitness evaluation, we use a sequence of intermediate fitness values

for the current individual, along with previous fitness values from fully-evaluated individuals, to decide whether to stop evaluating. If we stop, we use this information to estimate the final fitness value of the individual. We aim to stop as early as possible, without adversely affecting the algorithm behaviour.

There is a trade-off between evaluation time and reliability: full evaluation is reliable, but at high cost; entirely omitting evaluation eliminates that cost, but gives random results. The 'sweet spot' lies between. There is one other important consideration: the anticipated fitness of the individual should feed into the determination of effort. The best-of-run individual needs to be evaluated exactly, since we eventually need to know its exact fitness. Conversely we do not care whether an individual is the least fit in a population or the second-least-fit – it is highly unlikely to be selected, and its progeny are unlikely to survive. So predictions of high fitness justify more evaluation effort than predictions of low fitness.

Intermediate fitness values from multiple evaluations generate a kind of time-series problem, as the update to a previous intermediate fitness value with the current evaluation result determines the next intermediate fitness value. However the problem differs from classical time series prediction in that we care only about estimates of the final value, not about intermediate values. Conversely, in some ways it looks like an online learning problem, but violates the common assumptions of machine learning in at least the following ways:

1) Goal: what we wish to optimise is not the accuracy of estimation of the individual's final fitness per se, but its effect on the GP system's eventual solution
2) Cost: what we minimise is the evaluation effort on each instance, not the number of instances
3) Training distribution: we cannot afford to fully evaluate an unbiased set of training instances. We must learn from a biased sample: individuals which are likely to be fit (and thus justify full evaluation).
4) Independence: we could potentially treat estimation at each evaluation stage as a separate learning problem; but this ignores time relationships, discarding information in an already information-poor domain. If we treat the estimates from different stages together, we lose the assumption of independence.

So while this problem resembles both time-series estimation and machine learning, it is rather different from either; we have been unable to find any preceding theory.

We focus on the case where the evaluation order of fitness cases is fixed, although the proposed method can handle the non-fixed case as well. In the former case, other instance-based evaluation cost reduction techniques such as [16], [17] are inapplicable, since they rely on re-ordering or random sampling of fitness cases.

Our aim in this paper is to

1) Better understand how the problem domain affects the amount of speed-up gained from early termination
2) Better understand how early termination affects algorithm performance (quality of solutions), and its variation with problem domain

Algorithm 1 Fitness Computation with Early Termination

```
Initialize bestSoFar to a large value.
procedure COMPUTEFITNESS(individual)
    fitness ⇐ 0
    i ⇐ 0
    while i < NumFitcases do
        Update fitness of individual with fitness case i
        if fitness > bestSoFar then ▷ Early Termination
            return Extrapolate(fitness, i, NumFitcases)
        end if
        i ⇐ i + 1
    end while
    if fitness < bestSoFar then
        bestSoFar ⇐ fitness
    end if
    return fitness                          ▷ Full Evaluation
end procedure

procedure EXTRAPOLATE(fitness, i, NumFitcases)
    slope = fitness/(i + 1)
    return NumFitcases × slope  ▷ Linear Extrapolation
end procedure
```

For one class of problems, deciding when to stop, and how to extrapolate fitness, are easier. Many incremental evaluation problems minimise an error function over a set of points. Many use the l^1 norm (mean absolute error, MAE), with an important property: the absolute error monotonically increases as we evaluate cases. If we terminate after it exceeds the best individual so far, and extrapolate linearly over remaining cases, the actual MAE must exceed that of the best-so-far: we will fully evaluate the best individual. Others use higher norms, notably the l^2 norm (root mean square error, RMSE). We are not guaranteed to fully evaluate the fittest individuals, but its approximate monotonicity means we are unlikely to get very wrong estimates. In either case, the earlier we terminate, the more likely it is that the actual fitness is bad (in which case, any errors do not matter much). In this paper, we concentrated on such problems. The detail is shown in algorithm 1.

IV. EXPERIMENTS

A. Experimental Design

We studied the effects of early termination on a number of problems: some symbolic regressions, two toy differential equations, and a further one abstracted from a real-world application. For each, we ran a statistically meaningful number of two types of runs: with and without early termination.

B. Test Problems

The test problems are shown in table I, and detailed below:

1) Symbolic Regression Problems: Each consists of a target real-valued function, sampled at random points over a range; target, range, and number of samples are shown in table I.

2) Parity Problems: The target Boolean even parity function is sampled over all possible Boolean inputs; the problem size and number of samples are shown in table I.

TABLE I. PROBLEM DETAILS

Problem Class	Problem Name	Target Function	Range	Fitness Cases
Alg	Qrt	$x^4 + x^3 + x^2 + x$	$x \in [-1, 1]$	200
Alg	Root	$\sqrt{x}$	$x \in [0, 4]$	200
Alg	2vFrac	$\frac{1}{1+x^{-4}} + \frac{1}{1+y^{-4}}$	$x, y \in [-5, 5]$	400
Alg	2vQrt	$x^4 - x^3 + \frac{y^2}{2} - y$	$x, y \in [-3, 3]$	400
Trig	Trg1	$\cos 3x$	$x \in [0, 2]$	200
Trig	Trg2	$\sin x^2 \cos x$	$x \in [-2, 2]$	200
Trig	2vTrg1	$xy + \sin(x-1)(y-1)$	$x, y \in [-3, 3]$	400
Trig	2vTrg2	$6 \sin x \cos y$	$x, y \in [-5, 5]$	400
Parity	E5P	parity function of 5 inputs	$b_i \in \{0, 1\}$	32
Parity	E6P	parity function of 6 inputs	$b_i \in \{0, 1\}$	64
Parity	E7P	parity function of 7 inputs	$b_i \in \{0, 1\}$	128
DE	DE1	$e^{-\sin x} + 2$	$x \in [-2, 2]$	400
DE	DE2	$x^4 + x^3 + x^2 + x$	$x \in [-2, 2]$	400
Real	Lake	See detail below		2424

3) Differential Equation Problem: In Koza's form [1], differential equations are symbolic regression problems in which the target function is specified indirectly by a differential equation and sufficient boundary conditions.

The equations we used were:

$$\frac{dy}{dx} + y\cos x - 2\cos x = 0 \quad \text{where } y_0 = 3 \text{ for } x_0 = 0 \tag{1}$$

$$\frac{dy}{dx} - 4x^3 - 3x^2 - 2x - 1 = 0 \quad \text{where } y_0 = 0.6496 \text{ for } x_0 = 0.4 \tag{2}$$

with closed form solutions $e^{-\sin x} + 2$ and $x^4 + x^3 + x^2 + x$.

a) Fitness Calculation: The absolute values of the differential equation (substituted by the individual) are summed to obtain the first component of fitness (fit to the differential equation). The derivative, $\frac{dy}{dx}$ is obtained by numerical approximation: for endpoints, the slope to the nearest point; for others, the average of the slope to left and to right. The second component (satisfaction of the initial condition) is the absolute difference between y_0 and the value of an individual at x_0, multiplied by the number of fitness cases. The overall fitness is 75% of the first component plus 25% of the second.

C. Lake Problem

TABLE II. GRAMMAR FOR LAKE PROBLEM

T	→	BA BZ			
BA	→	EXP	BZ	→	EXP
EXP	→	EXP OP EXP	EXP	→	PRE EXP | VAR
PRE	→	e^	OP	→	$+ \mid - \mid \times \mid /$
VAR	→	$V_{\text{ba}} \mid V_{\text{bz}} \mid V_{\text{alk}}$	VAR	→	$V_{\text{si}} \mid V_{\text{tb}} \mid V_{\text{cd}}$
VAR	→	$V_{\text{do}} \mid V_{\text{lgt}} \mid V_{\text{n}}$	VAR	→	$V_{\text{p}} \mid V_{\text{ph}} \mid V_{\text{sd}}$
VAR	→	$V_{\text{tmp}} \mid R$			

This problem is abstracted from a real world problem (too expensive to use in a parametric study; moreover we don't know the optimum solution), modelling algal growth in Korea's Lower Nakdong River. It incorporates a complex river flow model (derived from knowledge) and an evolved algal growth model [19]. The river is regulated by a sea barrage, and often flows quite slowly, with algal blooms concentrated at these times – so it may be modelled, with some cost to fidelity, as a lake. Indeed, this has been in some previous models of its algal growth [20]. To produce a realistic but less expensive version of this problem, we took the best model output from the river modelling system. We simplified the model manually, omitting river flows and other complexities, while retaining the overall behaviour:

$$\begin{aligned} \frac{\partial BA}{\partial t} &= V_{\text{ph}} + [(-0.126 \times V_{\text{ba}}) - (V_{\text{bz}} \times Grazing)] \\ \frac{\partial BZ}{\partial t} &= V_{\text{bz}} \times [(3.35 \times Grazing) - 0.01] \\ Grazing &= 0.0123 \times \frac{V_{\text{ba}}}{V_{\text{ba}} + 42.949} \\ &\quad \times e^{-0.0057 * (V_{\text{tmp}} - 20)^2} \end{aligned} \tag{3}$$

BA and BZ mean measurements of phytoplankton and zooplankton respectively, while other variables in the equation represent various parameters used in the model.

We used this model to generate new artificial data, which we took as the target of a differential equation modelling process similar to the preceding. Environmental attributes such as water temperature, alkalinity, etc. were taken from the measured data. The overall structure of the model was fixed, but the details were evolved. An individual contains two growth functions (BA for algae and BZ for zooplankton concentrations), which update the corresponding variables. The target is a growth model that can accurately predict BA. The grammar for the GGGP system is shown in table II.

b) Fitness cases: (Simulated) data measured at 36 hour intervals from 1996 to 2005 were used, forming 2424 fitness cases in all. As is standard in ecological modelling, we used an RMSE error function (thus we could not guarantee that an early-terminated individual was less fit than the best-so-far).

D. Evolutionary Parameters

TABLE III. PARAMETER SETTINGS

Number of Runs	60	Generations per Run	51
Population	500	Tournament Size	3
Minimisation Objective	SAE	(for lake)	RMSE
Crossover prob.	0.9	Mutation prob.	0.1
Elite Size	2		
Function Set (except Lake, Parity)	$+, -, \times, /, exp, log, sin, cos$		
Function Set (Parity)	AND,OR,NAND,NOR		
Function Set (Lake)	See table II		

The evolutionary settings used in all experiments are shown in table III.

V. RESULTS

A. Costs and Benefits of Early Termination

Table IV presents the performance (in terms of mean best fitness) of the original and early termination algorithms on all problems, together with the ratio, showing how much the performance deteriorated. The first notable point is that in general it did not deteriorate: early termination actually enhanced performance in the majority of cases, and on average the performance improvements somewhat exceeded the performance decrements. Thus the fitness landscape smoothing discussed in [8] outweighed other effects – though not, unfortunately, in the semi-real-world Lake problem.

Table IV also shows the corresponding time performances. Here, as would be expected, the improvements ranged from

TABLE IV. MEAN BEST FITNESS AND RUNNING TIME SPEEDUP: ORIGINAL AND EARLY TERMINATING ALGORITHMS

Problem Class	Problem Name	Mean Best Fitness			Mean Running Time (millisecs)		
		Original	Early Term.	Ratio	Original	Early Term.	Ratio
Algebraic	Qrt	2.97±1.35	1.09±1.68	2.72	57817±1337	13850±809	4.17
	Root	2.93±2.28	3.43±3.42	0.85	72430±15781	31125±9070	2.33
	2vFrac	43.22±9.42	39.57±9.81	1.09	147723±23488	85853±19949	1.72
	2vQrt	618±310	818±280	0.76	150609±30796	77942±24708	1.93
Trigonometric	Trg1	5.20±4.05	3.01±2.74	1.73	76256±18901	35858±17541	2.13
	Trg2	3.58±1.97	3.49±2.41	1.02	89180±16233	46403±15448	1.92
	2vTrg1	191±19	167±36	1.14	128473±24779	102691±22913	1.25
	2vTrg2	228±273	194±255	1.18	149180±27371	73617±46872	2.02
Differential Equation	DE1	28.91±10.75	28.2±7.64	1.02	277641±86364	162727±43934	1.71
	DE2	153±132	105±76	1.466	200224±65314	103176±40509	1.94
Parity	E5P	6.95±0.91	7.38±1.08	0.94	17170±2053	14776±3080	1.16
	E6P	19.83±1.29	20.35±1.23	0.97	40083±3648	41361±4095	0.96
	E7P	50.15±1.67	49.72±1.51	1.01	93414±8548	97098±9980	0.96
Real	Lake	17.74±3.20	21.01±2.64	0.84	268642±71929	172266±87096	1.56

substantial to negligible or, in a couple of cases, slightly negative (presumably, in these cases, the cost of checking for early termination outweighed gains from early termination; it is notable that these were among the fastest runs, so that evaluation costs were relatively low). Overall, the benefits of early termination are modest, but this is to be expected of the conservative termination policy used (only terminating when we could be virtually certain that the individual was worse than the best-so-far, and terminating very early only if it was very much worse). The gains would undoubtedly increase with a more optimistic policy. However performance improvement is not the focus of this paper: our aim is to examine possible determinants of performance improvement.

B. Classifying Performance

TABLE V. CLASSIFICATION OF PROBLEMS BY EARLY TERMINATION EFFECTS

Performance	Running Time	
	Similar	Improved
Worse		Lake Root 2vQrt
Similar	E5P E6P E7P	DE1 Trg2 2vFrac
Improved		DE2 Qrt Trg1 2vTrg1 2vTrg2

Table V classifies the different problems by two characteristics: whether or not there is a speed-up from early termination, and whether the resulting performance is better or worse than, or similar to, the original performance. The combination yields six classes, two being unrepresented. We would like to find what is common among them, to understand when early termination will be useful.

The only group with little or no improvement in running time were the Boolean parity problems: changes were so small that most termination must have been late, close to the last stage. Thus there was little change in resulting fitness. But it would be desirable to understand why termination was generally so late. Of the problems which did see speed-up, some saw an improvement in performance, others a decrease, and still others little change. We would like to understand why we saw these effects: what problem properties determined this behaviour?

C. The Effect of Estimation Accuracy

One potential determinant of the effect of early termination is the accuracy of fitness estimates: if early estimates were accurate in all generations, and at all stages of the fitness estimation process, then we would expect early termination to have no effect on the progress of runs (because early-terminated individuals would be ranked in the same order as they would have been under normal conditions, and thus the same individuals would be selected, meaning that the trajectory of evolution would be unchanged). Of course this is unachievable: there will be some error, and so there will be some change to the evolutionary trajectory. So the interesting question is whether that trajectory, either over generations or over evaluation stages (or perhaps both) can help to explain the behaviours we observed.

To assess this, we completed a further series of runs, without early termination, recording at selected stages of fitness evaluation the estimated fitness that would have been generated by early termination. Due to space constraints, these stages were chosen logarithmically – more in the early stages, fewer later, the actual stages depending on the problem. At each stage of fitness estimation, over all individuals in the population, we compared the rank order resulting from early termination with that resulting from full evaluation. We needed some way to measure the difference between these orders. We used the sum of the absolute differences in ranks over the population, as shown in equation 4:

$$d_{\text{order}}(i) = \sum_{j=1}^{N} |P(\text{rank}(j,i)) - P(\text{rank}(j,N))| \qquad (4)$$

where i is the stage number, N is the population size, $\text{rank}(j,i)$ is the rank of individual j at stage i, $P(r)$ is the probability that individual of rank r is chosen in a tournament under the assumption that all individuals have distinct fitnesses and the worst one is ranked first, i.e. $\frac{r^k-(r-1)^k}{N^k}$, k being the tournament size [21].

In generating plots, we averaged the distance d_{order} across all runs. In figure 1, we further averaged across all stages so that we could see the trajectory over generations (since all problems used the same number of generations, no normalisation of the x axis was required). In figure 2, we instead averaged across all generations so that we could see the trajectory over the stages of fitness case evaluation. However in this case, different problems had different numbers of stages;

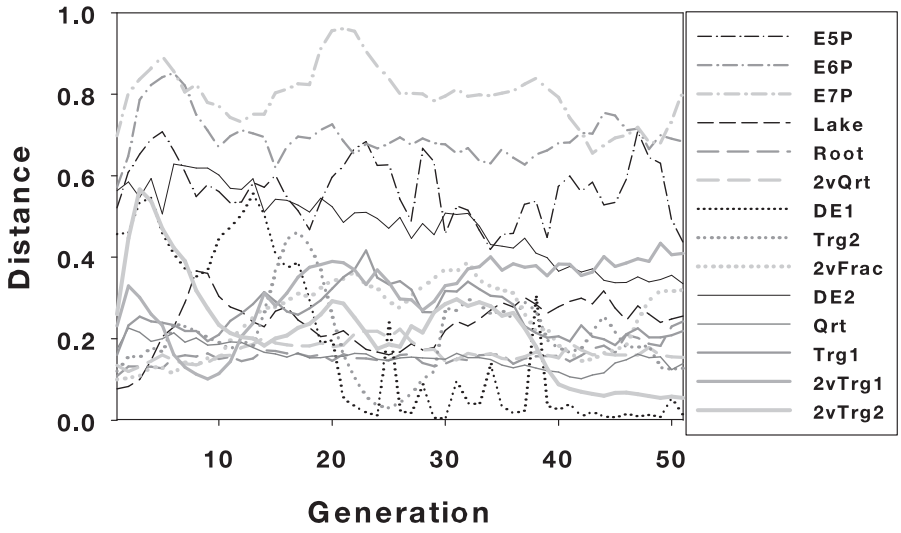

Fig. 1. Mean Order Difference d_{order} (Averaged over Runs and Stages) by Generation for All Problems

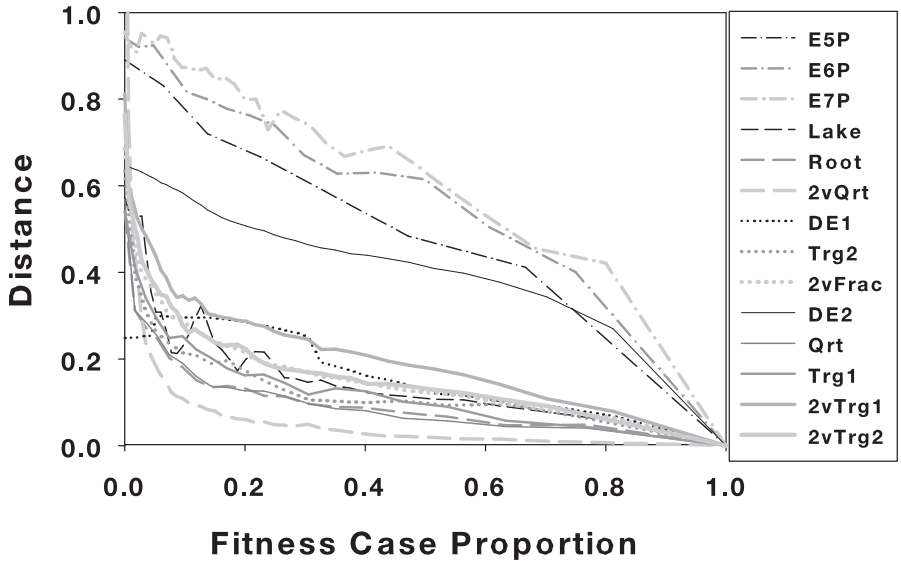

Fig. 2. Mean Order Difference d_{order} (Averaged over Runs and Generations) by Fitness Case Proportion for All Problems (the value for 2vQrt at 0.0 is off-scale at 1.78).

we normalised each problem by the total number of fitness case evaluations, so that all had the same x-axis scale, 0.0 to 1.0. In figures 1 and 2, we used similar line qualities for problems with the same behaviour in table V, so that where the property summarised in the table helps to classify the problem, we should see similar line qualities grouping together.

It is apparent from both that the Boolean problems group together: early termination generates the worst estimates, so that early termination will generate substantial noise in the estimates – perhaps large enough to deteriorate solution quality. However the case for the other problems is more complex.

Figure 1 does not give us much further useful information. There is little to differentiate the problem classes, and for most problems there is little trend in the order distance by generation, though it is worth noting that the 2vTrg2 and DE1 problems do exhibit a decreasing trend with generations (that is, the rank ordering generated by early termination improves as the run progresses); the other problems do not exhibit this trend, except perhaps weakly in DE2 (so perhaps this is a commonality generated by the form of the DE fitness function).

Figure 2 shows more structure: in general, the fitness ordering monotonically converges toward the final as more fitness cases are evaluated, but the shapes of the curves differ greatly, being upward convex in the case of the Boolean problems and DE2, and concave in the other cases: that is, early estimates are much better for the other problems. If anything, though, the problems with the worst early-termination performance (Lake, Root, 2vQrt) exhibited the *least* change in fitness ordering with

early termination – in the case of 2vQrt, termination after 10% of the fitness cases would yield a very accurate approximation of the final fitness ordering, and yet this problem saw the worst decrement in performance from early termination. Thus it seems that the extent or trajectory of fitness order changes with evaluation stage offer us only limited guidance as to which problems are likely to benefit from early termination.

We repeated this analysis with the symmetric Kullback-Leibler divergence, and both directions of the asymmetric divergence, with essentially the same results (omitted for lack of space): the exact metric does not seem to matter.

D. The Exploration - Exploitation Tradeoff and Early Termination

Another reasonable hypothesis is that early termination, in generating evaluation noise, provides additional exploratory power to the algorithm (the essence of the explanation in [8]), and thus helps it to overcome attraction to local optima. One way to evaluate this hypothesis is to look at the fitness curves of the evolution, since the shape of the fitness curve is correlated with the need for exploration. Figure 3 shows the mean best fitness by generation for runs with full evaluation. Again, we see that we can readily distinguish the Boolean problems, which are still seeing improvements in performance, but it is difficult to otherwise separate the early-termination performance classes in this figure. The 2vTrg1 performance curve stagnates at fairly poor fitness levels, suggesting that greater exploration might be desirable, and indeed we do see a performance improvement with early termination – yet the Lake problem exhibits a similar curve, but suffers a deterioration in performance with early termination.

We analysed this in another way by increasing the exploration in these algorithms more directly, increasing the rate of mutation in runs without early termination, on the basis that, if the effects of early termination noise were due to increased exploration, then directly increasing the exploration should show similar behaviour. The results (omitted due to lack of space) were similarly equivocal.

VI. Discussion and Conclusions

Early termination is a promising technique. In the very conservative form presented here it offers reasonably substantial reductions in computational cost with no loss, or even some quite substantial gains, in performance. However some problems see either little improvement in computational cost or significant decrements in performance. It is worth trying, since on average the effect is likely to be beneficial – and the cost of implementing it is low.

We have investigated candidate explanations for the differences in behaviour on different problems, based both on the quality of the fitness estimates generated by early termination, and on the probable exploration requirements of the problem domain. They were able to distinguish problems the likely improvement in computational cost. But we were not able to generate good predictions of the effect on performance. Thus the best advice we can give is to try it and see. Finding criteria that can provide good forecasts of the effects of early termination is an important future research direction.

Early termination is a potentially valuable addition to the arsenal of GP practitioners. In the very conservative form used here, its application is limited to problems with at least expected monotonicity in the fitness function. This is still, of course, a vast array of problem domains, covering for example all l^k norms, and indeed almost any conceivable error functions. But we probably don't need to be as conservative in terminating early – we don't need a guarantee that the current individual is fairly unfit before we terminate, we just need some reasonable confidence (precisely because of the well-known tolerance of evolutionary algorithms for error). Thus early termination should be extensible to other problem domains so long as we can find reasonable theory for the mixed time series/learning problem as delineated in section III. We see this also as an important direction for future research.

One important advantage of early termination is its seeming orthogonality to other methods for speeding up GP. In particular, there is no in-principle problem in combining early termination with surrogate or clustering methods; in neither case does the method need as accurate estimates of the fitness of its training cases when they are unfit as it does when they are highly fit (although current versions of surrogacy and clustering methods do not take advantage of this). For surrogacy methods in particular, noise in the less fit training examples may even be an advantage, forcing the surrogacy learner to concentrate on the highest fitness region of the search space, where its predictions are most important. So combining surrogacy and early termination methods is one of our highest priorities in future research.

As early termination changes the evolutionary trajectory, the generalisation ability of solutions may change as well. In a preliminary study, we measured the mean fitness of the best individuals over a test set (for all problems except parity), using the same settings. The ratio of mean best fitness between runs with and without early termination was generally similar to that for training error. In four of the eleven problems, the direction of change was the same, though the degree of change increased by between 10% and 20%.

While the current early termination controls only the termination point of each fitness evaluation, early stopping [22], which is used to fight overfitting in training a neural network, controls when the run should stop. They are completely orthogonal, so that they can be combined together, offering the prospect of both better generalisation and reduced evaluation. We intend to investigate this in the near future.

Thus more generally, we see early termination as a technique that warrants both experimental and theoretical research, to determine how it may most effectively be used. We hope that this paper represents a useful first step in that direction.

Acknowledgements

This research was supported by the Basic Science Research Program of the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (Project No. 2011-0004338). The ICT at Seoul National University provided research facilities for the study.

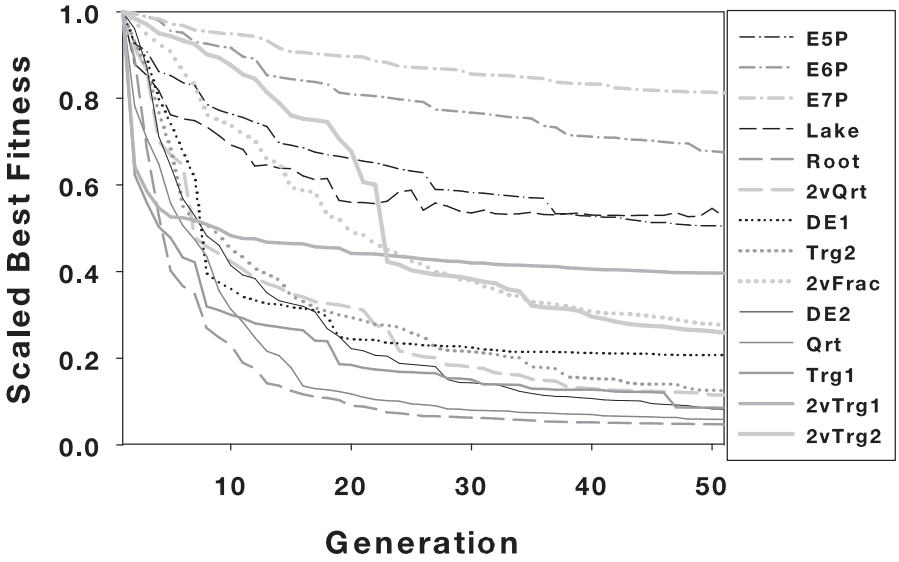

Fig. 3. Scaled Mean Best Fitness by Generation for All Problems (Full Evaluation)

REFERENCES

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

[2] ——, *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge Massachusetts: MIT Press, May 1994.

[3] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. San Francisco, CA, USA: Morgan Kaufmann, Jan. 1998.

[4] D. Andre and J. Koza, "A parallel implementation of genetic programming that achieves super-linear performance," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, vol. 3, 1996, pp. 1163–1174.

[5] H.-G. Beyer, "Evolutionary algorithms in noisy environments: theoretical issues and guidelines for practice," *Computer methods in applied mechanics and engineering*, vol. 186, no. 2, pp. 239–267, 2000.

[6] D. V. Arnold, *Noisy optimization with evolution strategies*, ser. Genetic Algorithms and Evolutionary Computation. Kluwer Academic Press, 2002, vol. 8.

[7] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments – a survey," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 303–317, 2005.

[8] T. Bäck and U. Hammel, "Evolution strategies applied to perturbed objective functions," in *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, jun 1994, pp. 40 –45 vol.1.

[9] P. Nordin and W. Banzhaf, "Evolving Turing-complete programs for a register machine with self-modifying code," in *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, L. J. Eshelman, Ed. Pittsburgh, PA, USA: Morgan Kaufmann, 15-19 Jul. 1995, pp. 318–325.

[10] S. Harding and W. Banzhaf, "Fast genetic programming on GPUs," in *Proceedings of the 10th European Conference on Genetic Programming*, ser. Lecture Notes in Computer Science, M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, and A. I. Esparcia-Alcázar, Eds., vol. 4445. Valencia, Spain: Springer, 11-13 Apr. 2007, pp. 90–101.

[11] W. B. Langdon and W. Banzhaf, "A SIMD interpreter for genetic programming on GPU graphics cards," in *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, ser. Lecture Notes in Computer Science, M. O'Neill, L. Vanneschi, S. Gustafson, A. I. Esparcia Alcazar, I. De Falco, A. Della Cioppa, and E. Tarantino, Eds., vol. 4971. Naples: Springer, 26-28 Mar. 2008, pp. 73–85.

[12] D. Hooper and N. S. Flann, "Improving the accuracy and robustness of genetic programming through expression simplification," in *Genetic Programming 1996: Proceedings of the First Annual Conference*, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds. Stanford University, CA, USA: MIT Press, 28–31 Jul. 1996, p. 428.

[13] P. Wong and M. Zhang, "SCHEME: Caching subtrees in genetic programming," in *2008 IEEE World Congress on Computational Intelligence*, J. Wang, Ed., IEEE Computational Intelligence Society. Hong Kong: IEEE Press, 1-6 Jun. 2008, pp. 2678–2685.

[14] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, vol. 9, no. 1, pp. 3–12, 2005.

[15] H. Xie, M. Zhang, and P. Andreae, "Population clustering in genetic programming," in *Proceedings of the 9th European Conference on Genetic Programming*, ser. Lecture Notes in Computer Science, P. Collet, M. Tomassini, M. Ebner, S. Gustafson, and A. Ekárt, Eds., vol. 3905. Budapest, Hungary: Springer, 10 - 12 Apr. 2006, pp. 190–201.

[16] I. Gonçalves, S. Silva, J. B. Melo, and J. a. M. B. Carreiras, "Random sampling technique for overfitting control in genetic programming," in *Proceedings of the 15th European conference on Genetic Programming*, ser. EuroGP'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 218–229.

[17] C. Gathercole and P. Ross, "Tackling the boolean even n parity problem with genetic programming and limited-error fitness," *Genetic programming*, vol. 97, pp. 119–127, 1997.

[18] P. Whigham, "Grammatically-based genetic programming," in *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, vol. 16, no. 3, 1995, pp. 33–41.

[19] D.-K. Kim, B. McKay, H. Shin, Y.-G. Lee, and X. H. Nguyen, "Ecological application of evolutionary computation: Improving water quality forecasts for the Nakdong River, Korea," in *World Congress on Computational Intelligence*, IEEE Computational Intelligence Society. Barcelona: IEEE Press, July 2010, pp. 2005–2012.

[20] K.-S. Jeong, D.-K. Kim, P. Whigham, and G.-J. Joo, "Modelling *microcystis aeruginosa* bloom dynamics in the Nakdong River by means of evolutionary computation and statistical approach," *Ecological modelling*, vol. 161, no. 1, pp. 67–78, 2003.

[21] T. Motoki, "Calculating the expected loss of diversity of selection schemes," *Evolutionary Computation*, vol. 10, no. 4, pp. 397–422, Dec. 2002.

[22] L. Prechelt, "Early stopping - but when?" in *Neural Networks: Tricks of the Trade*, ser. Lecture Notes in Computer Science, G. Orr and K.-R. Mller, Eds. Springer Berlin Heidelberg, 1998, vol. 1524, pp. 55–69.