Wave: Incremental Erosion of Residual Error

David Medernach
BDS Group
CSIS Department
University of Limerick
david.medernach@ul.ie

Jeannie Fitzgerald
BDS Group
CSIS Department
University of Limerick
jeannie.fitzgerald@ul.ie

R. Muhammad Atif Azad
BDS Group
CSIS Department
University of Limerick
atif.azad@ul.ie

Conor Ryan
BDS Group
CSIS Department
University of Limerick
conor.ryan@ul.ie

ABSTRACT

Typically, Genetic Programming (GP) attempts to solve a problem by evolving solutions over a large, and usually predetermined number of generations. However, overwhelming evidence shows that not only does the rate of performance improvement drop considerably after a few early generations, but that further improvement also comes at a considerable cost (bloat). Furthermore, each simulation (a GP run), is typically independent yet homogeneous: it does not re-use solutions from a previous run and retains the same experimental settings.

Some recent research on symbolic regression divides work *across* GP runs where the subsequent runs optimise the *residuals* from a previous run and thus produce a cumulative solution; however, all such subsequent runs (or iterations) still remain homogeneous thus using a pre-set, large number of generations (50 or more). This work introduces *Wave*, a divide and conquer approach to GP whereby a sequence of short but *sharp*, and dependent yet potentially *heterogeneous* GP runs provides a collective solution; the sequence is akin to a *wave* such that each member of the sequence (that is, a short GP run) is a *period* of the wave. Heterogeneity across periods results from varying settings of system parameters, such as population size or number of generations, and also by alternating use of the popular GP technique known as *linear scaling*.

The results show that Wave trains *faster* and *better* than both standard GP and multiple linear regression, can prolong discovery through constant restarts (which as a side effect also reduces bloat), can innovatively leverage a learning aid, that is, linear scaling at various stages instead of using it constantly regardless of whether it helps and performs reasonably even with a tiny population size (25) which bodes well for real time or data intensive training.

GECCO '15, July 11 - 15, 2015, Madrid, Spain

ISBN 978-1-4503-3488-4/15/07…$15.00

DOI: http://dx.doi.org/10.1145/2739482.2768503

Categories and Subject Descriptors

D.1.2 [**Programming Techniques**]: Automatic Programming; I.2.6 [**Artificial Intelligence**]: Learning—*Induction*

Keywords

Genetic algorithms; Genetic programming; Fitness landscapes; Performance measures; Machine learning; Semantic GP

1. INTRODUCTION

Performance curves of Genetic Programming (GP) [12] typically rise steeply in the early generations before flattening. Later generations gradually accumulate small improvements; however, these improvements spread over a large number of generations and typically accompany *code bloat* [13]. Thus, there are diminishing returns as the run progresses. Not surprisingly then, a lot of GP literature innovates to improve the quality of GP runs, in particular, to extend improvement enjoyed by earlier generations.

However, a question naturally arises; given that GP performs most efficiently in the earliest generations, why bother with the later generations at all? Instead, can GP not leverage the speed of the initial generations repeatedly through multiple re-starts which *build upon* the progress made in the previous run or runs?

A biological theory that supports this is the theory of *Punctuated Equilibrium* [6] which suggests that evolutionary changes are not necessarily uniform, but may instead emerge from long periods of stasis followed by rapid change (characterized for example by speciation). The biological literature reports multiple causes for such evolutionary changes, for example, *saltationism* [5] whereby important mutations quickly relocate offspring to a distant and better point on the fitness landscape than the parent, *ecological changes* [16] in the environment (and therefore in the fitness landscape) which may encourage a species to specialize to a new resource which can result in a rapid and important phenotypic change, and/or *peripatric speciation* [15] where the geographical separation of asymmetric groups can allow the

smallest to rapidly evolve phenotypic differences. The research question then is whether we can effect artificial punctuated equilibrium in GP through repeated *restarts* such that multiple GP runs co-operate to produce an effective combined solution.

We take the view that the interesting problems are so difficult that every computing cycle expended on GP should contribute to the eventual solution; multiple GP runs for the sake of generating statistics are all well and good, but when it comes to large scale problems rather than simple benchmarking, no cycle should be left behind.

Cascading evolutionary computation (EC) runs have existed for some time (see section 2 for some background). *Wave*, the proposed system differs from other approches because it embraces heterogeneity and leverages different parameters (population size, use of Linear Scaling, etc.) by permitting the constituent runs to be significantly different from each other. Similar to the manner in which a natural wave goes through several *periods*, we aggregate improvements, eventually ending with a joint solution which is the sum of solutions presented at the end of each period. This is similar to creating an ensemble, except that the joint solution does not comprise of independently evolved components.

This has some biological parallels where not only the environment changes over time, but the evolving entities themselves *cause* the environmental changes; moreover, changing target data after every period and using heterogenous settings for each period causes a change in the fitness landscape. This paper tests whether this metaphor works towards producing an efficient GP system that uses computationally cheap generations to match or even improve upon the results that GP normally produces over extended runs with significant code growth. We find that *Wave* results in an efficient GP system, which matches and often improves over the standard GP and only at a fraction of the cost.

The paper reads as follows: section 2 introduces Wave; section 3 discusses previous approaches to dynamic fitness landscape in GP; section 4 discusses the experimental setup and the problem suite used in this study; section 5 details the results and discusses their significance; and, finally, section 6 concludes the paper highlighting the achievements and directing further work.

2. WAVE

Wave (see Figure 1) uses a collection of heterogenous GP runs to create a dynamic fitness landscape. In GP, the fitness landscape is the direct consequence of the data points used to compute the training fitness. With Wave, instead of doing one GP run (with a fixed parameter setting) over a large number of generations, we use a succession of smaller heterogenous (in terms of parameter settings) runs, and simply linearly sum their best results (without semantic operators). Each short run stops when it ceases to produce a significant improvement. At the end of each of these short runs we modify the fitness landscape so that the next one *builds* on the previous work. These short runs could thus be seen as periods of a wave progressively eroding the fitness landscape, thus using a *divide and conquer* approach. Here, at the end of each period the best evolved individual is used to *reset* targets of our data-set, thus creating a new fitness landscape; this resetting results by posing the *residual* between the target and the best evolved output as the new target for the next period. The next period starts afresh with a new population.

First we formalise the terms for the Wave system; this will facilitate describing the experimental settings and discussing the results.

A **period** (as shown in Figure 1) is similar to a normal GP run (randomly initialized at the first generation and terminated when the last generation is reached) except that at the end of each period the best individual is used to compute new target values; this creates a new fitness landscape for the subsequent period. Thus, if t is the target value at the start of a period, and f is the best evolved function for this period, then t' is the new data-set for the next period such that $t' = t - f$. We end a wave when $MaxP$ periods have been processed.

While Wave is our proposed system, *a* **wave** is a particular collection of periods where the target varies across periods in the manner described above.

Notice, although given the definitions, a standard GP run can be described as a wave composed of a single period, to avoid confusion, we do not describe such runs as waves; instead, we refer to them as standard GP (GP) runs.

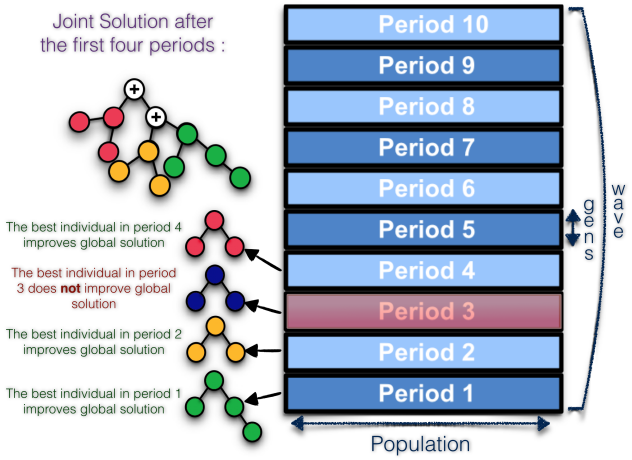


Figure 1: A simple Wave setup is depicted, where the population size and the number of generations stay fixed across the periods. Although the joint solution simply adds the best results of various periods, notice however, that each best result is added to the joint result only if it decreases the cumulative error thus far. Width of periods is proportional to the size of the population, height of periods is proportional to the number of generations.

2.1 Updating Target Values

As illustrated in Figure 1, the final solution of a wave results from summing the outputs of the best individuals of successive periods, where each period *potentially* optimises a unique target data t'. We say *potentially* because some times the target data may not be renewed for a new period. This happens if the current period, optimising over a current target set t, fails to produce an individual f_i such that $(f_i - t)^2 < (0 - t)^2$. In other words, adding f to the joint solution retains (at best) or worsens the error of the joint solution. In this case, we do not add f_i to the joint solution (we deem f_i *ineligible*) and, instead, launch the next period to again optimise over the same target set t. Therefore, $f_{joint} =$

$\sum_{i=1}^{n}(f_i)$ where f_{joint} is the joint solution of a wave, n is the number of periods that produced eligible functions and f_i is the function from such a period i.

2.2 Heterogenous periods

Key to Wave's design is the use of heterogenous peroids. Each wave can use completely different settings, even to the extend of using different search algorithms, as long as the output of the particular wave can be combined with the others.

In this work we consider four aspects that we vary throughout the entire run. These are the use of a flexible end point for periods, varying the population size, the number of generations and alternating the use of Linear Scaling.

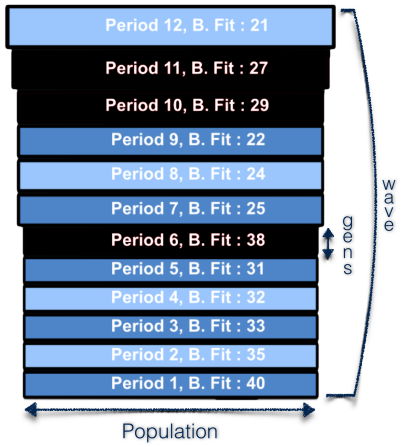


Figure 2: A wave is depicted. Its periods vary in population size & number of generations (periods in black failed to improve joint solution). Width of periods is proportional to the size of the population, height of periods is proportional to the number of generations.

2.2.1 *Flexible period ending*

We decide that a period should end if the best fitness has not improved significantly during the last few generations. Using the punctuated equilibrium analogy we stop a period during a static phase but not during a phase of rapid change. Therefore, to decide whether we are in a static phase, we compare the improvement over the last two generations (current improvement) with that over the three generations before the last two (previous improvement). If the current improvement is less than 0.5% of the previous improvement, we end the period. However, each period still undergoes a certain minimum number of generations before we take action. Therefore, the following condition formalises the period stopping criteria:

$g_c > g_m$ and
$(B^F(g_c) - B^F(g_c - 2)) \leq (B^F(g_c - 2) - B^F(g_c - 5))/200$
where g_c is the current generation, g_m is a minimum number of generations before a period stops and $B^F(g_c)$ is the best training fitness at generation g_c.

2.2.2 *Varying population size and minimum number of generations*

As Figure 2 shows, it is possible to vary the number of generations and population size for each period. Given that each problem with standard GP may require a different value for these two parameters [20], varying them across different periods of a wave may make sense. Thus, we observe the effect of increasing the population size and the minimum number of generations before stopping a period, but only when the previous period failed to improve the fitness of the joint solution. These changes are *cumulative*.

2.2.3 *Alternating between using and not using Linear Scaling:*

Linear scaling [11] optimises the slope and intercept of an evolving function and has been effective on symbolic regression problems. However, we question if it *consistently* improves performance: in some experiments for this study, standard GP outperformed *scaled* GP. Therefore, we also report experiments where periods alternate between using and not using linear scaling. In these cases the initial period uses linear scaling[1].

3. BACKGROUND

In this paper we investigate an approach to problem solving with GP which is analogous to ecological change in nature. We propose modifying the fitness landscape in order to make a problem more malleable for GP. This method could either be seen as a way to introduce ecological changes with a radical modification of the fitness landscape or as a form of *saltationism*. The creation of a new population after modifying the target that uses the previous evolutionary result as a black box unmodifiable part of the individuals (which is biologically realistic, as elements which appeared early in the evolutionary process are usually less likely to be modified), is akin to a high amount of mutation in the hope of skipping long periods of stasis out of the evolutionary process.

The Wave paradigm also offers an additional saving in computational effort as there is no need to re-evaluate the fixed (black box) sub-expressions. This provides the potential for increasing the total complexity of individuals without additional cost. Relatively similar methods have also proved to be efficient to solve time series problems [8]. Moreover, different *periods* of Wave can use different GP settings, so that the final joint solution emerges from a varied set of evolutionary simulations. For example, in this paper, we question whether *linear scaling* [11], a very successful aid to symbolic regression with GP, should be used throughout or only in alternate periods.

Outside the field of GP, the most similar method could be *cascade correlation (CasCor)* [22], where hidden layers are added to a previously trained artificial neural network and are trained to reduce its residual error.

There is also some precedent in the GP literature for taking this approach. For example, *Interleaved Sampling* [3, 9] creates an oscillating fitness landscape by alternating between different fitness measures every second generation. *Sequential Covering Genetic Programming* (SCGP) [19] sequentially decomposes the initial Boolean problem by running new iterations of GP only on the unsolved points in the training dataset and consequently on a new fitness landcaspe. A similar approach has also been used to address the class-imbalance problem in classification [7]. More generally, in EC based *Novelty Search* [14], the objective is to

[1]Exploratory experiments showed that Wave performs better this way.

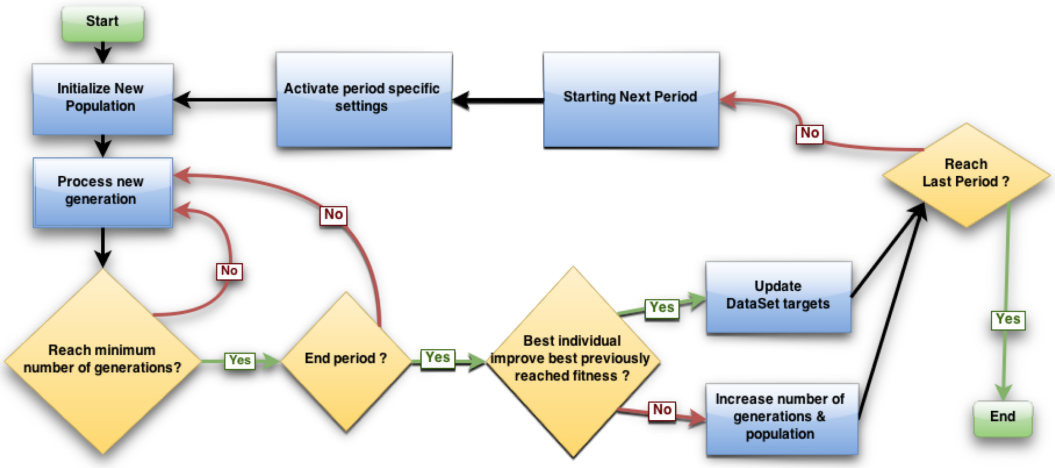


Figure 3: A flowchart summarizing the different steps of Wave is depicted. On the settings used in this work *activate perdiod specific settings* only occurs while alternating between using and not using Linear Scaling and *increase number of generations & population* while varying population size and minimum number of generations.

continuously find novel solutions, and consequently the target landscape continues to change.

Unlike these methods, *Sequential Symbolic Regression* [18] (SSR) tackles symbolic regression problems where the target output is not discrete. Like SCGP, SSR spreads the task of approximating the training data across a number of GP runs; each such run is termed an *iteration*. At the end of each iteration, outputs of the original problem are modified based on the use of a geometric semantic crossover [17] on the output of the best evolved solution in the current iteration[2]. Whereas Semantic GP combines individuals at random, SSR optimises the effect of the geometric semantic crossover operator in the next iteration by evolving the best match to the curent solution to solve the problem; however, each iteration is homogeneous and typically uses a large number of generations (50 or 100). A key difference is Wave's use of heterogenous periods compared to the homogenous ones employed by SSR. Similarly, Wave's periods are considerably shorter [3] and are variable rather than fixed length. Furthermore, the best individual evolved at the end of a period is added to the global solution only if it improves the fitness of the combined solution, and the way individuals are added to the global solution is simpler with wave (a simple sum of selected individuals against a geometric semantic crossover with a random r factor).

Another alternative to randomly combining individuals in semantic GP is ESAGP (Error Space Alignment GP) [23] which optimally combines individuals with an aligned error vector or, groups three individuals with co-planar error vectors (error vectors existing on the same bi-dimensional plane which intersects the origin of the error space). Another recent relevant approach is Multiple Regression Genetic Programming that combines a program's sub-expressions via multiple linear regression (MLR) [2] and claims this approach crucial to improving upon the results produced by MLR alone.

[2]The r factor is set randomly at the end of each iteration.
[3]The sharp rise of performance curves of GP in the early generations motivated this choice.

Each of these systems significantly modifies GP, so incorporating them into an existing system is not a trivial task. Even SSR, which needs the fewest changes, still requires the ability to perform geometric semantic crossover. Wave, on the other hand, only minimally changes the existing GP implementations, and yet significantly reduces the cost (short, non-bloating runs) and the complexity of the evolved solutions. Often superior quality also results. Next, we detail this system.

4. EXPERIMENTS

Table 1 lists the configuration parameters that we consistently use across all the experiments, except where stated otherwise.

Table 1: GP Parameters.

Parameter	Value
Replacement Strategy	Generational
Operator Probabilities	Xover: 0.9; Point mutation: 0.1
Tournament Size	10[a]
Max. depth	17
Max. size	100
Functions set	$+,-,\times,\div$[b]
Terminal set	Inputs and constants -1.0, -0.5, 0.0, 0.5 & 1.0
Fitness	RMSE
Initialisation	Ramped half & half
Max. initial depth	8

[a] A relatively high tournament size but recently successfully used in [9] and [3].
[b] Secured division.

4.1 Problem Suite

For this study we have used three multi-dimensional data-sets from the UCI Machine learning repository [4] and two mathematical functions. Those from the UCI: **Concrete Strength** where the objective is to predict the compressive strength of concrete and data-set includes 1030 instances each having 8 inputs; **Yacht** where the objective is to predict the hydrodynamic performances of a yacht, and data-set includes 308 instances each with 7 inputs; and **Powerplant** where the task is to predict the net hourly electrical energy

output of a power plant and data-set includes 9568 instances and 4 inputs.

The two mathematical functions are: **Poly-10 [21]** $y = x1 * x2 + x3 * x4 + x5 * x6 + x1 * x7 * x9 + x3 * x6 * x10$ and **Div-5 [24]** $y = \frac{10}{5+\sum_{i=1}^{5}(x_i-3)^2}$. For each function, we randomly generate 500 data points in the range $[0; 1]$ and for each wave we randomly split the given data-set into two subsets of equal size for training and testing purposes.

4.2 Benchmarks

The benchmarks we have chosen to compare Wave with include both EC based and non-EC based machine learning methods. The EC based methods are standard GP, both with and without linear scaling, with a population size of 500; each run spans 100 generations. The non-EC method is multiple linear regression (MLR), an efficient method to solve regression problems which, in an award winning recent effort, has been put forward as a tough benchmark for GP[2]. For all experiments we split the data randomly into equal partitions for training and testing purposes. At each generation, each individual's fitness is computed both on testing and training dataset. Selection, of course, is conducted using only training fitness. For clarity and space restrictions. we do not report completely homogeneous runs as in SSR; exploratory experiments with consistently long periods (100 or 50 generations) reported much larger individuals than with flexible periods over much smaller number of generations. However, the experiments which consistently either use or do not use linear scaling are an approximation of the homogeneous runs, and also present tougher benchmarks.

4.3 Naming Conventions

The naming convention we adopt for Wave GP set-ups has the following format:

$Wave : PeriodsNumber : Setting - P : PopulationSize$ where $PeriodsNumber$ is the number of periods in each wave, $P : PopulationSize$ is the population size of the first period of the wave (e.g. $P : 500$ is a population of 500 individuals) and $Setting$ indicates whether linear scaling was used (LS) or not (NS). However, $LS : NS - P$ indicates alternating between scaled and normal periods. Similarly, standard GP settings follow a similar pattern, that is, $GP : Setting - P : PopulationSize$.

Table 2: Different Wave and GP configurations.

Method	LS [a]	Pop[b]	Gen [c]	G. Inc[d]	P. Inc [e]	Period[f]
GP:LS-P:500	On	500	100	NA	NA	NA
GP:Norm-P:500	Off	500	100	NA	NA	NA
Wave:25:LS-P:100	On	100	10	1	18	25
Wave:25:NS-P:100	Off	100	10	1	18	25
Wave:200:LS-P:25	On	25	5	0	0	200
Wave:25:LS-P:500	On	500	10	0	0	25
Wave:25:NS-P:500	Off	500	10	0	0	25
Wave:25:LS:NS-P:500	Alt[g]	500	10	1	18	25

[a] Linear Scaling (LS)
[b] Initial population size
[c] Initial minimum number of generations
[d] Number added to current minimum number of generations after a Period failed to improve training fitness
[e] Number added to current population size after a Period failed to improve
[f] Number of consecutive Periods
[g] Alternation between LS and non LS, Gen 1 with LS. training fitness

5. RESULTS AND DISCUSSION

Because of phenomena such as over-fitting, the best test fitness is not necessarily reached either at the last period of a wave or at the last generation of an GP run. Therefore noting only the end of run results can be *unlucky* for any setup, and may require tailoring the end of runs to each setup. To avoid that and to make a fair comparison between GP and Wave, we measure the various run statistics at the end of each period for Wave experiments and every ten generations for GP; we call those reporting times *moments*. Measures at a particular moment include items such as the curent generation or the current period, the training fitness, the testing fitness, the number of nodes processed, amongst others.

In this study, at every moment, we report median values for training and test fitness as the median is a robust measure of central tendency and represents data more meaningfully [10]. We also present the best moment (in terms of testing fitness) for each configuration and the total number of nodes, which is the sum of the tree sizes of all evaluated trees until the moment reported and is expressed in nodes, 10^{10} nodes. By presenting the best moment, instead of a consistent, pre-determined fixed moment (or the last generation for GP), we avoid reporting at an unlucky point in time. We can however, see whether such a moment comes very early without evolving much or at a reasonably later point which tests whether the algorithm in question can battle over-fitting.

For each data-set, except powerplant, the reported results for Wave and GP experiments are median value on 100 runs (or waves) of each. However, for the powerplant data-set, we report results only for 30 runs at each configuration[4]. We also run MLR 100 times[5].

Due to the heterogeneity of the settings investigated, it can be difficult to compare them in a fair manner on testing fitness alone, as we cannot expect a wave with a small fixed population to do better than one with a large dynamic population. However, the smaller waves may nevertheless be useful if they can achieve acceptable testing fitness with a modest computational cost. To look into the trade off between efficiency and computational cost of the respective systems, Figure 4 plots the testing fitness against the number of nodes processed since the beginning of a run at each moment.

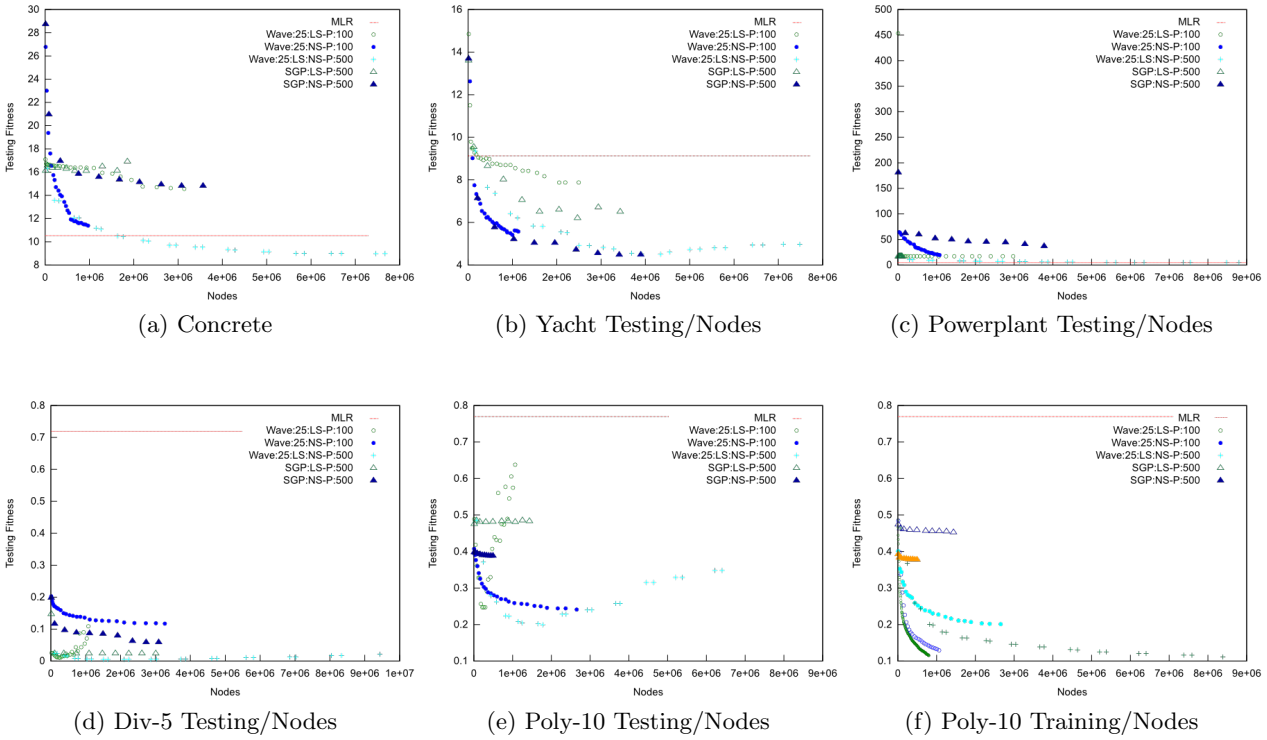
5.1 The speed/accuracy trade off

Usually testing fitness is the key criterion to compare the performance of machine learning algorithms; however, with a deluge of data (or *big data*), training efficiency also becomes important, particularly when initial training runs are conducted to estimate the distribution of the data or salience/relationships/dependencies within the data.

In cases such as these, it is more important to have *Fast Learners* that have a reasonable approximation to the solution, as they will keep the response time low. Thus, training speed of the cheapest Wave setup (Figure 4f), Wave:200:LS-P:25, is important for future applications.

In tables 3-7 we report the best median training and testing fitness among all moments for each setting on each data-set. Lowest testing fitness among Wave settings and among

[4]This is due to a high number of data points resulting in substantial computational cost.
[5]MLR method is deterministic but we use random training and testing data sets so result may vary between runs.



(a) Concrete (b) Yacht Testing/Nodes (c) Powerplant Testing/Nodes

(d) Div-5 Testing/Nodes (e) Poly-10 Testing/Nodes (f) Poly-10 Training/Nodes

Figure 4: Fitness / number of nodes is evaluated for each moment. The fitness is computed on the testing data-set for Figure 4a to 4e and on the testing data-set for Figure 4f.

GP settings are in bold. Additionally, we report the **Trade Off Wave** which, at some moment, produced the best test fitness while consuming fewer nodes than the best GP moment. If the Trade Off Wave also has better test fitness than the best result with GP, clearly it is better both in terms of solution quality and expense.

We also note the **Fastest Good Wave**; this is a Wave set up which outperforms the best GP moment on test fitness and consumes fewer nodes than any other Wave.

Finally, we also report results from the MLR benchmark. Of course, in this case we can not compare the nodes used, neither can we record moments.

To test the statistical significance of reported differences in performance, we use the Mann-Whitney U test at $p = 0.05$.

5.2 Discussion

The results clearly show that, except on Yacht (insignificant difference), Wave with a population size of 500 with alternating linear scaling performs the best among all the configurations investigated. Those results are statistically significant. This is rather surprising because one would expect linear scaling to be more effective if used consistently. While the best moment for the best Wave set-up consumed a higher node requirement than GP, Figures 4 shows that this setup is the best even after consuming a similar number of nodes. Also, a higher node count for the best Wave moment than that with GP shows that the extra computation with Wave is not promoting over-fitting, which shows an implicit bias in Wave towards *regularisation*.

Also notice that while the *best* moment of the best Wave set-up is computationally more expensive than that with GP, the *Trade Off Wave* and the *Fastest Good Wave* are by definition *considerably* cheaper for almost all of the problems tackled.

Although the cheapest Wave set-up, Wave:200:LS-P:25, is not among the best performers, Tables 5 and 6 demonstrate that it performs quite reasonably on some data-sets, despite a very small population size of only 25. This configuration can produce better solutions on out of sample data than GP, but even when it does not, the performance is not drastically worse, showing that it can operate reasonably with limited computational resources.

While figures 4 depict the good performance of Wave with a population size 500 in terms of test fitness, we see also that Wave with a population size of 100 achieves very good results with considerably fewer node evaluations. Therefore, the latter configuration may prove to be a good choice if computational resources are scarce. We also note that if for a particular data-set GP with LS does better than GP without LS, Wave with LS produces even better results.

Results indicate that Wave is an efficient method. The best wave moment is significantly better than the best GP moment on four of the five data-sets studied, and the improvement on testing fitness is substantial. Also as shown in table 8 the average tree sizes at the End of a Wave are usually significantly lower with Wave than with GP; note that we limit the number of nodes to 100 for GP which allows it generate surprisingly small individuals than with a typical limiting factors such as a depth limit of 17.

Table 3: Experimental results on Yacht dataset.

Method	Moment	Train	Test	Nodes
GP:LS-P:500	71g	5.16411	6.21251	2468k
GP:NS-P:500	81g	4.09004	**4.48998**	3410k
Wave:25:LS-P:100	23p	7.42025	7.87332	2056k
Wave:25:NS-P:100	22p	4.01469	5.41762	999k
Wave:200:LS-P:25	200p	9.02013	9.30088	450k
Wave:25:LS-P:500	25p	6.48936	7.08987	3229k
Wave:25:NS-P:500	9p	3.38198	4.73403	2464k
Wave:25:LS:NS-P:500	16p	3.35759	**4.50582**	4342k
MLR	NA	8.86019	9.11015	NA
Trade Off Wave :				
Wave:25:LS-P:500	9p	3.38198	4.73403	2464k
Fastest Good Wave :				
NA				

Table 4: Experimental results on Concrete dataset.

Method	Moment	Train	Test	Nodes
GP:LS-P:500	51g	14.1436	16.1199	671k
GP:NS-P:500	81g	14.6654	**14.8268**	3080k
Wave:25:LS-P:100	25p	14.3886	14.5569	3139k
Wave:25:NS-P:100	25p	10.3821	11.3939	970k
Wave:200:LS-P:25	12p	16.2683	16.3699	402k
Wave:25:LS-P:500	21p	13.8497	13.9208	1753k
Wave:25:NS-P:500	14p	8.72476	10.1065	4418k
Wave:25:LS:NS-P:500	24p	7.71554	**8.98308**	7463k
MLR	NA	10.3123	10.5693	NA
Trade Off Wave :				
Wave:25:LS:Norm;p:500	13 p	9.08195	9.69879	2950k
Fastest Good Wave :				
Wave:25:LS;p:500	2 p	14.4465	14.5430	133k

Table 5: Experimental results on Powerplant dataset.

Method	Moment	Train	Test	Nodes
GP:LS-P:500	1g	17.0788	**17.1299**	11k
GP:NS-P:500	91g	37.8269	37.2500	3775k
Wave:25:LS-P:100	2p	17.0738	17.0571	23k
Wave:25:NS-P:100	25p	18.8342	19.5098	1077k
Wave:200:LS-P:25	181p	17.0157	17.1246	403k
Wave:25:LS-P:500	2p	17.1188	17.0372	99k
Wave:25:NS-P:500	25p	10.7321	11.3897	7530k
Wave:25:LS:NS-P:500	24p	4.86973	**5.15350**	8480k
MLR	NA	5.12806	5.13028	NA
Trade Off Wave :				
Wave:200:LS;p:500	3 p	17.0561	17.0781	6k
Fastest Good Wave :				
Wave:200:LS;p:500	3 p	17.0625	17.0779	6k

Table 6: Experimental results on Div-5 dataset.

Method	Moment	Train	Test	Nodes
GP:LS-P:500	71g	0.02377	**0.02395**	2234k
GP:NS-P:500	81g	0.05738	0.05897	2731k
Wave:25:LS-P:100	7p	0.00698	0.00999	258k
Wave:25:NS-P:100	25p	0.10738	0.11704	3267k
Wave:200:LS-P:25	6p	0.02829	0.03099	15k
Wave:25:LS-P:500	6p	0.00442	0.00484	1050k
Wave:25:NS-P:500	23p	0.07700	0.08441	2143k
Wave:25:LS:NS-P:500	9p	0.00424	**0.00480**	1508k
MLR	NA	0.71604	0.71736	NA
Trade Off Wave :				
Wave:25:LS:Norm;p:500	10 p	0.00424	0.00480	1636k
Fastest Good Wave :				
Wave:25:LS;p:100	3 p	0.01808	0.02098	78k

The Fastest Good Wave results show that Wave not only achieves significantly better training fitness, but also produces at least equivalent testing fitness in four of the five data-sets, with significantly fewer node evaluations. For each dataset at least one moment of a wave reaches equal or statistically better testing fitness than the Best Standard GP moment, at a fraction of cost, even if the final combined program can reach huge sizes.

The best wave moment is also significantly better than MLR on four data-sets, and the difference in the remaining Powerplant dataset is not statistically significant.

In terms of training fitness, Wave with a larger population always reaches significantly better fitness than either GP or MLR. However, there is some evidence of over-fitting with this set-up, as can be seen in Figure 4f.

The setting which emerges as the most efficient is that which alternates between use and non-use of linear scaling (Wave:25:LS:NS-P:500). This outperforms the chosen benchmarks on four out of the five data-sets, and among Wave setups, is only outperformed by Wave:25:LS-P:500 on the poly-10 problem.

The heterogenity within the wave proves particularly useful because, rather surprisingly, linear scaling does not always produce the best results, even when employed with GP. A mixed approach using the Wave paradigm appears to be a more flexible strategy that produces consistently good results.

The Powerplant dataset is particularly interesting because both GP:LS and GP:Norm seem inefficient on this data. GP:LS does not seem to evolve at all and while evolution is apparent with GP:Norm, it is very slow. However, the combination of these two configurations in Wave:25:LS:NS-P:500 reaches good fitness in its early periods while also maintaining its ability to evolve as seen on Fig 4c.

6. CONCLUSIONS

In this paper, we present a novel approach to GP which we call Wave, where we use a small number of generations in each wave, compute the residual and delegate the outstanding improvement required to further runs. In this way we systematically leverage the inherent behaviour of GP by which rapid improvement typically occurs in early generations and we avoid the code growth associated with prolonged evolution.

The results of this preliminary investigation are very encouraging, demonstrating that Wave is a promising paradigm in the search for improved performance and scalability in GP. The Wave approach is extremely flexible, and we anticipate many possibilities for further study. For example, Wave could be totally or partially composed of periods using other EC methods such as Interleaved sampling. Another approach would be to vary evolutionary parameters such as function set or genetic operator types and/or probabilities during a Wave.

In fact, the Wave paradigm is so flexible that it does not necessarily need to be limited to EC and could, for example, be combined with MLR or other optimization methods.

There are, of course, some minor caveats, the most significant of which is that as the initial best sub-expression is systematically selected it is possible that if this first wave is not a good one we risk polluting the remaining periods. Informal experiments with alternating LS and normal GP tended to confirm this idea. Further work should address this potential issue.

Table 7: Experimental results on Poly-10 dataset.

Method	Moment	Train	Test	Nodes
GP:LS-P:500	1g	0.47447	0.47591	11k
GP:NS-P:500	71g	0.37882	**0.38868**	383k
Wave:25:LS-P:100	7p	0.35781	0.45138	32k
Wave:25:NS-P:100	25p	0.20110	0.24104	2653k
Wave:200:LS-P:25	9p	0.20627	0.24712	220k
Wave:25:LS-P:500	6p	0.16587	**0.18736**	1097k
Wave:25:NS-P:500	24p	0.17971	0.22286	2824k
Wave:25:LS:NS-P:500	10p	0.16333	0.19930	1777k
MLR	NA	0.76773	0.77197	NA
Trade Off Wave :				
Wave:25:LS;p:100	7 p	0.20627	0.24712	220k
Fastest Good Wave :				
Wave:25:Norm;p:100	3 p	0.35231	0.37657	51k

Table 8: Average size of individuals at the end of each period or GP run.

Settings	Yacht	Conc	P-10	Div-5	PowP
GP:LS-p:500	93.0	20.1	32.6	84.3	2.0
Wave:25:LS-p:100	11.0	2.0	2.0	40.9	2.8
Wave:25:NS-p:100	32.8	14.2	5.0	26.6	13.4
GP:NS-p:500	92.3	99.9	5.0	8.0	97.1
Wave:200:LS-p:25	14.2	2.4	2.4	2.0	2.2
Wave:25:LS-p:500	27.6	87.3	4.6	65.8	2.1
Wave:25:NS-p:500	47.3	36.2	30.9	10.2	33.8
Wave:25:LS:NS-p:500	21.3	8.0	14.9	34.0	2.0

In this initial study, we have focused on applying Wave to symbolic regression problems. It would be interesting to investigate how the Wave approach could be applied to other learning tasks such as classification.

7. REFERENCES

[1] LS Aiken, SG West, SC Pitts Multiple linear regression. *Handbook of psychology, Wiley Online Library*, 2003.

[2] I. Arnaldo, K. Krawiec, and U. O'Reilly. Multiple regression genetic programming. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 879–886. ACM, 2014.

[3] R. Azad, D. Medernach, and C. Ryan. Efficient interleaved sampling of training data in genetic programming. In *Proceedings of the 2014 conference companion on Genetic and evolutionary computation companion*, pages 127–128. ACM, 2014.

[4] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[5] D. G. Blackburn. Saltationist and punctuated equilibrium models for the evolution of viviparity and placentation. *Journal of Theoretical Biology*, 174(2):199–216, 1995.

[6] N. Eldredge and S. Gould. Punctuated equilibria: an alternative to phyletic gradualism. *Models in paleobiology. Freeman Cooper and Co*, pages 82–115, 1972.

[7] C. Gathercole and P. Ross. Dynamic training subset selection for supervised learning in genetic programming. *Parallel Problem Solving from Nature III*, volume 866 of *LNCS*, pages 312–321, Jerusalem, 9-14 October 1994. Springer-Verlag.

[8] L. Geum Yong. Genetic Recursive Regression for Modeling and Forecasting Real-World Chaotic Time Series. *Advances in Genetic Programming 3, MIT Press*, pages 401–423, 1999.

[9] I. Goncalves and S. Silva. Balancing learning and overfitting in genetic programming with interleaved sampling of training data. *Proceedings of the 16th European Conference on Genetic Programming, EuroGP 2013*, volume 7831 of *LNCS*, pages 73–84, Vienna, Austria, 3-5 April 2013. Springer Verlag.

[10] D. C. Hoaglin, F. Mosteller, and J. W. Tukey. *Understanding robust and exploratory data analysis*. Wiley series in probability and mathematical statistics. Wiley-Interscience, 1983.

[11] M. Keijzer. Scaled symbolic regression. *Genetic Programming and Evolvable Machines*, 5(3):259–269, September 2004.

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

[13] W B. Langdon and R. Poli. *Fitness causes bloat*. Springer, 1998.

[14] J. Lehman and K O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[15] E. Mayr. Speciation and macroevolution. *Evolution*, pages 1119–1132, 1982.

[16] B G. Milligan. Punctuated evolution induced by ecological change. *American Naturalist*, pages 522–532, 1986.

[17] A Moraglio, K Krawiec, C G. Johnson Geometric Semantic Genetic Programming. *Parallel Problem Solving from Nature, Springer Berlin Heidelberg*, 2012.

[18] LOVB Oliveira, FEB Otero, GL Pappa, J Albinati Sequential Symbolic Regression with Genetic Programming. *Genetic and Evolutionary Computation, Springer*, 2014.

[19] FEB Otero, CG Johnson. Automated Problem Decomposition for the Boolean Domain with Genetic Programming. *Genetic Programming, Springer Berlin Heidelberg*, pages 169-180, 2013.

[20] A. Piszcz and T. Soule. Genetic programming: Optimal population sizes for varying complexity problems. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 953–954. ACM, 2006.

[21] R. Poli. A simple but theoretically-motivated method to control bloat in genetic programming. In *Genetic Programming*, pages 204–217. Springer, 2003.

[22] Prechelt, Lutz Investigation of the CasCor family of learning algorithms. *Neural Networks 10, Elsevier*, pages 885–896, 1997.

[23] S. Ruberto, L. Vanneschi, M. Castelli, and S. Silva. ESAGP–a semantic GP framework based on alignment in the error space. In *Genetic Programming*, pages 150–161. Springer, 2014.

[24] E J. Vladislavleva, G F. Smits, and D. Den Hertog. Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *Evolutionary Computation, IEEE Transactions on*, 13(2):333–349, 2009.