Predict the Performance of GE with an ACO Based Machine Learning Algorithm

Gopinath Chennupati
BDS Group
CSIS Department
University of Limerick, Ireland
gopinath.chennupati@ul.ie

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

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

ABSTRACT

The quality of the evolved solutions of an evolutionary algorithm (EA) varies across different runs and a significant percentage of runs can produce solutions of undesirable quality. These runs are a waste of computational resources, particularly in difficult problems where practitioners have time bound limitations in repeating runs.

This paper proposes a completely novel approach, that of a *Run Prediction Model* (RPM) in which we identify and terminate evolutionary runs that are likely to produce low-quality solutions. This is justified with an Ant Colony Optimization (ACO) based classifier that learns from the early generations of a run and decides whether to continue or not.

We apply RPM to Grammatical Evolution (GE) applied to four benchmark symbolic regression problems and consider several contemporary machine learning algorithms to train the predictive models and find that ACO produces the best results and acceptable predictive accuracy for this first investigation. The ACO discovered prediction models are in the form of a list of simple rules. We further analyse that list manually to tune them in order to predict poor GE runs.

We then apply the analysed model to GE runs on the regression problems and terminate the runs identified by the model likely to be poor, thus increasing the rate of production of successful runs while reducing the computational effort required. We demonstrate that, although there is a high bootstrapping cost for RPM, further investigation is warranted as the mean success rate and the total execution time enjoys a statistically significant boost on all the four benchmark problems.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search - Heuristic methods

General Terms

Algorithms, Applications

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.

http://dx.doi.org/10.1145/2598394.2609860.

Keywords

Grammatical Evolution; Ant Mining; Machine Learning; Symbolic Regression; Training Set;

1. INTRODUCTION

Experiments in evolutionary computation (EC) follow a common practice of running the algorithm multiple times in order to obtain statistically significant data, particularly when the likelihood of achieving a solution of a certain quality is only estimated through a large number of runs. However, as problems get increasingly computationally expensive, the computational resources are best utilised if the Evolutionary Algorithm (EA) consistently generates solutions of desirable quality and the runs producing poor solutions are minimised. For example [21, 26, 33] are all more concerned with producing a single *useful* result rather making statistical affirmations about their ability to emulate the same result.

This paper details a pilot study into a mechanism to better utilise resources for EC by predicting which runs are most likely to fail and discontinuing them early on. Our technique will work with essentially all other techniques for improving GE (or Genetic Programming (GP)), such as Linear Scaling [11] and No Same Mates (NSM) [9]. We apply an ACO [6] induced prediction model to estimate the end of run performance based on the information gathered in the first few generations of GE. Then, we only allow the run to continue if it is predicted to be *successful* and terminate the remaining runs. This improves the rate of success while reducing the total amount of time significantly.

In this paper we establish the use of $cAnt-Miner_{PB}$ [18], an ACO based classifier that produces a model to predict the performance of GE; the results indicate that the predictive model effectively improves the proportion of successful GE runs on a selection of symbolic regression problems. The training data required for the classifier describes the *changes* in different parameters (such as fitness, genome lengths) at the early stages of an evolutionary cycle of GE, that is, how fast does a particular measure change. The ants then devise a rule based classification model that has been further scrutinized to select the best combination of rules to predict the performance. The predictive model thus produced is then used during the prediction of the evolutionary cycle of a given run. Finally, the runs that are predicted to be poor are immediately halted, freeing more resources for higher quality runs. Our approach in predicting the run time performance is novel: to the best of our knowledge, this is the first attempt to employ an ACO classifier to predict the per-

formance of an EA run along with a detailed understanding of how we use an ACO based classifier for model discovery in the current context.

The rest of the paper is laid out as follows: section 2 introduces GE, some previous approaches to improve the quality of evolutionary search with GE, describes the ACO and, presents the novel approach to predict the GE runs; section 4 details the selection of problems for this study, the experimental setups and the results achieved; section 5 analyses the ACO based predictive models; and finally, section 6 concludes the paper and outlines our future aspirations.

2. BACKGROUND

GE [25] is an evolutionary algorithm that evolves computer programs in an arbitrary language. Unlike standard GP [14], GE endorses genotypes to phenotypes distinction through grammars to generate the computer programs; a genotype is a variable length binary string genome whereas the phenotype is a computer program in a language defined by a CFG of choice specified in Backus Naur Form (BNF).

GE uses a simple *mapping* process to convert the genotypes to phenotypes. In GE, a genotype is a string of 8 bit integers each integer termed a *codon*; each of these codons helps mapping a derivation tree from the selection of production rules in the CFG. To facilitate this, each codon selects a production rule from the given grammar as follows:
Rule=(Codon Integer Value)%(# rules of this non-terminal)
where *Rule* is an index to a production rule that is applicable in the present context of mapping a derivation. Naturally, the codons that appear earlier in the genome shape the derivation tree so that the codons appearing later only select rules as required by the partially generated derivation tree. Thus, there is a left to right dependency in how the codons in a GE genome are interpreted by the GE mapping process; therefore, if we change a codon at the start of the genome, the interpretation of the codons appearing later changes. Thus the effect of the change at the start of the genome *ripples* through the rest of the genome; correspondingly, the one point crossover in GE is called the *ripple crossover* [13]. If the mapping remains incomplete, then a special operator, *wrapping*, can be used to reuse the genotype from the start.

It is possible that the mapping can finish before using the entire genotype. Thus, in GE, the *actual length* of a genotype may be different from the *effective length* which is the number of codons mapped. The actual length of a genome is usually longer than the effective length, on any problem. A complete description of GE can be found in [20].

Note that the nature of GE mapping easily allows us to use both the actual and effective lengths of the genomes as inputs to our predictive model, while in GP, it is not trivial to distinguish between functional and non-functional parts of the genome[1]. This is why we currently use GE, and will later extend the work to GP.

2.1 Run Improvement

The efforts to improve the performance of GP date back to its inception; GE being a form of GP [14], can also benefit from a number of such approaches. Typically, this work tries to improve the success rate of the results, which, in turn, impacts the total execution time.

For example, Keijzer [10] improved GP based symbolic regression results using interval arithmetic and linear scaling. An improved GP search was presented in [9] for the symbolic regression problems with an analysis on crossover and diversity. [15, 23] proposed improvement techniques that work by killing the bloating individuals which occur at the later stages of the evolutionary cycle. Tsakonas et al. [31] applied kill tournaments [28], replacing the worst (fitness) individual with the best, in predicting the classification rules with GP. Work shown in [1, 7] improved the quality of GP on symbolic regression problems.

Although this is just a tiny subset of vast amount of research, in general, they all have the same focus, that is to make the best out of the available genetic material. This paper attempts to ensure that the available genetic material is as good as it can be, so that all these methods should be able to benefit from it.

2.2 Predicting The Quality of Runs

There are many possible reasons for a run to fail to achieve good quality solutions, but many manifest themselves in the form of the premature convergence [14] of the population to a sub-optimal solution leaving little diversity in the population to explore new solutions. In the case of GE, Keijzer et al. [13] showed that the *ripple* crossover (a variant of one-point crossover specific to GE) delays premature convergence. However, GE suffers from the problem of locality [24] that worsens the search results.

Some studies suggested that multiple independent runs could result in useful solutions. For example [27] argued that multiple small runs can reach global solutions with fewer function evaluations than a single run for a genetic algorithm (GA).

The quality can be improved by proposing modifications to the search process as explained in literature or by performing multiple isolated runs [27], but, a different solution is to some how catch the runs that fail to produce high-quality solutions and kill them. We accomplish this goal by predicting the success or otherwise of a GE run using an ACO based machine learning algorithm. As a result, we find that, along with an improvement in the final result of run, our approach reduces the execution time using fitness prediction that leverages the difference between the individuals in the current and past generations.

A key difference with this work is that we are more concerned with producing a small number of high quality runs than producing a large number of statistically useful runs. However, RPM can operate in tandem with virtually any system from the literature.

2.3 Ant Miner Overview

ACO [6] is an optimization meta-heuristic inspired by the foraging and pheromone strategies of real ants. In essence, ants cooperate each other in the form of colonies in order to find an optimal solution to a given problem. For example, if we consider the travelling salesman problem (TSP): ants start from a random city (vertex) and then select the routes (edges) to add new cities that iterative process, in return, produces a solution in the form of pheromones present on the edges. Many researchers have applied ACO to a number of optimization problems and classification is one such domain.

[1]This is not to say that GE cannot produce useless parts of the genome, e.g. (X*0). However, identifying this is beyond the scope of this work.

The first ACO based classification algorithm, *Ant-Miner*, was introduced in [22] that dealt only with the nominal attributes in a given data set. Ant-Miner produces a list of *IF-THEN* classification rules of the form *IF BFC <= 0.0268 THEN no*. Following this, several variations were proposed, some dealt with pheromone update strategies and heuristic information [17], others with discovering fuzzy classification rules [8]. One such extension is $cAnt{-}Miner_{PB}$ [18], showed much better accuracy with a list quality function that can deal with the numerical attributes also.

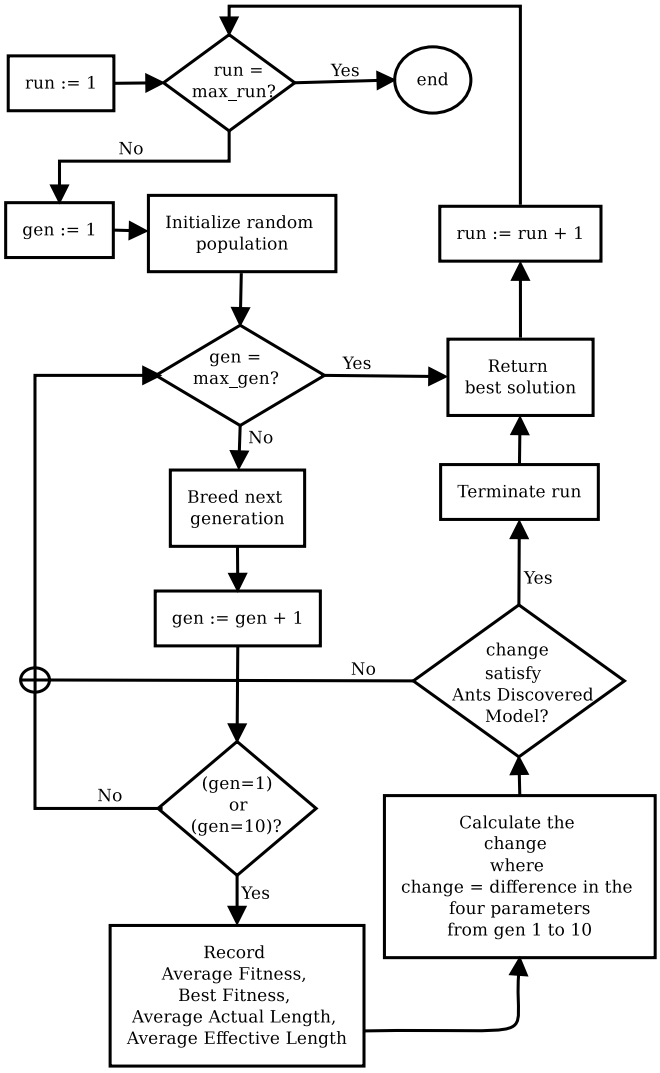

Figure 1: The flowchart for the RPM applied GE that predicts the performance of GE runs.

$cAnt - Miner_{PB}$ algorithm starts with an empty rule list and the training set in each iteration and, repeated until the maximum number of iterations reached or the algorithm converges. The ant then creates a rule from the data points in the training set. There after, the ant prunes the rule and removes all the covered data points from the training set. This process is repeated until the remaining data points in the training set become less than the user specified number of uncovered training points. This way a rule list is prepared in each iteration. The quality of the current list of rules is compared with the previous list of rules, and replaces if the current list is better than the best so far. This way, ants construct a list of best rules. A data point is considered correctly classified if the predicted class value is same as the actual class value. $cAnt - Miner_{PB}$ has produced competitive results when compared against the state-of-the-art machine learning methods. With this motivation for the first time we employ $cAnt - Miner_{PB}$ to improve the quality of GE results as proposed in this paper.

3. THE RUN PREDICTION MODEL

This section describes an overview of the ACO learning process in predicting the GE runs then, the implementation details of the novel approach. The ACO based learning algorithm uses the training data (shown in section 4.1), that discovers a prediction model. The model takes GE evolutionary cycle as an input then, tries to identify the quality of the solution attainable at the end of a GE run. This way, the model allows only the *promising* runs to continue and terminates the remaining runs early.

Figure 1 presents the flowchart of run prediction model (RPM) applied GE. Except for a small change to interrupt the execution of GE, the proposed approach follows the standard GE algorithm. The variable *gen* stands for the current generation number. In summary, the proposed approach starts with an initial random population of individuals. Then, the next generation of individuals are evolved iteratively performing genetic operations, mapping the genotypes to phenotypes, evaluating the fitness of the phenotypes, incrementing the generation count. During this process a new step, the algorithm records the change in the parameters: best fitness, average fitness, average actual length and, average effective length; we note the change in these parameters that records the difference (change) from generation 1 to 10. This change in the respective parameters is used to identify the success or failure of a GE run. An interesting question here is that what if we consider the difference between generation 1 and 5 or generation 1 and 20? The former, in fact achieves a better optimization in the execution time while the calculated change resulted in an approximately zero difference in most of the runs, that in turn resulted in omitting this choice. With the latter possibility, we end up with less optimization in total execution time. Considering these factors into count, we leaned towards experimenting with the difference between generations 1 and 10. If the change in any one or all of the recorded parameters satisfy the conditions in the ACO classifier discovered model then, that run is terminated recording the best so far fitness that helps for the comparison. Otherwise, the algorithm continues to run reporting the best of run result. This process is repeated until the maximum number of runs have reached.

4. EXPERIMENTS

This section describes the regression problems and the generated training data for the respective problem, the experimental approach and the results.

4.1 Problems and Training data

We followed the experimental procedure explained in [3] in preparing the data sets for the four symbolic regression problems.

Table 1: The problem set considered. The "yes" column counts the number of runs producing results above the threshold; the "no" column counts the runs that remained below the threshold.

#	Problem:Source	no	yes	Success threshold (Best Fitness <)
f_1	$(1+x)^3$:[5]	590	410	0.7
f_2	$x^4 - x^3 - y^2 - y$:[30]	606	394	0.65
f_3	$x^3 - y^3 - y - x$:[30]	848	152	0.7
f_4	x^y:[29]	518	482	1

Table 1 describes the experimental problems and the data sets used in this paper. The first column in Table 1 gives an index to the problem; henceforth, we use this index to refer to the corresponding problem. The second column gives the target function in each problem; we also cite the source of the problem from the GP literature.

For each problem, a training set has been prepared to train the ACO algorithm. Each data set contains 1000 data points that are classified with two (*yes, no*) labels. The number of *yes*, *no* class labels are presented in third and fourth columns for the respective problem. The classification of a data point depends upon the user defined success threshold, Best Fitness (BF). For problem f_1, if $BF > 0.7$ then those data points belong to *yes* class, *no* otherwise. Since, the regression problems are hard to solve and the solution evolved by an EA varies largely from a problem to problem, we consider a different threshold value as a success measure on each problem. All the attributes in the resulting data sets are continuous with no duplicates.

Table 2: A brief description of the data sets used in the experiments. BFC (Best Fitness Change), AFC (Average Fitness Change), AALC (Average Actual Length Change), AELC (Average Effective Length Change), are the attributes of the data set with the data points that represent the change from generation 1 to 10.

BFC	AFC	AALC	AELC	Class
0.0220119	0.0481755	2.005	-18.995	no
0.0530694	0.0641799	6.105	-12.565	yes
⋮	⋮	⋮	⋮	⋮
0.0868888	0.0622562	10.945	-17.825	no

Table 2 presents an example description of f_1 training set. The data set contains four attributes: BFC, AFC, AALC, AELC that represent the change in the respective parameters between generation 1 and generation 10. The two fitness related attributes (BFC and AFC) are intuitive; however, these are not the only possible attributes, and later work will examine more. The data sets were prepared recording the change in the respective attributes and classifying them as per the threshold values defined in Table 1 for the respective problem. Although the computational effort in preparing the training sets is rather high for this preliminary investigation, it has resulted highly in improving the success rate and optimizing the execution time with the help of the proposed run prediction model.

4.2 Experimental Approach and Results

We have conducted two sets of experiments. In the first set, we compare the predictive accuracy of the ACO based classifier with those produced by a number of contemporary machine learning algorithms. We find that the ACO based classifier outperforms the benchmark machine learning methods. Next, we apply the ACO devised models to actual GE runs after a manual inspection in order to detect and terminate poor runs and present the results showing the efficacy of the proposed approach.

Listing 1: Ant discovered model for problem f_1

```
IF AALC <= -12.8875 THEN no;
IF BFC <= 0.0268 THEN no;
IF AELC <= -23.31 THEN yes;
IF AFC > 0.0731 THEN yes;
IF 0.0614 < AFC <= 0.0633 THEN no;
IF AELC <= -21.725 THEN yes;
IF AELC <= -20.99 THEN no;
IF AALC <= -6.035 THEN yes;
IF 7.2475 < AALC <= 8.3425 THEN no;
IF BFC <= 0.0268 THEN no;
IF AFC <= 0.0534 THEN no;
IF BFC > 0.0656 THEN yes;
IF AELC > -17.7924 AND
   AALC > -1.5025 THEN yes;
IF BFC > 0.0558 THEN no;
IF BFC > 0.0496 AND
   AFC <= 0.0613 THEN yes;
IF AFC > 0.0558 AND
   AELC > -22.855 THEN no;
IF BFC > 0.0292 THEN yes;
IF <empty> THEN yes;
```

4.2.1 Discovered Models

We ran $cAnt - Miner_{PB}$ mining algorithm on all the four data sets to generate four different predictive models, one for each problem. The set parameters for the algorithm were: *ant_colony_size* = 5, *iterations* = 500, *minimum_cases_per_interval* = 10, and, *evaporation_probability* = 0.9. A complete description of these parameters can be found in [22].

The ant learning algorithm discovered four prediction models, one for each problem. For example, Listing 1 presents the ant discovered model for problem f_1, the remaining models are concealed owing to the space restrictions. In the model, each line that starts with an *IF* keyword and ends with *;* (semi-colon) represents a rule. All the models consist of the best list of rules derived by ants. Note that the rules in the models are simple enough for a manual inspection and analysis. As the focus of the paper is to predict the *poor* runs that fail to produce high-quality solution, we consider the rules that predict the class value *no* from the discovered rule list.

Examining the classification model shown in Listing 1, ants devised a manageable number of rules with a single attribute alone or in combination of more than one attribute and a class. While it is difficult to identify clearly which rule of the model suits best in run prediction, we manually explored different possibilities of these rules on the training data set of the respective problem in order to identify the best combination of rules.

Table 3: Comparison of accuracy measure (*mean* [*standard deviation*]) in %, among J48, ZeroR, $cAnt{-}Miner_{PB}$, PART and JRip measured by applying 10-fold cross-validation. The value of the most accurate algorithm is shown in bold for a given data set with the Wilcoxon tests at $\alpha = 5\%$.

Problem	J48	ZeroR	$cAnt - Miner_{PB}$	PART	JRip
f_1	58.4 [0.49]	54.12 [0.95]	**78.72 [0.84]**	57.85 [0.63]	59.18 [0.81]
f_2	60.61 [0.89]	60.25 [1.32]	**69.32 [0.61]**	61.19 [1.58]	62.60 [1.02]
f_3	79.44 [0.35]	78.43 [0.26]	**86.96 [0.45]**	80.88 [1.01]	80.24 [0.56]
f_4	54.21 [0.54]	51.82 [0.64]	**68.12 [0.99]**	56.10 [0.38]	58.50 [0.47]

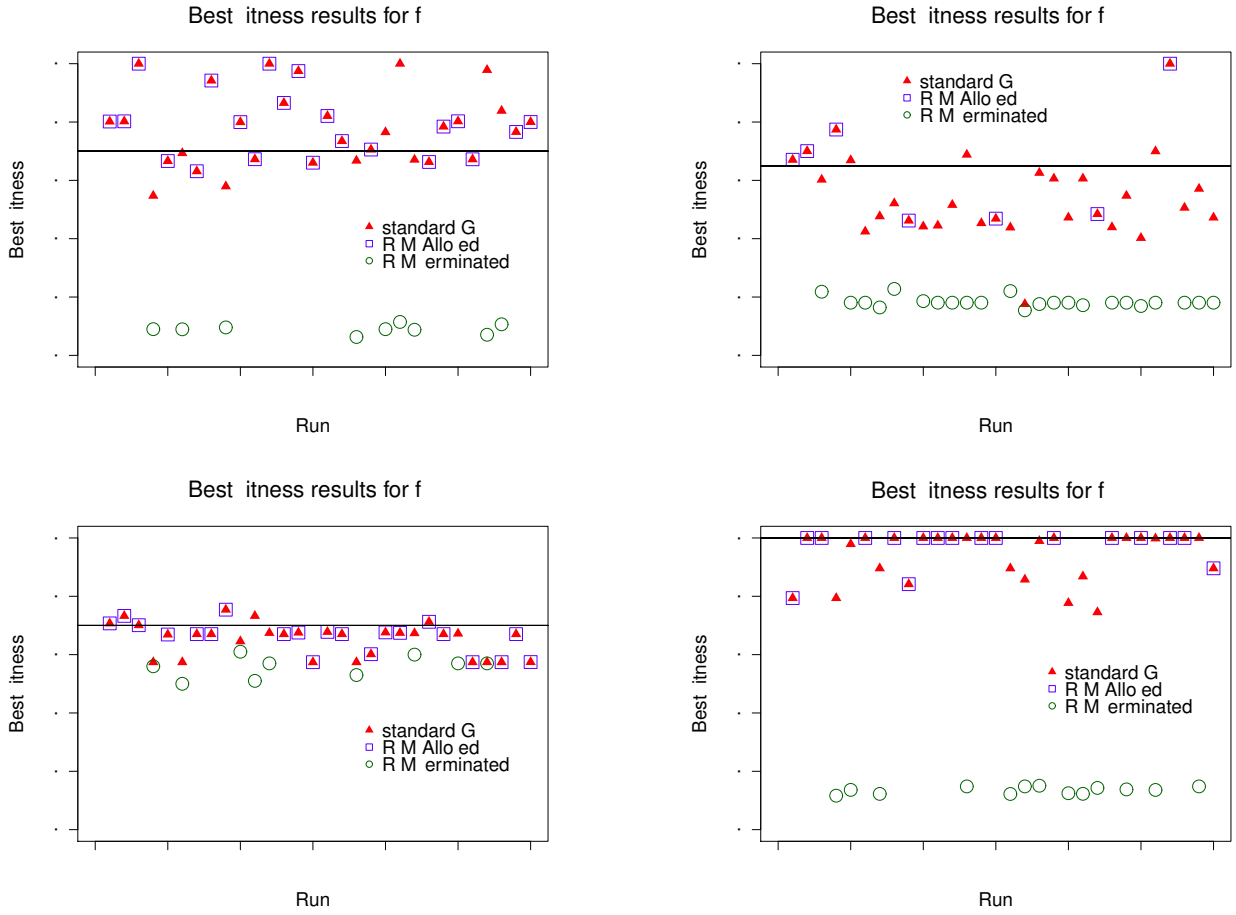

Figure 2: The best fitness measure of standard GE (shown in triangles) vs. RPM applied GE (shown in squares and circles) on the four symbolic regression problems. The end of run results of all four problems illustrate an improvement in the quality of runs after applying the prediction model.

Table 4: A comparison in the success rate of the standard GE vs. Run Prediction Model (RPM) applied GE out of 30 evolutionary runs on each approach.

Problem	GE		GE+RPM		
	# of successful runs	% of success	# of successful runs	% of success	terminated runs
f_1	18	56.66	15	71.43	9
f_2	7	23.33	3	65.15	23
f_3	6	20.00	5	23.81	9
f_4	17	56.66	12	70.59	13

Table 5: Selected rules for run prediction.

Problem	Selected Rule(s)
f_1	if(bfc <= 0.027)
f_2	if((bfc > 0.022) && (afc > 0.087))
f_3	if(aalc > 5.97)
	if((bfc <= 0.19) && (aelc > 3.84))
f_4	if(bfc > 0.035) and
	if((aalc < -17.79) && (aelc > -22.85))

Table 5 presents the selected rules from the manual analysis on all the problems and shows the simplicity of the selected rules. We focused on the rules that can identify the low-quality run only. In fact the selected list of rules may sometimes fail to exactly find the poor solution producing runs as is the case that it might try to terminate a few useful runs also. We treat this as the error exerted by the prediction that we explain it in detail later in section 5.

Table 3 compares the predictive accuracy achieved by $cAnt - Miner_{PB}$ with some state of the art classification algorithms on the four data sets considered in this study. Each entry in Table 3 shows the mean accuracy measure (in %) followed by the standard deviation, calculated as a result of 10-fold cross-validation. $cAnt - Miner_{PB}$ exhibited significantly better predictive accuracy than its counter parts when performed the Wilcoxon two-tailed significance tests at a significance level of $\alpha = 5\%$. Overall, $cAnt - Miner_{PB}$ models exhibit significantly better predictive accuracy outperforming the remaining classifiers on all the four data sets.

4.2.2 Impact of Predictors

With the predictors generated, the next step was to establish how effective they were. We assess the effectiveness of the predictors by conducting the next set of experiments in two steps. We used libGE [19], an open source framework for GE in C++. In the first step, all four problems were run with standard GE system recording the best fitness at the end of run. Similarly, in the second step, the same problems were run with RPM applied GE system recording the best fitness and, terminating the runs that satisfy the respective model of the corresponding problem. Finally, a comparative analysis has been presented with the results obtained in both the steps.

The following parameters were kept consistent in both the experimental setups. The parameters were: *number of runs* = 30, *population size* = 200, *number of generations* = 120, *crossover probability* = 0.9, *mutation probability* = 0.01, randomly initialized the initial population with a *minimum genotype length* of 15, *maximum length* of 25, *seed* value was incremented for each run, and a steady state replacement strategy. We used an *S-Lang* evaluator for the fitness evaluation. A run in this experimental settings is considered as long (with 120 generations) when compared to the standard GE settings (50 generations). The main reason for this setting is that [2] showed that a single large run reaches the global solution faster than multiple small runs saving the computational resources.

Figure 2 shows the best fitness measure for 30 runs that summarizes the results comparing the standard GE system with that of the run prediction model applied GE. Standard GE results are shown in triangles, whereas the model applied GE results are represented with squares and circles. In that, the circles represent the terminated runs, whereas the squares represent the model allowed runs that completed their execution. At $BestFitness = 0.70$ there exists a black straight line parallel to x-axis. This represents the success measure (defined in section 4.1) on the respective problems, if the end of run result is greater than that value then it is a high-quality run, or else a low-quality run. Some of the squares and the triangles in Figure 2 overlap each other. This is due to the fact that the proposed approach tries to predict the success or failure of a run without making any modifications to the GE search process. This results in an improvement in the success rate of the GE results.

The results shown in Table 4 summarizes the number of successful (high-quality) runs and, the rate of success for 30 runs with standard GE and RPM applied GE approaches. In addition to this the count of terminated runs resulted from the run prediction model are reported. The RPM+GE exhibits significant improvement in the % of successful results over standard GE on all the four problems. While f_3 enjoys only a small (but still statistically significant) improvement of 3.81%, the remaining three problems f_1, f_2 and f_4 exhibit 14.77%, 41.82%, and 13.93% respectively. Notice that f_2 exhibits huge jump in the success rate; it is because, the model has identified 23 runs as the *poor* runs and terminated. This behaviour of the model requires us to consider how many *good* runs are killed by the model? We explain the answer later in section 5 with an analysis on the inherent error of the new prediction approach.

Table 6: Statistical test results for the total execution time of standard GE (T_{GE}) vs. that of RPM applied GE (T_{GE+RPM}) over 30 runs. Note that the "Yes" represents that the model applied GE performed significantly better than standard GE with Wilcoxon Signed rank test at $\alpha = 5\%$. When there is significant difference, *A* measure value is shown, otherwise (-).

Problem	Significant	A measure	T_{GE} (in sec)	T_{GE+RPM} (in sec)
f_1	No	-	91.83	66.78
f_2	Yes	0.831	88.24	25.43
f_3	Yes	0.661	87.65	61.93
f_4	Yes	0.692	87.73	52.31

Since the amount of processing time required to evolve the solution is one of the major concerns in EA, we illustrate the total execution time results as shown in Table 6, whereas T_{GE} stands for total execution time of standard GE and T_{GE+RPM} stands for the total execution time of the RPM applied GE. We observe that the proposed approach reduces the time on all the problems. Execution time reduction in f_1 is insignificant as a result of the Wilcoxon test, whereas it is significant for the remaining three problems. The non-parametric Wilcoxon Signed Rank test states whether the model applied GE is significantly better or not, but it fails to state how much better it is. We use *Vargha-Delaney A* [32] measure that tells us the probability that the model applied GE achieve better performance over the standard GE. When

the A measure is above 0.5, standard GE outperforms the model applied GE. When it is 0.5, both are equal, otherwise, the model applied GE outperforms the standard GE. The A measure tells us that run prediction model applied GE outperforms standard GE, 83.16%, 66.11% and, 69.17% of times on f_2, f_3 and f_4 respectively.

From the results shown in Table 4 and 6, we conclude that the rate of success of GE has increased while reducing the total execution time on the respective regression problems with the introduction of the new run prediction approach.

5. DISCUSSION

Recall that the proposed RPM+GE inherently exerts error so that it misclassified a few GE runs as either low-quality or high-quality runs and terminates or allows to continue the execution. We describe this issue of the model incorrect predictions in this section.

Table 7: RPM applied GE predictions on the four benchmark problems.

Problem	TP	FP	FN	TN
f_1	15	6	3	6
f_2	3	4	4	19
f_3	5	16	1	8
f_4	12	5	5	8

Table 7 shows the prediction results of the four RPM applied GE problems. The terms are defined as follows: *TP* is the actual number of correctly identified high-quality runs, *FP* is the number of runs that are actually low-quality but identified as high-quality, *FN* is the number of runs that are actually high-quality but identified as low-quality, *TN* is the number of runs that are actually low-quality and identified as low-quality. The results shown in Table 7 describe that there is a possibility that the model terminates a few number of successful runs (FN), even though, the results in Table 4 confirm the improvement in the success rate. Considering FP rate, problem f_3 has high FP rate. If we refer back to Figure 2, we note that most of these runs fall just short of crossing the threshold. Therefore, we need a too sensitive predictor to classify such points as negative examples. As it is, this performance is not a disaster if we allow for a small margin of error around the threshold.

In summary, the analysis clearly suggests that the ACO discovered models are simple enough to analyse and easy to use in an EA run prediction. Combined use of ACO with EA produced a reasonably good prediction system. As a result, a significant improvement has been observed in the number of successful runs while significantly reducing the total execution time.

6. CONCLUSIONS

We have successfully accomplished the set goals of improving the quality of the solution as well as optimizing the speed of execution by employing an ACO based classification with the introduction of the new run prediction approach. This approach was more akin to reduce the total execution time while improving the rate of success. This work can have huge implications for scaling EC algorithms which, due to their stochastic nature, often confront prediction when it comes to the length of the runs. One assumption that we have made is the existence of good predictive data.

The next step will be concerned with bootstrapping the system, and we are currently examining two approaches. First, the use of data produced on smaller but related versions of the problem, similar to the technique used in [12] will be examined. Although not perfect, this could get over the initial bootstrapping phase, particularly if the problem being tackled is a scaled up version of the one the data came from.

Secondly, and probably in tandem with the first method, we will investigate rapidly retraining the predictors during the runs as increasingly more data becomes available, similar to work by [4, 16, 26]. This would facilitate increasingly accurate predictors as more data becomes available.

7. REFERENCES

[1] R. M. A. Azad and C. Ryan. Abstract functions and lifetime learning in genetic programming for symbolic regression. In *Proceedings of the Twelth International Conference on Genetic and Evolutionary Computation Conference, GECCO'10*, pages 893–900, 2010.

[2] E. Cantú-Paz and D. Goldberg. Are multiple runs of genetic algorithms better than one? volume 2723 of *LNCS*, pages 801–812. Springer, Berlin, Heidelberg, 2003.

[3] G. Chennupati, C. Ryan, and R. M. A. Azad. An empirical analysis through the time complexity of GE problems. In R. Matousek, editor, *19th International Conference on Soft Computing, MENDEL'13*, pages 37–44, Brno, Czech Republic, jun, 26-28 2013.

[4] D. Costelloe and C. Ryan. On improving generalisation in genetic programming. In L. Vanneschi, S. Gustafson, A. Moraglio, I. Falco, and M. Ebner, editors, *Genetic Programming*, volume 5481 of *LNCS*, pages 61–72. Springer, Berlin, Heidelberg, 2009.

[5] J. M. Daida, R. R. Bertram, S. A. Stanhope, J. C. Khoo, S. A. Chaudhary, O. A. Chaudhri, and J. A. I. Polito. What makes a problem gp-hard? analysis of a tunably difficult problem in genetic programming, 2001.

[6] M. Dorigo and T. Stützle. *Ant colony optimization*. MIT Press, 2004.

[7] J. Fitzgerald, R. M. A. Azad, and C. Ryan. A bootstrapping approach to reduce over-fitting in genetic programming. In *Proceeding of the Fifteenth Annual Conference Companion on Genetic and Evolutionary Computation Conference Companion*, pages 1113–1120, 2013.

[8] M. Galea and Q. Shen. Simultaneous ant colony optimization algorithms for learning linguistic fuzzy rules. In A. Abraham, C. Grosan, and V. Ramos, editors, *Swarm Intelligence in Data Mining*, volume 34 of *Studies in Computational Intelligence*, pages 75–99. Springer, Berlin, Heidelberg, 2006.

[9] S. Gustafson, E. Burke, and N. Krasnogor. On improving genetic programming for symbolic regression. In *IEEE Congress on Evolutionary Computation*, volume 1, pages 912–919, 2005.

[10] M. Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, editors, *Genetic Programming*, volume 2610 of *LNCS*, pages 70–82. Springer, Berlin, Heidelberg, 2003.

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

[12] M. Keijzer, C. Ryan, and M. Cattolico. Run transferable libraries - learning functional bias in problem domains. In K. Deb, editor, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 3103 of *LNCS*, pages 531–542. Springer, Berlin, Heidelberg, 2004.

[13] M. Keijzer, C. Ryan, M. O'Neill, M. Cattolico, and V. Babovic. Ripple crossover in genetic programming. In J. Miller, M. Tomassini, P. Lanzi, C. Ryan, A. Tettamanzi, and W. Langdon, editors, *Genetic Programming*, volume 2038 of *LNCS*, pages 74–86. Springer, Berlin, Heidelberg, 2001.

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

[15] J. R. Koza, S. Al-Sakran, and L. Jones. Cross-domain features of runs of genetic programming used to evolve designs for analog circuits, optical lens systems, controllers, antennas, mechanical systems, and quantum computing circuits. In *Proceedings of NASA/DoD Conference on Evolvable Hardware*, pages 205–212, 2005.

[16] K. Krawiec and J. Swan. Pattern-guided genetic programming. In *Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference*, pages 949–956, 2013.

[17] D. Martens, M. De Backer, R. Haesen, J. Vanthienen, M. Snoeck, and B. Baesens. Classification with ant colony optimization. *IEEE Transactions on Evolutionary Computation*, 11(5):651–665, 2007.

[18] M. Medland and F. E. B. Otero. A study of different quality evaluation functions in the cant-miner(pb) classification algorithm. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference, GECCO '12*, pages 49–56, 2012.

[19] M. Nicolau and D. Slattery. libge - grammatical evolution library. http://bds.ul.ie/libGE/index.html, 2006.

[20] M. O'Neill and C. Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[21] U.-M. O'Reilly, M. Wagy, and B. Hodjat. Ec-star: A massive-scale, hub and spoke, distributed genetic programming system. In R. Riolo, E. Vladislavleva, M. D. Ritchie, and J. H. Moore, editors, *Genetic Programming Theory and Practice X*, pages 73–85. Springer New York, 2013.

[22] R. Parpinelli, H. Lopes, and A. Freitas. Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 6(4):321–332, 2002.

[23] R. Poli. A simple but theoretically-motivated method to control bloat in genetic programming. In *Proceedings of the 6th European Conference on Genetic Programming*, EuroGP'03, pages 204–217. Springer, Berlin, Heidelberg, 2003.

[24] F. Rothlauf and M. Oetzel. On the locality of grammatical evolution. In *Proceedings of the 9th European Conference on Genetic Programming*, EuroGP'06, pages 320–330. Springer, Berlin, Heidelberg, 2006.

[25] C. Ryan, J. J. Collins, and M. O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty, editors, *Proceedings of the First European Workshop on Genetic Programming*, volume 1391 of *LNCS*, pages 83–95. Springer, 1998.

[26] C. Ryan, M. Keijzer, and M. Cattolico. Favourable biasing of function sets using run transferable libraries. In U.-M. O'Reilly, T. Yu, R. Riolo, and B. Worzel, editors, *Genetic Programming Theory and Practice II*, volume 8, pages 103–120. Springer US, 2005.

[27] R. Shonkwiler. Parallel genetic algorithms. In *ICGA*, pages 199–205. Morgan Kaufmann, 1993.

[28] J. Smith and F. Vavak. Replacement strategies in steady state genetic algorithms: static environments. In W. Banzhaf and C. Reeves, editors, *Foundations of Genetic Algorithms 5*, pages 219–234, 1999.

[29] M. Streeter and L. Becker. Automated discovery of numerical approximation formulae via genetic programming. *Genetic Programming and Evolvable Machines*, 4(3):255–286, 2003.

[30] A. Topchy and W. F. Punch. Faster genetic programming based on local gradient search of numeric leaf values. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the third international conference on Genetic and evolutionary computation conference, GECCO '01*, pages 155–162, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[31] A. Tsakonas, G. Dounias, J. Jantzen, H. Axer, B. Bjerregaard, and D. G. von Keyserlingk. Evolving rule-based systems in two medical domains using genetic programming. *Artificial Intelligence in Medicine*, 32(3):195–216, 2004.

[32] A. Vargha and H. D. Delaney. A critique and improvement of the "cl" common language effect size statistics of mcgraw and wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[33] D. Wilson, K. Veeramachaneni, and U.-M. O'Reilly. Cloud scale distributed evolutionary strategies for high dimensional problems. In *Proceedings of the 16th European Conference on Applications of Evolutionary Computation*, EvoApplications'13, pages 519–528, Berlin, Heidelberg, 2013. Springer-Verlag.