Multiple Regression Genetic Programming

Ignacio Arnaldo
Massachusetts Institute of Technology, CSAIL
Cambridge, MA
iarnaldo@mit.edu

Krzysztof Krawiec*
Institute of Computing Science
Poznan University of Technology
60965 Poznan, Poland
krawiec@cs.put.poznan.pl

Una-May O'Reilly
Massachusetts Institute of Technology, CSAIL
Cambridge, MA
unamay@csail.mit.edu

ABSTRACT

We propose a new means of executing a genetic program which improves its output quality. Our approach, called Multiple Regression Genetic Programming (MRGP) decouples and linearly combines a program's subexpressions via multiple regression on the target variable. The regression yields an alternate output: the prediction of the resulting multiple regression model. It is this output, over many fitness cases, that we assess for fitness, rather than the program's execution output. MRGP can be used to improve the fitness of a final evolved solution. On our experimental suite, MRGP consistently generated solutions fitter than the result of competent GP or multiple regression. When integrated into GP, *inline* MRGP, on the basis of equivalent computational budget, outperforms competent GP while also besting *post-run* MRGP. Thus MRGP's output method is shown to be superior to the output of program execution and it represents a practical, cost neutral, improvement to GP.

Categories and Subject Descriptors

I.2.2 [**Artificial intelligence**]: Automatic Programming

Keywords

Genetic Programming, Multiple Regression

1. INTRODUCTION

In principle Genetic Programming (GP), given enough time, should be capable of simultaneously identifying and combining useful program subexpressions to yield an overall program that maximizes fitness. In reality, our existing GP algorithm designs often fall short of this capability. While there has been prior work on building block identification, promotion and aggregation, in this contribution we take a new tact toward fitness maximization. We decouple program subexpressions and combine them linearly via regression on the target variable. Our regression yields an alternate output for a GP program: the prediction of the resulting multiple regression model. It is this output, over many fitness cases, that we assess for fitness, rather than the program's execution output. We call our approach Multiple Regression Genetic Programming (MRGP) because it hybridizes GP with multiple regression.

In GP's current paradigm, selection pressure is not explicitly on a program's subexpressions but is instead on the entire program. Because a program's final output determines its fitness, not the intermediate outputs of its subexpressions, expressions are only optimized indirectly. This may lead to suboptimal fitness maximization because an explicit focus on the evolution of good building blocks is missing. We posit that it is possible to improve subexpression contribution to fitness without resorting to selection on the unit of a subexpression. We propose to optimize a program prior to selection by changing the way its output is calculated, i.e. by optimizing the fitness contributions of its subexpressions. We decouple the subexpressions from their nesting in the program after we have saved their outputs during execution. We then linearly combine them by means of a multiple linear regression which regresses the target variable on the saved outputs from every fitness case. This method basically places optimal coefficients in front of each subexpression within a linear model in an attempt to improve the overall combination of subexpressions. It thus acts as a surrogate for direct selection on them.

Our contribution is primarily to change how a program's output is derived so that this new output provides a more reliable fitness signal (one amenable to evolutionary selection and variation) and, overall, so that the ability of GP to evolve highly fit programs improves. Specifically we propose two variants of MRGP and compare them, based upon testing performance, to multiple regression (MR) directly on the input variables and to a competent GP that uses multi-objective optimization based upon accuracy and complexity. In the *post-run* approach we simply use MRGP on the best solution of a run. We experiment with regressing 5 alternate sets of subexpressions. We find that, for a suite of 6 symbolic regression problems, at least one of these optimizations is always superior to our competent GP and MR. Given the low cost of trying all variants and the option to also apply them to other programs at the end of a run, MRGP appears to offer a viable post-run optimization. Its Success confirms the basic proposal that a program can be improved

*research conducted during K. Krawiec's stay at the ALFA Group, Computer Science and Artificial Intelligence Laboratory (CSAIL), MIT

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

ACM 978-1-4503-2662-9/14/07 ...$15.00.
http://dx.doi.org/10.1145/2576768.2598291.

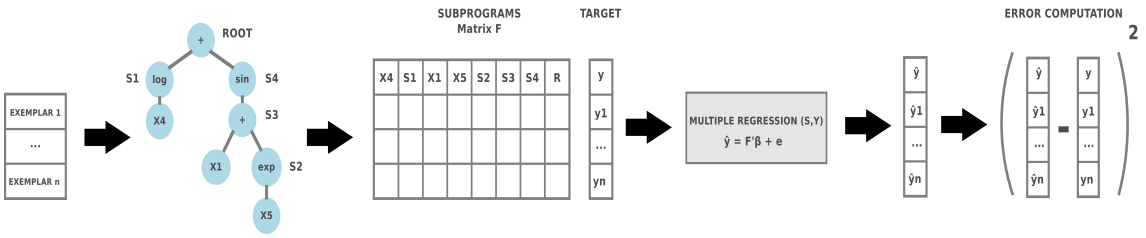


Figure 1: The outline of regressing program subexpressions.

by targeting its subexpressions explicitly for tuning within a linear framework. It also sets up the question of whether inlining multiple regression of subexpressions, within GP's evolutionary cycle is helpful. Our experiments provide a positive response: *inline* MRGP is superior even to *post-run* MRGP and is cost-neutral vs competent GP. By cost-neutral, we mean that, if we fix the optimization time, *inline* MRGP is superior despite using fewer fitness evaluations.

We proceed as follows: We describe the general approach in Section 2. In Sections 3 and 4 we respectively describe the *post-run* variant and empirically examine its effectiveness. In Sections 5 and 6 we respectively describe the *inline* variant and empirically examine its effectiveness. We describe related work in Section 7 and then conclude and mention future work in Section 8.

2. THE MRGP APPROACH

2.1 Terminology

The objective in a regression task is to find a *model* that maps one or more *input variables* onto a single *target variable* (desired output). When solving such a task using GP-based symbolic regression, models are instantiated by *programs* (expression trees in case of tree-based GP), where *program inputs* (*terminals*, *leaves*) map to the input variables' values, and *program output* (the expression's value when the root node is executed) maps to the target variable's value.

2.2 Subexpression Multiple Linear Regression

MRGP differs from conventional GP primarily in eliminating direct comparison of the final program output against the target variable, y. Instead, we substitute a new calculation of a program's output which is based on the linear combination of subexpression behavior as regressed against the target variable. We assume that all subexpressions of a program can be tuned in linear combination with respect to the target output. We also expect the resulting regression model to be superior over program output because it disregards the nested order in which subexpressions are considered in program execution to arrive at the final program output. We compare the target variable y to the output of the regression model.

Given a dataset D (also known as a set of fitness cases) composed of m columns of input variables and n rows of example values and a target vector y with target values for each example, we proceed as follows:

1. We step by step execute the program (with the conventional inorder tree parse) and store the output of each subexpression after it is executed. For tree-based GP, this means pausing the program execution process at each tree node (including leaves and the root node) and storing the value calculated at that node. By doing this for each training example, we obtain an $n \times k$ matrix of subexpressions F, where k is the size of the GP tree and n is the number of exemplars of D.

2. We map the values of F onto the desired output y using multiple linear regression (MR), which produces an optimal linear combination that minimizes the prediction error of $\hat{y}$. Multiple regression determines the vector of coefficients β that minimizes the sum of squares of residuals e of mapping the k subexpressions (predictors) onto the desired output y:

$$y = F'\beta + e$$

where F' is a $n \times (k+1)$ matrix obtained from F by prepending it with an additional column of ones, so that the corresponding coefficient β_1 implements the intercept of the linear model.

3. To assess the quality of the regressed model, we compute its output as $\hat{y} = F\beta$ and compare it to the original targets of the dataset in a conventional way, i.e., as $(y - \hat{y})^T(y - \hat{y}) = e^T e$.

Fig. 1 outlines this process for tree-based GP.

2.3 Enlisting Least Angle Regression

Major challenges with multiple regression can arise because the least squares approach fails if the matrix F is not of full rank. This may happen when the number of features k is large with respect to the number of exemplars n or if some of the columns of F are not linearly independent. The former circumstance will occur whenever a GP tree size exceeds the number of examples. Given GP tree size parameters and bloat, it should be anticipated. The former circumstance occurs when the outputs of different subexpressions are correlated. This correlation arises from one of the following scenarios:

- a subexpression appears multiple times in the GP tree.
- two or more subexpressions are syntactically different but semantically equivalent, i.e., present the exact same output. For instance, the subprograms X_2 and $exp\ lnX_2$ produce identical outputs,
- two subexpressions are linearly dependent, e.g., $(x_1 + x_2)$ and $(x_1 + x_2) * (x_1 + x_1)/x_1$.

To deal with these challenges, we employ the Least Angle Regression (LARS) algorithm presented by Efron et al. [4]. LARS is a stepwise process in which the variables are added one by one to the *active set* while the model's coefficients are continuously moved toward its least-square value until

all the variables are included in the model. At the end of the process, the obtained model is the full least-squares fit. The steps involved in the LARS algorithm are listed below as presented in [6]:

1. Standardize the predictors to have zero mean and unit norm. Set the initial coefficients to 0, $\hat{\beta}_1, \hat{\beta}_2, ..., \hat{\beta}_p = 0$. The error or residual is then $r = y - \hat{y} = y$.
2. Initialize the active set with the variable x_i that is most correlated with r (equivalent in this first iteration to the target vector y).
3. Move $\hat{\beta}_i$ from 0 towards its least-squares coefficient $\langle x_i, r \rangle$, until another variable x_j is as correlated with the current residual as x_i. Add x_j to the active set.
4. Move $\hat{\beta}_i$ and $\hat{\beta}_j$ in the direction defined by their joint least squares coefficient of the current residual on (x_i, x_j), until some other competitor x_l has as much correlation with the current residual. Add x_l to the active set.
5. Repeat step 4 until all k predictors have been added to the active set.

The LARS algorithm is efficient since it takes k steps to get to the full least squares estimates and can be adapted [2] to solve the family of regression problems written as follows:

$$\min_{\beta} \frac{1}{2}||X\beta - y||_2^2 + \lambda_1||\beta||_1 + \frac{1}{2}\lambda_2||\beta||_2^2$$

where β is the vector of regression coefficients. We set $\lambda_1 = 0$ and $\lambda_2 = 0$ so the solution is unregularized and we ensure the active set includes all input variables. Without loss of generality we refer to this as multiple regression (MR).

3. POST-RUN REGRESSION OF PROGRAMS EVOLVED BY COMPETENT GP

The *post-run* multiple regression strategy consists of applying MR to programs evolved by a competent implementation of GP. Our goal is to determine whether GP programs, when 'behaviorally re-interpreted' in various ways by MR, have better fitness than when interpreted in the conventional way. Note that, in this case, GP does not receive any feedback from the outcomes of MR, i.e., the evolutionary process is not *driven* by MRGP (contrary to the approach studied in Section 5). We propose 5 strategies of the *post-run* variant (see Figure 2):

(a) Root: root node only (in which case the MR problem collapses to the simple univariate regression problem).

(b) Root and leaves: the leaves of the GP tree and the outputs of the root node are included in F.

(c) Subexpressions: the outputs of all the nodes of the tree (including the root node and leaves) are in F.

(d) Root and variables: matrix F contains outputs of the root and all the explanatory variables of the problem.

(e) Subexpressions and variables: F contains the outputs of all subexpressions and all the explanatory variables.

Note that in strategies (d-e) the explanatory variables are added to the matrix F even if they do not appear in the program tree. This is justified for high-dimensional problems, where GP trees would need to be very large to reference all the variables of a problem.

4. POST-RUN MRGP EVALUATION

We now compare the performance of the *post-run* multiple regression to a competent GP and multiple regression solely on the problem's explanatory variables.

4.1 Evaluation Problems

Table 1 summarizes the five symbolic regression problems used in this study. We chose problems of various sizes (number of explanatory variables and training set size) and characteristics. Energy problems (datasets ENH and ENC) come from simulation experiments while the NOX emissions dataset is a collection of real power plant data. For the Wine Quality Datasets, both red and white, the input variables are physical measurements, but the dependent variable is the median score of human 'sensory assessors', expressed on a scale from 0 (very bad) to 10 (excellent). In the case of the NOX dataset, the train/test split was fixed beforehand. To obtain the training and test sets of the remaining datasets, we performed random 0.66/0.33 splits.

To provide comparability of results across the benchmarks, their target variables are normalized to the interval $[0, 1]$.

4.2 Competent GP

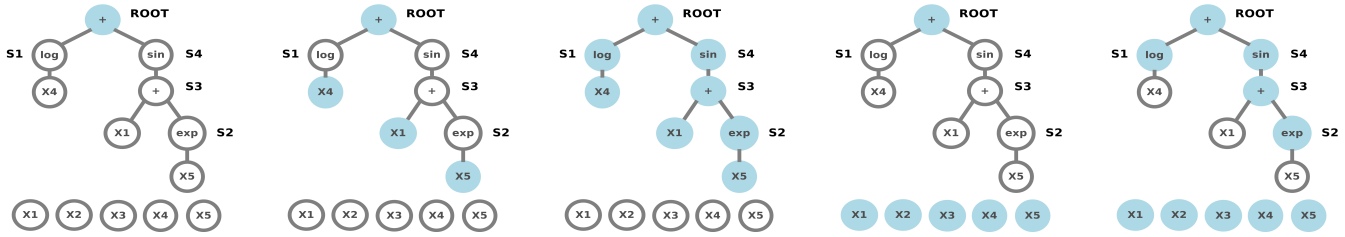
We compare MRGP and MR to our "competent" GP system. It performs multi-objective optimization based on Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [3] working with two minimized objectives: model error and model complexity. In the experiment reported in this section, the former is simply the error of the (normalized) program output with respect to target (calculated using L2 metric), while the latter is the Subtree Complexity measure introduced in [18].

We run 10 replicas of each experiment. Therefore, a total of 50 GP runs (5 datasets $\times$ 10 replicas) are performed to analyze the suitability of the different *post-run* multiple regression strategies. In each run, we select the solution presenting the lowest error and apply the five different post-run linear regression strategies introduced in Section 3 and depicted in Fig. 2, namely Root, Root and Leaves, Subexpressions, Root and variables, Subexpressions and variables. Table 2 presents the settings of the essential evolutionary parameters, most of which remain constant throughout the rest of this paper (unless otherwise stated). We execute each GP replica for 15 minutes, computing however many generations fit into this deadline.

4.3 Regression process

The data derived from GP subexpressions, before undergoing regression by the LARS algorithm (Section 2.3), requires the following preprocessing. First, if a given subexpression's output list contains not-a-number (NaN) values (resulting from, e.g., numerical over- or underflow), they are replaced by median value of that output list. If a subexpression's outputs are exclusively NaN values, it is discarded altogether as an input variable to the multiple regression (equivalently as a column of F). Subexpressions with constant outputs are also removed.

Once the regression model is induced, we adopt the following policy when applying it to data (either training or testing): if the model's prediction is beyond the interval $[-0.5, 1.5]$ (i.e., 0.5 below or above the assumed $[0, 1]$ range of target variable), it is clamped/trimmed to that value. This avoids contaminating the results with outlier predic-



(a) Root (b) Root and leaves (c) Subexpressions (d) Root and Variables (e) Subexpressions and variables

Figure 2: Post-run multiple regression strategies for the expression $logX_4 + sin(X_1 + exp(X_5))$. Outputs of blue colored nodes are included in the multiple regression.

Acronym	Description	Number of attributes	Number of instances			Reference
			Total	Training	Test	
ENC	Energy efficiency, cooling load	8	768	512	256	[17]
ENH	Energy efficiency, heating load	8	768	512	256	[17]
NOX	NOX	18	5227	4017	1210	[20]
WIR	Wine quality assessment, red wine	11	1559	1066	533	[1]
WIW	Wine quality assessment, white wine	11	4898	3265	1633	[1]

Table 1: Benchmark datasets.

Parameter	Value
pop size	1000
selection	NSGAII with crowded Tournament
crossover	Single Point Crossover
mutation	Subtree mutation
MSE	Normalized tree output
Complexity	Subtree Complexity
stop criterion	15 minutes

Table 2: Parameters settings of our competent GP.

tions from degenerate models which may happen to be induced from, e.g., very small GP trees, or GP trees producing many NaNs. For fairness of comparison, we adopt this interpretation policy for all considered methods, including MR and GP.

4.4 Post-Run MRGP Results

Table 3 presents the mean square error (MSE) on the test set returned by the compared methods. The striking observation is that GP fails to outperform the baseline MR on all benchmarks, both on train set and test set. This indicates that evolutionary search was not effective enough at finding useful subexpressions and assembling them into expressions, at least within the selected computational budget.

More importantly however, almost all variants of *post-run* MRGP reduce the error rate when compared to standard GP and MR. Interestingly, the only exception is the 'root' case, where MRGP typically attains error in between that of MR and GP. Thus, we can expect that applying regression to the root node's outputs results in less error than when that output is directly compared to the target. However, this regression is not sufficient (at least for this benchmark suite) to outperform MR; that becomes possible only when more features, collected from various subexpressions, are used of predictors in a MR process.

The practical upshot is that providing the *post-run* regression process of end-of-run GP outcomes with additional predictors derived from program subexpressions is always worth trying. The extra computation incurred is marginal, while there may be significant gains in prediction accuracy. Moreover, as this experiment shows, the improvements on the training and testing set are usually correlated: if additional predictors lower the error on the training set, then it is very likely that they will do so also for the test set. Therefore, a recommended practice could be to first verify whether performing multiple regression on the subexpressions of the best-of-run GP program improves accuracy on the training set, and adopt it for the testing set if it does, otherwise falling back to the pure, non-augmented GP model.

This result also prompts the question as to whether it would be advantageous to apply MRGP to each program during a GP run, i.e. inline with evolution. Hypothetically evolutionary pressure on fitness arising from considering a different output from a program might be favorable. Thus, we next propose *inline* MRGP.

5. GUIDING EVOLUTIONARY SEARCH USING INLINE MRGP

In the inline approach, we integrate multiple regression into fitness evaluation in the GP flow. This substantially changes the evolutionary dynamics: programs are not forced anymore to produce execution outputs that are useful since the signal used for the fitness of a given program is the output of the model regressed as a linear combination of its subexpressions.

In proceeding, we anticipate that evolutionarily selecting programs solely on the basis of multiple regression error may be ill-advised. In particular, programs of different sizes have different quantities of subexpressions and, in turn, lead to models of different sizes (vector $\hat{\beta}$ varies in length from program to program). Given more subexpressions (many of

Table 3: Testing-set mean absolute error (MAD) of models considered in **post-hoc analysis**. The best result in each column is highlighted. 0.95-confidence intervals provided in smaller font.

Method	ENC		ENH		NOX		WIR		WIW	
MR	0.0612	—	0.0541	—	0.0773	—	0.1012	—	0.0992	—
GP	0.0909	0.0358	0.1278	0.1109	0.0832	0.0034	0.1084	0.0131	0.1048	0.0050
leaves+root	0.0515	0.0079	0.0320	0.0108	0.0799	0.0039	0.0999	0.0019	0.1001	0.0009
leaves+FEFs	0.0584	0.0523	0.0497	0.0613	0.0802	0.0044	0.1049	0.0039	0.0990	0.0012
root	0.0809	0.0291	0.0727	0.0504	0.0834	0.0035	0.1045	0.0058	0.1028	0.0019
vars+root	0.0513	0.0077	0.0320	0.0108	0.0759	0.0011	0.0989	0.0017	0.0985	0.0006
vars+FEFs	0.0585	0.0523	0.0488	0.0580	0.0755	0.0021	0.1036	0.0043	0.0977	0.0011

them potentially uncorrelated), regression is likely to regress a better fitting model with lower overall error. This may bias the search towards programs that yield more subexpressions, i.e. larger trees, and hence contribute to program bloat.

To combat the anticipated bloat, similarly to the *post-run* approach, we rely on two-objective selection based on NSGA-II algorithm. In this case, the first objective is the error of the regression model induced from the outputs collected from tree subexpressions, and the second objective is model complexity. However, it is far from obvious how to measure the complexity of the compound model that involves both *symbolic* GP execution and *numeric* MR transform. To address that, we consider four different complexity measures.

GP Tree complexity (TC). This measure is simply the number of subexpressions, which coincides with program tree size, a criterion often used in GP to prefer smaller programs and lessen program bloat.

Sum of t-statistics (Sum-t). One may argue assessing model complexity by simply counting the number of subexpressions (or columns of F) is too crude. First, k is discrete, so ties between programs are likely, particularly for small programs. Secondly, and more importantly, k is insensitive to the impact of specific subexpressions. One may expect that among the subexpressions within a program, the outputs of only some of them will serve as useful predictors for the target output. In an extreme case, regression may build a perfect model using a single subexpression and neglecting all the remaining ones. Thus, two equally sized program trees may give rise to models that involve different numbers of significant parameters.

Following this observation, we define our second complexity measure as:

$$c(\hat{\beta}) = \sum_{i=2}^{k} |\frac{\hat{\beta}_i}{SE(\hat{\beta}_i)}| \tag{1}$$

where $SE(\hat{\beta}_i)$ is the standard error of the sample coefficient $\hat{\beta}_i$. The $\frac{\hat{\beta}_i}{SE(\hat{\beta}_i)}$ term in Formula (1) is the t-statistic that reflects the significance of $\hat{\beta}_i$. The greater its absolute value, the more it is likely that the corresponding predictor is a significant element of the model. Thus, $c(\hat{\beta})$ penalizes the models that use large number of significant predictors. Note that the summation in Eq. (1) starts from $i = 2$, so models are not penalized for using intercepts. Including $\hat{\beta}_1$ in the complexity measure would promote MR models that use no intercept, and compensate for its absence by engaging other predictors.

Minimum Description Length (MDL). In [16], Stine, following the work by Elias [5] and Rissanen [13], proposes a complexity measure of regression models that takes into account the significance of particular model parameters. Based on those findings, we define the complexity of a multiple regression model as the total *description length* of its parameters, i.e.,

$$c(\hat{\beta}) = \sum_{i=2}^{k} lu(\frac{\hat{\beta}_i}{SE(\hat{\beta}_i)}) \tag{2}$$

where $lu(x)$ is an idealized length of the universal code of number x [5, 13]:

$$lu(x) = 2 + \log_2 |x| + 2 \log_2 \log_2 |x|$$

where we set $\log_2(x) \equiv 0$ for $|x| < 1$.

Derivation of this measure is beyond the scope of this paper; it is however worth noting that the t-statistic of $\hat{\beta}_i$ is also present in this metric, thus penalizing models with many significant predictors. Moreover, because $lu(t)$ is concave, a model that involves a single variable is favored over a model that uses two variables that are 'half as significant', i.e., $lu(t) < 2lu(\frac{t}{2})$.

Saturated Minimum Description Length (SMDL). This measure is an extension of MDL in which the encoding length for large t-statistics are saturated. The motivation is that models that use very useful predictors (high t) should not be excessively penalized. The Saturated Minimum Description Length is computed as follows:

$$c(\hat{\beta}) = \sum_{i=2}^{k} slu(\frac{\hat{\beta}_i}{SE(\hat{\beta}_i)}) \tag{3}$$

where

$$slu(x) = \begin{cases} lu(x) & |x| < 2.626 \\ lu(2.626) & otherwise \end{cases}$$

where 2.626 is a critical value of t-statistics for significance level 0.01.

6. EXPERIMENT WITH INLINE MRGP

The experiment reported below has two primary goals: (i) to analyze whether *inline* MRGP attains better predictive accuracy than MR, conventional GP, and *post-run* MRGP, and (ii) to compare the suitability of different complexity measures to drive the *inline* MRGP process.

The inline MRGP implements a multi-objective approach based on NSGA-II. In this case, the targeted objectives are *multiple regression error* (see Section 2.2) and one of the complexity measures introduced in the previous section, namely TC, Sum-t, MDL, or SMDL. Therefore, we consider four different MRGP configurations:

(a) MRGP-TC: MR error and TC complexity

(b) MRGP-SumT: MR error and Sum-t complexity

(c) MRGP-MDL: MR error and MDL complexity

(d) MRGP-SMDL: MR error and SMDL complexity

We compare these four configurations to MR, conventional GP, and to an additional method called Root Regression GP (RRGP). RRGP represents the simplest inline MRGP strategy and consists of regressing only the program's output onto the target variable. In this sense, it is the inline version of the 'root' variant of *post-run* GP (cf. Table 3). In contrast to GP's conventional approach where program output y_p is directly compared to the target y, RRGP runs simple linear regression on y_p (and intercept) and assume that the resulting predicted values $\hat{y}$ constitute the final output. This approach finds the best linear mapping between y_p and y and is thus closely linked to using correlation coefficient to assess program quality, which has been used in the past [11, 15]. Note however that it is *not* equivalent to scaling of y_p and y to the same interval, a technique often practiced in symbolic regression.

Table 4 presents the settings of the essential evolutionary parameters of inline MRGP. MRGP presents a higher cost per fitness evaluation than conventional GP. As a result, given that we will execute with a fixed time budget, a lower number of generations will be executed. However, we anticipate that multiple regression provides a finer discrimination between subexpressions modifying the exploration/exploitation balance involved in GP search. To investigate how this balance impacts the inline MRGP process, we consider two population sizes, namely 100 and 1000.

In summary, a total of 400 MRGP runs (4 configs $\times$ 5 datasets $\times$ 10 replicas $\times$ 2 population sizes) were performed to investigate the search dynamics involved in MRGP. We assume that the final outcome of an evolutionary run is the lowest-error model found in the last generation of a run.

6.1 The results

Table 5 compares the mean absolute deviation (MAD) error on the test set, for the models evolved using the above variants of MRGP. We present there also the MAD of conventional linear regression applied to the original features (MR) and standard GP (GP).

The hybrid methods quite consistently outperform MR, and often do so by a large margin. This is not surprising, given the rich repertoire of nonlinear transformations available to GP which MR, being a linear technique, cannot model. The relative size of the gap between the evolutionary techniques and MR is problem dependent: it is rather small for the test sets of WIR and WIW, substantial for NOX, and dramatic for ENC and ENH. As expected, some problems may require more sophisticated models, while others not.

Parameter	Value
pop size	100,1000
selection	NSGAII with crowded Tournament
crossover	Single Point Crossover
mutation	Subtree mutation
MSE	Multiple regression error
Complexity	Subtree Complexity
stop criterion	15 minutes

Table 4: Parameters settings of the inline MRGP strategy.

Concerning the different MRGP variants with a population size of 1000, TC, SumT and MDL outperform MR and GP on four out of five benchmarks, while on WIR they are worse. On the other hand, MRGP-SMDL presents a significantly higher average error for the ENC and ENH problems. Moreover, this last approach has also great variance between runs.

Similar remarks apply to the runs performed with a population size of 100 individuals. In fact, no significant differences can be noticed with respect to the setup considering a larger population. The TC, SumT, and MDL variants still outperform MR and GP, while SMDL fails to provide satisfactory results for the ENC and ENH problems. MRGP thus shows to be robust since its results seem insensitive to population size.

Overall, there is no clear winner among the MRGP variants. However, MGRP-SumT seems most reliable. It often produces the smallest or the close-to-smallest error, and for many benchmarks its error varies the least across evolutionary runs (see the confidence intervals shown in smaller font). Also, for the challenging WIR benchmark, where all GP-based methods (except RRGP) behave badly, MGRP-SumT is worse than MR only by $\sim$10% for population size 1000 and even by a smaller margin for population size 100. On the other hand, on ENC it halves the test-set error of MR, and on ENH it yields a test-set error four times smaller than MR. Apparently, the fine-grained model complexity assessment implemented by this method proves effective as a second objective of the search. Nevertheless, MRGP-TC also often attains quite good results, despite its discrete nature (Section 5). The possible reason for this is computational overhead: calculating program size (MRGP-TC) is much faster than computing t-statistics (MRGP-SumT), so the former method has the chance to evaluate more individuals and execute more generations.

Except for the WIR benchmark, MRGP variants are almost always better than RRGP. This corroborates the rationale for intra-execution behavioral evaluation: the regression process provided with insight into internal program behavior (MRGP), rather than only into program output (RRGP), has the chance to build better predicting models and thus helps guide the evolutionary search process more efficiently.

The general observation resulting from this experiment follows from comparison of the results of *post-run* analysis (Table 3) with the inline approach (Tables 5). With a few exceptions, the inline approach provides lower regression error. Given that the same execution time budget was available to all methods, we may conclude that using partial program outcomes for fitness evaluation is most often beneficial. Postponing the usage of partial outcomes till the end of evolutionary runs (as the *post-run* approach does) still brings some improvements compared to conventional approaches (Table 3), but not as substantial as for the inline approach. Apparently the redefined selection pressure is capable of guiding the evolutionary search toward better performing candidate solutions, despite making fewer fitness evaluations due to computational overhead introduced by multiple regression. In fact, for a fixed time budget, MRGP yields better results than conventional GP, even if less generations are executed or a reduced population is considered.

Table 5: Test-set mean absolute error (MAD) of models evolved using standard GP, RRGP, and four variants of MRGP, averaged over 10 evolutionary runs, for population sizes 1000 and 100. MR marks the result for multiple regression of original problem variables. Two best results in each column are highlighted. 0.95-confidence intervals provided in smaller font.

Population size	Method	ENC		ENH		NOX		WIR		WIW		Avg. rank
1000	MR	0.0612	—	0.0541	—	0.0773	—	0.1012	—	0.0992	—	6.7
	GP	0.0909	0.0358	0.1278	0.1109	0.0832	0.0034	0.1084	0.0131	0.1048	0.0050	9.2
	RRGP	0.0497	0.0358	0.0193	0.0111	0.0826	0.0037	0.1012	0.0017	0.1094	0.0187	6.5
1000	MRGP-TC	0.0536	0.0288	0.0115	0.0005	0.0710	0.0015	0.1172	0.0046	0.0958	0.0008	4.4
	MRGP-SumT	0.0300	0.0037	0.0123	0.0010	0.0704	0.0023	0.1115	0.0026	0.0959	0.0005	3.0
	MRGP-MDL	0.0365	0.0098	0.0982	0.1247	0.0724	0.0018	0.1120	0.0037	0.0960	0.0009	6.4
	MRGP-SMDL	0.2549	0.2453	0.2028	0.1931	0.0732	0.0012	0.1347	0.0425	0.0961	0.0004	9.1
100	MRGP-TC	0.0325	0.0082	0.0128	0.0028	0.0692	0.0018	0.1085	0.0059	0.0961	0.0004	3.7
	MRGP-SumT	0.0307	0.0023	0.0237	0.0245	0.0718	0.0015	0.1076	0.0030	0.0963	0.0007	4.6
	MRGP-MDL	0.0739	0.0526	0.0140	0.0044	0.0705	0.0009	0.1030	0.0025	0.0964	0.0009	5.0
	MRGP-SMDL	0.1176	0.1434	0.0868	0.1222	0.0720	0.0012	0.1080	0.0037	0.0968	0.0011	7.4

7. RELATED WORK

In the past literature, there were relatively many attempts to hybridize linear regression with GP. An early example is STROGANOFF by Iba *et al.* [7], where multiple regression was used to tune the *internal* parameters of GP programs (the weights of inputs of particular program instructions). However, program fitness was defined in a conventional way, no attempt was made to map the behavior of entire program onto target, and model complexity was not considered there.

McKay et al. [12] used continuum regression (a generalization of multiple regression, principal component regression, and partial least squares) in combination with GP to build nonlinear models. The regression algorithm iterates over latent variables, and for each of them invokes an independent GP run aimed at matching that variable (this may be likened to, e.g., cascade correlation learning algorithms developed for neural networks).

Keijzer proposed to use univariate (simple) symbolic regression to map program output onto the desired output, in every act of fitness evaluation independently. Apart from demonstrating the usefulness of this approach in practice [9], Keijzer provided also theoretical evidence that evaluating GP individuals in this manner leads to programs that are on average better than programs evolved using conventional GP [10]. Similarly to McKay *et al.*'s work, only the program output is used as a predictor variable in regression. This method is equivalent to RRGP considered in Section 6.

The approach proposed by Sobester *et al.* in [14] is similar to [9] in also using regression within fitness evaluation. In particular, GP was used there to evolve 'intervening variables' for linear regression. Interestingly, for multivariate problems a basis function was evolved for every dimension independently, by means of cooperative coevolution. However, only final program outcomes were taken into account. Also, model complexity was not considered.

By linearly combining program components, MRGP becomes distantly related to approaches that optimize constants in programs. One of the earliest examples is here the work by Jiang and Wright [8], who used the Levenberg-Marquardt algorithm for this purpose. The same algorithm is used for that purpose in a purely deterministic search method proposed in [19]. However, such algorithms implicitly assume that the structural form of a GP program being tuned is correct, and that its performance can be improved by adjusting the constants. MRGP, as argued in Introduction, is based on fundamentally different premises, viz. that the arrangement of program components can be inappropriate, and thus deteriorate program fitness, and that the hidden yet valuable program components can be detected by a MR analysis.

In the context of past studies, MRGP remains novel in (i) exploiting multiple features generated by program subexpressions, (ii) taking into account model complexity, and (iii) using state-of-the-art stage-wise least-angle regression to avoid rank deficiency problem harassing conventional multiple regression.

8. SUMMARY AND FUTURE WORK

We have introduced Multiple Regression Genetic Programming, a new means of executing a genetic program that provides a practical and cost-neutral improvement to GP. MRGP hybridizes GP with multiple regression by decoupling program's subexpressions and linearly combining them via regressing their outputs on the target variable. The use of Least Angle Regression, a stepwise linear regression algorithm, was important for the success of the approach, since it deals with the ways using GP subexpression outputs in this manner could trip up regression.

When applied in a *post-run* fashion, MRGP always improves the output of the programs obtained with a competent GP system and, as a consequence, often outperforms MR. In this mode, multiple regression only needs to be applied once, to the best-of-run individual. We consider these to be strong arguments to systematically employ MRGP *after every GP run*.

MRGP turns out to be even more beneficial when performed *inline*, i.e., when the feedback provided by the multiple regression process drives the evolutionary process. In this case, the search algorithm is provided with more robust and more reliable information on the value of individual's genetic code, which captures a complete decomposition of the program's behavior, rather than only its output. Compared to post-run MRGP, it introduces a higher cost for each fitness evaluation but when evaluated on an equal execution duration it yields substantial improvement of predictive accuracy. In this sense, both these usage scenarios (post-run and inline) complement each other.

We have analyzed the impact of considering different complexity measures corresponding both the complexity of genetic programs and that of the retrieved linear models. In-

line MRGP has shown to be robust to this parameter, outperforming MR, GP, and post-run MRGP.

As a future work, we plan to extend the post-run MRGP approach so that the final linear model combines features retrieved from *multiple models*. With this approach, we expect to obtain a wider variety of features, many of them possibly uncorrelated, and increase the accuracy of the final regressed linear model. Apart from that, we would like to consider use of *regularized* linear regression, which would automatically promote simpler linear models and thus possibly impose qualitatively different selection pressure on the population of evolving programs.

9. ACKNOWLEDGMENTS

This work was supported by the Li Ka Shing Foundation. The authors would like to thank Dr. Kalyan Veeramachaneni for his contributions to this paper. Krzysztof Krawiec acknowledges financial support from Fulbright Commission and grants no. 91-543 and DEC-2011/01/B/ST6/07318.

10. REFERENCES

[1] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547 – 553, 2009. Smart Business Networks: Concepts and Empirical Evidence.

[2] R. R. Curtin, J. R. Cline, N. P. Slagle, W. B. March, P. Ram, N. A. Mehta, and A. G. Gray. MLPACK: A scalable C++ machine learning library. *Journal of Machine Learning Research*, 14:801–805, 2013.

[3] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182 –197, apr 2002.

[4] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.

[5] P. Elias. Universal codeword sets and representations of the integers. *Information Theory, IEEE Transactions on*, 21(2):194–203, 1975.

[6] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition, 2009.

[7] H. Iba, T. Sato, and H. de Garis. Numerical genetic programming for system identification. In J. P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 64–75, Tahoe City, California, USA, 9 July 1995.

[8] M. Jiang and A. H. Wright. An adaptive function identification system. In *Proceedings of the IEEE/ACM Conference on Developing and Managing Intelligent System Projects, Vienna, Virginia, USA*, pages 47–53, Mar. 1993.

[9] 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, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 70–82, Essex, 14-16 Apr. 2003. Springer-Verlag.

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

[11] S.-Y. Liong, T. R. Gautam, S. T. Khu, V. Babovic, M. Keijzer, and N. Muttil. Genetic programming: A new paradigm in rainfall runoff modeling. *Journal of American Water Resources Association*, 38(3):705–718, June 2002.

[12] B. McKay, M. Willis, D. Searson, and G. Montague. Non-linear continuum regression using genetic programming. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1106–1111, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

[13] J. Rissanen. A Universal Prior For Integers And Estimation By Minimum Description Length. *Annals Of Statistics*, 11(2):416–431, 1983.

[14] A. Sobester, P. B. Nair, and A. J. Keane. Evolving intervening variables for response surface approximations. In *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, number AIAA 2004-4379 in , pages 1–12, Albany, New York, USA, 30-Aug. 1-Sept. 2004.

[15] K. Stanislawska, K. Krawiec, and Z. W. Kundzewicz. Modeling global temperature changes with genetic programming. *Computer & Mathematics with Applications*, 64(12):3717–3728, 2012.

[16] R. A. Stine. Model Selection Using Information Theory and the MDL Principle. *Sociological Methods Research*, 33(2):230–260, Nov. 2004.

[17] A. Tsanas and A. Xifara. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings*, 49(0):560 – 567, 2012.

[18] E. Vladislavleva. *Model-based Problem Solving through Symbolic Regression via Pareto Genetic Programming*. PhD thesis, Tilburg University, Tilburg, the Netherlands, 2008.

[19] T. Worm and K. Chiu. Prioritized grammar enumeration: symbolic regression by dynamic programming. In C. Blum, E. Alba, A. Auger, J. Bacardit, J. Bongard, J. Branke, N. Bredeche, D. Brockhoff, F. Chicano, A. Dorin, R. Doursat, A. Ekart, T. Friedrich, M. Giacobini, M. Harman, H. Iba, C. Igel, T. Jansen, T. Kovacs, T. Kowaliw, M. Lopez-Ibanez, J. A. Lozano, G. Luque, J. McCall, A. Moraglio, A. Motsinger-Reif, F. Neumann, G. Ochoa, G. Olague, Y.-S. Ong, M. E. Palmer, G. L. Pappa, K. E. Parsopoulos, T. Schmickl, S. L. Smith, C. Solnon, T. Stuetzle, E.-G. Talbi, D. Tauritz, and L. Vanneschi, editors, *GECCO '13: Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*, pages 1021–1028, Amsterdam, The Netherlands, 6-10 July 2013. ACM.

[20] F. Xue, R. Subbu, and P. Bonissone. Locally weighted fusion of multiple predictive models. In *Neural Networks, 2006. IJCNN '06. International Joint Conference on*, pages 2137–2143, 2006.