# Improving Symbolic Regression through a Semantics-driven Framework

Quang Nhat Huynh, Hemant Kumar Singh and Tapabrata Ray
School of Engineering and Information Technology, University of New South Wales, Australia.
Email: quang.huynh@student.adfa.edu.au, {h.singh, t.ray}@adfa.edu.au. Web: www.mdolab.net

***Abstract*—The process of identifying analytical relationships among variables and responses in observed data is commonly referred to as Symbolic Regression (SR). Genetic Programming is one of the commonly used approaches for SR, which operates by *evolving* expressions. Such relationships could be explicit or implicit in nature, of which the former has been more extensively studied in literature. Even though extensive studies have been done in SR, the fundamental challenges such as bloat, loss of diversity and accurate determination of coefficients still persist. Recently, semantics and multi-objective formulation have been suggested as potential tools to alleviate these issues by building more intelligence in the search process. However, studies along both these directions have been in isolation and applied only to selected components of SR so far. In this paper, we intend to build a framework that integrates semantics deeper into more components of SR. The framework could be operated in conventional single objective as well as multi-objective mode and is capable of dealing with both explicit and implicit functions. Semantics are used in the proposed framework for improving compactness and diversity of expressions, crossover and local exploitation. Numerical experiments are presented on a set of benchmark problems to demonstrate the strengths of the proposed approach.**

## I. Introduction

The problem of finding comprehensible relationships among input variables and output responses is of significant interest in data analytics. It is used by several industries including science, engineering, business and management to understand model behaviors, for validation of theories and better decision making. Traditional approaches require "guessing" an equation based on prior knowledge and applying the corresponding regression methods such as linear, quadratic or exponential regressions to determine the coefficients [1], [2]. However, realistically this approach has limited application as it is often not possible to predict the complexity and types of existing relations within the data in advance. Symbolic regression (SR), a branch of genetic programming (GP), comes in handy in such situations, which *evolves* expressions to model the given data.

In SR, an equation is usually represented as a binary tree, which is constructed using different types of nodes [3], [4]. For real-valued SR, the intermediate nodes represent the mathematical operators (binary or unary), while the terminal *leaf* nodes are either constants or variables. Due to the nature of mathematical operators, certain additional rules may be needed to handle singularities, e.g. division by zero or logarithm of zero may be set to return a value of 1 [3]. Similarly, if an operator returns a value too large in magnitude, it may be set to a predefine cut-off value to avoid out-of-range errors.

The expressions that may define the relationships among the input and output data can be broadly classified into two categories: explicit and implicit. Most of the equations are of the former type, where an observed output is an explicit function of the input variables ($y = f(\mathbf{x})$). In the process of identifying the explicit equations, SR usually suffers a limitation called *bloat*, which implies that the tree sizes grow too large relative to the expressions they represent. This has an adverse effect on the time required for genetic evolution, as well as on the diversity. These drawbacks are further aggravated for the case of implicit equations, which are of the form $f(\mathbf{x}, y) = 0$. These expressions are commonly used to describe physical laws/phenomena such as conservation of energy and momentum [5]. In implicit equation mining, commonly used $L_1$ or $L_2$ norm cannot be used as a measure of accuracy (unlike explicit equations), as it would simply return a set of trivial expressions [6], for example $x - x = 0$. To counter this, a measure based on derivatives of the functions was proposed in [6], which was further integrated in a multi-objective formulation in [5] to mine compact implicit equations. The determination of correct coefficients for both types of equations is also often challenging [5]. In literature, many methods have been suggested to tackle these critical problems, such as ramped half-and-half initialization [3] or fitness sharing using edit distance metric [7] to produce more diverse individuals, rule-based components for algebraic manipulation of expressions [2] to reduce bloat, and trials of set of random scalars [5] to rectify the coefficients. Incorporation of these methods have demonstrated improvements in accuracy, compactness and convergence rate of expressions evolved using SR.

One of the emerging approaches to improve GP/SR is incorporation of *semantics*. The use of semantics makes the GP somewhat *intelligent* by being more informed about the expressions being evaluated [8] instead of blindly searching based on syntax alone. Semantics is rather domain dependent and no universal definition exists that can cover all types of GPs. In [4], the approaches representing and extracting semantics were categorized into three groups: grammar-based, formal methods, and GP s-tree representation. Some of the recent noteworthy works in the domain include geometric semantics [9] and approximate semantics by surrogate [10], among others. Semantics has been individually integrated into different main components of GP such as initialization,

crossover and mutation [4], [9], [11] to show promising improvements. The authors of the review paper [8] strongly recommend applying semantics into more components to enhance the strength of GP further.

Another prominent development recently has been use of multi-objective GP formulations. In this context, a few different types of objective functions have been investigated. Tree size (or sum of tree sizes) was used as a second objective in addition to accuracy in [12], [5], [1] to reduce bloat. Testing error of a subset of fitness cases was used in [13] to improve generality of equations. Application specific fitness objective function pairs, such as sensitivity and specificity, false-positive and false-positive rate, recall and precision, etc. were studied in [14] to generate Pareto optimal decision trees.

Both semantics and multi-objective approaches may often increase the complexity of GP. Therefore, the trade-off between their performance and complexity needs to be carefully managed to achieve a good balance of accuracy, compactness and generalization. In this paper, the critical challenges facing SR mentioned above are intended to be addressed by integrating semantics deeper in SR. Towards this goal, a semantics based framework is developed, which can be executed both in single objective or multi-objective mode, and is capable of dealing with both explicit and implicit problems. While the previously reported studies have only used semantics typically for one component, in this study we use it for a number of different components, such as compaction , uniqueness, crossover, and local exploitation. The phrase *uniqueness* mentioned in this refers to genotype. Additionally, a local search has been introduced to determine coefficients accurately. New mutation and local exploitation methods are also suggested to enhance the global exploration and local exploitation abilities. In order to objectively assess the benefits of proposed approach, we compare the results to those reported in the literature using an existing semantics based algorithm [4] for explicit functions and a multi-objective algorithm [5] for implicit functions.

The rest of this paper is structured as follows. Section II briefly discusses relevant background of the semantics relevant to this study. Thereafter, in Section III, the details of the proposed algorithm are discussed. Numerical experiments using the proposed approach are presented in Section IV, and summary and future work are discussed in Section V.

## II. Sampling semantics

As mentioned in the previous section, semantics provide GP with a better interpretation of the expression being evaluated, rather than blindly searching for expressions. In computational sense, semantics could be defined/extracted in a number of different ways, out of which we consider a particular one. The authors of [4] defined semantics for real-value SR based on a method called *sampling semantics*, where a number of points in the problem domain are randomly sampled and used to evaluate the equations represented by a tree. If the difference between evaluated values of two expressions is less than a given threshold, they are considered as semantically equivalent. Thereafter, two types of semantic crossover methods were proposed based on this definition, named *semantics aware crossover* (SAC) and *semantic similarity-based crossover* (SSC). The former forces the sub-trees selected for crossover/swapping to have a semantic difference higher than a given threshold $(t_1)$. The latter adds one more condition, where the difference in semantics of two sub-trees must also less than another threshold $(t_2)$. The interpretation of these two thresholds are relatively straightforward: the semantics of two sub-trees must not be the same to prevent the redundant swapping and must not be too different to avoid dramatic changes in semantics of the offspring compared to their parents. Along with all the advantages of SAC and SSC, there is one major disadvantage: the values of the thresholds are hard to predetermine for different problems.

The computational effort required for SR is measured by counting number of node evaluations in [4], and an upper limit on it used to terminate the algorithm. SAC and SSC require the evaluations of sub-trees to determine the semantics every time, thus using up a large proportion of the number of node evaluations. This implementation may be an accurate way to obtain the sampling semantics, but it is evidently costly. In our implementation, we reduce this computational effort significantly by calculating the semantics at the same time as the fitness evaluation, only based on the *given fitness cases*. As a result, a large proportion of allotted computational resource is saved for other mechanisms. The implication of this change may depend on the distribution of the fitness cases in the sampled domain. While the original implementation is intended to make the equations more *generalized*, the proposed implementation proves to be as useful as the original one in our experiments, for all problems studied in [4].

## III. Proposed Framework

To reduce the adverse effect of bloat, a multi-objective genetic programming approach (MOGPA) was proposed in [5], in which the error and the depth of the trees were minimized concurrently. The accuracy was quantified using mean squared error (MSE) for explicit problems and mean log of errors in derivatives (MLED) for implicit equations. The resulting final trees were more compact and easier to interpret. However, the operators implemented in the framework were relatively basic, leaving further scope of improvement.

In the multi-objective context, our work in this paper attempts to improve upon the MOGPA using semantics and other additional mechanisms as will be detailed in the next section. The new mechanisms include compactness, uniqueness, multipoint mutation, constant optimization and local exploitation. However, the benefits are not limited to multi-objective mode only, and the framework could instead also be executed in single-objective mode (minimization of error alone), in which case the advantages over existing single-objective approaches such as [4] could be demonstrated. The enhanced framework is referred to here as semantics based symbolic regression (SSR) and summarized in Algorithm 1. Subsequently, we describe

the enhanced mechanisms first (denoted with asterisk ($^*$) in Algorithm 1), since they are integrated in various stages of the algorithm. Finally, we discuss each of the main stages of the overall algorithm.

**Algorithm 1** Semantics based Symbolic Regression (SSR)

1: **Initialization:** Generate initial population; Evaluate and sort the trees in the population;
2: **while** termination condition not met **do**
3: **Crossover:** Perform semantic crossover of parents selected using binary tournament; Find the **compact**$^*$ form of the child population;
4: **Mutation:** Perform multi-point mutation on the child population; Find the compact form of the child population again;
5: **Evaluation:** Evaluate the fitness on all objectives of the child population;
6: Remove duplicated trees through **uniqueness**$^*$ check;
7: **Sorting:** Sort the trees in child and parent population based on either single or multi-objective criteria as applicable;
8: Select the best trees to carry to next generation;
9: Perform **constant optimization**$^*$ and **local exploitation**$^*$; Re-sort the population;
10: **end while**

### A. Compactness

The mechanism to find the compact form of a tree can be divided into two major steps. The first is to replace the operator nodes which always return constant values by terminal constant nodes. The second step is to identify the sub-trees having the form of $(0+expression)$, $(expression+0)$, $(expression-0)$, $(1\times expression)$, $(expression\times 1)$ and $(expression\div 1)$ and shorten them. For example, an equation including a bloat such as $((((x_1-1)+(x_1/x_1))-x_1)+x_2)$ will first be transformed into $(0+x_2)$ and then to simply $x_2$.

In order to determine if an operator node always returns a constant value, the semantics obtained in tree fitness evaluations are used. For a particular node, if all the values corresponding to all the fitness cases are within a preset threshold with respect to the mean of these values, this operator node is considered as always returning a constant. Without semantics, it would be difficult to shorten $(((x_1-1)+(x_1/x_1))-x_1)$ to 0. Note that the process of finding the compact form will not incur any extra node evaluations, except in the initialization phase where the fitness of the trees need to be evaluated right after they are (randomly) constructed. In order to find the compact form of this new tree, node evaluations need to be carried out. Finding compactness based on semantics has also been applied in [15], however, our proposed mechanism returns a more compact form of the equation by dealing with the constants 0 and 1 if necessary.

Both steps of the compactness mechanism are implemented in recursive manner and stop only when all the nodes of the tree have been visited. Algorithm 2 illustrates the first step of this mechanism.

### B. Uniqueness

To compare if two trees are identical, the trees are converted to the equations in the form of strings and a simple string comparison method is executed first. If the strings are different, then the number of nodes, number of operator nodes, identities of the operator nodes, number of variable nodes, identities of the variable nodes are examined. If all the conditions above fail to differentiate the two trees, the semantics is made use of. If the original semantics are different, all the constant nodes of the two trees are reset to contain value of 1 first. Then, the two trees are evaluated using new values of constant nodes to obtain their semantics. If new semantics are identical for both trees, it means that the structure of the trees are the same and the only difference is the values of the constant nodes; in which case the one with better fitness is kept.

The intent of employing this mechanism is to maintain the diversity of the population by maintaining unique trees. In [16], it was stated that "diversity does not necessary cause better performance but better performance is seen with higher diversity". In the proposed SSR, the diversity is achieved by ensuring the uniqueness of all the equations in the population from the beginning till the end. In the initialization phase, whenever a new tree is built, it is compared with the existing trees in the population. If this new tree is not unique, it will be deleted and rebuilt. The cycle will continue and only stop once there the population is filled with unique trees. The process of comparing the trees is outlined in Algorithm 3.

When the child population is created by crossover and mutation, each child is compared with the rest of the child population so that the duplications can be removed. Among a group of duplicated children, only one with the best fitness is saved. Thereafter the unique children are compared with the parents to further remove the duplications between the two populations. Once again, only the fittest equation among the duplications is kept.

In addition, right after the constant optimization or local exploiting is finished (discussed in next section), the new tree is again examined for its uniqueness. If this tree is a duplicate, it is discarded. The constants of this tree are then flagged not to be optimized again subsequently.

**Algorithm 2** Replace_Operator_By_Constant

**Require:** $N_{id}$: root node of a tree/sub-tree;
**Ensure:** $N_{id-new}$: root node of a tree/sub-tree with operator nodes, which always return constant values, are replaced by terminal constant nodes

1: **if** $N_{id}$ is not an operator node **then**
2: **return**
3: **end if**
4: $is\_const \leftarrow true$;
5: **for** $i=0$ TO $|fitness\_cases|-1$ **do**
6: **if** $fitness\_cases[i]-mean\_fitness\_case>\varepsilon$ **then**
7: $is\_const \leftarrow false$;
8: break the loop;
9: **end if**
10: **end for**;
11: **if** $is\_const == true$ **then**
12: Replace this node by a terminal constant node;
13: **else**
14: Replace_Operator_By_Constant(child node of $N_{id}$);
15: **end if**

### C. Constant Optimization and Local Exploitation

Crossover and mutation are useful in finding the correct structures of the equations but inadequate for identifying accurate values of the constant coefficients. Thus, a new constant optimization mechanism is proposed to address this drawback. To perform this operation, the terminal variable nodes are treated as constants and the terminal constant nodes are treated

**Algorithm 3** Trees_Comparison

**Require:**
$T_1$: First tree for comparison;
$T_2$: Second tree for comparison;
**Ensure:**
*True*: if two tree are identical;
*False*: if two tree are nonidentical;

Convert the trees into strings $epr1$, $epr2$;
**if** $epr1 == epr2$ **then**
  **return** *True*;
**end if**
**if** $num\_node\_tree\_1 \neq num\_node\_tree\_2$ **then**
  **return** *False*;
**end if**
**if** $num\_operator\_tree\_1 \neq num\_operator\_tree\_2$ **then**
  **return** *False*;
**end if**
**if** $list\_operator\_tree\_1 \neq list\_operator\_tree\_2$ **then**
  **return** *False*;
**end if**
**if** $num\_variable\_tree\_1 \neq num\_variable\_tree\_2$ **then**
  **return** *False*;
**end if**
**if** $list\_variable\_tree\_1 \neq list\_variable\_tree\_2$ **then**
  **return** *False*;
**end if**
**if** $value\_tree\_1 == value\_tree\_2$ **then**
  **return** *True*;
**end if**
Replace all the values of terminal constant nodes by 1;
Re-evaluate the semantics of two trees;
$is\_semantics\_same \leftarrow False$;
**if** $value\_tree\_1 == value\_tree\_2$ **then**
  $is\_semantics\_same \leftarrow True$;
**end if**
Revert all the terminal constant nodes back to the original values;
Re-evaluate the semantics of two trees;
**if** $is\_semantics\_same == True$ **then**
  **return** *True*;
**else**
  **return** *False*;
**end if**

**Algorithm 4** Constant_Optimization & Local_Exploitation

**Require:** $P$: Current population;
**Ensure:** $P_{new}$: Population with some trees have been optimized or locally exploited;

$P_{CST} \leftarrow \emptyset$;
At each depth, find the best tree whose terminal constant nodes have not been optimized and assign it to $P_{CST}$;
**for** each tree $T$ in $P_{CST}$ **do**
  Treat terminal constant nodes as variables and use gradient descent method to optimized there values;
  $is\_const\_optimized \leftarrow True$;
  $Local\_Exploit(T)$;
  **if** $is\_local\_exploit\_successful$ **then**
    $is\_const\_optimized \leftarrow False$;
  **end if**
**end for**
$P_{LE} \leftarrow \emptyset$;
At each depth, find the best tree and assign it to $P_{LE}$;
**for** each tree $T$ in $P_{LE}$ **do**
  $Local\_Exploit(T)$;
  **if** $is\_local\_exploit\_successful$ **then**
    $is\_const\_optimized \leftarrow False$;
  **end if**
**end for**

as variables. The forms of the equations are known at this stage and any reasonable local search method can be applied to identify the most appropriate values for the terminal constant nodes. The optimization process is guided by the fitness value of the trees, which is either $L_1$, $L_2$ or *MLED* depending on the formulation. In other words, the optimizer will try to adjust the values of the constants in a particular equation, which are treated as variables now, improve its fitness.

Each tree has a flag to mark if its constants have been optimized before. In each generation, among all the trees having the same depth and optimized flags set to *false*, the fittest tree will qualify for constant optimization. As a result, each depth will have exactly one tree to be optimized in each generation. Note that irrespective of whether constant optimization is able to improve a tree or not, the optimized flag is switched to set to *true* so that further computation is not wasted on this particular tree again. However, in the case the tree is modified by compactness mechanism or local exploitation later, the flag is reset to *false* and this tree can undergo constant optimization again.

Local exploitation is simply a basic mutation operator combined with fitness checking. Basic mutation means that a binary node will be replaced by another binary node, an unary node will be replaced by another unary node and a terminal node will be replaced by another terminal node, either a constant or variable. After the mutation, the fitness will be re-evaluated. If the mutated is fitter and unique (doesn't already exist), it is kept; otherwise it is discarded. The number of trials for local exploitation is set to 4 in our work. Besides, if the local exploitation is successful, the flag for constant optimization of this tree is reset to *false*. This operator is suggested to enhance the exploitation in the search space surrounding the equations in the current population. It increases the probability of finding the accurate form for the equations with high fitness and improves the structures of the equations with poor fitness.

With this scheme, all trees in the population eventually undergo constant optimization and local exploitation, resulting in improved fitness. The constant optimization and local exploitation mechanisms are summarized in Algorithm 4.

### *D. Overall algorithm*

Having outlined the key mechanisms incorporated in the framework, we now discuss the main steps of the overall algorithm, as outlined earlier in Algorithm 1. The differences from MOGPA [5] are also highlighted where applicable.

*1) Initialization:* Unlike a standard initialization process in MOGPA [5], SSR not only seeks a diverse initial population but also controls the depths of the trees in the population. In order to achieve this, whenever a new tree is created, it undergoes compactness and uniqueness operations. Subsequently, it is tested for the maximum allowable depth (pre-defined). The tree is deleted and rebuilt if any of the above conditions is/are violated.

*2) Crossover:* The crossover operators used in SSR are either SSC or SAC [4] depending on the test problems. In our experiments, we use SSC for cases where upper bound is available from previous studies ($F_1 - F_{10}$), else we use SAC. This is because results using SSC are sensitive to the upper threshold used, and there is no definitive way of identifying its value. The use of semantics based crossover can be considered as an upgrade to MOGPA [5] in which only the standard subtree swap crossover was used.

*3) Mutation Operator:* SSR contains two different mutation mechanisms. First is a standard mutation, where a chosen node is replaced with a node of the same class; i.e., a binary is mutated to another binary node, a unary to another unary node and a terminal to another terminal node. This mutation is used for problems $F_1 - F_{10}$ in the numerical experiments (Section IV) for a fair comparison with [4]. Second is a new mutation operator used for problems otherwise, in which multiple nodes of a tree are mutated. A fixed percentage of nodes in a tree are selected for mutation and half of the selected nodes are forced to change to a new node in the same class. The other half of selected nodes are forced to mutate to a new node belonging to other classes, which are chosen randomly. This new operator is referred to as multi-point mutation and it helps to enhance the exploration ability of SSR, since the semantics of a tree after mutation can be significantly changed. However, as is the usual practice, the mutation rate is always kept low (0.05) to prevent it from happening too often.

*4) Depth Reserving:* There is no depth control mechanism in crossover and mutation operators, so the depths of the trees in the child population may exceed the specified maximum depth. The trees are pruned to a maximum prescribed depth if necessary. In some specific cases, if a lower limit on tree depth is known, this tree is expanded instead to satisfy the minimum depth requirement if needed. After the depth of a tree is altered, compactness mechanism will find the most compact form of the tree again. The depth reserving is implemented after the crossover and mutation operations.

*5) Evaluation:* The fitness of each tree is calculated based on the type of problem. For explicit problems, the error could be quantified using a number of quantities, such as the sum of absolute error on all data points ($L_1$ norm), sum of squared errors ($L_2$ norm) or the mean squared error (MSE). In this study, $L_1$ norm is used to quantify the error, which is set as the objective to be minimized. For implicit problems, none of these measures could be used, and the mean logarithmic error in derivatives (MLED) as suggested in [6], [5] is used as the objective instead, which is calculated shown in Equation 1.

$$MLED = \frac{1}{N}\sum_{i=1}^{N} log(1+\mid \frac{\Delta x_{1,i}}{\Delta x_{2,i}} - \frac{\delta x_{1,i}}{\delta x_{2,i}} \mid), \tag{1}$$

where $N$ is the number of fitness cases, $\Delta x_{1,i}/\Delta x_{2,i}$ is an implicit derivative estimated from the data, and $\delta x_{1,i}/\delta x_{2,i}$ is the implicit derivative derived from the candidate implicit equation

When the SSR is run in multi-objective mode, depth of the tree is assigned as the additional objective to be minimized. Consequently, a range of trees with best possible trade-off between accuracy and complexity of expressions are returned in the output.

*6) Sorting:* Quick sort is used to rank the population based on fitness (e.g. $L_1$ norm) in the case of single-objective mode, while non-dominated sorting is used in the case of multi-objective mode. The sorting is applied on the set of (parent+child) population in order to select the best trees to carry to next generation. Additionally, after constant optimization and local exploitation, the population is sorted again since the fitness of the trees may have been altered in the process.

## IV. Numerical Experiments

In this section, the proposed SSR algorithm is tested with two sets of problems. The first includes ten explicit problems from [4] and the second contains three implicit problems from [6], [5]. Considering the nature of previously reported results available for comparison and the space limitations, we present the results using single objective mode for explicit problems only, and the results using multi-objective mode for implicit problems only.

### A. Single-objective Mode

Ten explicit problems, listed in Table I, are studied in [4] to illustrate the superiority of SSC against other crossover operators. Here, these problems are used to mainly demonstrate that the use of compactness and uniqueness can enhance the results further. As mentioned previously, the semantics at each node of a tree is not calculated based on random points in the problem's domain as in [4] but the points in the given data. When a tree is evaluated for its fitness, the semantics of the nodes are stored in the memory. Therefore, the crossover operator can compare the semantics of the nodes without any additional calculations. As a result, more number of node evaluations can be reserved for finding compactness and uniqueness.

For a fair comparison in single objective mode with SSC12 [4], the proposed framework is executed using SSC crossover and simple mutation (where a node can only be changed to another in the same class). All the parameters are set to be as close as those of SSC12 [4], shown in Table II. The thresholds $\{10^{-4}, 0.4\}$ and number of trials (12) of SSC12 are kept the same in our implementation. Binary tournament is used for selecting parents for crossover, and $L_1$ is used as the objective function. In addition, problems $F_1$ to $F_8$ in [4] only allow terminal constant nodes to take single value of 1, so constant optimization and local exploitation are excluded in these problems. For the other two problems, terminal constant nodes are not allowed at all so the compactness scheme also becomes irrelevant.

Thirty independent runs are performed for each problem using SSR. A run is considered as *successful* if there is at least one tree in the population hitting all the fitness cases. As in [4], a fitness case is regarded as *hit* if the difference between predict output and observed output is less than 0.01.

As can be seen in Table III, the proportion of successful runs is significantly improved using SSR. Compactness and uniqueness manage to boost the successful rate to more than 90% for 6 out of 8 first problems, the exceptions being $F_4$ and $F_8$. Problem $F_4$ is relatively hard to solve since it is a polynomial equation with a high degree of 6. The problem $F_8$ has a simple square root form, but the square root operator is not included in the unary operator set, and thus only an approximate form could be found.

TABLE I: 10 explicit equations

| Equations | Fitcases |
|---|---|
| $F_1 = x_1^3 + x_1^2 + x_1$ | 20 random points $\subseteq$ [-1,1] |
| $F_2 = x_1^4 + x_1^3 + x_1^2 + x_1$ | 20 random points $\subseteq$ [-1,1] |
| $F_3 = x_1^5 + x_1^4 + x_1^3 + x_1^2 + x_1$ | 20 random points $\subseteq$ [-1,1] |
| $F_4 = x_1^6 + x_1^5 + x_1^4 + x_1^3 + x_1^2 + x_1$ | 20 random points $\subseteq$ [-1,1] |
| $F_5 = sin(x_1^2)cos(x_1) - 1$ | 20 random points $\subseteq$ [-1,1] |
| $F_6 = sin(x_1) + sin(x_1 + x_1^2)$ | 20 random points $\subseteq$ [-1,1] |
| $F_7 = log(x_1 + 1) + log(x_1^2 + 1)$ | 20 random points $\subseteq$ [0,2] |
| $F_8 = \sqrt{x_1}$ | 20 random points $\subseteq$ [0,4] |
| $F_9 = sin(x_1) + sin(x_2^2)$ | 100 random points $\subseteq$ [-1,1] |
| $F_{10} = 2sin(x_1)cos(x_2)$ | 100 random points $\subseteq$ [-1,1] |

TABLE II: Runs and evolutionary parameter values of single-objective SSR

| Parameter | Value |
|---|---|
| Population size | 500 |
| Tournament size | 2 |
| Crossover probability | 1 |
| Mutation probability | 0.05 |
| Max. depth | 15 |
| Operators | +, -, *, /, sin, cos, exp, log |
| Terminals nodes | $X_1$, 1 for single variable problems and $X_1$, $X_2$ for bi-variable problems |
| No. of node evaluations | $15 \times 10^6$ |

For problems $F_9$ and $F_{10}$, uniqueness alone is observed to raise the success rate up to 90% and 96% respectively, thus justifying the advantage of high diversity. Tables IV indicates the number of runs over 30 runs that single-objective SSR returns the exact equations. Table V lists out samples of the expressions obtained for the exact equations and their corresponding depths returned by the algorithm. Figure 1 reports the median convergence rates for all explicit problems. It can be seen that SSR exhibits a remarkable convergence rate, reaching the hit threshold within $2.5 \times 10^6$ node evaluations.

TABLE III: Percentage of successful runs of SSC12 and single-objective SSR for 10 explicit problems

| | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ | $F_7$ | $F_8$ | $F_9$ | $F_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| SSC12 | 67 | 33 | 14 | 12 | 47 | 47 | 66 | 38 | 37 | 51 |
| SSR | 100 | 93 | 96 | 73 | 100 | 100 | 80 | 73 | 90 | 96 |

TABLE IV: Number of runs out of 30 that single-objective SSR returns the exact equations for 10 explicit problems

| | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ | $F_7$ | $F_8$ | $F_9$ | $F_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Number of runs | 30 | 2 | 20 | 8 | 2 | 21 | 0 | 0 | 16 | 27 |

The mean depths of median runs of problem $F_3$ and $F_9$ are shown in Figure 2. Due to the compactness operation, the mean depth of $F_3$ starts low at the beginning of the run and increases to approximately 12 at generation 40. In contrast, the mean depth of problem $F_9$, which is run without the compactness, starts with a bigger value and reaches the value of 12 at generation 20.

Despite the improvement in the results, there are still a reasonable scope for further enhancement in the algorithm. Even though the correct and most compact equations were found, the single-objective mode of SSR usually returns the equations with maximum depth in the final population. Observing the

TABLE V: Examples of accurate equations returned by single-objective SSR for 10 explicit problems

| | Equations | Depth |
|---|---|---|
| $F_1$ | $(((x_1 + (x_1 * x_1)) + 1.0) * x_1)$ | 5 |
| $F_2$ | $(((x_1 + (x_1 * (x_1 + (x_1 * x_1)))) * x_1) + x_1)$ | 7 |
| $F_3$ | $((x_1 * ((1.0 + (((x_1 * ((x_1 * x_1) + x_1)) + x_1) * x_1)) + x_1)) * 1.0)$ | 10 |
| $F_4$ | $((x_1 * (((((x_1 + (((x_1 * x_1) + x_1) * x_1)) * x_1) + x_1) * x_1) + x_1)) + x_1)$ | 11 |
| $F_5$ | $((cos(x_1) * sin(exp(log((x_1 * x_1))))) - 1.0)$ | 7 |
| $F_6$ | $(sin(x_1) + sin((x_1 * (x_1 + 1.0))))$ | 5 |
| $F_7$ | no exact equation found | - |
| $F_8$ | no exact equation available | - |
| $F_9$ | $(sin(x_1) + sin((x_2 * (x_2 - (x_1 - x_1)))))$ | 6 |
| $F_{10}$ | $((cos(x_2) * sin(x_1)) + (sin(x_1) * cos(x_2)))$ | 4 |

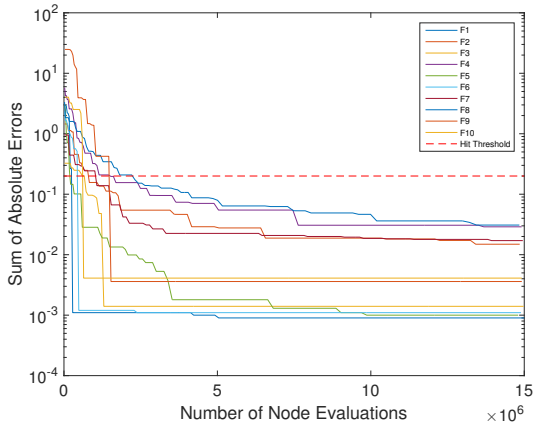


Fig. 1: Convergence rate for 10 explicit problems

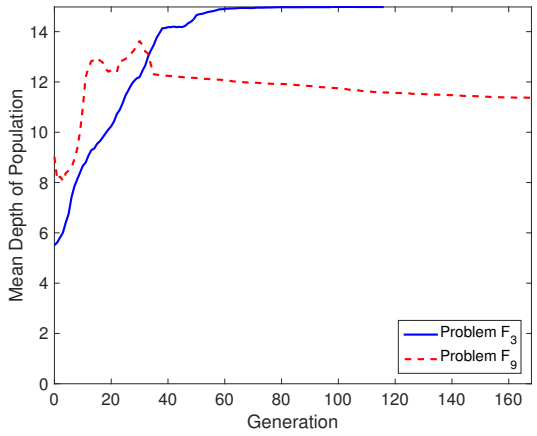


Fig. 2: Mean tree depths for median runs of $F_3$ and $F_9$

trees over generations indicates that the depths of the trees in general gradually increase. Hence if the search process is led at certain point to expressions with high accuracy and high tree sizes, it may not be possible to revert back to find the accurate forms of the equations any more. In that case, most of the final equations would be long and hard to interpret despite their good fitness.

The increase in depth also affects the efficiency of crossover and mutation operators. A node which is closer to the root usually has more influence on the fitness of the tree [17]. In a tree with high depth, the nodes close to the leaf nodes have a

higher chance to be picked up for crossover and mutation, so these operators may only change the fitness of the tree slightly. Consequently, the fitness of the trees in the final population has very small standard deviation. In other words, even though the genotype of the population is diverse, the phenotype/semantics is relatively similar. Running the SSR framework in multi-objective mode could provide improved results in both of the above cases, where the pressure towards producing smaller size trees prevents accurate compact equations from getting eliminated [5]. In next subsection, we present the results using SSR in multi-objective mode for implicit problems.

### *B. Multi-objective Mode*

MOGPA in [5] uses rudimentary checks to remove bloat and able to deal with relatively straightforward cases such as $(x_2 - x_2)$ or $(2-2)$ which are equivalent to 0. In contrast, SSR makes use of semantics hence it can detect the bloat in much more complex forms, such as $(((x_2 - x_1) + x_1) \times (1/x_2))$ which is equivalent to 1. SSR is further supported by better diversity owing to the uniqueness operator.

Realizing the difficulty in identifying the coefficients of the equations, MOGPA samples 100 values randomly in a given range and substitutes those values into the terminal constant nodes of a tree. Then, the fitness of the tree is reevaluated to determine which sampling values are most appropriate for this tree. However, in this scheme each sampling only returns a given number of discrete values, which may not capture the exact coefficients. In addition, if a tree has more than one constant node, more values need to be sampled for all the nodes, otherwise only 100 values may be too small to cover all the nodes. Our propose constant optimization scheme can overcome these drawbacks. The search process is performed on a continuous range and does not suffer if the number of terminal constant nodes increases.

Thirty independent runs are performed on each of the implicit equations in Table VI, taken from [5]. The parameters and settings for multi-objective SSR are listed in Table VII. The crossover and mutation methods used here are SAC and multi-point mutation respectively. To examine our proposed algorithm in much stricter conditions, a run is only considered successful if it can find the exact equation from the input data. Correct equations are the ones with the forms of $x_1^2 + x_2^2$ for $F_{11}$ and $x_1^3 - x_1 - x_2^2$ for $F_{12}$ and $F_{13}$. The last constants in $F_{11}$, $F_{12}$ and $F_{13}$ can be easily determined by substituting the found equations into one given fitness case as in [5]. In addition, the number of runs in which at least one tree has MLED less than $10^{-3}$ is also observed and compared with the result of MOGPA.

As can be seen in Table VIII, the result of SSR is competitive with that of MOGPA because 100% of runs can find equations with MLED less than $10^{-3}$. The number of function evaluations used by SSR are only half of those used for MOGPA, which demonstrates its relative efficiency. More importantly, the numbers of successful runs, where the exact equation is found in the final population is also 100% for each problem. The exact equations are found in very early generations for the problem $F_{11}$. The maximum number of generations (among all runs) used by SSR for finding the exact equation for $F_{12}$ and $F_{13}$ are 70 and 75 respectively. Some of the accurate equations found with different depths are reported in Table IX.

TABLE VI: 3 implicit equations

| Problems | Equations |
|---|---|
| $F_{11}$ | $x_1^2 + x_2^2 - 4 = 0$ |
| $F_{12}$ | $x_1^3 - x_1 - x_2^2 + 0.15 = 0$ |
| $F_{13}$ | $x_1^3 - x_1 - x_2^2 + 1.8750 = 0$ |

TABLE VII: Runs and evolutionary parameter values of multi-objective SSR

| Parameter | Value |
|---|---|
| Population size | 500 |
| Number of generations | 100 |
| Crossover probability | 1 |
| Mutation probability | 0.05 |
| Max depth | 15 |
| Operators | +, -, *, square |
| Terminals | $X_1$, $X_2$ and constant nodes with values in [0,10] |

TABLE VIII: Results of multi-objective SSR for 3 implicit problems

| | $F_{11}$ | $F_{12}$ | $F_{13}$ |
|---|---|---|---|
| Total number of runs | 30 | 30 | 30 |
| Number of successful runs | 30 | 30 | 30 |
| Number of runs having MLED $< 10^{-3}$ | 30 | 30 | 30 |

TABLE IX: Examples of accuracy equations returned by multi-objective SSR for 3 implicit problems

| | Equations | Depth |
|---|---|---|
| $F_{11}$ | $(squ(x_1) + (x_2 * x_2))$ | 3 |
| | $(((x_1 * x_1) + x_2) - ((4.422535 + x_2) - squ(x_2)))$ | 4 |
| | $((((x_1 * x_1) + 6.681197) + x_2) - (x_2 - squ(x_2)))$ | 5 |
| $F_{12}$ | $((x_1 * (x_1 * x_1)) - (x_1 + squ(x_2)))$ | 4 |
| | $(((x_1 * squ(x_1)) - squ(x_2)) - x_1)$ | 5 |
| | $((((x_1 * squ(x_1)) - squ(x_2)) + 4.661589) - x_1)$ | 6 |
| $F_{13}$ | $((squ(x_1) * x_1) - (squ(x_2) + x_1))$ | 4 |
| | $((squ(x_1) * x_1) - (squ(x_2) + ((x_1 - 6.164129) - 9.235206)))$ | 5 |
| | $((x_1 + (squ(x_2) - (squ(x_1) * x_1))) + 9.975000)$ | 6 |

One point needs to be taken note when implementing the MLED as in Equation 1. If a tree has only $x_1$ as variable, the MLED is undefined because $\delta x_{2,i}$ is always equal to 0. Because the roles of the two variables are the same, we set the MLED of a tree to be *Inf* in the cases either $x_1$ or $x_2$ is missing. As a result, among the trees with the same depths, those with both variables always dominate those with only one variable. This could potentially make the algorithm unable to identify redundant variables in the input data, which is one of the drawbacks of using MLED.

While observing the equations in the final population, it is noted that there are many groups of trees having same MLED and depths. These sets of trees may occupy several places in the population occasionally, thus reducing the diversity of the population. In order to maximize the chance to find out

the accurate implicit equations, we may consider to increase the minimum depth of the trees in the population. With some simple checks, it can be concluded that trees with depth of 2 do not form the correct equation– for example by constructing all trees of depth 2 (this number is relatively low). In such a case, these trees can be removed to make places for trees with bigger depths. Another approach that probably makes positive impact is incorporating $L_1$ value in the sorting process. When a set of trees have same MLED and depths, they can be ranked using their $L_1$ values. The $L_1$ values of the trees can be directly obtained from the summing the values in the semantics vector in the case of implicit equations, since the observed output is always 0.

### C. Further Discussion

As mentioned in previous sections, the semantics is calculated only based on the given fitness cases so that it won't take excessive time compared to the conventional GP. In our framework, the uniqueness operator takes up the most of the allowed time in each generation because it compares each child to the other in the children population and to all the parents. Evidently, as the population size increases, the run time will also increase with the maximum complexity of $O(2 * n^2)$, where $n$ is the population size.

The main focus of this work was to demonstrate that semantics can be used in various operators to improve the search results. In single objective case, the experiment results showed that applying semantics to more operators was beneficial in improving success rate compared to existing study which used crossover alone [4]. For bi-objective optimization, it was observed that semantics helped SSR to find exact equations more consistently compared to MOGPA [5].

## V. Summary and Future Work

In this paper, we present a semantics based symbolic regression (SSR) framework to mine explicit and implicit expressions from given data. SSR utilizes benefits of semantics for various components, such as compactness, uniqueness, crossover and local exploitation. In addition, a new constant optimization technique and multi-point mutation are used to strengthen the approach further. These enhancements help to deal with the persistent obstacles in SR, such as bloat, diversity and coefficient identification as well as enhance the exploration and exploitation abilities of the algorithm. Numerical experiments are conduced using both single and multi-objective forms of SSR. The results obtained are compared with existing techniques for both implicit and explicit functions to demonstrate the ability of SSR to find accurate, compact expressions in low computational budget.

For the future work, we would like to investigate an adaptive way to determine the values for SSC thresholds. SAC is used for the multi-objective experiments in previous section, since it is not clear how to determine an appropriate value of for the upper threshold if SSC is used. In addition, MLED is useful in solving the implicit equations but its detection ability for redundant variables is limited in some cases. Thus, looking for a better fitness function seems to be an interesting topic. One consideration may be the sum of weighted MLED and either $L_1$ or $L_2$. Besides, limiting the range of tree depth usually helps to find the more accurate forms of the tested equations. Thus, how to adaptively limit the tree depth is also worth investigating.

## Acknowledgment

The second author would like to acknowledge the support from the Ian Potter Foundation Travel Grant 2016.

## References

[1] G. Smits and M. Kotanchek, "Pareto-front exploitation in symbolic regression," in *Genetic Programming Theory and Practice II*, ser. Genetic Programming, 2005, vol. 8, pp. 283–299.

[2] J. W. Davidson, D. A. Savic, and G. A. Walters, "Method for the identification of explicit polynomial formulae for the friction in turbulent pipe flow," *Journal of Hydroinformatics*, vol. 1, no. 2, pp. 115–126, 1999.

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

[4] N. Uy, N. Hoai, M. ONeill, R. McKay, and E. Galvan-Lopez, "Semantically-based crossover in genetic programming: application to real-valued symbolic regression," *Genetic Programming and Evolvable Machines*, vol. 12, no. 2, pp. 91–119, June 2011.

[5] B. Wang, H. Singh, and T. Ray, "A multi-objective genetic programming approach to uncover explicit and implicit equations from data," in *IEEE Congress on Evolutionary Computation (CEC) 2015*, May 2015, pp. 1129–1136.

[6] M. Schmidt and H. Lipson, "Symbolic regression of implicit equations," in *Genetic Programming Theory and Practice VII*, ser. Genetic and Evolutionary Computation, October 2010, pp. 73–85.

[7] A. Ekart and S. Nemeth, "A metric for genetic programs and fitness sharing," in *Genetic Programming*, ser. Lecture Notes in Computer Science, 2000, vol. 1802, pp. 259–270.

[8] L. Vanneschi, M. Castelli, and S. Silva, "A survey of semantic methods in genetic programming," *Genetic Programming and Evolvable Machines*, vol. 15, no. 2, pp. 195–214, January 2014.

[9] A. Moraglio, K. Krawiec, and C. Johnson, "Geometric semantic genetic programming," in *Parallel Problem Solving from Nature - PPSN XII*, ser. Lecture Notes in Computer Science, 2012, vol. 7491, pp. 21–31.

[10] A. Kattan and Y.-S. Ong, "Surrogate genetic programming: A semantic aware evolutionary search," *Information Sciences*, vol. 296, pp. 345 – 359, 2015.

[11] N. Uy, N. Hoai, and M. ONeill, "Semantics based mutation in genetic programming: The case for real-valued symbolic regression," in *15th International Conference on Soft Computing, Mendel'09*, 2009.

[12] S. Bleuler, M. Brack, L. Thiele, and E. Zitzler, "Multiobjective genetic programming: reducing bloat using SPEA2," in *Proceedings of the 2001 Congress on Evolutionary Computation*, vol. 1, 2001, pp. 536–543.

[13] G. Dick, "Bloat and generalisation in symbolic regression," in *Simulated Evolution and Learning*, ser. Lecture Notes in Computer Science, 2014, vol. 8886, pp. 491–502.

[14] H. Zhao, "A multi-objective genetic programming approach to developing pareto optimal decision trees," *Decision Support Systems*, vol. 43, no. 3, pp. 809 – 826, 2007.

[15] M. Amir Haeri, M. Ebadzadeh, and G. Folino, "Statistical genetic programming: The role of diversity," in *Soft Computing in Industrial Applications*, ser. Advances in Intelligent Systems and Computing, 2014, vol. 223, pp. 37–48.

[16] E. Burke, S. Gustafson, and G. Kendall, "Diversity in genetic programming: an analysis of measures and correlation with fitness," *Evolutionary Computation, IEEE Transactions on*, vol. 8, no. 1, pp. 47–62, 2004.

[17] M. Affenzeller, S. Winkler, G. Kronberger, M. Kommenda, B. Burlacu, and S. Wagner, "Gaining deeper insights in symbolic regression," in *Genetic Programming Theory and Practice XI*, ser. Genetic and Evolutionary Computation, 2014, pp. 175–190.