Enhanced Symbolic Regression Through Local Variable Transformations

Jiří Kubalík[1], Erik Derner[1,2] and Robert Babuška[1,3]
[1]*Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague, Prague, Czech Republic*
[2]*Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, Prague, Czech Republic*
[3]*Cognitive Robotics, Faculty of 3mE, Delft University of Technology, Delft, The Netherlands*
{*jiri.kubalik, erik.derner*}*@cvut.cz, r.babuska@tudelft.nl*

9th International Joint Conference on Computational Intelligence (IJCCI 2017), `http://www.ijcci.org`

Keywords: Symbolic Regression, Single Node Genetic Programming, Nonlinear Regression, Data-driven Modeling

Abstract: Genetic programming (GP) is a technique widely used in a range of symbolic regression problems, in particular when there is no prior knowledge about the symbolic function sought. In this paper, we present a GP extension introducing a new concept of *local transformed variables*, based on a locally applied affine transformation of the original variables. This approach facilitates finding accurate parsimonious models. We have evaluated the proposed extension in the context of the Single Node Genetic Programming (SNGP) algorithm on synthetic as well as real-problem datasets. The results confirm our hypothesis that the transformed variables significantly improve the performance of the standard SNGP algorithm.

1 INTRODUCTION

Symbolic regression (SR) is a regression analysis method formulated as an inductive learning task. The main principle is to search in a predefined space of mathematical expressions for a model that fits the given data as accurately as possible. SR has been successfully used in nonlinear data-driven modeling or data mining, often with quite impressive results (Schmidt and Lipson, 2009; Vladislavleva et al., 2013; Staelens et al., 2013; Brauer, 2012). The main advantage of SR is that it generates parsimonious and human-understandable models. This is in contrast to other widely used nature-inspired algorithms, such as artificial neural networks, which are black box and do not offer any insight or interpretation of the underlying model.

Symbolic regression is usually solved by means of genetic programming (GP). On the one hand, this approach is suitable as there is generally no prior knowledge on the structure and shape of the symbolic function sought. On the other hand, the search space in SR is huge so when a pure GP is applied to a SR task, it needs a long time to find an acceptable solution. Besides the standard Koza's tree-based GP (Koza, 1992), many other variants have been proposed such as Grammatical Evolution, Gene Expression Programming, Cartesian GP or Single Node Genetic Programming. Grammatical Evolution (Ryan et al., 1998) evolves programs whose syntax is defined by a user-specified grammar. Gene Expression Programming (Ferreira, 2001) evolves linear chromosomes that are expressed as tree structures through a genotype-phenotype mapping. Cartesian GP (CGP) (Miller and Thomson, 2000) uses the genotype-phenotype mapping to express linear chromosomes as programs in the form of a directed graph. Single Node Genetic Programming (SNGP) (Jackson, 2012a; Jackson, 2012b) is a GP technique evolving a population organized as an ordered linear array of interlinked individuals, each representing a single program node.

It has been widely reported in the literature that evolutionary algorithms work much better when hybridized with local search techniques or other means of final solution optimization. An example is the memetic algorithm (Hart et al., 2005), where the local search is used to fine-tune the candidate solutions along the whole evolution process. A similar approach can be used to develop efficient GP-based methods for symbolic regression. Recently, several evolutionary SR methods emerged that explicitly restrict the class of generated models to generalized linear models, i.e., models formed as a linear combination of non-linear basis functions, also called features. Examples of these methods are Evolutionary Feature Synthesis (EFS) (Arnaldo et al., 2015), Multi-Gene Genetic Programming (MGGP)

(Hinchliffe et al., 1996; Searson et al., 2010), Fast Function Extraction (FFX) (McConaghy, 2011), and Single-Run SNGP with LASSO (s-SNGPL) (Kubalík et al., 2016). Candidate basis functions are generated through a standard evolutionary process, while the coefficients of the basis functions are found by using a multiple regression technique. In this way, accurate linear-in-parameters nonlinear models can efficiently be evolved.

In this paper, we go one step further towards robust GP methods for SR. We introduce *local transformed variables*, which are coordinate transformations introduced in the individual nonlinear functions. The motivation for using transformed variables is that in many cases a good symbolic model is hard to produce with features in the original coordinate system. The search process for an acceptable model can be much more efficient if some of its components use transformed variables.

We chose a variant of Single Node Genetic Programming (Jackson, 2012a; Jackson, 2012b) for a proof-of-concept implementation of the transformed variables into GP. It has been shown in (Kubalík et al., 2016) that SNGP is good at solving the SR problem. The concept of transformed variables is general and can be implemented with other types of GP as well. In order to facilitate operations with constants, which is crucial for a proper functioning of the transformed variables, we used the concept of *partitioned population*, similar to the one introduced in (Alibekov et al., 2016). The partitioned population is organized so that it contains a continuous segment of nodes producing only constant output. These constant-output nodes are used in and evolved simultaneously with other expressions in the population.

We evaluated the proposed extension on synthetic as well as real-problem datasets with the number of dimensions ranging from 2 to 4.

The paper is organized as follows. Section 2 describes the standard Single Node Genetic Programming algorithm. The novel concept of transformed variables is introduced in Section 3. Section 4 presents the experiments performed in order to compare the algorithm variants. Finally, the achieved improvement of the proposed algorithm extension is discussed in Section 5.

2 STANDARD SNGP

Single Node Genetic Programming is a graph-based GP technique that evolves a population organized as an ordered linear array of individuals, each representing a single program node. Program nodes can be of various types depending on the particular problem solved. In the context of SR the program node can either be a terminal, i.e. a constant or a variable, or some operator or function chosen from a set of functions defined for the problem at hand. The individuals are interconnected in the left-to-right manner, meaning that an individual can act as an input operand only of those individuals which are positioned to the right of it in the population. Thus, the whole population represents a graph structure similar to that of the CGP with multiple expressions rooted in individual nodes. The population is evolved through a first-improvement local search procedure using a single reversible mutation operator.

Formally, the SNGP population is a set $M = \{m_0, m_1, \ldots, m_{L-1}\}$ of L individuals, with each individual m_i being a single node represented by the tuple $m_i = \langle e_i, Succ_i, Pred_i, o_i, f_i \rangle$, where

- $e_i \in T \cup F$ is either an element chosen from a function set F or a terminal set T defined for the problem;
- $Succ_i$ is a set of successors of this node, i.e. the nodes whose output serves as the input to this node;
- $Pred_i$ is a set of predecessors of this node, i.e. the nodes that use this node as an input operand;
- o_i is a vector of its outputs;
- f_i is the individual's fitness.

The population is organized as shown in Figure 1, starting with constants and variables, followed by function nodes.

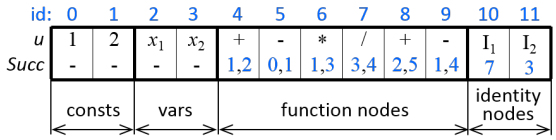

Figure 1: Structure of the standard SNGP population.

Links between the nodes in the population must satisfy the condition that any function node can use as its successor (i.e. the operand) only nodes that are positioned lower down in the population. Similarly, predecessors of individual i must occupy higher positions in the population. Note that each function node is in fact a root of an expression that can be constructed by recursively traversing its successors towards the leaf terminal nodes.

The fitness of each individual (i.e. of each expression) is for SR problems typically calculated as the root mean squared error observed on the set of training samples.

A simple evolutionary operator called *successor mutation* (*smut*) is used to modify the population. It picks one individual of the population at random and then replaces one of its successors by a reference to another individual chosen among all individuals positioned to the left of it in the population.

The population is evolved using a local search procedure. In each iteration, a new population is derived from the current one by the *smut* operator. The new population is then accepted for the next iteration if it satisfies the chosen acceptance rule (typically, if its performance is not worse than the performance of the original population). Otherwise the original population remains for the next iteration.

The baseline SNGP implementation used in this work differs from the one described in (Jackson, 2012a; Jackson, 2012b) in several aspects described in the following paragraphs.

Form of regression models. A hybrid SNGP denoted as the Single-Run SNGP with LASSO proposed in (Kubalík et al., 2016) is used in this work. It produces generalized linear regression models. In particular, candidate features, i.e. expressions rooted in individual nodes, are combined into generalized linear regression model using some multiple regression technique. In this way, precise linear-in-parameters nonlinear regression models can efficiently be produced.

Original s-SNGPL uses the Least Absolute Shrinkage and Selection (LASSO) regression technique to generate models of limited complexity, i.e. models composed of a limited number of features. In particular, all features in the population with a non-constant output are eligible candidates for the final model. Though, only some of the candidate features are used in the final model returned by the LASSO procedure in the end. An advantage of this LASSO approach is that a suitable subset of features used in the model is chosen automatically. On the other hand, the LASSO procedure is more computationally expensive than a simple least squares regression. Here, instead of the LASSO regression operating with large set of features, we use the simple least squares regression explicitly operating on a fixed number of features. The complexity of the resulting regression models is controlled by (i) the maximum depth of features evolved in the population, d, and (ii) the maximum number of features the models can be composed of, n_f.

In order to have a possibility to identify the nodes to be used as features in the linear regression model, a special type of node called *identity node* is introduced. This node has a single successor (i.e. an input), which is a link to another node in the population. The identity node then returns outputs of the node it refers to. The identity nodes are positioned at the end of the population, see Figure 1. The number of identity nodes n_i is equal to the maximum number of features to be used in the generated models. By changing the successors of the identity nodes, the set of features for the generated regression models can be altered. This is realized in two ways. The first one is by the standard mutation operator during the evolution. The second one is through a local search procedure at line 21 in Algorithm 1. It is launched in the end of each epoch and runs for a predefined number of iterations, l_i. In each iteration, one identity node is randomly chosen and its successor reference is changed to a new node randomly chosen from the set of tail partition nodes. Such a modified population is evaluated, meaning its linear regression model is constructed using the modified set of identity nodes (i.e. the features). If the new model's fitness is not worse than the fitness of the original population model, the modification made to the identity node is accepted. Otherwise, the original setting of the identity node remains for the next iteration.

Fitness. Here, the fitness represents a quality of the whole generalized linear regression model calculated as the root mean squared error observed on the set of training samples. In addition, we also assess a quality of each single node in the population. This is denoted as the *node's utility* and it is calculated as the Pearson product-moment correlation coefficient between the node's output and the desired output over the set of training samples.

Selection strategy. In the original SNGP, nodes to be mutated are selected purely at random. Here we use a variant of tournament selection (see line 12 in Algorithm 1). It operates with utility values assigned to the nodes during the population evaluation step at line 3 and 15, respectively. When selecting the node to be mutated, t candidate individuals are randomly picked from the population at first. Then, out of the candidates the one that is involved in the best-performing expression with respect to the root node utility value is chosen. In this way, not only nodes that have high utility values themselves are being favoured. Instead, all nodes at least *contributing* to high-quality expressions are preferred to the ones not playing an important role in the population (i.e. the nodes not involved in any well-performing expression).

Evolutionary model. The process of evolving the population is carried out in epochs. In each epoch, multiple independent parallel threads are run for a predefined number of generations, all of them starting from the same population, which is the best final

population out of the previous epoch threads. This reduces the chance of getting stuck in a local optimum. The model is parameterized by the number of threads n_t, the epoch length l_e, and the number of epochs n_e.

3 PROPOSED EXTENSIONS

In this section, we describe the concept of *transformed variables*, introduced in order to make the SNGP algorithm more efficient at solving the SR problem.

3.1 Transformed Variables

The transformed variables are obtained by local affine transformations of the coordinate system. Assume n-dimensional input space $\mathcal{X} \subset \mathbb{R}^n$. Each transformed variable v is represented by the tuple

$$v = \langle w_0, c_0, x_0, \ldots, w_n, c_n, x_n \rangle$$

where c_j is a pointer to a constant-valued node in the population, $w_j \in \mathbb{R}$ is a real-valued constant, x_j is the jth original variable, with $x_0 = 1$ corresponding to the offset of the transformation. The constant-valued nodes referred to by the c_j pointers are either basic constant nodes (i.e. nodes from the first section of the partitioned population in Figure 2) or head partition nodes. The value of the transformed variable v is calculated as

$$v = \sum_{j=0}^{n} b_j x_j,$$

where $b_j = w_j o_j$ with o_j the output of the node referenced by c_j. The weighting coefficients are formed as $b_j = w_j o_j$ in order to allow for both global and local tuning of their values. In particular, changing the reference c_j to a different constant-valued node is likely to lead to a significant change of the corresponding b_j. On the contrary, fine-tuning of the coefficient b_j can be achieved by applying small changes to w_j. The values of w_j are initialized to 1. The links c_j are tuned by means of the standard mutation operator.

Both c_j and w_j can further be tuned through a local search procedure, see line 22 in Algorithm 1. It is launched in the end of each epoch and runs for a predefined number of iterations, l_v. In each iteration, one transformed variable v_k is randomly chosen and either its reference to the constant-valued node c_k or the value of w_k is changed with equal probability. The value of w_k is modified by adding a random value drawn from a normal distribution as follows:

$$w_k = w_k + N(0,1).$$

The population with the modified node v_k is evaluated and if its fitness is not worse than the fitness of the original population, the modification made to the node v_k is accepted. Otherwise, the original setting of the transformed variable node remains for the next iteration.

3.2 Partitioned Population

To support the concept of transformed variables, the population is organized so that the function nodes are further divided into two continuous segments, the *head* and *tail* partitions as introduced in (Alibekov et al., 2016). The head partition nodes are allowed to represent root nodes of expressions producing only constant output. The tail nodes can be roots of both the constant-output as well as variable-output expressions. The head partition therefore represents a pool of constants that are used in and evolved simultaneously with other features in the population.

The number of transformed variables n_v and the number of head partition nodes n_h are user-defined parameters. Note that besides the transformed variables the population always contains also all the original variables $x \in \{x_1, \ldots, x_n\}$. The structure of the partitioned population is shown in Figure 2.

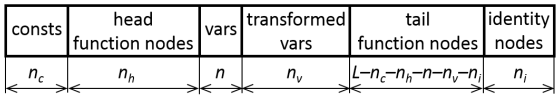

Figure 2: Structure of partitioned population.

3.3 Algorithm SNGP-TV

The whole algorithm, denoted here as SNGP with transformed variables (SNGP-TV), is outlined in Algorithm 1. It iteratively evolves current best population of individuals P and the corresponding model M. The main loop, lines 5–29, iterates through epochs. In each epoch, multiple independent threads are run, lines 7–24, all of them started with the same population P, line 8. Each thread runs for a predefined number of generations, lines 11–20. In the end of each thread, the identity nodes and transformed variables are tuned for a specified number of trials, lines 21–22. The best final population and its model are saved in P and M, lines 25–27, and P becomes the starting population for next epoch. Finally, the last version of model M is returned as the solution.

Algorithm 1: SNGP with transformed variables.

Input: training data set, L, n_c, n_h, n, n_v, n_f, d, l_i, l_v, l_e, n_e, n_t

initialize population P
build regression model M using P
evaluate M
$e \leftarrow 0$
do
  $t \leftarrow 0$
  do
    $P_t \leftarrow P$
    $M_t \leftarrow M$
    $generation \leftarrow 0$
    do
      $s \leftarrow selectNode(P_t)$
      $P'_t \leftarrow mutate(P_t, s)$
      build model M'_t using P'_t
      evaluate M'_t
      if (M'_t is not worse than M_t)
        $P_t \leftarrow P'_t$
        $M_t \leftarrow M'_t$
      $generation \leftarrow generation + 1$
    while ($generation < l_e$)
    $[P_t, M_t] \leftarrow optIdentityNodes(P_t, l_i)$
    $[P_t, M_t] \leftarrow optTransfVars(P_t, l_v)$
    $t \leftarrow t + 1$
  while ($t < n_t$)
  $b \leftarrow \underset{t=1,\ldots,n_t}{argbest}(M_t)$
  $P \leftarrow P_b$
  $M \leftarrow M_b$
  $e \leftarrow e + 1$
while ($e < n_e$)
return M

Output: symbolic model M

4 EXPERIMENTS

4.1 Test Datasets

For the proof-of-concept experiments we used thirteen datasets, featuring both synthetic and real data, with the number of dimensions ranging from 2 to 4. Figure 3 shows mesh plots of all 2-dimensional datasets used.

To demonstrate how the transformed variables facilitate finding models of functions in a transformed space, we used the synthetic datasets *sig0*, *sigπ/8* and *sigπ/4* shown in Figures 3(a) through 3(c). The dataset *sig0* was calculated using a sigmoid-like function, defined by Equation 1, sampled on a regular grid of 31×31 points for $x_1, x_2 \in [-10, 10]$.

$$f_1(\mathbf{x}) = 0.1x_1 + \frac{2}{1+e^{-x_1}} \tag{1}$$

Note that x_2 is not used in the equation and therefore, the values of the function do not change along the x_2 dimension. The function was then rotated by $\pi/8$ rad (dataset *sigπ/8*) and $\pi/4$ rad (dataset *sigπ/4*) in the counterclockwise direction in order to simulate the transformation of the coordinate system.

Other two synthetic datasets *f2* and *f3* (Figures 3(d) and 3(e)) were generated using the following functions:

$$f_2(\mathbf{x}) = x_1^2 \cos(10x_1 - 15x_2), \tag{2}$$

$$\begin{aligned} f_3(\mathbf{x}) = {} & (x_1-1)^2 \cos(10(x_1-1) - 15(x_2-1)) \\ & - (x_2-1)^2 \sin(10(x_1-1) + 15(x_2-1)), \end{aligned} \tag{3}$$

sampled on a regular grid of 31×31 points in the interval $x_1, x_2 \in [0, 1]$.

The following datasets come from the domain of reinforcement learning applications.

The *1dof_dir* dataset (Figure 3(f)) consists of samples of the value function approximator for a 1-DOF inverted pendulum swingup (Adam et al., 2012). The pendulum moves in a state space defined by variables $\alpha \in [-\pi, \pi]$, $\dot{\alpha} \in [-40, 40]$, where α denotes the angle and $\dot{\alpha}$ is the angular velocity of the pendulum. The pendulum is pointing up for $\alpha = 0$. The 961 data points were sampled on a regular grid with 31 samples in each dimension. The *1dof_inv* dataset (Figure 3(g)) is a variant of the 1-DOF swingup problem, which differs in the interpretation of the angle α. The pendulum is pointing down for $\alpha = 0$, whereas both for $\alpha = -\pi$ and $\alpha = \pi$, the pendulum is pointing up.

The *magman* dataset (Figure 3(h)) is based on a magnetic manipulation (Hurák and Zemánek, 2012) problem. The magnetic manipulation setup consists of two electromagnets in a line positioned at $a = 0\,\mathrm{m}$ and $a = 0.025\,\mathrm{m}$. The dataset is represented by samples of the value function approximator on the domain $x_1 \in [0, 0.05]$, $x_2 \in [-0.4, 0.4]$, where $x_1 = a$ is the position and $x_2 = \dot{a}$ is the velocity. The data are sampled on a regular grid of 31×31 points.

The *policy* dataset represents samples of the policy function for the direct inverted pendulum task, described above. The policy function was sampled on the range $x_1 \in [-\pi, \pi]$, $x_2 \in [-40, 40]$. From the whole state space, only the samples in the transition region were used and the plateaus were removed from the dataset, see Figure 3(i). In order to retain the number of training data points comparable with the rest of the datasets used in the experiments, these data were sampled on a regular grid with a higher resolution of

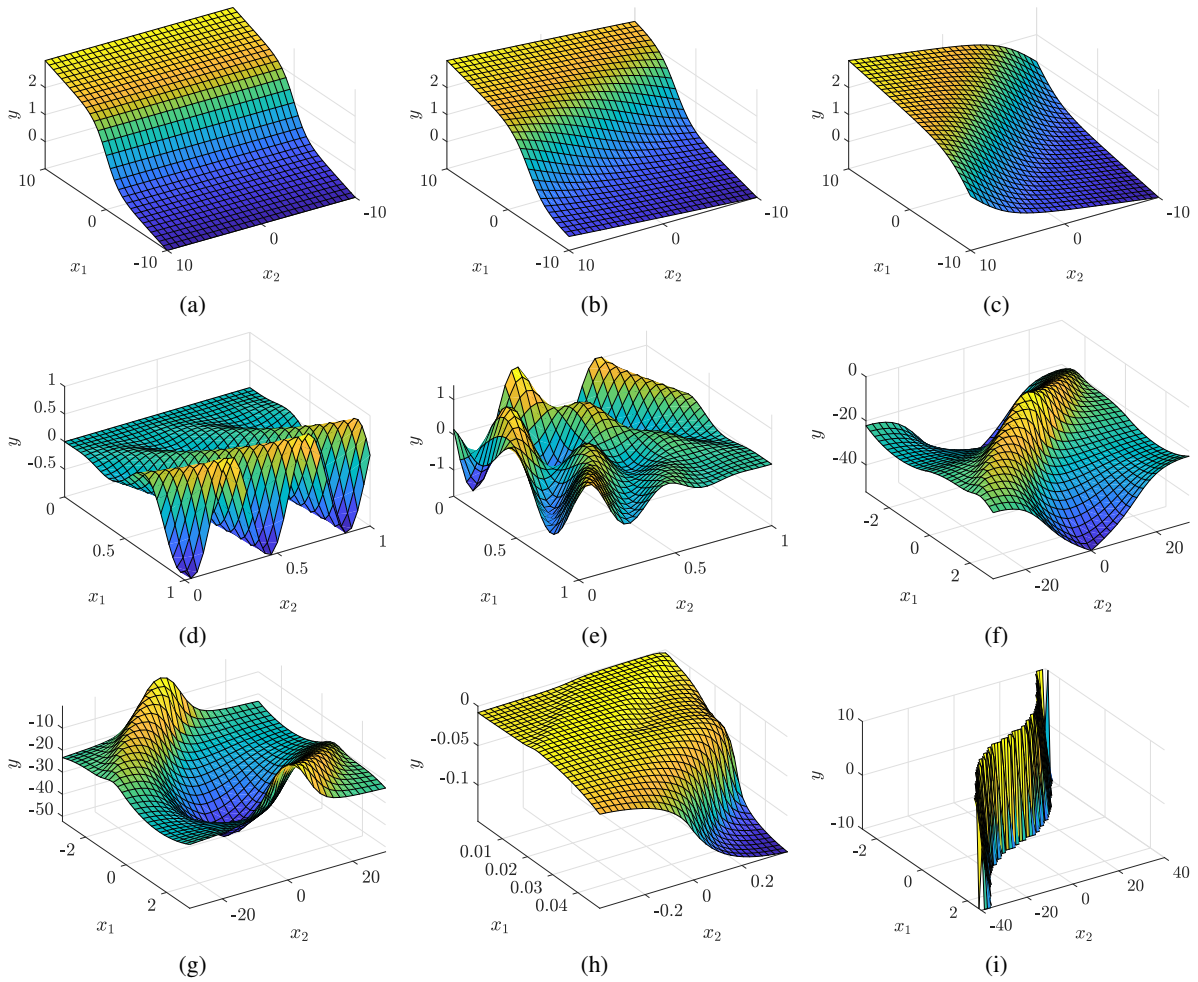

Figure 3: Mesh plots of the datasets. The intersections of the grid represent the data used as the input for the SNGP algorithm and its variants. The following datasets were used in the experiments: from (a) to (c) – synthetic sigmoid-like function described by Equation 1, rotated by (a) 0 rad, (b) $\pi/8$ rad and (c) $\pi/4$ rad; (d), (e) – synthetic functions described by Equation 2 and 3, respectively; (f), (g) – variants of inverted pendulum, (h) – magnetic manipulation; (i) – policy function for inverted pendulum (transition region only).

61×61 points. The *policy* dataset comprises 313 data points.

The last dataset related to the reinforcement learning, denoted *2dof*, is derived from the value function for an inverted pendulum with two links. It is a 4-dimensional problem with the angle α_i and angular velocity $\dot{\alpha}_i$ describing the state of each link $i \in \{1,2\}$. The dataset consists of 6561 data points sampled on a regular grid with nine samples evenly distributed in each dimension.

4.2 Experiment Setup

We experimentally evaluated the proposed **SNGP-TV** algorithm and compared it to two other variants of SNGP – namely the SNGP with simple population (see Figure 1) denoted as **SNGP-SP** and the SNGP with partitioned population (see Figure 2) but not using the transformed variables, denoted as **SNGP-PP**. Note, both the variants can be described by the Algorithm 1 if the population type and control parameters are set accordingly. The SNGP-SP and SNGP-PP differ in the population type while none of them uses the transformed variables, so $n_v = 0$ and $l_v = 0$. The SNGP-TV was tested with the number of transformed variables being equal to the number of the original variables, i.e. $n_v = n$.

Other control parameters were set as follows:

- population size: $L = 500$
- total number of fitness evaluations: 20.000
- epochs: $n_e = 20$, $n_t = 2$
- trivial constants: $C = \{0, 0.5, 1, 2, 3, 4\}$, $n_c = 6$
- original variables: $n = 2$
- maximum tree depth: $d = 7$
- basic functions:
  $F^1 = \{*, +, -, square, cube, BentIdentity^1\}$,
  $F^2 = \{*, +, -, square, cube, Sigmoid\}$
- maximum number of features: $n_f^1 = 10$, $n_f^2 = 3$
- SNGP-SP: $l_e = 650$, $l_i = 150$
- SNGP-PP: $l_e = 650$, $l_i = 150$, $n_h = 121$
- SNGP-TV: $l_e = 500$, $l_i = 150$, $l_v = 150$, $n_h = 120$

We carried out three experiment series:

1. **experiment-1** – all compared algorithms were run on all datasets with function set F^1 and the maximum number of features $n_f^1 = 10$.
2. **experiment-2** – all compared algorithms were run on all datasets with function set F^1 and the maximum number of features $n_f^2 = 3$.
3. **experiment-3** – algorithms SNGP-PP and SNGP-TV were run on synthetic datasets *sig*π*0*, *sig*π*/8*, and *sig*π*/4* with function set F^2 and the maximum number of features $n_f^2 = 3$.

The first two experiments were run to support our hypothesis that the use of the transformed variables in GP can be beneficial. The two experiments differ in the complexity of the evolved general regression models. Our hypothesis is that when more complex models (in terms of the maximum number of evolved features combined in the linear regression models) are allowed, then even less effective algorithms (i.e. the ones not using the transformed variables) can be able to construct a good model. On the other hand, when less complex models are allowed, the advantage of using the transformed variables should prove more significant. Experiment-3 is proposed to illustrate that when the population contains some feature or its part that might well describe the data if its input was properly transformed, then this can efficiently be achieved with the transformed variables. In this case the key element of the model sought is the sigmoid function that was added to the set of basic functions in place of the *BentIdentity* function.

Each experiment was replicated 30 times with the root mean squared error (RMSE) calculated for each single run. The median RMSE over the 30 runs is used as the performance measure in the tables with results. The Wilcoxon rank sum test (also known as the Mann-Whitney U-test) was used to evaluate statistical significance of the difference in performance between two algorithms.

Besides the RMSE, the complexity of the produced models was analyzed as well. In general, the complexity of a model was calculated as the total number of nodes in the model. More precisely, the model complexity equals the sum of the complexities of the features that the model is composed of, plus the operators and weights that are used to join the features into one expression (the whole regression model). The following assumptions were made for the calculation of a complexity of the SNGP-PP and SNGP-TV models. In the case of SNGP-PP and SNGP-TV, each constant-valued tree rooted in a head partition node was treated as a single constant-valued node. Thus, if some tail partition function node takes a head partition node as its input, then this head partition node is counted as a single node, even though it is effectively represented by a more complex tree structure. The rationale behind this assumption is that the head partition was introduced into the population

[1] Bent identity is a non-linear, unbounded, monotonic function that approximates identity near the origin, see `http://www.wow.com/wiki/Activation_function`.

in order to facilitate a production of constant-valued expressions that act as single constants in the final models. Similarly, each transformed variable node in SNGP-TV is treated as a single variable node, even though it is effectively represented by a more complex structure, see Section 3.1.

Note, we restrict our experiments to compare only the SNGP variants – SNGP with and without the transformed variables. We do not include any other GP method into the analysis since we simply want to demonstrate that if the transformed variables are used in some GP method for solving the symbolic regression problem, then it leads to an improved performance of that method. We did not tune control parameters of the compared SNGP algorithms either since the purpose of the experiments was not to present the best possible performance of the algorithms on the test data sets. Again, we just wanted to show the difference between using and not using the transformed variables with the other control parameters set the same for both variants. It is very likely that if we tuned the control parameters of the SNGP, the performance of the compared variants, both the one with and the one without the transformed variables, would improve.

4.3 Results

Results for experiment-1 are summarized in Table 1, Table 2 and Table 3. Table 1 presents the median RMSE values. Table 2 shows which algorithms outperform the other algorithms. A given algorithm outperforms another algorithm on a given dataset if both of the following two conditions are met:

- its RMSE median is smaller than the RMSE median of the compared algorithm,
- the Wilcoxon rank sum test rejects the null hypothesis that the RMS errors in the 30 runs for each of the two algorithms are samples from distributions with equal medians, at the statistical significance level of 1 % and 5 %, respectively.

In the case of 1 % significance level, the null hypothesis for the pair of SNGP-PP and SNGP-TV algorithms was not rejected on 3 datasets. When the significance level was increased to 5 %, the null hypothesis was not rejected on 2 datasets. Table 3 presents the median complexity of models produced by the three compared variants. It shows that the complexity of SNGP-TV models, as defined in Section 4.2, is in most cases less than the complexity of the SNGP-SP and SNGP-PP models.

Results for the experiment-2 are summarized in Table 4. Here, the null hypothesis was rejected at the 1 % significance level for all datasets with the exception for the *policy* dataset. When the significance level was increased to 5 %, the null hypothesis was rejected for all datasets. Thus, the improvement provided by the transformed variables is even more significant when smaller models are evolved as demonstrated by the increased number of datasets on which SNGP-TV proved to be significantly better than SNGP-PP compared to the results observed in experiment-1. Similarly as in the experiment-1, the complexity of SNGP-TV models is in most cases less than the complexity of the SNGP-PP models, see Table 5.

Results for the experiment-3 are summarized in Table 6. Both SNGP-PP and SNGP-TV variants perform equally well on the unrotated dataset *sig0* as the null hypothesis was rejected neither at the 1 % nor 5 % significance level. This is in accordance with our expectation that this dataset is easy for the algorithms

Table 1: Medians of RMSE for all tested algorithms and datasets in experiment-1.

Dataset	**SNGP variant**			**Scale**
	SP	**PP**	**TV**	
sig0	8.75	8.51	6.98	$\times 10^{-4}$
sigπ/8	5.42	5.45	0.46	$\times 10^{-2}$
sigπ/4	1.64	2.85	0.26	$\times 10^{-2}$
f2	2.76	2.64	1.59	$\times 10^{-1}$
f3	4.06	4.00	3.26	$\times 10^{-1}$
1dof_dir	2.82	2.82	1.34	$\times 10^{0}$
1dof_inv	2.50	2.20	2.22	$\times 10^{0}$
magman	8.54	7.51	3.55	$\times 10^{-3}$
policy	1.88	1.88	1.84	$\times 10^{0}$
2dof	5.24	5.24	4.66	$\times 10^{1}$

Table 2: Comparison of performance of algorithms in experiment-1. The table shows which algorithms are outperformed by the algorithm in the column heading for each dataset. The performance is compared using the Wilcoxon rank sum test with a significance level of 1 %. The entry with an asterisk would be included if the significance level was increased to 5 %.

Dataset	**SNGP variant**		
	SP	**PP**	**TV**
sig0			
sigπ/8			SP, PP
sigπ/4			SP, PP
f2			SP, PP
f3			SP, PP
1dof_dir			SP, PP
1dof_inv		SP	SP
magman		SP	SP, PP
policy			SP, *PP**
2dof			SP, PP

Table 3: Medians of complexity for all tested algorithms and datasets in experiment-1.

Dataset	SNGP variant		
	SP	PP	TV
sig0	108	106	100
sigπ/8	119	121	104
sigπ/4	119	117	102
f2	137	144	125
f3	128	133	119
1dof_dir	122	122	106
1dof_inv	126	119	112
magman	132	144	122
policy	118	118	119
2dof	103	106	106

Table 4: Medians of RMSE obtained with SNGP-PP and SNGP-TV in experiment-2.

Dataset	SNGP variant		Scale
	PP	TV	
sig0	4.07	2.41	$\times 10^{-2}$
sigπ/8	1.43	0.72	$\times 10^{-1}$
sigπ/4	1.45	0.42	$\times 10^{-1}$
f2	2.98	2.74	$\times 10^{-1}$
f3	4.31	4.17	$\times 10^{-1}$
1dof_dir	5.20	4.03	$\times 10^{0}$
1dof_inv	5.00	4.31	$\times 10^{0}$
magman	1.04	0.73	$\times 10^{-2}$
policy	3.21	2.92	$\times 10^{0}$
2dof	8.40	7.01	$\times 10^{1}$

since they only need to generate and combine simple linear and sigmoid features with a single input x_1. On dataset *sigπ/4*, a significant difference in performance between SNGP-PP and SNGP-TV can already be observed. However, even SNGP-PP was able to find a solution with RMSE lower than 6.5×10^{-10} in

Table 5: Medians of complexity obtained with SNGP-PP and SNGP-TV in experiment-2.

Dataset	SNGP variant	
	PP	TV
sig0	32	32
sigπ/8	32	37
sigπ/4	37	35
f2	57	47
f3	49	49
1dof_dir	40	36
1dof_inv	40	35
magman	53	42
policy	41	39
2dof	39	38

Table 6: Medians of RMSE obtained with SNGP-PP and SNGP-TV in experiment-3.

Dataset	SNGP variant		Scale
	PP	TV	
sig0	6.26	6.75	$\times 10^{-10}$
sigπ/8	9.61	0.17	$\times 10^{-2}$
sigπ/4	3.72	0.13	$\times 10^{-2}$

Table 7: Medians of complexity obtained with SNGP-PP and SNGP-TV in experiment-3.

Dataset	SNGP variant	
	PP	TV
sig0	24	22
sigπ/8	38	25
sigπ/4	33	29

10 out of 30 runs. These results correspond to models that employ a sigmoid feature with a simple argument $(x_1 + x_2)$. A similar trend of SNGP-TV outperforming SNGP-PP was observed on the *sigπ/8* dataset where SNGP-TV achieves by more than one order of magnitude better RMSE than SNGP-PP. Moreover, the best solution obtained with SNGP-PP had an RMSE of 7.7×10^{-5}, while SNGP-TV generated five solutions with RMSE lower than 6.5×10^{-10}. Again, the complexity of SNGP-TV models is lower than the complexity of the SNGP-PP models, see Table 7. The most significant difference being observed on the *sigπ/8* dataset, which is the one with the most "difficult" transformation of the input space.

5 CONCLUSIONS

The concept of local transformed variables for symbolic regression was introduced, so that the genetic programming search for an accurate model is more efficient. We have integrated transformed variables into the Single Node Genetic Programming algorithm, using a partitioned population with certain nodes explicitly devoted to producing constant-output expressions.

The experimental results show that the proposed SNGP with transformed variables outperforms the baseline SNGP algorithm on the majority of test datasets. We found that the transformed variables are especially useful when parsimonious models are sought.

Our future research will focus on incorporating model complexity control into the algorithm, which may further boost the performance of the approach.

6 Acknowledgment

This research was supported by the Grant Agency of the Czech Republic (GAČR) with the grant no. 15-22731S titled "Symbolic Regression for Reinforcement Learning in Continuous Spaces" and by the European Regional Development Fund under the project Robotics 4 Industry 4.0 (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000470).

REFERENCES

Adam, S., Buşoniu, L., and Babuška, R. (2012). Experience replay for real-time reinforcement learning control. *IEEE Transactions on Systems, Man, and Cybernetics. Part C: Applications and Reviews*, 42(2):201–212.

Alibekov, E., Kubalík, J., and Babuška, R. (2016). Symbolic method for deriving policy in reinforcement learning. In *Proceedings of the 55th IEEE Conference on Decision and Control (CDC)*, pages 2789–2795, Las Vegas, USA.

Arnaldo, I., O'Reilly, U.-M., and Veeramachaneni, K. (2015). Building predictive models via feature synthesis. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, pages 983–990, New York, NY, USA. ACM.

Brauer, C. (2012). Using Eureqa in a Stock Day-Trading Application. Cypress Point Technologies, LLC.

Ferreira, C. (2001). Gene expression programming: a new adaptive algorithm for solving problems. *Complex Systems*, 13(2):87–129.

Hart, W. E., Smith, J. E., and Natalio, K. (2005). *Recent Advances in Memetic Algorithms*. Springer, Berlin, Heidelberg.

Hinchliffe, M., Hiden, H., McKay, B., Willis, M., Tham, M., and Barton, G. (1996). Modelling chemical process systems using a multi-gene genetic programming algorithm. In Koza, J. R., editor, *Late Breaking Papers at the Genetic Programming 1996 Conference Stanford University July 28-31, 1996*, pages 56–65, Stanford University, CA, USA. Stanford Bookstore.

Hurák, Z. and Zemánek, J. (2012). Feedback linearization approach to distributed feedback manipulation. In *American control conference*, pages 991–996, Montreal, Canada.

Jackson, D. (2012a). A new, node-focused model for genetic programming. In Moraglio, A., Silva, S., Krawiec, K., Machado, P., and Cotta, C., editors, *Genetic Programming: 15th European Conference, EuroGP 2012, Málaga, Spain, April 11-13, 2012. Proceedings*, pages 49–60. Springer, Berlin, Heidelberg.

Jackson, D. (2012b). Single node genetic programming on problems with side effects. In Coello, C. A. C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., and Pavone, M., editors, *Parallel Problem Solving from Nature - PPSN XII: 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part I*, pages 327–336. Springer, Berlin, Heidelberg.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. MIT Press Ltd.

Kubalík, J., Alibekov, E., Žegklitz, J., and Babuška, R. (2016). Hybrid single node genetic programming for symbolic regression. In Nguyen, N. T., Kowalczyk, R., and Filipe, J., editors, *Transactions on Computational Collective Intelligence XXIV*, pages 61–82. Springer, Berlin, Heidelberg.

McConaghy, T. (2011). FFX: Fast, scalable, deterministic symbolic regression technology. In Riolo, R., Vladislavleva, E., and Moore, J. H., editors, *Genetic Programming Theory and Practice IX*, pages 235–260. Springer New York, New York, NY.

Miller, J. F. and Thomson, P. (2000). Cartesian genetic programming. In Poli, R., Banzhaf, W., Langdon, W. B., Miller, J., Nordin, P., and Fogarty, T. C., editors, *Genetic Programming: European Conference, EuroGP 2000, Edinburgh, Scotland, UK, April 15-16, 2000. Proceedings*, pages 121–132. Springer, Berlin, Heidelberg.

Ryan, C., Collins, J., and Neill, M. O. (1998). Grammatical evolution: Evolving programs for an arbitrary language. In Banzhaf, W., Poli, R., Schoenauer, M., and Fogarty, T. C., editors, *Genetic Programming: First European Workshop, EuroGP'98 Paris, France, April 14–15, 1998 Proceedings*, pages 83–96. Springer, Berlin, Heidelberg.

Schmidt, M. and Lipson, H. (2009). Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85.

Searson, D. P., Leahy, D. E., and Willis, M. J. (2010). GPTIPS : An open source genetic programming toolbox for multigene symbolic regression. In *Proceedings of the International Multiconference of Engineers and Computer Scientists 2010 (IMECS 2010)*, volume 1, pages 77–80, Hong Kong.

Staelens, N., Deschrijver, D., Vladislavleva, E., Vermeulen, B., Dhaene, T., and Demeester, P. (2013). Constructing a no-reference H.264/AVC bitstream-based video quality metric using genetic programming-based symbolic regression. *IEEE Transactions on Circuits and Systems for Video Technology*, 23(8):1322–1333.

Vladislavleva, E., Friedrich, T., Neumann, F., and Wagner, M. (2013). Predicting the energy output of wind farms based on weather data: Important variables and their correlation. *Renewable Energy*, 50:236–243.