An Improved Single Node Genetic Programming for Symbolic Regression

Jiří Kubalík[1] and Robert Babuška[1,2]
[1]*Czech Institute of Informatics, Robotics, and Cybernetics, CTU in Prague, Prague, Czech Republic*
[2]*Delft Center for Systems and Control, Delft University of Technology, Delft, The Netherlands*

Keywords: Genetic Programming, Single Node Genetic Programming, Symbolic Regression.

Abstract: This paper presents a first step of our research on designing an effective and efficient GP-based method for solving the symbolic regression. We have proposed three extensions of the standard Single Node GP, namely (1) a selection strategy for choosing nodes to be mutated based on the depth of the nodes, (2) operators for placing a compact version of the best tree to the beginning and to the end of the population, and (3) a local search strategy with multiple mutations applied in each iteration. All the proposed modifications have been experimentally evaluated on three symbolic regression problems and compared with standard GP and SNGP. The achieved results are promising showing the potential of the proposed modifications to significantly improve the performance of the SNGP algorithm.

1 INTRODUCTION

This paper presents a first step of our research on genetic programming (GP) for the symbolic regression problem. The ultimate goal of our project is to design an effective and efficient GP-based method for solving dynamic symbolic regression problems where the target function evolves in time. Symbolic regression (SR) is a type of regression analysis that searches the space of mathematical expressions to find the model that best fits a given dataset, both in terms of accuracy and simplicity[1].

Genetic programming belongs to effective and efficient methods for solving the SR problem. Besides the standard Koza's tree-based GP (Koza, 1992) there have been many other variants proposed that proved to perform well on the SR problem. They include, for instance, Grammatical Evolution (GE) (Ryan et al., 1998) which evolves programs whose syntax is defined by a user-specified grammar (usually a grammar in Backus-Naur form). Gene Expression Programming (GEP) (Ferreira, 2001) is another GP variant successful in solving the SR problems. Similarly to GE it evolves linear chromosomes that are expressed as tree structures through a genotype-phenotype mapping. A graph-based Cartesian GP (CGP) (Miller and Thomson, 2000), is a GP technique that uses a very simple integer based genetic representation of a program in the form of a directed graph. In its classic form, CGP uses a variant of a simple algorithm called $(1 + \lambda)$-Evolution Strategy with a point mutation variation operator. When searching the space of candidate solutions, CGP makes use of so called *neutral mutations* meaning that a move to the new state is accepted if it does not worsen the quality of the current solution. This allows an introduction of new pieces of genetic code that can be plugged into the functional code later on and allows for traversing plateaus of the fitness landscape.

A Single Node GP (SNGP) (Jackson, 2012a), (Jackson, 2012b) is rather new graph-based GP system that evolves a population of individuals, each consisting of a single program node. Similarly to CGP, the evolution is carried out via a hill-climbing mechanism using a single reversible mutation operator. The first experiments with the SNGP were very promising as they showed that the SNGP significantly outperforms the standard GP on various problems including the SR problem. In this work we take the standard SNGP as the baseline approach and propose several modifications to further improve its performance.

The goals of this work are to verify performance of the SNGP compared to the standard GP on various SR benchmarks and to investigate the impact of the following three design aspects of the SNGP algorithm:

[1]https://en.wikipedia.org/wiki/Symbolic_regression

Kubalík, J. and Babuška, R..
An Improved Single Node Genetic Programming for Symbolic Regression.
In *Proceedings of the 7th International Joint Conference on Computational Intelligence (IJCCI 2015) - Volume 1: ECTA*, pages 244-251
ISBN: 978-989-758-157-1

- a strategy for selection of the nodes to be mutated,
- a strategy according to which the nodes are ordered in the population,
- and a type of the local search strategy used to guide the optimization process

The paper is organized as follows. In Section 2, the SNGP algorithm is described. In Section 3, three modifications of the SNGP algorithm are proposed. Experimental evaluation of the modified SNGP and its comparison to the standard SNGP and standard Koza's GP is presented in Section 4. Finally, Section 5 concludes the paper and proposes directions for the further work on this topic.

2 SINGLE NODE GENETIC PROGRAMMING

2.1 Representation

The Single Node Genetic Programming is a GP system that evolves a population of individuals, each consisting of a single program node. The node can be either terminal, i.e. a constant or a variable node, or a function from a set of functions defined for the problem at hand. Importantly, individuals are not entirely distinct, they are interlinked in a graph structure similar to that of CGP, with population members acting as operands of other members (Jackson, 2012a).

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

- $u_i \in T \cup F$ is either an element chosen from a function set F or a terminal set T defined for the problem,
- f_i is the fitness of the individual,
- $Succ_i$ is a set of successors of this node, i.e. the nodes whose output serves as the input to the node,
- $Pred_i$ is a set of predecessors of this node, i.e. the nodes that use this individual as an operand,
- O_i is a vector of outputs produced by this node.

Typically, the population is partitioned so that first N_{term} nodes, at positions 0 to $N_{term} - 1$, are terminals (variables and constants in case of the SR problem), followed by function nodes. Importantly, a function node at position i can use as its successor (i.e. the operand) any node that is positioned lower down in the population relative to the node i. This means that for each $s \in Succ_i$ we have $0 \leq s < i$ (Jackson, 2012a). Similarly, predecessors of individual i must occupy higher positions in the population, i.e. for each $p \in Pred_i$ we have $i < p < N$. Note that each function node is in fact a root of a tree that can be constructed by recursively traversing the successors until the leaf terminal nodes.

2.2 Evolutionary model

In (Jackson, 2012a), a single evolutionary operator called *successor mutate* (*smut*) has been proposed. It picks one individual of the population at random and then one of its successors is replaced by a reference to another individual of the population making sure that the constraint imposed on the successors is satisfied. Predecessor lists of all affected individuals are updated accordingly. Moreover, all individuals affected by this action must be reevaluated as well. For more details refer to (Jackson, 2012a).

The evolution is carried out via a hill-climbing mechanism using a *smut* operator and an acceptance rule. The acceptance rule can have various forms. In (Jackson, 2012a), it was based on fitness measurements across the whole population, rather than on single individuals. This means that once the population has been changed by a single application of the *smut* operator and all affected individuals have been reevaluated the new population is accepted if and only if the sum of the fitness values of all individuals in the population is no worse than the sum of fitness values before the mutation. Otherwise, the modifications made by the mutation are reversed. In (Jackson, 2012b) the acceptance rule is based only on the best fitness in the population. The latter acceptance rule will be used in this work as well. The reason for this choice is explained in Section 3.4.

3 PROPOSED MODIFICATIONS

In this section, the following three modifications of the SNGP algorithm will be proposed:

1. A selection strategy for choosing nodes to be mutated based on the depth of the nodes.
2. Operators for placing a compact version of the tree rooted in the best performing node to the beginning and to the end of the population, respectively.
3. A local search strategy with multiple mutations applied in each iteration.

In the following text, the term "best tree" is used to denote the tree rooted in the best performing node.

3.1 Depthwise Selection Strategy

The first modification focuses on the strategy for selecting the nodes to be mutated. In the standard SNGP, the node to be mutated is chosen at random. This means that all function nodes have the same probability of selection irrespectively of (1) how well they are performing and (2) how well the trees of which they are a part are performing. This is not in line with the evolutionary paradigm where the well fit individuals should have higher chance to take part in the process of an evolution of the population.

One way to narrow this situation is to select nodes according to their fitness. However, this would prefer just the root nodes of trees with high fitness while neglecting the nodes at the deeper levels of such well-performing trees which themselves have rather poor fitness. In fact, imposing high selection pressure on the root nodes might be counter-productive in the end as the mutations applied on the root nodes are less likely to bring an improvement than mutations applied on the deeper structures of the trees.

We propose a selection strategy that takes into account the quality of the mutated trees, so that better performing trees are preferred, as well as the depth of the mutated nodes so that deeper nodes of the trees are preferred to the shallow ones. The selection procedure has four steps:

1. A function node n is chosen at random.
2. A tree t with the best fitness out of all trees that use the node n is chosen.
3. All nodes of the tree t are collected in a set S. Each node is assigned a score equal to its depth in the tree t.
4. One node is chosen from the set S using a binary tournament selection considering the score values in the higher the better manner.

3.2 Organization of the Population

The second modification aims at improving the exploration capabilities of the SNGP algorithm. Two operators for placing a compact version of the tree rooted in the best performing node to the beginning or to the end of the population are proposed.

moveLeft Operator. Let us first describe the idea behind and the realization of the operator that places the compact version of the best tree to the beginning of the population. The motivation for this operator, denoted in this work as *moveLeft* operator, is that well-performing nodes (and the whole tree structure rooted in this node) can represent a suitable building block for constructing even better trees when used as a successor of other nodes in the population. Since the chance of any node of being selected as a successor by other nodes is higher if the node is more to the left in the population and vice versa, it might be beneficial to store the well performing nodes at lower positions in the population. Thus, the operator takes the best tree, T_{best}, and puts its compact version to the very beginning of the population. The operator works as follows:

1. Start in the best performing node and traverse its tree, T_{best}, in the breadth-first manner so that nodes with the lowest index i are processed first. Put the nodes of the tree into a list of nodes, L, in the order reversed to the order in which they were processed.
2. Set all successor and predecessor links within L so that the structure represents the same tree as the original tree T_{best}.
3. Place the tree stored in L to the beginning of the population, i.e. the first node of L being at the first function node position of the population.
4. Set the successor and predecessor links within the original tree T_{best} so that the tree retains the same functionality as it had before the action.
5. Update the successor and predecessor links of the other nodes in the population so that as many original links as possible are preserved while the functionality of the compact tree remains intact.

Note that after the application of the T_{best} operation the population contains two versions of the best tree, the original one and the compact one. It is made sure that all nodes of the original tree have properly set their successors sets. If for example some successors of a node of the original best tree gets affected by the T_{best} operation (i.e. the successor falls into the portion of the population newly occupied by the compact version of the tree), the successor of the respected node is redirected to the new version of the successor. Moreover, some links have to be added to the list of predecessors of nodes of the compact tree in order to ensure a feasibility of other nodes in the population.

moveRight Operator. Similarly, an operator that places the compact version of the best tree to the end of the population is proposed. The motivation for this operator, denoted in this work as *moveRight* operator, is that a performance of some well-performing tree can be further improved by mutations applied to the nodes on deeper levels of the tree more likely than by mutations applied to the root node or shallow nodes of the tree. In order to increase the number of possible structural changes to the deeper nodes of the best tree the compact version of the tree is placed to the end of the population. The operator works similarly to the

moveLeft operator with the difference the successor and predecessor links are updated. Here, the predecessor links of nodes that refer to the nodes within the area of the compact version of the best tree must be removed. Otherwise, they would affect a functionality of the best tree. Note that the application of the *moveRight* operator might result in the final population that contains just a single occurrence of the best tree. This might happen when the nodes of the original best tree fall into the area of the compact version of the tree.

3.3 Local Search Strategy

The last modification of the standard SNGP algorithm consists in allowing multiple mutation in a single iteration of the local search procedure. The idea behind this modification is rather straightforward. During the course of the optimization process the population might converge to the local optimum state where it is hard to find further improvement by just one application of the smut operator. With multiple mutations applied in each iteration, the probability of getting stuck in such local optimal should be reduced. In this work, a parameter *upToN* specifying the maximum number of mutation applications is used. Thus, if the parameter is set for example to 5, a randomly chosen number from interval $\langle 1,5\rangle$ of mutations are applied to the population in each iteration.

3.4 Outline of Tested SNGP Algorithm

This section presents an outline of the generic SNGP algorithm with possible utilization of the proposed modifications, see Fig. 1. In each iteration, k mutations are applied to nodes of the population, see steps 8-10. In case of the standard SNGP just a single mutation is applied in each iteration. After all k mutations have been applied, the nodes affected by this action gets reevaluated. If the best fitness of the modified population is not worse than the current best-so-far fitness than the modified population becomes the current population for the next iteration, see step 15. In step 16, the operators moving the best tree to the beginning or to the end of the population are applied to the population, if applicable. Then the fitness evaluation counter is incremented and if there are still some fitness evaluations left the next iteration is carried out. Once the maximal number of fitness evaluations is used the best node (and its tree) of the population is returned.

In this work, we use the acceptance criterion, step 13, working with the best fitness in the population, not the average fitness of the population. The reason is that when the *moveLeft* and *moveRight* operators are used, they might significantly change the average fitness of the population while the best fitness stays intact.

```
1   Initialize population of nodes, P
2   evaluate P
3   bestSoFar ← best of P
4   i ← 0                  // number of fitness evaluations
5   do
6       P' ← P             // work with a copy of P
7       Choose the no. of mutations, k ∈ (1, upToN),
        to be applied to nodes from P'
8       for(n = 1 ... k)
9           Choose the node to be mutated, N
10          P' ← Apply mutation to node N of P'
11      evaluate P'
12      currBest ← best of P'
13      if(currBest is not worse than bestSoFar)
14          bestSoFar ← currBest
15          P ← P'         // update the population
16          If applicable, apply either moveLeft
            or moveRight operator to P
17      i ← i + 1
18  while (i ≤ maxFitnessEvaluations)
19  return bestSoFar
```

Figure 1: Outline of the SNGP algorithm.

4 EXPERIMENTS

This section presents experiments carried out with standard GP, standard SNGP and SNGP with the proposed modifications.

4.1 Test Cases

The algorithms have been tested on five symbolic regression benchmarks

- $f_1(x) = 4x^4 - 3x^3 + 2x^2 - x$, 32 training samples equidistantly sampled from $\langle 0, 1.0)$,
- $f_2(x) = x^6 - 2x^4 + x^2$, 100 training samples equidistantly sampled from $\langle -1.0, 1.0)$
- $f_3(x) = x^6 - 2.6x^4 + 1.7x^2$, 100 training samples equidistantly sampled from $\langle -1.0, 1.0)$
- $f_4(x) = x^6 - 2.6x^4 + 1.7x^2 - x$, 100 training samples equidistantly sampled from $\langle -1.4, 1.4)$
- $f_5(x_1, x_2) = \frac{(x_1-3)^4+(x_2-3)^3-(x_2-3)}{(x_2-2)^4+10}$, 100 training samples equidistantly sampled from $\langle 0.05, 6.05)$

Table 1: Results obtained with the compared algorithms on the function f_1.

algorithm	fitness	sample rate (%)	solution rate (%)	size
GP	0.14	82.8	49	151
SNGP_1_random_noMove	0.65	37.2	5	26.8
SNGP_1_depthwise_noMove	0.29	63.4	18	33.7
SNGP_1_random_moveLeft	0.62	38.1	3	22.5
SNGP_1_depthwise_moveLeft	**0.25**	68.4	24	33.9
SNGP_1_random_moveRight	0.66	35.9	4	34.5
SNGP_1_depthwise_moveRight	0.28	66.9	17	56.6
SNGP_5_depthwise_noMove	0.16	78.8	49	32.3
SNGP_5_depthwise_moveLeft	0.17	77.5	49	28.5
SNGP_5_depthwise_moveRight	0.14	84.7	53	52.4

Table 2: Results obtained with the compared algorithms on the function f_2.

algorithm	fitness	sample rate (%)	solution rate (%)	size
GP	0.78	88.7	69	175.1
SNGP_1_random_noMove	0.85	73.4	33	27.7
SNGP_1_depthwise_noMove	0.17	94.4	78	27.6
SNGP_1_random_moveLeft	0.75	78	33	30.8
SNGP_1_depthwise_moveLeft	0.25	92.8	65	27.7
SNGP_1_random_moveRight	0.84	72.7	17	47
SNGP_1_depthwise_moveRight	0.15	94.3	82	40.9
SNGP_5_depthwise_noMove	**1e-6**	100	100	22.6
SNGP_5_depthwise_moveLeft	0.08	97.8	87	21.2
SNGP_5_depthwise_moveRight	0.05	98.2	91	37.2

The first two functions are rather simple polynomials with small integer constants. We chose the function f_1 since it was used in the original SNGP paper (Jackson, 2012a). Function f_2 is the Koza-3 function taken from (McDermott, 2012). Functions f_3 and f_4 are modifications of the Koza-3 function so that they involve non-trivial decimal constants. Thus, these functions should represent harder instances than f_1 and f_2. The function f_4 is made even harder than f_3 while breaking the symmetry by adding the term "$-x$". The last function f_5 is a representative of a rational function of two variables. This function, known as Vladislavleva-8 function (McDermott, 2012), represents the hardest SR problem used in this work.

The fitness of each individual is calculated as the sum of absolute errors in the $f(x)$ values computed by the individual over all training samples.

4.2 Experimental Setup

All the tested variants of the SNGP use a population of size 400. The population starts with terminal nodes representing the variable x_1 and x_2 and a constant 1.0 followed by function nodes of types {+, -, *, /}. SNGP was run for 25,000 iterations, in each iteration just a single population reevaluation is computed (note, just the nodes that were affected by the mutation are reevaluated). The number of iterations was chosen so as to make the comparisons of the GP and SNGP as fair as possible. This way a balance between processed nodes and fitness evaluations is found, see (Ryan and Azad, 2014).

The proposed modifications of the SNGP algorithm are configured with the following parameters:

- $upToN \in \{1,5\}$,
- $selection \in \{random, depthwise\}$
- $moveType \in \{noMove, moveLeft, moveRight\}$.

Names of the tested configurations of the SNGP are constructed as follows "SNGP_upToN_selection_moveType". The standard SNGP is denoted as SNGP_1_random_noMove.

Standard GP with generational replacement strategy was used with the following parameters:

Table 3: Results obtained with the compared algorithms on the function f_3.

algorithm	fitness	sample rate (%)	solution rate (%)	size
GP	1.19	68.4	0	155
SNGP_1_random_noMove	2.7	40	0	28.9
SNGP_1_depthwise_noMove	1.5	54.0	0	33.6
SNGP_1_random_moveLeft	**2.39**	43.7	0	30.1
SNGP_1_depthwise_moveLeft	1.4	59.2	0	35
SNGP_1_random_moveRight	2.75	37.2	0	43.4
SNGP_1_depthwise_moveRight	1.6	51.6	0	56.2
SNGP_5_depthwise_noMove	1.37	59.2	0	32.6
SNGP_5_depthwise_moveLeft	**1.23**	65.6	0	30.3
SNGP_5_depthwise_moveRight	1.35	60.8	0	57.2

Table 4: Results obtained with the compared algorithms on the function f_4.

algorithm	fitness	sample rate (%)	solution rate (%)	size
GP	11.0	19.4	0	146.8
SNGP_1_random_noMove	10.5	12.9	0	26.1
SNGP_1_depthwise_noMove	7.8	21.7	0	34.2
SNGP_1_random_moveLeft	11.2	10.8	0	26.5
SNGP_1_depthwise_moveLeft	8.3	19.0	0	32.9
SNGP_1_random_moveRight	**8.9**	19.0	0	45.8
SNGP_1_depthwise_moveRight	7.4	22.4	0	53.9
SNGP_5_depthwise_noMove	7.15	25.8	0	31.5
SNGP_5_depthwise_moveLeft	7.4	24.0	0	28
SNGP_5_depthwise_moveRight	**6.7**	27.2	0	50.8

Table 5: Results obtained with the compared algorithms on the function f_5.

algorithm	fitness	sample rate (%)	solution rate (%)	size
GP	71.2	4.4	0	194.3
SNGP_1_random_noMove	64.1	4.0	0	26.0
SNGP_1_depthwise_noMove	61.6	3.8	0	35.6
SNGP_1_random_moveLeft	64.9	3.8	0	27.3
SNGP_1_depthwise_moveLeft	**60.0**	4.1	0	34.9
SNGP_1_random_moveRight	63.6	4.4	0	44.3
SNGP_1_depthwise_moveRight	61.1	4.1	0	51.6
SNGP_5_depthwise_noMove	60.7	3.7	0	32.8
SNGP_5_depthwise_moveLeft	60.9	3.6	0	31.9
SNGP_5_depthwise_moveRight	60.4	4.0	0	51.1

- Function set: {+, -, *, /}
- Terminal set: $\{x_1, x_2, 1.0\}$
- Population size: 500
- Initialization method: Ramped half-and-half
- Tournament selection: 5 candidates
- Number of generations: 55, i.e. 54 generations plus initialization of the whole population
- Crossover probability: 90%
- Reproduction probability: 10%
- Probability of choosing internal node as crossover point: 90%

For the experiments with the GP we used the Java-based Evolutionary Computation Research System ECJ 22[2].

One hundred independent runs were carried out with each tested algorithm on each benchmark and the observed performance characteristics are

- *fitness* – the mean best fitness over 100 runs;
- *sample rate* – the mean number of successfully solved samples by the best-fitted individual calculated over 100 runs, where the sample is considered to be successfully solved by the individual iff the absolute error achieved by the individual on this sample is less then 0.01;
- *solution rate* – the percentage of complete solutions found within 100 runs, where the runs completely solves the problem iff the best individual generates on all training samples the absolute error less than 0.01;
- *size* – the mean number of nodes of the best solution found calculated over 100 runs.

4.3 Results

The results obtained with the compared algorithms are presented in Tables 1-5. The first observation is that our results obtained on the benchmark f_1 are quite different than the results presented in (Jackson, 2012a), as the performance of the SNGP is not as good as the SNGP performance presented there whilst the standard GP performs much better than presented in (Jackson, 2012a). This might be caused by different configurations of the SNGP and GP used in our work and in (Jackson, 2012a). We used different acceptance criterion in SNGP and the generational instead of the steady-state replacement strategy in GP. This observation might indicate that both approaches are quite sensitive to the proper setting of their individual components.

The second observation is that the modified versions of SNGP systematically outperform the standard SNGP w.r.t. the *fitness*, *sample rate* and *solution rate* performance measures. On the other hand, the modified SNGP is not a clear winner over the standard GP. The SNGP outperforms GP on f_2, f_4 and f_5. On f_1 it performs equally well as the GP On f_3, all versions of SNGP get outperformed by the GP w.r.t. the $fitness$. It turns out functions f_3, f_4 and f_5 represent a real challenge for all tested algorithms since no one was able to find a single correct solution within the 100 runs. We hypothesize the hardness of f_3 and f_4 stems from the fact these benchmarks involve non-trivial constants that might be hard to evolve. Function f_5 is hard since it is a rational function.

The third observation is that there is a clear trend showing that the depthwise node selection works significantly better than the random one. Whenever the SNGP configurations differ just in the selection type the one using the depthwise selection outperforms the one with the random selection.

The fourth observation is that the reorganization of the population using either the $moveLeft$ or *moveRight* operator does not have any significant impact on the overall performance of the algorithm. It happens only rarely that the SNGP using $moveLeft$ or *moveRight* outperforms its counterpart configuration with no move operator used. In particular, the $moveLeft$ operator was significantly better[3] than *noMove* in four cases, the *moveRight* operator was significantly better than *noMove* in two cases as indicated by values written in bold in Tables 1-5. On the other hand, the *noMove* configuration happened to outperform both the $moveLeft$ and *moveRight* configuration on function f_2.

The fifth observation is that the local search strategy allowing multiple mutations in one iteration outperforms the standard local search procedure with just a single application of the mutation operator per iteration. This is with agreement with our expectations.

Last but not least, the SNGP consistently finds much smaller trees than the GP. This is very important since very often solutions of small size that are interpretable by human are sought in practice.

5 CONCLUSIONS

This paper proposes three extensions of the standard Single Node GP, namely (1) a selection strategy for choosing nodes to be mutated based on the depth of the nodes, (2) operators for placing a compact version of the best tree to the beginning and to the end of the population, and (3) a local search strategy with multiple mutations applied in each iteration.

All proposed modifications have been experimentally evaluated on three symbolic regression problems and compared with standard GP and SNGP. The achieved results are promising showing the potential of the proposed modifications to significantly improve the performance of the SNGP algorithm.

The next step of our research will be to carry out a thorough experimental evaluation of the modified SNGP algorithm and other GP paradigms such as the

[2]https://cs.gmu.edu/ eclab/projects/ecj/

[3]Checked using the T-test at the 5% significance level

standard GP and GE with the primary objectives being the speed of convergence and the ability to react fast to the changes of the environment. Further extensions of the SNGP algorithm will include a utilization of new mutation operators, for example a mutation operator altering the function type of the mutated node. Since the local strategy is very prone to get stuck in a local optimum, we will also investigate mechanisms for escaping from the local optima such as the restarted search with perturbations.

As the experimental results suggest, the SNGP algorithm might become ineffective when trying to approximate a target function that involves non-trivial constants. Thus, we will investigate possibilities of hybridization of the SNGP with local tuning techniques such as multivariate linear regression.

ACKNOWLEDGEMENT

This research was supported by the Grant Agency of the Czech Republic (GAČR) with the grant no. 15-22731S entitled "Symbolic Regression for Reinforcement Learning in Continuous Spaces".

REFERENCES

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

Jackson, D. (2012a). A new, node-focused model for genetic programming. In *EuroGP 2012*, pages 49–60.

Jackson, D. (2012b). Single node genetic programming on problems with side effects. In *PPSN XII*, pages 327–336.

Koza, J. (1992). *Genetic programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, 2nd edition.

McDermott, J. e. a. (2012). Genetic programming needs better benchmarks. In *Proceedings of the GECCO '12*, pages 791–798, New York, NY, USA. ACM.

Miller, J. and Thomson, P. (2000). Cartesian genetic programming. In *Poli, R. et al. (eds.) EuroGP 2000, LNCS, vol. 1802, pp. 121–132*. Springer.

Ryan, C. and Azad, R. M. A. (2014). A simple approach to lifetime learning in genetic programming-based symbolic regression. *Evolutionary Computation*, 22(2):287–317.

Ryan, C., Collins, J., Collins, J., and O'Neill, M. (1998). Grammatical evolution: Evolving programs for an arbitrary language. In *LNCS 1391, Proceedings of the First European Workshop on Genetic Programming*, pages 83–95. Springer-Verlag.