Sequential Model Building in Symbolic Regression

Jan Žegklitz[1,2] and Petr Pošík[1]

[1] Czech Technical University in Prague, Faculty of Electrical Engineering,
Technická 2, 166 27 Praha 6, Prague, Czech Republic
[2] Czech Academy of Sciences, Institute of Computer Science,
Pod Vodárenskou věží 271/2, 182 07 Praha 8, Prague, Czech Republic

Abstract: Symbolic Regression is a supervised learning technique for regression based on Genetic Programming. A popular algorithm is the Multi-Gene Genetic Programming which builds models as a linear combination of a number of components which are all built together. However, in recent years a different approach emerged, represented by the Sequential Symbolic Regression algorithm, which builds the model sequentially, one component at a time, and the components are combined using a method based on geometric semantic crossover. In this article we show that the SSR algorithm effectively produces linear combination of components and we introduce another sequential approach very similar to classical ensemble method of boosting. All algorithms are compared with MGGP as a baseline on a number of real-world datasets. The results show that the sequential approaches are overall worse than MGGP both in terms of accuracy and model size.

1 Introduction

Symbolic Regression (SR) is a supervised learning task with the goal to find a mathematical function (preferably a simple one) of a number of variables that fits the training data available for training. Regression is a well known task with a number of well known, well tested and successful approaches, e.g. neural networks, support vector machines and random forests to name a few. However, these conventional approaches produce models which, while useful, are often essentially black boxes which are difficult to interpret. One of the goals of SR is to produce "white-box" models in the form of a mathematical expression. The overwhelming majority of SR approaches utilize some form of Genetic Programming (GP) [1].

This article is structured as follows: Section 2 introduces previous research relevant for this article and and provide a new view on the main algorithm examined in this article, in Section 3 we propose a simple boosting-based sequential SR algorithm, Sections 4 and 5 present the experiments and results respectively and provide a discussion of these results. Finally, Section 6 concludes the article and proposes further research.

2 Related Work

In this section we briefly revisit important approaches and algorithms related to the sequential model building and to the experiments performed later in this article.

First we introduce Multi-Gene Genetic Programming which is not a sequential algorithm but is important for this article. Then we briefly revisit boosting, a classical machine learning ensemble method. Finally we introduce the most important related research, the Sequential Symbolic Regression algorithm.

2.1 Multi-Gene Genetic Programming

Multi-Gene GP (MGGP) [2, 3, 4] is an extension of classical GP for symbolic regression. The main idea behind MGGP is that each individual is composed of one *or more* independent trees, called genes, and these then form the complete model together. To make the complete model, the outputs of the genes are linearly combined with the coefficients determined using linear regression. In this article, we call this linear combination of genes a "top-level linear combination".

MGGP does not build the model sequentially. However, since we use the concept of top-level linear combination to put SSR (see the next section) in a similar framework and since we use the algorithm in the experiments, it is necessary that it is mentioned.

2.2 Boosting

Building predictive models sequentially is a well known machine learning technique. The prominent example of this approach is the *boosting* ensemble method [5, 6, 7]. Boosting in the context of least-squares regression (i.e. regression using the squared error as the criterion) works in the following way (simplified):

1. Start with a constant model $f_0(\mathbf{x}) = c$, e.g. using the mean target value: $f_0(\mathbf{x}) = \bar{\mathbf{y}}$
2. Iterate for $k = 1, 2, ..., M$, where M is the desired number of iterations:
   (a) Compute the residuals $\tilde{\mathbf{y}} = \mathbf{y} - f_{k-1}(\mathbf{x})$.
   (b) Train a model component $h(\mathbf{x})$ using $\tilde{\mathbf{y}}$ as the target values.

(c) Update the complete model $f_k(\mathbf{x}) = f_{k-1}(\mathbf{x}) + h(\mathbf{x})$.

3. $f_M(\mathbf{x})$ is the final model.

One possible view is that the latter model components correct the mistakes of the previous ones.

2.3 Sequential Symbolic Regression

SSR [8, 9] is a GP-based method that builds the model in a sequential fashion. The overall structure of the SSR method is following:

1. An SR algorithm (e.g. GP) is executed with the training set, producing a model component.
2. The model component is *added* to the complete model and the training data is modified.
3. Unless stopping criteria are met, go to step 1 and repeat, using the modified training data.

This structure is somewhat similar to boosting – a model component is trained, added to the complete model and the next one is trained using a modified training data. However, SSR does not do boosting. The difference is in step 2, i.e. how is the new component combined with the other components of the complete model and how is the training data modified.

For the first iteration, the target values are not modified and the new component simply becomes the complete model as at the time it is the only component. In the subsequent iterations, SSR utilizes the Geometric Semantic Crossover (GSX) from Geometric Semantic Genetic Programming (GSGP) [10] to add the new component to the complete model. GSX performs a convex combination of two functions:

$$f^*(\mathbf{x}) = rf(\mathbf{x}) + (1-r)f'(\mathbf{x}) \tag{1}$$

where r is a random number from the interval $[0,1)$, and f and f' are the two combined functions.

In SSR, the GSX is employed in a different way than to combine two known functions. Only one of the two functions is known (the component from the previous iteration, i.e. f). SSR therefore performs an "incomplete" GSX, meaning that f' is only a placeholder for a function to be found in the next iteration using a modified training data. The training data is derived by the following reasoning. The result of the GSX should optimally produce zero error, i.e. $f^*(\mathbf{x}) = \mathbf{y}$. Substituting this into Equation 1 and slightly rearranging it, the following equation is obtained

$$f'(\mathbf{x}) = \frac{\mathbf{y} - rf(\mathbf{x})}{1-r}. \tag{2}$$

This is, in effect, a requirement on how the outputs of f' should look like, i.e. it is a definition of its target values for the next iteration. Therefore the problem is transformed and the next iteration with different training data is started. It is important to note that the convex combinations are "nested" in depth in the second term of the GSX expression. In the last iteration, the f' placeholder is not replaced with another GSX result but with the function found in that iteration, terminating the nesting. The complete SSR procedure has the following form:

1. $\mathbf{y}_1 = \mathbf{y}$, $k \leftarrow 1$
2. Run the base algorithm with $\mathbf{y}_k$ as target values, producing a component f_k.
3. Sample a random number $r_k \in [0,1)$.
4. Update the complete model:
   - if $k = 1$ and it is not the last iteration: $f^* = r_k f_k + (1-r_k)f'$
   - if $k = 1$ and it is the last iteration: $f^* = f_k$, terminate
   - if $k > 1$ and it is not the last iteration: $f' \leftarrow r_k f_k + (1-r_k)f'$
   - if $k > 1$ and it is the last iteration: $f' \leftarrow f_k$, terminate
5. Adjust target values $\mathbf{y}_{k+1} = \frac{\mathbf{y}_k - r_k f_k(\mathbf{x})}{1-r_k}$
6. $k \leftarrow k+1$, repeat from step 2 unless a stopping condition is met.
7. return f^*

where f' is the placeholder. The nesting happens in the third case in step 4, where the placeholder is replaced with the GSX of the new function and the placeholder again.

2.4 SSR as Top-Level Linear Combination

In previous subsection we have briefly described the SSR procedure and how it constructs the models via GSX. Now we rewrite the procedure in such a way that the result is a linear combination of expressions as in MGGP and Boost-GP.

By iterating Equation 1 (with added subscripts to indicate the iteration number) and expanding the multiplication at each step, we can see how the model looks like as if it was finished at that iteration, i.e. in the second and fourth case of step 4 of the SSR procedure from the previous subsection (for the sake of simplicity, let $\bar{r}_k = 1 - r_k$):

$$\begin{aligned}
k=1: &\quad f_1^* = f_1 \\
k=2: &\quad f_2^* = r_1 f_1 + \bar{r}_1 f_2 \\
k=3: &\quad f_3^* = r_1 f_1 + \bar{r}_1 (r_2 f_2 + \bar{r}_2 f_3) \\
&\qquad = r_1 f_1 + \bar{r}_1 r_2 f_2 + \bar{r}_1 \bar{r}_2 f_3 \\
k=4: &\quad f_4^* = r_1 f_1 + \bar{r}_1 r_2 f_2 + \bar{r}_1 \bar{r}_2 (r_3 f_3 + \bar{r}_3 f_4) \\
&\qquad = r_1 f_1 + \bar{r}_1 r_2 f_2 + \bar{r}_1 \bar{r}_2 r_3 f_3 + \bar{r}_1 \bar{r}_2 \bar{r}_3 f_4 \\
&\quad \vdots
\end{aligned} \tag{3}$$

From the above iteration we can see that the model in fact is a linear combination of several expressions. We can also see a pattern of the multiplicative coefficients: each time a component f_k is added to the model, the coefficient of component f_{k-1} is multiplied by r_{k-1} and the coefficient of the new component f_k is the coefficient of component f_{k-1} multiplied by $\bar{r}_{k-1}$. This allows for efficient implementation using a list of expressions and a corresponding list of their coefficients, just multiplying a number in the second list and adding an element to the end of both lists. This view also allows further manipulation of the coefficients and model components in a way similar to MGGP, e.g. performing an additional linear regression.

3 Boosting-GP

In the previous section we briefly introduced Sequential Symbolic Regression algorithm (SSR) which utilizes Geometric Semantic Crossover to sequentially combine the components of the complete model and to derive modified training data for subsequent iterations. We propose an alternative approach that is almost identical to the boosting scheme (see Section 2.2), hence we call it Boost-GP.

The procedure matches that of boosting we already described, except for a few details. In Boost-GP, we do not start with a constant model, instead we start with no prior model at all and the first component is found using GP too. The procedure that finds the individual components is a GP-based algorithm. If the underlying algorithm produces a linear combination of several expressions (as MGGP does), those are treated as several components and are added to the model at the same time, summing possible constant offsets. Other than these details, the procedure is exactly the same as that of boosting.

4 Experiments

We perform a series of experiments with the aim to compare the performance of a number of algorithms both in terms of model accuracy and complexity. First we describe the algorithms used in the experiments and their settings, then we describe the datasets and finally we describe the experimental protocol.

4.1 Algorithms

For the experimental evaluation we selected a number of algorithms and their variants based on SSR, boosting with GP, MGGP and linear regression as a baseline.

LR The first algorithm that serves purely as a baseline is an ordinary least-squares linear regression on the problem's features. Its purpose is to frame all the other algorithms to a well established context.

MGGP This algorithm represents an algorithm that produces models composed of multiple components but which are all evolved simultaneously.

1G The same as MGGP but the individuals are limited to a single gene, making it effectively a Scaled Symbolic Regression algorithm [11] (not to be confused with Sequential Symbolic Regression that is discussed in this article, which has the same abbreviation SSR). In order to provide similar complexity, the tree depth limit is increased to allow for at least the same number of nodes as MGGP.

SSR-GP, SSR-1G Sequential Symbolic Regression algorithm (see Section 2.3) with ordinary GP and 1G (see above) as the base algorithm respectively.

nSSR-GP, nSSR-1G Similar to SSR-GP and SSR-1G but using the normalization and denormalization as described in [9].

SSR with final fitting All the SSR-based algorithms mentioned above (coded identically but then suffixed with the word "fit", e.g. "SSR-GP fit") are also used in such way that after the algorithm completes, the coefficients of the individual components are recalculated using linear regression in the same manner as is used in MGGP.

Boost-GP, Boost-1G Boosting-type sequential approach (see Section 3) with ordinary GP and 1G (see above) as the base algorithm respectively.

4.2 Algorithm settings

The description of settings of the algorithms (except for LR which has no settings) follows. MGGP, SSR and Boost algorithms are set to produce a model of 10 components with the maximum depth of 10. That means that MGGP has the gene number limit set to 10 and SSR and Boost algorithms are run for 10 iterations. The stopping criterion for MGGP is a wall-clock time limit of 1800 s, for SSR and Boost algorithms it is the number of 10 iterations of which each has a wall-clock time limit of 180 s, therefore all algorithms have an equal amount of time available to find a good solution. The 1G algorithm produces only one component and in order to have roughly the same amount of nodes available, the maximum depth is set to 16. 1G is also run for 1800 s. Other settings which are common for the algorithms are described in Table 1.

4.3 Datasets

We use four real-world datasets freely available from the UCI repository [12]. The datasets are described in the following paragraphs. Summary information about these datasets is in Table 2.

parameter	value
population size	800
# of elites	10 % of population
tournament size	4 % of population
prob. of mutation	0.2
prob. of crossover	0.7
constant/subtree mutation ratio	0.3/0.7
σ of constant mutation	10
function set	$a+b$, $a-b$, $a \cdot b$, $\frac{a}{\sqrt{1+b^2}}$, e^a, $\lvert a \rvert$, $\sin a$, a^2

Table 1: Common algorithm settings.

dataset	# of dimensions	# of samples	
		training	testing
ASN	5	1052	451
CCS	8	721	309
ENC	8	537	231
ENH	8	537	231

Table 2: Summary information on the datasets used in the experiments. The division into training and testing samples comes from a random 70:30 split of the original dataset.

ASN Airfoil self-noise (ASN) is a 5-dimensional dataset describing acoustic pressure for various airfoils during wind tunnel tests.

CCS Concrete compressive strength (CCS) is an 8-dimensional dataset describing compressive strength of concrete based on its ingredients.

ENC, ENH Energy efficiency of cooling/heating (ENC, ENH) are 8-dimensional datasets describing energy efficiency of cooling and heating buildings.

4.4 Experimental protocol

Each dataset is randomly split into training/testing subset 25 times. Each algorithm is run once for each split of each dataset, i.e. 25 runs per algorithm per dataset in total. For each of the 25 runs of an algorithm on a dataset a different seed for the random number generator used by the algorithm (except for LR which is fully deterministic).

During the algorithm run, only the training set is used. After the run completes, the models produced by the algorithms are evaluated on the testing set. Performance is measured using the coefficient of determination, or R^2 score.[1] The model complexity is measured as the number of nodes in the model. The coefficients of the linear combination that combines the components of the model is not counted towards this number. These two measures (performance and complexity) are then aggregated over the 25 runs of each algorithm for each dataset.

All runs were performed on machines of identical configuration that were part of the MetaCentrum grid (see Acknowledgements).

5 Results

In this section we present the results of the experiments. As described in Section 4.4, we focus on the performance (R^2) the complexity (number of nodes) of the models. Test-set performance along with complexity is depicted in Figure 1. The result is displayed in the form of two crossing lines whose intersection is at the median R^2 and median number of nodes, and they stretch from 1st to 3rd quartile in both the measures. SSR algorithms of the same configuration except for final fitting are plotted in the same color for easier visual assessment of the effect of final fitting.

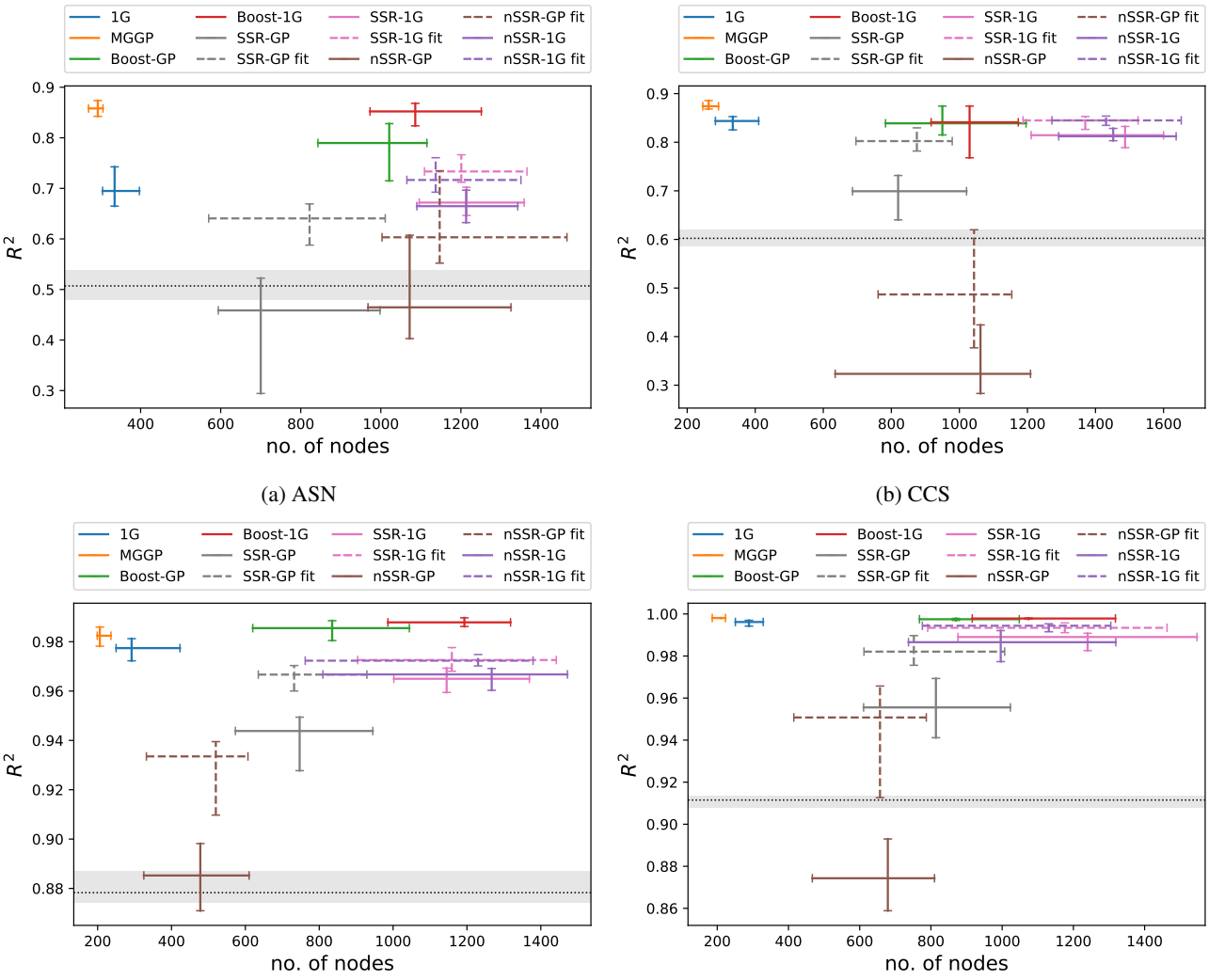
5.1 Discussion

From Figure 1 it is clear that MGGP produces the smallest models for all four datasets. We hypothesize that this is caused by the fact that all the components are evolved together so they can complement each other while in the other algorithms (except for 1G) each component is forced to evolve on its own. When the component is being evolved, it doesn't "know" about the other components so the evolution tries to solve the current problem with the single component and it produces a big component as it tries to do that. When it is time for the next component, the previous one is fixed and it cannot get smaller anymore.

In terms of test-set performance, MGGP, Boost-GP and Boost-1G are the best performers overall, with (n)SSR-1G fit being close. On the other hand, SSR-GP and nSSR-GP are, overall, the worst performers, performing even worse than LR in some cases. Even though the Boost algorithms are competitive in terms of performance, they are in line with the SSR algorithms regarding the number of nodes, which is much larger than that of MGGP or 1G.

The SSR algorithms, especially the variants without final fitting, are performing rather poorly compared to the other algorithms except for 1G and LR. It can be seen that the final fitting improves the performance considerably. We hypothesize that this is caused by the fact that the original coefficients are chosen randomly and from a bounded interval. Even though the target values are modified in such way that the result should be close to optimum if the new component models the new target values well,

[1] $R^2 = 1 - \frac{\sum_{i=0}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{N}(y_i - \bar{y})^2}$



(a) ASN

(b) CCS

(c) ENC

(d) ENH

Figure 1: Complexity-performance plot of the final models for all four datasets. The horizontal lines are vertically placed at the median R^2 value and stretch in the horizontal direction from 1st to 3rd quartile of the number of nodes. The vertical lines are horizontally placed at the median number of nodes and stretch in the vertical direction from 1st to 3rd quartile of R^2. The intersections show the point of median number of nodes and median R^2. The closer the lines are to the upper left corner of the plot, the better (i.e. well performing and small) the models are. The dotted line shows the median R^2 of LR and the gray band stretches from 1st to 3rd quartile of R^2 of LR.

the contribution of previous component can be multiplied by a very small number which makes the new target values not much "easier" than for previous components. Also, looking at Equation 3, each component is multiplied by a number of values, each smaller than or equal to 1 and the later the component is added the more such multipliers it has and therefore it contributes less and less to the final model. However, SSR-1G (and its fitted version) perform much better. The reason is that with 1G, the component is scaled by a factor, mitigating the just described issue to some extent.

Another notable pattern is that algorithms using 1G as the base algorithm perform considerably better than those using GP, which is to be expected since even the simple linear scaling can considerably reduce the error, especially if the inner structure of the component is good but it only lacks such scaling.

The final interesting pattern is that the normalization in SSR has almost no effect in terms of performance when the base algorithm is 1G, and quite inconsistent effect when the base algorithm is GP. The reason of the small effect with 1G is again the fact that 1G introduces its own multiplicative and additive constant so the normalization and denormalization coefficients become redundant to some extent.

5.2 Overfitting and underfitting

Figure 2 displays the results on training and testing set side by side. With a few exceptions, it can be said that no algorithm suffered from overfitting as the testing performance values are about as much smaller than the training ones as is exhibited by the linear regression. The most notable exceptions are the Boost algorithms, especially on ASN and CCS datasets. Boost-1G produced two significant outliers in both datasets and the other testing values are also generally smaller than the training ones. It is best seen in the CCS dataset where Boost-1G is the best performer by training performance but the testing performance is significantly[2] worse than the training performance.

With the exception of nSSR-GP fit on the CCS dataset, all the SSR algorithms have, overall, the smallest difference in training and testing performance. In some cases the testing performance is even better than the training performance. This could indicate potential underfitting, either of the whole algorithm or the individual components.

6 Conclusions and Future Work

In this article we reviewed the idea of sequential approach to symbolic regression. We reviewed an existing approach, called Sequential Symbolic Regression, which is unique by using Geometric Semantic Crossover to combine the individual model components and derive training data for subsequent component evolution.

We have shown that the models SSR produces are effectively a linear combination of a number of components, similar to MGGP, but constructed by a different approach. We have also proposed a simple boosting-inspired approach which uses the residuals directly as the training target values for the next component and combines the components just by adding them together.

We have performed a series of experiments with the algorithms, using different base algorithms of GP and 1G, and different variants of SSR, most importantly with final fitting of the coefficients with linear regression. The results have shown that MGGP, a non-sequential algorithm, is the best or one of the best algorithms on each dataset and producing by far the smallest models. Of the sequential algorithms, the SSR is, overall, the worst, except for the variants using final fitting and 1G as the base, which are comparable to the boosting-inspired algorithms on two of the four datasets.

Except for the boosting-inspired algorithms in some cases, no algorithm experienced overfitting, which might be a useful information for some users, and it also shows that big and complex models don't necessarily mean overfitting.

[2] Mann-Whitney U test with significance level $\alpha = 0.01$.

6.1 Future Work

This article investigated the sequential algorithms in their basic form only, using a predefined number of components and a predefined scheme of the evolution of the individual components. However, the sequential algorithms naturally lend themselves to tuning and tweaking. One of possible approaches is not to switch components regularly but based on some other scheme, e.g. the performance of the current component. Another approach worth investigating might be using full MGGP as the base algorithm in the sequential algorithms which would mean that multiple components will be produced at each stage, not just one. This would allow for a "continuum" of algorithms with MGGP on one side as a totally non-sequential algorithm, and SSR or boosting-inspired algorithms as described in this article on the other side, as totally sequential algorithms.

Another area worth exploring is the interplay of the complexity of individual components or limits on their size, the number of components, and the amount of computation resources available for a single component. The classical boosting works with so-called weak learners, i.e. models which by themselves are not capable of solving the problem to any reasonable degree, and uses many of such models. A similar approach could be used for the sequential approaches reviewed in this article, by setting tighter limits on the complexity and/or computational resources for the evolution of each component.

Acknowledgements

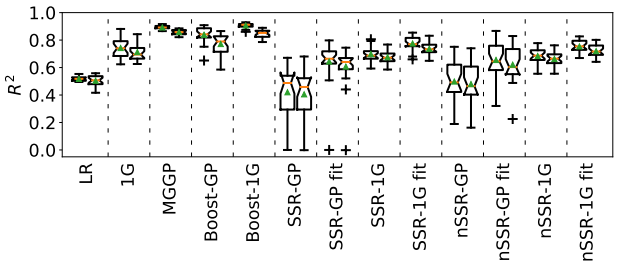
The reported research was supported by the Czech Science Foundation grant No. 17-01251. Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures " (CESNET LM2015042), is greatly appreciated.

References

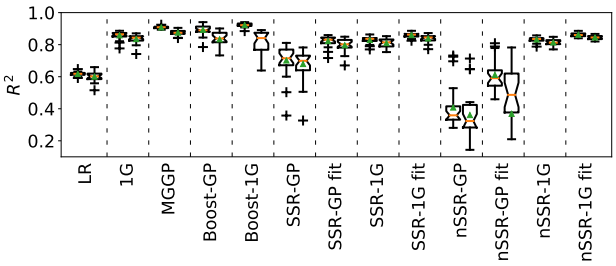
[1] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992. ISBN 0-262-11170-5. URL `http://mitpress.mit.edu/books/genetic-programming`.

[2] Mark Hinchliffe, Hugo Hiden, Ben McKay, Mark Willis, Ming Tham, and Geoffery Barton. Modelling chemical process systems using a multi-gene genetic programming algorithm. In *Late Breaking Paper, GP'96*, pages 56–65, Stanford, USA, 1996.

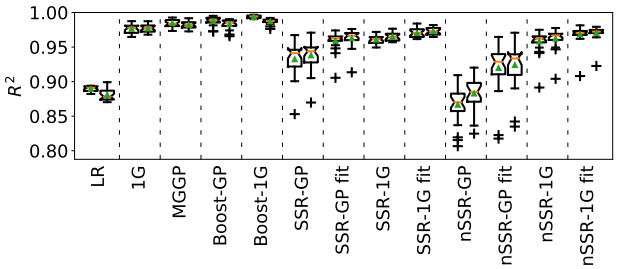
[3] Dominic P Searson, David E Leahy, and Mark J Willis. GPTIPS: an open source genetic programming toolbox for multigene symbolic regression. In *Proceedings of the International MultiConference of*



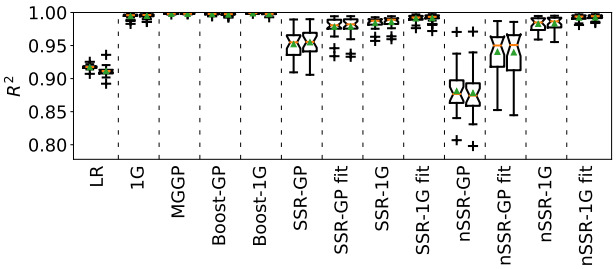
(a) ASN; two outlier values in testing Boost-1G at -206 and -2.85 are not shown.



(b) CCS; two outlier values in testing Boost-1G at -127797 and -85.6 and one value in testing nSSR-GP fit at -2.91 are not shown.



(c) ENC



(d) ENH

Figure 2: Performance box plots of the final models for all four datasets. Each algorithm has two box plots, the left displays training set performance, the right one displays the testing set performance. In plots for ASN and CCS datasets some outlier values are not shown in in the plot for the sake of clarity.

Engineers and Computer Scientists, volume 1, pages 77–80, March 2010.

[4] Dominic P. Searson. *GPTIPS 2: An Open-Source Software Platform for Symbolic Data Mining*, pages 551–573. Springer International Publishing, Cham, 2015. ISBN 978-3-319-20883-1. doi: 10.1007/978-3-319-20883-1_22. URL `http://dx.doi.org/10.1007/978-3-319-20883-1_22`.

[5] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990. ISSN 1573-0565. doi: 10.1007/BF00116037. URL `http://dx.doi.org/10.1007/BF00116037`.

[6] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997. ISSN 0022-0000. doi: doi:10.1006/jcss.1997.1504. URL `http://dx.doi.org/10.1006/jcss.1997.1504`.

[7] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag New York, 2 edition, 2009. ISBN 978-0-387-84858-7.

[8] Luiz Otávio V.B. Oliveira, Fernando E.B. Otero, Gisele L. Pappa, and Julio Albinati. *Sequential Symbolic Regression with Genetic Programming*, pages 73–90. Springer International Publishing, Cham, 2015. ISBN 978-3-319-16030-6. doi: 10.1007/978-3-319-16030-6_5. URL `https://doi.org/10.1007/978-3-319-16030-6_5`.

[9] L. O. V. B. Oliveira, F. E. B. Otero, L. F. Miranda, and G. L. Pappa. Revisiting the sequential symbolic regression genetic programming. In *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 163–168, Oct 2016. doi: 10.1109/BRACIS.2016.039.

[10] Alberto Moraglio, Krzysztof Krawiec, and ColinG. Johnson. Geometric semantic genetic programming. In CarlosA.Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7491 of *Lecture Notes in Computer Science*, pages 21–31. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-32936-4. doi: 10.1007/978-3-642-32937-1_3. URL `http://dx.doi.org/10.1007/978-3-642-32937-1_3`.

[11] Maarten Keijzer. Scaled symbolic regression. *Genetic Programming and Evolvable Machines*, 5(3):259–269, 2004. ISSN 1573-7632. doi: 10.1023/B:GENP.0000030195.77571.f9. URL `http://dx.doi.org/10.1023/B:GENP.0000030195.77571.f9`.

[12] K. Bache and M. Lichman. UCI machine learning repository, 2013. http://archive.ics.uci.edu/ml.