Feature Extraction for Surrogate Models in Genetic Programming

Martin Pilát[1(✉)] and Roman Neruda[2]

[1] Faculty of Mathematics and Physics, Charles University in Prague, Malostranské Náměstí 25, 11800 Prague, Czech Republic
Martin.Pilat@mff.cuni.cz

[2] Institute of Computer Science, The Czech Academy of Sciences, Pod Vodárenskou Věží 271/2, 182 07 Prague, Czech Republic
roman@cs.cas.cz

Abstract. We discuss the use of surrogate models in the field of genetic programming. We describe a set of features extracted from each tree and use it to train a model of the fitness function. The results indicate that such a model can be used to predict the fitness of new individuals without the need to evaluate them. In a series of experiments, we show how surrogate modeling is able to reduce the number of fitness evaluations needed in genetic programming, and we discuss how the use of surrogate models affects the exploration and convergence of genetic programming algorithms.

Keywords: Surrogate model · Genetic programming · Random forest

1 Introduction

Evolutionary algorithms are great optimizers, however, they require a large number of objective function evaluations to find a suitable solution for a given problem. This large number of evaluations may be problematic in practice. Surrogate modeling [4] helps to reduce the number of fitness evaluations needed to find a solution of a given quality. Its main idea is to build an approximate model of the fitness function, which is used during the optimization as a cheap replacement of the expensive fitness. In the most common case, the model is built using various machine learning techniques from the individuals evaluated earlier in the evolution. The surrogate model is usually a standard regression model. So far, this technique is used almost exclusively in the field of continuous optimization, i.e. the optimization of functions $\mathbb{R}^n \to \mathbb{R}$.

The spread of surrogate modeling to different areas of evolutionary optimization is limited by the higher complexity of machine learning in these cases. Creating a surrogate model that maps, for example, a genetic program to a real number is a much more challenging task than running a regression algorithm on a vector of real numbers. In genetic programming, the surrogate modelling additionally includes, at least, the extraction of features from the genetic programs. So far, there are few applications of surrogate modeling outside the field

J. Handl et al. (Eds.): PPSN XIV 2016, LNCS 9921, pp. 335–344, 2016.
DOI: 10.1007/978-3-319-45823-6_31

of continuous optimization, one such example is provided by Li *et al.* [6] who use surrogate models to solve a problem in mixed integer programming.

Hildebrandt and Branke [3] proposed a method based on phenotypic features to predict the fitness values of an individual in genetic programming and used it to evolve dispatching rules for job shop scheduling. To create the feature vectors, they evaluate the individual on a few tasks and use the results of the individual as features. Then, they use a nearest neighbor model to predict the fitness of the individual (the fitness of the closest evaluated individual is used as a fitness of the new one). Such approach can be used in scenarios where the evolved program is used as a controller and it can be evaluated on a smaller task. However, in other scenarios the applicability of this method may be limited. For example, our motivation for this work is based on our previous work [5], where we attempted to evolve machine learning workflows with genetic programming. In such a scenario a partial evaluation of the individual does not make sense.

In this work we investigate the extraction of features directly from the tree individuals used in GP, with no need to evaluate the individuals. Our main goal is to create an algorithm, which can use these features to predict the quality of an individual based solely on it genotypic representation. To this end, we first extract as many features as possible from the individual and then train a surrogate model based on random forests. We also investigate the importance of individual features and how well the predictions match the real quality of individuals.

2 Surrogate-Based Genetic Programming

We propose a set of features, that can be extracted in a single pass through the tree without the need to evaluate the program. The features contain information of different kinds: general features regarding the tree, features concerning the primitives (i.e. functions) used in the tree, features on the arguments of the program, features regarding the constants used in the tree, and also the fitness of the parents of an individual. Particularly, the following features were used in the experiments in this paper:

- tree features – depth of the tree, size of the tree (number of nodes)
- constant features – maximum, minimum, and mean, number of constants and distinct constants divided by the size of tree
- argument features – average number of times an argument is used and proportion of arguments used
- for each terminal or primitive – the number of times it is used divided by the length of the individual
- parents' fitness – minimum, maximum and mean of the fitness of parents

Therefore, the number of features extracted from each individual equals the number of different non-terminals + the number of arguments of the program + 1 (for the constants as terminals) + 12 (the general features, the counts of arguments and constants, the statistics on the constants values, and the statistics

Algorithm 1. Surrogate-based Genetic Programming

Require: n: population size, τ: number of evaluations before surrogate modeling, A_m: maximum size of training set, w: the proportion of worst individuals to discard

1: $t \leftarrow 0$, $A = \emptyset$, $P_0 \leftarrow$ InitRandomPopulation(n)
2: **for** i in P_0 **do**
3: $f_i \leftarrow$ Evaluate(i)
4: $\varphi_i \leftarrow$ ExtractFeatures(i); $A \leftarrow A \cup \{(i, f_i, \varphi_i)\}$
5: **end for**
6: **while** termination criterion not met **do**
7: $t \leftarrow t + 1$
8: $S \leftarrow$ Selection(P_{t-1})
9: $O_t \leftarrow$ GenerateOffspring(S)
10: **if** $|A| > \tau$ **then**
11: $T \leftarrow A$ ▷ training set
12: **if** $|T| > A_m$ **then** $T \leftarrow$ RandomSample(T, A_m)
13: $M \leftarrow$ BuildModel(Features(T), Targets(T))
14: $I \leftarrow \{i \in O | \text{FitnessNotEvaluated}(i)\}$
15: $\hat{f}_i = \{\text{PredFit}(i, \text{ExtractFeatures}(i), M) | i \in I\}$
16: $W \leftarrow$ the indices of $w|I|$ worst individuals
17: **for** i in I **do**
18: **if** $i \in W$ **then** $I \leftarrow$ replace i with its parent from S in I
19: **end for**
20: **end if**
21: $I \leftarrow \{i \in O | \text{FitnessNotEvaluated}(i)\}$
22: **for** i in P_t **do**
23: $f_i \leftarrow$ Evaluate(i)
24: $\varphi_i \leftarrow$ ExtractFeatures(i); $A \leftarrow A \cup \{(i, f_i, \varphi_i)\}$
25: **end for**
26: $P_t \leftarrow$ Best(P_t) $\cup$ RemoveWorst(O)
27: **end while**

on fitness of parents). We investigate the importance of these features and also the performance of some models based on these features in Sects. 3.1 and 3.2.

We also considered a number of different features, i.e. the numbers times each tree of depth one is used. However, such structural features would increase the length of the feature vector significantly and would make the model training slower.

2.1 Baseline Algorithm

The main loop of the proposed algorithm (cf. Algorithm 1) is a relatively standard GP algorithm. It first generates a random initial population (line 1) and evaluates its fitness (line 3). Then, the evolution loop starts, the offspring are generated (line 9), those with unknown fitness are evaluated (line 23) and, finally, the environmental selection is performed (line 26). In the environmental selection, we use a weak elitism, i.e. the best individual from the parents replaces the worst offspring.

The part of the algorithm described in the paragraph above is also what we call the baseline algorithm in our experiments.

2.2 Surrogate Modeling

The surrogate version of the algorithm contains an archive of all individuals evaluated during the run which is initialized in the beginning (line 1) and updated after each evaluation of the real fitness with the value of the fitness and the features of the evaluated individual (lines 4 and 24).

The main part regarding the surrogate modeling lies between lines 10 and 20. First, there is a test, whether there are enough (at least τ) evaluated individuals in the archive, if not, the surrogate part is skipped and the algorithm works precisely as the simple algorithm described above. Otherwise, the training set is created. It contains either the whole archive, if there are less then A_m individuals, or a random sample of A_m individuals from the archive if there are more. The sampling step improves the speed of the surrogate model training.

The training of the model is performed on line 13. The features and the fitness of the individuals from the training set are collected and used for the training of the surrogate model. The output variable considered by the models is the fitness of the respective individual. Then, the individuals with unknown fitness, denoted by set I, are evaluated by the surrogate model. To this end, the features are extracted from each such individual and its fitness is predicted by the surrogate model. On line 16, the individuals are sorted by the estimated fitness and the indices of $w|I|$ worst individuals are put into set W. Each individual with its index in W is replaced by its parent. This ensures that such an individual does not need to be evaluated by a real fitness function, and it gives the parent another opportunity to generate new individual in the next generation (if it survives the mating selection).

Finally, the rest of the unevaluated individuals (now only $(1-w)|I|$) is evaluated using the real fitness function. The newly evaluated individuals are added to the archive and the next generation begins.

2.3 Discussion

We use a rather unusual way of discarding the individuals predicted to be bad by the model – we replace them by their parents. In preliminary experiments, we have also tried a more traditional approach, i.e. discarding the individuals completely and replacing them by random or best parents. However, both these cases (random or best parents) lead to a fast loss of diversity in the population, as some of the parents get repeated. That in turn significantly slows down the convergence of the algorithm and can even lead to pre-mature convergence.

Another feature of the algorithm, which may be slightly unusual in genetic programming, is the use of weak elitism. We do not need the elitism from the point of view of preserving the best individual, as we already save the archive of all of them, however, preliminary experiments have shown, that it may slightly improve the performance of this simple algorithm.

3 Experiments

In order to evaluate the performance of our approach, we used four symbolic regression benchmarks described by White *et al.* [11] (keijzer-6, vladislavleva-4, nguyen-7, and pagie-1). The benchmarks represent a range of different functions with one to five arguments and each of the benchmarks uses a different set of primitives. For their precise definition refer to [8,11]. In preliminary experiements, we also used the korns-12 benchmark, however neither the baseline nor the surrogate algorithm were able to improve the random solutions from the initial population significantly, so these results are not presented here.

The objective is defined as the base 10 logarithm of the root mean square error (RMSE) of the prediction. The logarithm is used to make the values smaller and also easier for the surrogate model to train. As the algorithm uses only tournament selection (i.e. only comparisons of values), the objective is equivalent to RMSE.

The GP algorithm described above uses a population of 200 individuals and is run for a maximum of 15,000 objective function evaluations. The population is initialized by the ramped half-and-half method with maximum depth of trees set to 5. The mating selection is a tournament of 3 individuals, and the environmental selection uses a weak elitism (i.e. the best parent is added to the offspring population and the worst offspring is removed). The algorithm uses a simple one-point crossover (i.e. random subtrees are selected from each individual and swapped) and a uniform mutation which replaces a random subtree by a random subtree of a random depth between 1 and 4. The probability of crossover is 0.2 and the probability of mutation is 0.7. Those parents that do not undergo any of the operators are simply copied to the offspring population.

Additionally, the surrogate version of the algorithm starts using the surrogate model after $\tau = 1000$ evaluations and uses at most $A_m = 5000$ random individuals from the archive for the surrogate training. The surrogate model, unless otherwise noted, is a random forest for regression with 100 trees with the depth of at most 14 (the rest of the parameters of the trees uses the default values from scikit-learn [9]). The worst $w = 2/3$ of the individuals according to the model are discarded and replaced by their parents. The whole algorithm is implemented in Python using the deap framework [2] and the source codes are available as supplementary materials at the authors' webpage (www.martinpilat.com).

3.1 Model Quality

Before we use the surrogate model in the GP algorithm, we first test the performance of the features and different models for predicting the quality of solutions generated by the algorithm. To this end, we start the baseline GP and in each generation evaluate the new (therefore never before seen by the model) individuals with the surrogate model and also the real fitness function. Then, we compute the Spearman's rank coefficient, which expresses how similar the ranking provided by two methods is.

The Spearman's rank coefficient is used instead of correlation, as in the algorithm, we only use comparisons between pairs of individuals and not the actual values of the fitness (the algorithm uses tournament selection).

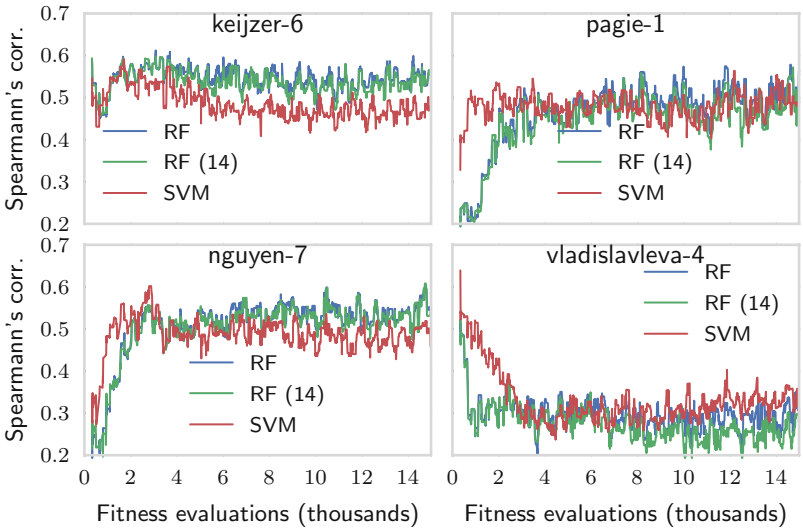

Fig. 1. The Spearman's correlation coefficient between the prediction provided by the model and the real fitness function. Median over 25 independent runs. The predictions made by the model were not used by the algorithm.

We compare support vector machines for regression (SVM) [10] and random forests [1] of 100 trees with two different depth limits - either unlimited (RF) or limited to 14 (RF (14)). Both methods use the default parameters from the scikit-learn framework and a standardization of the inputs is performed for the SVM. The limit on the depth of the random forest was set after preliminary experiments. It should be noted that the algorithm is quite sensitive to these settings and the provided values are a compromise, which seems to provide the best overall results. Different settings of the random forests were able to provide better results for some of the benchmarks than those presented here. In the preliminary experiments, we also used several types of linear models, however, their performance in this setting is unsatisfactory.

The results are presented in Fig. 1. The plots show the median of the Spearman's correlation coefficient computed over 25 independent runs as a function of the number of fitness evaluations. These runs were performed without the algorithm using the model in any way. In most cases, the model needs around 2,000 individuals in the training set before its performance becomes stable. After that, the Spearman's correlation coefficient is between 0.5 and 0.6 for most of the problems (except vladislavleva-4), which indicates a correlation of medium strength between the model and the observed fitness. For vladislavleva-4 the Spearmann's correlation drops to 0.3 after a thousand evaluations, this also coincides with the fact, that after thousand evaluations the convergence speed drops significantly (c.f. Fig. 2). This may be caused by the fact that the differences between the individuals get smaller and are thus harder to predict.

SVM provides the worst results of the three types of models used in this comparison, while the random forest with unlimited tree depth provides the best. However, the differences are rather small for some of the problems (mainly pagie-1 and vladislavleva-4). As expected, the performance of random forest with the depth of trees limited to 14 is slightly worse than that of unlimited random forest, but the difference is negligible. On the other hand, the smaller random forests are trained much faster, therefore, the RF (14) will be used in the rest of the paper.

3.2 Importance of Features

We have proposed a number of features. Naturally the importance of some of them is larger than the importance of others. In this section, we investigate which of the variables are the most important and therefore provide the most information about the quality of the programs. Such an information about the features is interesting not only for surrogate modeling in GP, but also for the design of better genetic programming operators. One can, for example, mutate more often those terminals that are more important for the quality of the program. We use the feature importance as computed by the random forest algorithm to judge the importance of the variables. This importance is computed [7] as the sum of the decreases in the performance metric used by the tree (mean square error in this case) caused by a given variable divided by the number of trees in the ensemble. Higher values indicate more important features.

To measure the importance, we run the same algorithm as in the previous section, i.e. the non-surrogate baseline with the model trained in each generation. This time, we log the importance of each variable provided by the model. The results of this experiment are summarized in Table 1. We report the four most important features for each of the problems after 5,000 and 10,000 evaluations together with their importance.

We can see that among the most important features are almost always some of the features which relate to the size of the tree – either the height of the tree or the size (len) of the tree. This should not be surprising, as too small trees can have a poor performance.

Other often important features are the counts of some of the primitives. Interestingly, among them the *exp* is the most important feature for the pagie-1 benchmark and one of the two most important features for the nguyen-7 dataset. In these cases, the features are important in the negative sense – programs with *exp* are inferior to programs without it, as none of the benchmarks use this function. Otherwise, the more simple primitives (multiplication, addition, subtraction) seem more important.

The vladislavleva-4 benchmark is interesting – it is the only one, where the importances of the different statistics on constants are among the top five features. The V-F1 feature in this benchmark corresponds to the special function in this benchmark defined as n^{ϵ}, where ϵ is a constant evolved by the algorithm. As this function is important to find the correct solution, it makes sense the number of times it is used is an important feature.

Table 1. The four most important features for each of the benchmarks after a given number of function evaluations. The importance is represented by median importance of each feature computed over 25 independent runs with the random forest regressor with 100 trees with maximum depth of 14. "depth" and "size" are the respective tree features, "const-mean" is the mean value of constants used in the individual and "arg-count" is the number of arguments used by the individual. The rest of the features represent the counts of the respective functions used by the individual.

Bench	Evals	Most important features
keijzer-6	5000	depth (0.397), safesqrt (0.097), mul (0.058), inverse (0.055)
keijzer-6	10000	depth (0.350), safesqrt (0.069), mul (0.050), size (0.050)
pagie-1	5000	exp (0.260), add (0.067), sub (0.060), size (0.060)
pagie-1	10000	exp (0.275), size (0.061), mul (0.058), arg-count (0.050)
nguyen-7	5000	depth (0.134), exp (0.096), safediv (0.068), mul (0.062)
nguyen-7	10000	depth (0.160), exp (0.085), size (0.065), mul (0.064)
vladisl.-4	5000	const-mean (0.150), V-F1 (0.076), depth (0.068), size (0.062)
vladisl.-4	10000	const-mean (0.159), size (0.062), V-F1 (0.059), depth (0.057)

3.3 Algorithm Performance

To test the algorithm, we made 25 runs on each of the benchmarks described above. The results are presented in Fig. 2. The plots show the dependence of the fitness (logarithm of the RMSE) on the number of fitness evaluations. Moreover, the red dotted line shows the p-value of one-sided Mann-Whitney U-test to test the statistical significance of the differences.

The best results were obtained for the keijzer-6 and pagie-1 benchmarks. Here, the surrogate version is able to decrease the number of evaluations needed to find a solution of given quality by almost 50 the evolution (approx. between 2,000 and 7,000 evaluations). After this phase, the baseline version performs similarly on keijzer-6. For pagie-1, the surrogate is better during the whole 15,000 evaluations given as a budget to the algorithm, however, the differences get smaller. For the nguyen-7 benchmark, the median run of the surrogate algorithm is better between 2,000 and 6,000 fitness evaluations. However, the difference is rather small and the standard deviations are large, which means we can draw no definitive conclusion from this experiment.

The performance was the poorest for the vladislavleva-4 benchmark. There is no significant difference between the two algorithms in this case. We believe, there are two reasons for this behavior. First, most of the improvement happens before the surrogate modelling is even enabled – the baseline converges fast in the first 1,000 evaluations and does not improve much further. It may indicate that the problem becomes too difficult for the simple GP used in this work. Second, the performance of the model, as indicated by the tests in the previous sections was quite poor for this benchmark, which also may affect the results in a negative way.

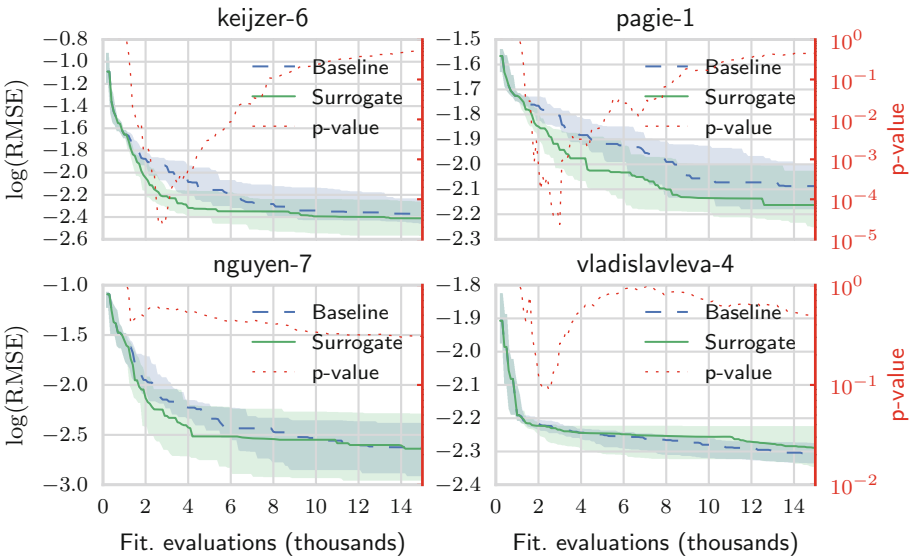

Fig. 2. The convergence rate of the algorithm on the four selected benchmarks. The lines represent the median of 25 runs, the shaded areas represent the first and third quartile. On the right axis, the red dotted line represents the p-value of the one-sided Mann-Whitney U-test computed after each 100 evaluations.

4 Conclusions and Future Work

We have proposed a simple surrogate-based genetic programming algorithm which provides promising results on the selected benchmark problems. We have shown that the surrogate models are capable of predicting the real fitness value without the need to evaluate the program, only by utilizing some static features. This may help to improve the effectiveness of genetic programming for problems with hard-to-evaluate fitness functions.

We also proposed a basic set of features which can be used for the building of surrogate models in genetic programming and we have evaluated the performance of these features. It seems that one of the most important features is the size of the tree, and among the more important features are the number of times each of the primitives is used. On the other hand, statistical features on the constants and features regarding the arguments actually used by the problem seem less important. In the case of the arguments, it may be caused by the fact that most of the benchmarks use all of the arguments, thus making these feature useless.

The presented approach should be considered mostly a proof of concept. There are definitely still many things that require more attention before it can be successfully used to solve practical tasks. The strategies for replacing the individuals deemed un-promising by the surrogate model can be refined, as well as the method for the selection of the training set from the archive – selecting a diverse set of samples instead of a random one may lead to better models.

Acknowledgments. This work is supported by Czech Science Foundation project no. P103-15-19877S.

References

1. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
2. De Rainville, F.-M., Fortin, F.-A., Gardner, M.-A., Parizeau, M., Gagné, C.: DEAP: a python framework for evolutionary algorithms. In: Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO 2012, pp. 85–92. ACM, New York (2012)
3. Hildebrandt, T., Branke, J.: On using surrogates with genetic programming. Evol. Comput. **23**(3), 343–367 (2015)
4. Jin, Y.: Surrogate-assisted evolutionary computation: recent advances and future challenges. Swarm Evol. Comput. **1**(2), 61–70 (2011)
5. Křen, T., Pilat, M., Neruda, R.: Evolving workflow graphs using typed genetic programming. In: 2015 IEEE Symposium Series on Computational Intelligence, pp. 1407–1414, December 2015
6. Li, R., Emmerich, M., Eggermont, J., Bovenkamp, E., Back, T., Dijkstra, J., Reiber, J.: Metamodel-assisted mixed integer evolution strategies and their application to intravascular ultrasound image analysis. In: IEEE Congress on Evolutionary Computation, 2008. CEC 2008 (IEEE World Congress on Computational Intelligence), pp. 2764–2771, June 2008
7. Louppe, G., Wehenkel, L., Sutera, A., Geurts, P.: Understanding variable importances in forests of randomized trees. In: Burges, C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K. (eds.) Advances in Neural Information Processing Systems, vol. 26, pp. 431–439. Curran Associates Inc., Red Hook (2013)
8. McDermott, J., White, D.R., Luke, S., Manzoni, L., Castelli, M., Vanneschi, L., Jaśkowski, W., Krawiec, K., Harper, R., Jong, K.D., O'Reilly, U.-M.: Genetic programming needs better benchmarks. In: Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference, pp. 791–798. ACM, Philadelphia (2012)
9. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
10. Smola, A., Vapnik, V.: Support vector regression machines. Adv. Neural Inf. Process. Syst. **9**, 155–161 (1997)
11. White, D.R., McDermott, J., Castelli, M., Manzoni, L., Goldman, B.W., Kronberger, G., Jaśkowski, W., O'Reilly, U.-M., Luke, S.: Better GP benchmarks: community survey results and proposals. Genet. Prog. Evol. Mach. **14**, 3–29 (2013)