GP with Ranging-Binding Technique for Symbolic Regression

Wen-Zhong Fang
Taiwan Evolutionary Intelligence Laboratory
Department of Electrical Engineering
National Taiwan University
r10921071@ntu.edu.tw

Chi-Hsien Chang
Taiwan Evolutionary Intelligence Laboratory
Department of Electrical Engineering
National Taiwan University
d07921004@ntu.edu.tw

Jung-Chun Liu
Taiwan Evolutionary Intelligence Laboratory
Department of Electrical Engineering
National Taiwan University
r10921043@ntu.edu.tw

Tian-Li Yu
Taiwan Evolutionary Intelligence Laboratory
Department of Electrical Engineering
National Taiwan University
tianliyu@ntu.edu.tw

ABSTRACT

This paper proposes a model-based genetic programming algorithm for symbolic regression, called the ranging-binding genetic programming algorithm (RBGP). The goal is to allow offspring to retain the superiority of their promising parents during evolution. Inspired by the concept of model building, RBGP makes use of syntactic information and semantics information in a program to capture the hidden patterns. When compared with GP-GOMEA, ellynGP, and gplearn, RBGP outperformed the others on average in the Penn machine learning benchmarks, RBGP achieving statistically significant improvements over all other methods on 44% of the problems.

CCS CONCEPTS

• **Software and its engineering → Genetic programming**.

KEYWORDS

genetic programming, evolutionary computation

ACM Reference Format:
Wen-Zhong Fang, Chi-Hsien Chang, Jung-Chun Liu, and Tian-Li Yu. 2023. GP with Ranging-Binding Technique for Symbolic Regression. In *Genetic and Evolutionary Computation Conference Companion (GECCO '23 Companion), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3583133.3590605

1 INTRODUCTION

Genetic programming (GP) is one of the evolutionary computation algorithms proposed by Koza [5]. GP and genetic algorithms (GAs) are similar frameworks: Both of them are population-based black-box optimization methods. Unlike GAs, GP evolves the encodings of programs, instead of decision variables.

GP aims to let computers write programs by themselves. Since its debut, GP has been widely applied in the field of symbolic regression [3, 6, 18], which is slightly different from numerical regression. Numerical regression aims to find the best parameters for a given fixed equation to fit the predicted outputs of the equation to the targets. Yet, the choice of the given equation often requires prior knowledge and expertise. In contrast, symbolic regression requires less prior knowledge and instead uses a set of basic operators such as +, −, ×, ÷, exp, and sin, and then finds a good (if not the best) combination from the set to fit the target.

After the 90s, GAs have developed along the direction of model-building genetic algorithms (MBGAs). DSMGA-II [2, 4] and LT-GOMEA [1, 15, 16] are two branches of state of the art (SOTA) in MBGAs. The concept is to learn the hidden patterns among potential chromosomes by machine learning techniques. Then, by utilizing these hidden patterns, MBGAs can generate potentially better offspring during the recombination process. We believe that GP benefits from a similar strategy.

This paper aims to develop model-based genetic programming (MBGP) for symbolic regression, called ranging-binding genetic programming algorithm (RBGP), which combines the concepts of model building and symbolic regression genetic programming (SRGP). During the evolution of SRGP, the offspring generated by recombination of its parents with better fitness may not be able to retain the superior characteristics [9, 10]. In order to preserve the superiority of the parents as much as possible, syntactic and semantic information of a program are utilized.

The proposed algorithm shows stronger optimization ability on 38 problems out of real-world datasets on average than GP-GOMEA [17], ellynGP [7], and gplearn [13]. The rest of this paper is organized as follows. Section 2 briefly introduces the GP framework and variants. Section 3 explains the proposed range and binding MBGP. Section 4 describes experiment settings and results in comparison to GP-GOMEA, ellynGP, and gplearn. In addition, a discussion of the findings is given. Finally, section 5 makes a conclusion.

2 RANGING-BINDING GENETIC PROGRAMMING

In the following section, the details of the proposed MBGP algorithm are described. Firstly, the framework of the algorithm is presented. Then, the two crucial components, ranging and binding, are presented. Essentially, the RBGP algorithm is built upon the SGP framework, incorporating binding and ranging to exploit the syntax and semantics information.

ACM ISBN 979-8-4007-0120-7/23/07.
https://doi.org/10.1145/3583133.3590605

2.1 The framework of RBGP

The proposed method has three main components. The first is to determine the frequency of each two-layer function. The second is to select cut points for crossover without breaking commonly used two-layer functions. The third is to pair programs for crossover by aligning the range of predicted output with the target range. In more detail, a cut point can separate a program tree into two parts during the crossover. One of the parts is to be remained, and the other part, which is also a valid subtree of the program, is the part to be swapped. The subtree chosen from other programs in the population with a narrower output range is selected to be combined with the remaining part when the program's output range is wider than the target value range, and a subtree with a wider output range is chosen otherwise. The algorithm flowchart is shown in Figure 1.

Algorithm 1: RBGP framework

```
$\mathcal{P}$ : population, $C$ : cut points for each program,
$O$ : offspring, $\mathcal{B}$ : binding counter information,
$\mathcal{R}$ : ranging information

randomly initialize population $\mathcal{P}$
while not meet the terminal condition do
    $\mathcal{B} \leftarrow$ UpdateBindingInformation($\mathcal{P}$)
    $C \leftarrow$ DecideCutPoint($\mathcal{P}, \mathcal{B}$)
    $\mathcal{R} \leftarrow$ UpdateRangingInformation($\mathcal{P}$, $C$)
    for $i \leftarrow 1$ to $|\mathcal{P}|$ do
        if should do the range crossover then
            $O_i \leftarrow$ RangingCrossover($\mathcal{P}_i$, $C_i$, $\mathcal{R}$)
        else
            $O_i \leftarrow$ do other SGP operators
        end
    end
    $\mathcal{P} \leftarrow (\lambda + \mu)$-Selection($\mathcal{P}$, $O$)
end
return the best program in $\mathcal{P}$
```

Algorithm 1 demonstrates the pseudo-code of the proposed algorithm. The detail of binding-related operators and ranging-related operators is introduced in the following subsection. In this paper, the population is denoted by $\mathcal{P}$, and $\mathcal{P}_i$ is the i-th program in the population. Furthermore, the frequency information of two-layer functions is denoted by $\mathcal{B}$, and the cut point information for each program in one generation is denoted by C. O are offsprings, which are candidates for the next generation. The ranging information is denoted by $\mathcal{R}$. Furthermore, The ranging information is denoted by $\mathcal{R}$. Furthermore, $\mathcal{R}_{i,d}$ is the ratio between the range of output of i-th program and the range of target value , and $\mathcal{R}_{i,s}$ is the ratio between the range of terminal variable at a specific dimension and the range of the output of the subtree which is decided by the cut point of i-th program.

Each iteration of the "while loop" is one generation in the proposed method. Each generation is based on the SGP framework, with each program choosing the operator such as mutation, crossover at random to generate offspring. In our algorithm, crossover based on ranging is one of the operators that can be chosen with high probability. After generating all offspring, the $(\lambda + \mu)$-selection is adopted to get the population in the next generation.

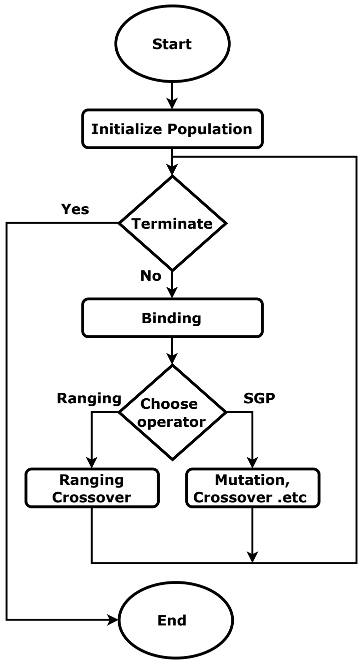

Figure 1: The algorithm flowchart of RBGP

3 TEST PROBLEMS AND EXPERIMENTS

In this section, our proposed RBGP with and without the binding mechanism are compared with GP-GOMEA, ellynGP and gplearn. Then, benchmark problems and experiment settings used in this paper are described. The empirical results are shown and discussed.

3.1 Test Problems

Penn machine learning benchmarks (PMLB) [8, 12] is a collection of benchmark datasets for supervised machine learning including classification and symbolic regression algorithms. PMLB collects ground truth problems and black-box problems. The ground truth problems include two sources, the Feynman symbolic regression database, and the ODE-Strogatz repository. The former comes from Feynman Lectures on Physics [11], representing static physical systems. The latter comes from Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering [14], which include non-linear and chaotic dynamical problems, such as bacterial respiration and models for gliders. On the other hand, black-box problems are collected from diverse domains such as health information, business, technology, *etc.* In this paper, 38 problems in PMLB are used for experiments, and problem names are shown in Table 1.

3.2 Experiment settings

In this paper, experiments are conducted in two scenarios. The first is the black-box scenario. In this scenario, we split the training set and testing set at a ratio of 75%/25%, and use k-fold validation to select the best model in the training set and take the fitness on the testing set as the result. The second is the ground-truth situation, in which we take all data points as input and take the fitness of the final evolved model as the result. The empirical results are the average fitness of the best program after running each method over 100 independent runs. The parameter settings for each of the four methods in each run are as follows: population size of 500, 50 generations, 5-fold cross-validation, terminal set consisting of variables for all dimensions, and function set consisting of add, sub, mul, div*, log*, exp*, sqrt* and pow2.

The fitness is defined as the mean absolute error between target value points and program output points by executing with input data. The lower fitness means the model is fitter in this definition. Furthermore, some functions we used are protected versions. The definitions of the protected version of those functions are as Equation 1 to 4.

$$div^*(x,y) = \begin{cases} \frac{x}{y}, & \text{if } |y| \geq 0.001 \\ 1, & \text{otherwise} \end{cases} \tag{1}$$

$$log^*(x) = \begin{cases} log(x) & \text{if } x \geq 0.001 \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$exp^*(x) = \begin{cases} exp(x), & \text{if } exp(x) \leq 10^{10} \\ 10^{10}, & \text{otherwise} \end{cases} \tag{3}$$

$$sqrt^*(x) = sqrt(|x|). \tag{4}$$

3.3 Results and Discussions

The results of the black-box scenario are shown in Table 2 and the results of the ground-truth scenario are shown in Table 3. On average, RBGP without binding results in a lower average fitness value in both scenarios, showing a stronger optimization ability than three SOTA. In addition, in the black-box scenario, RBGP without binding shows a more generalized ability with lower MSE on the testing data. Then the results also show that RBGP generally improves performance, increasing the first rank on 52% of the problems to 71%.

Table 4 displays the outcomes of the t-test that was carried out between RBGP and each of the other compared methods, using a significance level of 0.05. The null hypothesis in this t-test is that there is no significant difference between the means of the two groups being compared. The results are divided into three categories: Significantly lowest, where RBGP fitness is significantly lower than the other methods. No significant difference, where RBGP fitness does not show a significant difference compared with at least one other method. Not significantly lowest, where RBGP fitness is higher than at least one other method.

4 CONCLUSION

This paper proposed a model-based SRGP with ranging and binding mechanisms to utilize the syntactic and semantic information of programs. For semantics, the range difference ratio between the output of the program and the target can be information for pairing. For syntax, similar to the concept of ADF, two-layer functions were viewed as binding units and decreased the probability of separating commonly used binding units. The experiments showed

Table 1: Test problems

No.	Name	No.	Name
f1	strogatz_shearflow2	f20	fri_c3_500_5
f2	strogatz_shearflow1	f21	fri_c3_250_5
f3	strogatz_predprey2	f22	fri_c3_100_5
f4	strogatz_predprey1	f23	rmftsa_ladata
f5	strogatz_lv2	f24	fri_c2_250_10
f6	strogatz_lv1	f25	chatfield_4
f7	strogatz_glider2	f26	fri_c0_500_10
f8	strogatz_glider1	f27	fri_c1_250_10
f9	strogatz_barmag2	f28	fri_c3_500_10
f10	strogatz_barmag1	f29	feynman_test_9
f11	strogatz_bacres2	f30	feynman_test_7
f12	strogatz_bacres1	f31	feynman_test_6
f13	visualizing_galaxy	f32	feynman_I_10_7
f14	sleuth_ex1605	f33	feynman_I_12_5
f15	fri_c1_100_5	f34	feynman_I_15_10
f16	fri_c0_500_5	f35	feynman_I_15_3x
f17	fri_c1_500_5	f36	feynman_II_3_24
f18	fri_c3_1000_5	f37	feynman_II_37_1
f19	fri_c0_100_5	f38	feynman_II_35_18

Table 2: Black-box scenario empirical results

No.	Comparisons			Ours	
	GP-GOMEA	ellynGP	gplearn	RBGP - binding	RBGP
f1	0.448	0.612	0.146	0.092	**0.085**
f2	**0.105**	0.225	0.192	0.133	0.129
f3	1.670	0.529	0.385	0.374	**0.224**
f4	1.162	0.469	0.570	0.443	**0.255**
f5	0.299	0.218	0.146	0.107	**0.073**
f6	0.473	0.634	0.211	0.132	**0.076**
f7	0.860	0.323	0.335	0.251	**0.242**
f8	2.550	0.671	0.668	**0.615**	0.633
f9	0.143	0.222	0.120	0.072	**0.071**
f10	1.299	0.140	0.103	**0.063**	0.089
f11	2.305	0.723	0.507	0.554	**0.407**
f12	1.854	1.302	0.602	0.593	**0.530**
f13	90.093	227.590	88.113	84.969	**81.333**
f14	22.111	19.974	**12.213**	12.513	12.647
f15	1.299	0.812	0.798	**0.783**	0.789
f16	1.178	**0.792**	0.824	0.805	0.807
f17	1.496	**0.800**	0.824	0.806	0.816
f18	2.701	0.793	0.795	0.787	**0.769**
f19	1.688	0.856	**0.833**	0.850	0.835
f20	1.213	**0.781**	0.805	0.804	0.785
f21	1.903	**0.786**	0.796	0.798	0.791
f22	1.326	0.792	0.785	**0.743**	0.764
f23	2.048	1.920	1.834	1.821	**1.807**
f24	1.246	0.795	0.845	**0.767**	0.775
f25	364.006	33.566	34.232	**33.316**	34.606
f26	1.473	0.821	0.815	0.845	**0.812**
f27	1.199	0.825	0.838	0.836	**0.819**
f28	1.289	0.769	0.792	0.780	**0.763**

Table 3: Ground-truth scenario empirical results

No.	Comparisons			Ours	
	GP-GOMEA	ellynGP	gplearn	RBGP - binding	RBGP
f29	2080.423	1887.389	1919.632	1834.754	**1443.437**
f30	1.896	1.020	1.007	1.035	**1.006**
f31	0.288	0.238	0.211	0.219	**0.195**
f32	1.298	**1.041**	1.045	1.046	1.052
f33	10.189	4.291	4.140	4.239	**4.001**
f34	2.820	1.736	1.740	1.728	**1.713**
f35	1.972	1.363	**1.356**	1.357	1.393
f36	0.267	0.036	0.036	0.036	**0.032**
f37	30.796	19.020	17.799	17.695	**17.439**
f38	0.363	**0.260**	0.265	0.267	0.264

Table 4: T-test results

Category	Problem No.
Significantly lowest	f1,f3,f4,f5,f6,f7, f8,f9,f11,f12,f13,f18, f22,f24,f29,f31,f33
No significant difference	f10,f15,f17,f19,f20,f21, f23,f25,f26,f27,f28,f30, f32,f34,f36,f37,f38
Not significantly lowest	f2,f14,f16,f35

that both RBGP without binding and RBGP outperform the SOTA GP algorithms on 38 problems out of PMLB on average both in the black-box scenario and the ground-truth scenario. Furthermore, RBGP achieved statistically significant improvements over all other methods on 44% of the problems.

ACKNOWLEDGMENTS

The authors would like to thank the support by the National Science and Technology Council in Taiwan under grant No. NSTC 111-2221-E002-189 and National Taiwan University under grant No. 112L891103.

REFERENCES

[1] Peter A.N. Bosman and Dirk Thierens. 2012. Linkage Neighbors, Optimal Mixing and Forced Improvements in Genetic Algorithms. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation* (Philadelphia, Pennsylvania, USA) *(GECCO '12)*. Association for Computing Machinery, New York, NY, USA, 585–592. https://doi.org/10.1145/2330163.2330247

[2] Ping-Lin Chen, Chun-Jen Peng, Chang-Yi Lu, and Tian-Li Yu. 2017. Two-Edge Graphical Linkage Model for DSMGA-II. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Berlin, Germany) *(GECCO '17)*. Association for Computing Machinery, New York, NY, USA, 745–752. https://doi.org/10.1145/3071178.3071236

[3] Vinícius Veloso De Melo. 2014. Kaizen Programming. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation* (Vancouver, BC, Canada) *(GECCO '14)*. Association for Computing Machinery, New York, NY, USA, 895–902. https://doi.org/10.1145/2576768.2598264

[4] Shih-Huan Hsu and Tian-Li Yu. 2015. Optimization by Pairwise Linkage Detection, Incremental Linkage Set, and Restricted / Back Mixing: DSMGA-II *(GECCO '15)*. Association for Computing Machinery, New York, NY, USA, 519–526. https://doi.org/10.1145/2739480.2754737

[5] John R Koza. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA.

[6] Krzysztof Krawiec. 2016. *Program synthesis.* Springer International Publishing, Cham, 1–19. https://doi.org/10.1007/978-3-319-27565-9_1

[7] William La Cava, Thomas Helmuth, Lee Spector, and Kourosh Danai. 2015. Genetic Programming with Epigenetic Local Search. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* (Madrid, Spain) *(GECCO '15)*. Association for Computing Machinery, New York, NY, USA, 1055–1062. https://doi.org/10.1145/2739480.2754763

[8] William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabricio de Franca, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason Moore. 2021. Contemporary Symbolic Regression Methods and their Relative Performance. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, J. Vanschoren and S. Yeung (Eds.), Vol. 1. https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/file/c0c7c76d30bd3dcaefc96f40275bdc0a-Paper-round1.pdf

[9] Quang Uy Nguyen, Xuan Hoai Nguyen, and Michael O'Neill. 2009. Semantic Aware Crossover for Genetic Programming: The Case for Real-Valued Function Regression. In *Genetic Programming*, Leonardo Vanneschi, Steven Gustafson, Alberto Moraglio, Ivanoe De Falco, and Marc Ebner (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 292–302.

[10] Quang Uy Nguyen, Michael O'Neill, Nguyen Hoai, Robert McKay, and Edgar López. 2009. Semantic Similarity Based Crossover in GP: The Case for Real-Valued Function Regression. 170–181.

[11] Robert B. Leighton Richard P. Feynman and Matthew Sands. 2015. *he Feynman Lectures on Physics, Vol. I: The New Millennium Edition: Mainly Mechanics, Radiation, and Heat.*

[12] Joseph D Romano, Trang T Le, William La Cava, John T Gregg, Daniel J Goldberg, Praneel Chakraborty, Natasha L Ray, Daniel Himmelstein, Weixuan Fu, and Jason H Moore. 2021. PMLB v1.0: an open source dataset collection for benchmarking machine learning methods. *arXiv preprint arXiv:2012.00058v2* (2021).

[13] Trevor Stephens. 2016. Genetic Programming in Python, with a scikit-learn inspired API: gplearn.

[14] Steven H. Strogatz. 2000. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering.* Westview Press.

[15] Dirk Thierens. 2010. The Linkage Tree Genetic Algorithm. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature: Part I* (Kraków, Poland) *(PPSN'10)*. Springer-Verlag, Berlin, Heidelberg, 264–273.

[16] Dirk Thierens and Peter A.N. Bosman. 2011. Optimal Mixing Evolutionary Algorithms. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (Dublin, Ireland) *(GECCO '11)*. Association for Computing Machinery, New York, NY, USA, 617–624. https://doi.org/10.1145/2001576.2001661

[17] Marco Virgolin, Tanja. Alderliesten, Cees Witteveen, and Peter. Bosman. 2020. Improving Model-Based Genetic Programming for Symbolic Regression of Small Expressions. *Evolutionary Computation* 29 (06 2020), 1–27. https://doi.org/10.1162/evco_a_00278

[18] Jinghui Zhong, Liang Feng, Wentong Cai, and Yew Ong. 2018. Multifactorial Genetic Programming for Symbolic Regression Problems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* PP (07 2018), 1–14. https://doi.org/10.1109/TSMC.2018.2853719