Latest updates: https://dl.acm.org/doi/10.1145/3712255.3726697

POSTER

TinyverseGP: Towards a Modular Cross-domain Benchmarking Framework for Genetic Programming

ROMAN KALKREUTH, RWTH Aachen University, Aachen, Nordrhein-Westfalen, Germany

FABRÍCIO OLIVETTI DE FRANÇA, Federal University of ABC, Santo Andre, SP, Brazil

ANJA JANKOVIĆ, RWTH Aachen University, Aachen, Nordrhein-Westfalen, Germany

MARIE ANASTACIO, RWTH Aachen University, Aachen, Nordrhein-Westfalen, Germany

JULIAN DIERKES, RWTH Aachen University, Aachen, Nordrhein-Westfalen, Germany

ZDENĚK VAŠÍČEK, Brno University of Technology, Brno, South Moravian Region, Czech Republic

View all

Open Access Support provided by:

RWTH Aachen University

Federal University of ABC

Brno University of Technology

TinyverseGP: Towards a Modular Cross-domain Benchmarking Framework for Genetic Programming

Roman Kalkreuth
RWTH Aachen University
Aachen, Germany
kalkreuth@aim.rwth-aachen.de

Fabricio Olivetti de França
Universida de Federal do ABC
Santo André, Brazil
folivetti@ufabc.edu.br

Julian Dierkes
RWTH Aachen University
Aachen, Germany
dierkes@aim.rwth-aachen.de

Marie Anastacio
RWTH Aachen University
Aachen, Germany
anastacio@aim.rwth-aachen.de

Anja Jankovic
RWTH Aachen University
Aachen, Germany
jankovic@aim.rwth-aachen.de

Zdenek Vasicek
Brno University of Technology
Brno, Czech Republic
vasicek@fit.vutbr.cz

Holger Hoos
RWTH Aachen University
Aachen, Germany
hh@aim.rwth-aachen.de

Abstract

Over the years, genetic programming (GP) has evolved, with many proposed variations, especially in how they represent a solution. Being essentially a program synthesis algorithm, it is capable of tackling multiple problem domains. Current benchmarking initiatives are fragmented, as the different representations are not compared with each other and their performance is not measured across the different domains. In this work, we propose a unified framework, dubbed TinyverseGP (inspired by tinyGP), which provides support to multiple representations and problem domains, including symbolic regression, logic synthesis and policy search.

Keywords

Genetic Programming, Implementation, Benchmarking, Symbolic Regression, Logic Synthesis, Python

1 Introduction

Genetic programming (GP) is an evolutionary algorithmic paradigm from the field of randomised search heuristics. It was first introduced by Cramer [4] for the automated discovery of programs, and later popularised by Koza, with his GP-driven search to hierarchical structures [10], which led to meaningful applications in symbolic regression, one of the major problem domain of GP. Other problem domains have emerged, notably classification and digital circuit design, and revealed the need for different solution encodings, such as linear [15], graph-based [13] and grammar-guided [16] representations. The interplay between new representations and applications has led to a versatile landscape, but also to fragmentation and encapsulation, hindering the development of unified and cross-domain knowledge. Obsolete real-world problems have been proposed as benchmarks and bundled in domain-specific benchmark suites for program synthesis, symbolic regression and logic synthesis. However, cross-domain evaluation entails a large implementation overhead.

In this work, we take a step towards a unified framework, combining representation models and benchmarks. Our approach, dubbed *TinyverseGP*, is inspired by tinyGP [17], following the philosophy of keeping a minimalist implementation of each variant. The shared concepts are kept in a common library, making it easy to integrate new representations and to support different problem domains. We intend to create a collaborative space to incorporate the different variants and problem domains, facilitating the understanding of the characteristics of each representation. As a starting point, we showcase the integration of a tree-based and graph-based GP in the domains of logic synthesis, symbolic regression, and policy search.

2 Related Work

Genetic Programming. GP is an evolutionary search methodology, located in the wider field of randomised search heuristics, and was originally proposed for the synthesis of computer programs. GP as a heuristic paradigm aims at *evolving* a population of candidate programs towards a functionally specified solution for a given search problem. Fundamental to GP-driven search is the iterative transformation of a population of candidates into new populations consisting of programs with an improved fitness score. To represent a program, GP traditionally uses parse trees, which have been inspired by LISP S-expressions. Commonly used variation operators applied to the traditional tree-based representation

are subtree crossover and mutation. Subtree crossover swaps subtrees between two trees selected for recombination, while subtree mutation exchanges a subtree with a randomly generated one.

Problem Domains in GP. Because the very essence of GP is to search for a computer program, it can be readily applied to different machine learning tasks that would normally require major adaptations. In most cases, merely a grammatical specification for the respective task is required to seek a solution with GP paradigm. For example, in regression and classification tasks, the grammar can be composed of mathematical operators and conditional branching. Likewise, in policy search, GP can search for the function that estimates the value of a given action at the current state, reverting to a symbolic regression problem, or a computer program that will return the optimal action for that particular state, thus incorporating branching and loops in the grammar. Additionally, GP can also search for Boolean functions, if we constrain the grammar to the corresponding set of operators, a problem that is often performed with heuristics or exhaustive search.

Symbolic Regression. Symbolic regression (SR) [10] searches for a mathematical expression representing a regression model that accurately and compactly describes the data. Given a set of m points $\{x_i, y_i\}_{i=1...m}$, this task seeks the function $f(x) \approx y$.

Logic Synthesis. Logic synthesis (LS) [6], as approached with heuristic methods, can be defined as a black-box optimisation task that aims at the synthesis of Boolean expressions. As in symbolic regression, the expression to be synthesised has to match the correct input-output mapping of a (Boolean) function. LS tackled with GP predominantly aims at synthesis of Boolean expressions that match the correct input-output mapping of a Boolean functions.

Policy Search. Policy search focuses on finding decision-making strategies that maximise performance through interaction with an environment [18]. The environment is formalised as a Markov decision process (MDP) $\mathcal{M} := (\mathcal{S}, \mathcal{A}, p, r, \rho_0, \gamma)$, with state space $\mathcal{S}$, action space $\mathcal{A}$, unknown transition probability distribution $\rho : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$, reward function $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, distribution of the initial state $\rho_0 : \mathcal{S} \mapsto \mathbb{R}$ and discount rate $\gamma \in (0, 1)$. A policy $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ assigns a probability to each action for a given state. Interacting with the MDP, π collects episodes $\tau = (s_0, a_0, r_1, s_1, \cdots, s_T)$ over time steps $t = 0, \cdots, T$. The fitness of a policy is measured as the return $\mathbb{E}_{\tau \sim \pi}[\sum_{t=1}^{T} \gamma^t \cdot r_t]$.

3 Benchmarking in Genetic Programming

GP benchmarks have, for many years, been criticised by their lack of rigour, caused by the fragmentation of the field [12]. It has been argued that a first step towards better benchmarks is the standardisation to ensure a challenging environment and fair comparison between different flavours of GP as well as alternatives from machine learning. A major step forward in GP benchmarking has been made by the proposal of two benchmark suites for program synthesis, PSB1 [8] and PSB2 [7], which cover a diverse set of introductory and college-level coding problems. Moreover, benchmark suites for other problem domains, such as SRBench [14] and GBFS [9], have emerged, and new problem domains have been discovered in recent years [1].

SRBench: A Living Benchmark for Symbolic Regression. Orzechowski et al. [14] compared a selection of Symbolic Regression (SR) with traditional machine learning techniques, addressing a common criticism regarding the lack of comparison between these approaches. Their results indicated that SR was a competitive alternative to opaque models in terms accuracy, with the benefit of returning interpretable models. La Cava et al. [11] advanced this effort even further by incorporating new algorithms into the benchmark, dubbed SRBench, and proposing a collaborative environment facilitating the benchmarking of new algorithms using a common Python interface and a verified installation environment that enabled external peers to replicate the benchmark. This effort spanned multiple competitions, one of which led to a publication highlighting the challenges still faced by the field of SR in general, not only GP [5].

GBFS: General Boolean Function Benchmark Suite. The General Boolean Function Benchmark Suite (GBFS) [9] is a highly versatile benchmark suite for logic synthesis proposed with the intention to facilitate comprehensive assessment of the performance of GP models as well as to simplify reproducibility of existing results in this problem domain. GBFS covers a set of 29 problems carefully selected from seven different types of Boolean functions: arithmetic, transmission, comparison, counting, mixed, parity and cryptography. Most of the problems are characterised by having multiple outputs, such as the digital adder or multiplier function.

Reinforcement and Policy Learning. A variety of benchmarks have been developed and proposed for Reinforcement Learning (RL) in recent years, tailored to diverse use cases such as games, planning and robotics [2, 3, 20]. A widely used platform is Gymnasium [19], which provides diverse environments aimed at evaluating the core capabilities of RL algorithms. Additionally, the Atari Learning Environment (ALE) [2] offers a standardised and challenging testbed for RL agents. Benchmarking GP on Gymnasium has already been explored in [1].

4 The Proposed Framework

Major Motivation. The landscape of existing GP frameworks can be considered fragmented, in terms of representation as well as application domain. We acknowledge the implementation efforts made over time to facilitate broad use of GP. However, existing implementations are meant to provide an end-user experience, thus reducing the burden of choosing the most suitable representation for each task. Notably, there is clearly a gap on benchmarking and understanding the different representations proposed so far across different domains, as evidenced by the paper *Genetic programming needs better benchmarks* [12], which received the 2022 SIGEVO Impact Award and led to initiatives such as SRBench [11] and GBFS [9], which provide tight guidelines to ensure fairness of comparison among the competing algorithms.

Key Features and Properties. For the first version, we concentrated on the development of the following features and properties that represent the fundamental infrastructure of TinyverseGP. In general, we pursue object-oriented design, to ensure that the design goals communicated in the previous subsection can be adhered to.

- **Light-weight representation modules:** Each representation derives from a model base class, which promotes uniformity among different representations. Each representation is implemented within a tiny module that handles initialisation, decoding, breeding and evaluation of candidate programs concerning the requirements of the respective model.
- **Broad range of applications:** TinyverseGP can already be applied to several problems that differ greatly and require different methods on how to evaluate an evolved program, showing the versatility of this tool.
- **Benchmarking support:** We provide an interface to SRBench and GBFS, covering two problem domains in combinational synthesis. Additionally, we provide an interface to policy learning benchmarks from Gymnasium, which is accomplished with an implementation of an agent class that bridges the gap between the respective GP model and the environment.

5 Top-level Architecture

TinyverseGP hybridises object-oriented design with modularisation, two programming paradigms commonly used to establish an efficient reusable architecture. With modularisation, we also facilitate encapsulation of diverse techniques and methodologies, to maintain an organised and interpretable code structure, as TinyverseGP is expected to become larger and more heterogeneous in the future. We therefore leverage modularisation and object-oriented features to implement a design that is well-equipped to apply various GP models to a diverse set of benchmarks.

Figure 1 provides an overview of the modular architecture of TinyverseGP. The GP module represents the core of our framework, and the current architecture allows further integration with less effort; existing Python implementations of other GP models can be easily integrated into the already existing framework model, which already provides essential features to handle function and terminal sets, as well as the hyperparameter and model configuration.

The fundamental architecture is illustrated in Figure 2 on the class level. The most common problem type that is tackled with the GP paradigm are black-box problems, where known input-output pairs are given in the respective training dataset. Policy search problems, on the other hand, are commonly applied to environments where an agent executes actions in accordance to a policy for which he receives rewards. For this type of problem we provide an agent class that receives a candidate policy from the respective GP model, as illustrated in Figure 3.

6 Discussion, Vision and Future Work

The primary intention behind TinyverseGP is to take the first step towards a community-driven framework that simplifies the process of performing comparative studies among various representations of GP, thereby promoting best practices for benchmarking randomised search heuristics within this field. We acknowledge the efforts that have already been made in this direction by the proposal of various benchmark suites for major GP problem domains. We believe that facilitating cross-domain benchmarking that leverages previous efforts for specific domains is a natural next step and will bring the field forward. However, unifying both the representation models and a diverse landscape of application domains in one
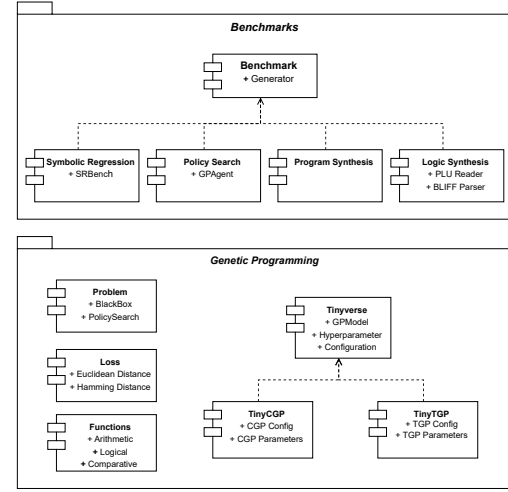

Figure 1: Modular top-level view of TinyverseGP

framework is a task that requires well-considered design choices to be equipped for further expansion. Our vision for TinyverseGP can be summarised as follows:

- **Community driven development:** The source code will be open-sourced and freely available.
- **Easy to extend:** By providing a common API requiring just a few methods, the framework can be easily extended without the need for understanding the entire codebase.
- **Simplicity of implementation:** The minimalistic approach allows us a direct comparison between representations without the influence of other external agents that can influence the performance of the algorithm, such as different selection mechanisms, island models and local search.

Table 1: Planned support for different benchmarks and problem domains.

Problem Domains	Support	Benchmarks
Logic Synthesis	✓	Classic, GBFS
Symbolic Regression	(✓)	Classic, SRBench, Feynman
Policy Search	(✓)	Gymnasium
Program Synthesis	✗	Leet Code, PBS 1 & 2, SyGuS

Planned Features and Extensions. Two specific extensions can be considered as natural next steps for the project. Firstly, we plan to include linear-based and grammar-based GP in the context of our tiny module approach, enabling a scope on the representation level that can be used for a first broad comparative study performed with TinyverseGP. Secondly, on the domain level, we will concentrate on the integration of program synthesis benchmarks by providing an interface for the General Program Synthesis Benchmark Suites (PSB1 and PSB2) proposed by Helmuth *et al.* [7, 8]; a summary is given in Table 1, where supported benchmarks are
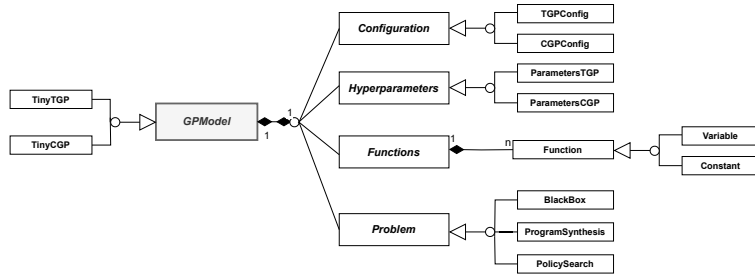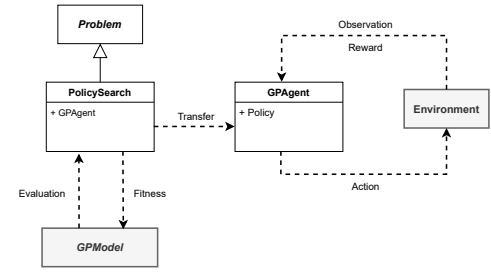
Figure 2: High-level architecture of TinyverseGP

Figure 3: Policy search with TinyverseGP

marked using ✓, partially supported ones using (✓), and those under development using ✗.

7 Conclusion

We have proposed the first prototype of a modular cross-domain benchmarking framework for GP that is meant to be the blueprint for a large and scalable framework. TinyverseGP can already be used to evaluate various benchmarks from three problem domains to which GP is highly applicable, including the underrepresented domain of policy search. to achieve a better balance against overused benchmarks.

8 Resources

The source code of the actively developed main branch, test scripts and a handbook are available on GitHub.[1] The version proposed in this paper is provided in a locked branch.[2]

9 Acknowledgements

References

[1] R. Amaral, A. Ianta, C. Bayer, R. J. Smith, and M. I. Heywood. 2022. Benchmarking Genetic Programming in a Multi-Action Reinforcement Learning Locomotion Task. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '22)*. 522–525.

[2] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. 2013. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research* 47 (2013), 253–279.

[3] M. Chevalier-Boisvert, B. Dai, M. Towers, R. Perez-Vicente, L. Willems, S. Lahlou, S. Pal, and P. S. Castro. 2023. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. In *Proceedings of the 37th International Conference on Neural Information Processing Systems (NeurIPS)*. Article 3209.

[4] N. L. Cramer. 1985. A Representation for the Adaptive Generation of Simple Sequential Programs. In *Proceedings of the 1st International Conference on Genetic Algorithms*. 183–187.

[5] F. O. de Franca, M. Virgolin, M. Kommenda, M. S. Majumder, M. Cranmer, G. Espada, L. Ingelse, A. Fonseca, M. Landajuela, B. Petersen, R. Glatt, N. Mundhenk, C. S. Lee, J. D. Hochhalter, D. L. Randall, P. Kamienny, H. Zhang, G. Dick, A. Simon, B. Burlacu, Jaan Kasak, Meera Machado, Casper Wilstrup, and W. G. La Cava. 2024. SRBench++: Principled Benchmarking of Symbolic Regression With Domain-Expert Interpretation. *IEEE Transactions on Evolutionary Computation* (2024).

[6] G. D. Hachtel and F. Somenzi. 1996. *Logic Synthesis and Verification Algorithms*. Kluwer.

[7] T. Helmuth and P. Kelly. 2021. PSB2: The Second Program Synthesis Benchmark Auite. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '21)*. 785–794.

[8] T. Helmuth and L. Spector. 2015. General Program Synthesis Benchmark Suite. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '15)*. 1039–1046.

[9] R. Kalkreuth, Z. Vasícek, J. Husa, D. Vermetten, F. Ye, and T. Bäck. 2023. General Boolean Function Benchmark Suite. In *Proceedings of the 17th ACM/SIGEVO Conference on Foundations of Genetic Algorithms (FOGA '23)*. 84–95.

[10] J. R. Koza. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

[11] W. La Cava, P. Orzechowski, B. Burlacu, F. de Franca, M. Virgolin, Y. Jin, M. Kommenda, and J. Moore. 2021. Contemporary Symbolic Regression Methods and their Relative Performance. In *Proceedings of the 35th International Conference on Neural Information Processing Systems (NeurIPS), Track on Datasets and Benchmarks*. 1–16.

[12] J. McDermott, D. R. White, S. Luke, L. Manzoni, M. Castelli, L. Vanneschi, W. Jaskowski, K. Krawiec, R. Harper, K. De Jong, and U.-M. O'Reilly. 2012. Genetic Programming Needs Better Benchmarks. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '12)*. 791–798.

[13] J. F. Miller and P. Thomson. 2000. Cartesian Genetic Programming. In *Proceedings of the European Conference on Genetic Programming (EuroGP '00)*, Vol. 1802. 121–132.

[14] P. Orzechowski, W. La Cava, and J. H. Moore. 2018. Where Are We Now?: A Large Benchmark Study of Recent Symbolic Regression Methods. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '18)*. 1183–1190.

[15] T. Perkis. 1994. Stack-Based Genetic Programming. In *Proceedings of the 1st IEEE Conference on Evolutionary Computation (CEC)*, Vol. 1. 148–153.

[16] C. Ryan, J. J. Collins, and M. O'Neill. 1998. Grammatical Evolution: Evolving Programs for an Arbitrary Language. In *Proceedings of the European Conference on Genetic Programming (EuroGP '98)*, Vol. 1391. 83–96.

[17] M. Sipper. 2019. Tiny Genetic Programming in Python. https://github.com/moshesipper/tiny_gp.

[18] R. S. Sutton and A. G. Barto. 2018. *Reinforcement Learning: An Introduction*. MIT Press.

[19] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, and O. G. Younis. 2024. Gymnasium: A Standard Interface for Reinforcement Learning Environments. arXiv:2407.17032 [cs.LG]

[20] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, Y. Zhu, and K. Lin. 2020. robosuite: A Modular Simulation Framework and Benchmark for Robot Learning. arXiv:2009.12293 [cs.RO]

---

[1]https://github.com/GPBench/TinyverseGP
[2]https://github.com/GPBench/TinyverseGP/tree/gecco-2025