Genetic Programming Benchmarks: Looking Back and Looking Forward

By James McDermott, School of Computer Science, University of Galway, Ireland james.mcdermott@universityofgalway.ie; Gabriel Kronberger, University of Applied Sciences Upper Austria gkronber@heuristiclab.com; Patryk Orzechowski, University of Pennsylvania, USA and AGH University of Science and Technology, Krakow, Poland patryk.orzechowski@gmail.com; Leonardo Vanneschi, NOVA Information Management School (NOVA IMS), Universidade Nova de Lisboa, Campus de Campolide, 1070-312 Lisboa; Portugal lvanneschi@novaims.unl.pt; Luca Manzoni, Department of Mathematic and Geosciences, University of Trieste, Via Alfonso Valerio 12/1, 34127 Trieste, Italy lmanzoni@units.it; Roman Kalkreuth, Natural Computing Research Group, Leiden Institute of Advanced Computer Science, Leiden University, Netherlands roman.kalkreuth@tu-dortmund.de; Mauro Castelli, NOVA Information Management School (NOVA IMS), Universidade Nova de Lisboa, Campus de Campolide, 1070-312 Lisboa; Portugal mcastelli@novaims.unl.pt.

The top image shows a set of scales, which are intended to bring to mind the ideas of balance and fair experimentation which are the focus of our article on genetic programming benchmarks in this issue. Image by Elena Mozhvilo and made available under the Unsplash license on https://unsplash.com/photos/j06gLuKK0GM.

Introduction

In 2011, something which was latent in discussions of Genetic Programming (GP) was crystallised by Sean Luke in an email to the GP mailing list: “I think GP has a toy problem”. There was a damaging *mismatch* between the problems used to test GP performance in research papers and the real-world problems that researchers actually cared about. This idea was picked up and expanded by a group of GP researchers to become a discussion, a project, and then two papers:

- *Genetic programming needs better benchmarks*, James McDermott, David R White, Sean Luke, Luca Manzoni, Mauro Castelli, Leonardo Vanneschi, Wojciech Jaśkowski, Krzysztof Krawiec, Robin Harper, Kenneth De Jong, Una-May O’Reilly, GECCO 2012.
- *Better GP benchmarks: community survey results and proposals*, David R White, James McDermott, Mauro Castelli, Luca Manzoni, Brian W Goldman, Gabriel Kronberger, Wojciech Jaśkowski, Una-May O’Reilly, Sean Luke, GPEM 2013.

The issues raised in these papers included:

- There is a mismatch between benchmark problems and real-world problems;
- Easy benchmark problems may give misleading information about performance;
- Experimental practice is sometimes inadequate, e.g. in relation to train-test splits and statistical testing;
- GP is highly flexible in many respects, making experimental comparisons difficult.

Neither paper went so far as to curate a new benchmark suite, but the 2013 paper found community support for “blacklisting” certain problems and proposed possible replacements.

On the occasion of the ACM SIGEvo 10-year Impact award for the GECCO 2012 paper, recently awarded at GECCO 2022, we (some of the original authors, and some new ones) take the opportunity to look back and to look forward.

A changed environment

In the years since these publications we have seen interesting trends in GP benchmarking and experimental practice. Many papers have avoided the "blacklist" problems, and have explicitly discussed experimental methods and standards. Several authors have proposed new benchmarks and benchmark suites, as we will discuss. There has been further research on benchmarking issues such as train-test splits in GP (Nicolau et al., 2015). Authors are moving to discard baselines such as "standard GP" (Koza, 1992) and the grammatical evolution equivalent "GE98" (Ryan et al., 1998) which are rather easy to out-perform (Nicolau, 2017). The field is perhaps less inward-looking, with more use of non-GP baselines. In the last few years, neural approaches to program synthesis (e.g. Devlin et al., 2017, Parisotto et al., 2017, Balog et al., 2017, Chen et al., 2018) have become more common, creating a new "competitor" to GP. In general there has been a trend of higher and higher experimental standards and rigour, driven jointly by editors, reviewers and authors.

Increased interoperability has benefited our field. For example, the Scikit-Learn library (Pedregosa et al., 2011) emerged as a *de facto* standard API for many ML problems, and some GP projects have taken advantage of this, e.g. GPLearn (Stephens, 2016) and SRBench (La Cava et al., 2020). This API makes it easy to benchmark GP ML approaches against standard ML approaches.

Although open-source code was common in GP ever since early GP publications by Forsyth (1981) and Koza (1992), a trend towards scientific replicability in recent years has resulted in both source code and open data accompanying the majority of papers now. (A prominent exception is the *Eureqa* symbolic regression product, based on the work of Schmidt and Lipson (2009), which is popular but closed-source.) In principle, source code is not as important as accurate description of algorithms, but in practice, most GP papers cannot specify every detail (and in GP, there are many details) to enable replicability without source code. This impacts on benchmarking as fair evaluation of new methods depends on close re-implementation of previous methods and use of precisely the same data.

Not only the field of GP but the broader fields of AI and ML have seen huge changes. Compute capabilities have greatly increased since 2012, including GPU and cloud

computing. It is amazing to look back and realise that these were not commonplace at the time. The synergy of large compute and large datasets have enabled great progress especially in deep learning, which seems to implicitly create a challenge to GP with each new advance. Meanwhile, benchmarking issues are also being discussed in the broader fields, e.g. with a meta-analysis and taxonomy of generalisation failures in ML, which begins with the comment “[a]dvances in machine learning often evaporate under closer scrutiny or turn out to be less widely applicable than originally hoped” (Liao et al., 2021). New competitions have helped to drive progress, as part of academic conferences and via sites like Kaggle.com.

Why benchmarking?

There are several reasons why benchmarks are useful.

First, benchmarks aim to establish as fair as possible a comparison between methods. This comparison may span different aspects, e.g. accuracy (according to a single or multiple metrics), percentage of solved problems with sufficient accuracy, running time, scalability, number of times a given method outperformed others by some margin. Sometimes different authors compare methods in different ways, which may lead to some approaches appearing inadequate to others. Benchmark suites can try to overcome those challenges by using an isolated environment and providing each of the benchmarked methods the same amount of resources (CPU, RAM, timeout) etc., or a fixed fitness evaluation budget.

Second, benchmark suites can serve to consolidate known experimental results. When taking a widely-used benchmark problem off the shelf, authors can trust that their results can be compared fairly against previously published results on the same problem, with regard to hyper-parameter tuning, resource budget, train-test splits, and other details. This avoids the need for every author to re-implement and re-run previous methods, which is error-prone and time-consuming. So, benchmark suites are good for the environment!

Third, benchmarks take some of the burden from the author. An author may wish both to provide a new and improved *method*, and to demonstrate a real-world *application*. But aiming for both in the same paper presents a problem: dealing with a real application may require many application-specific details. Running multiple previous methods on the same problem may be difficult or impossible for this reason, and these details may not be of interest to most GP-focussed readers and may be more suited to an application-oriented publication venue. Meanwhile, including many results on unrelated problems may distract application-focussed readers. Separating out the application and the method may improve the impact of both.

Fourth, benchmarks can provide a standardised API to be implemented by problems (on one side) and methods (on the other). This makes it much simpler for the end user to switch between different methods and find the one that is the most likely to solve a given problem. The Scikit-Learn API for ML problems is the central example.

Lastly, benchmark suites help prevent the suspicion that the author has cherry-picked those challenges where their method actually shines. They achieve this in three ways. Benchmark suites are (usually) created by a third party, and (usually) already accepted by the community. And they often include a wide diversity of problems. (Christie et al. (2018) warn that performance on some problems can be correlated, so a large number of problems is not sufficient). Also, benchmark suites include problems that span several different levels of problem difficulty and complexity. For benchmark suites where this is true, it is important to remember that solving difficult problems well is more meaningful than being only marginally better in many simple ones.

Dangers

At the same time, benchmarks can be dangerous.

In real-world problems, authors usually understand the semantics of each dependent and independent variable, whereas authors using a benchmark suite may be more inclined to apply algorithms blindly. This can lead to nonsensical results such as those identified by McDermott and Forsyth (2016).

A related danger is that when authors aim only for benchmark performance, playing the "up-the-wall game" (Burke et al., 2009), they may miss opportunities for more novel research. A benchmark problem may mandate the fitness function in order to ensure fair comparisons, but that prevents researchers from trying out new fitness functions. Here, a solution might be to de-couple the fitness function (used to drive evolution) from the evaluation function (post-run). Only the latter has to be mandated.

Another example is the GP language. For some problems, researchers can "game" the problem by specifying a GP language in which a solution is available as a single primitive. Synthetic problems are particularly vulnerable. So, we might want to specify the GP language which is allowed for this problem. But then future GP researchers trying to investigate language issues, e.g. grammar design, or languages which generalise across many problems, won't be able to use our problem in their setting.

Sometimes the best way to make progress is to reformulate the problem. An example arises in recent work in neural program synthesis, e.g. the GitHub CoPilot program synthesis product (GitHub Team, 2022). It does not work with a training set of program input-output pairs as in most GP, but with a quite different view of how program synthesis could be used in the real world. But because of the reformulation, comparisons of performance between CoPilot and a GP system on standard benchmarks aren't obvious (but see Sobania et al., 2022).

"Goodhart's law" states that when a metric becomes a target, it ceases to be useful, because there is an incentive to "game" the metric. A well-known but possibly apocryphal example is the story of the Soviet nail factory: productivity was measured in tons of nails produced, which incentivised the production of a few, useless, enormous nails. Many examples of Goodhart's law in the fields of optimisation, machine learning and reinforcement learning were collected by Lehman et al. (2020). The same effect could arise in benchmarking: if many authors in a field focus too strongly on a particular suite, they may lose sight of the bigger picture, and might even "meta-over-fit" to the chosen problems. For the specific case of meta-over-fitting to unseen ML test sets, Roelofs et al. (2019) showed that this was not a major problem, in a study of ML competitions on Kaggle.com.

Perhaps most important of all, there can be a mismatch between benchmark problems and the real-world problems on which we want to demonstrate progress. We will discuss this in more detail under “Reflections on Mismatch”, below. First, we consider some of the progress of recent years.

New benchmark suites

At the time of the original GP Benchmarks papers, there were some clear gaps in the market, but there have been several developments since.

PSB and PSB2

The Program Synthesis Benchmark suite (PSB; Helmuth and Spector, 2015) and its successor PSB2 (Helmuth and Kelly, 2021) filled one of the main gaps identified in the original GP benchmarks papers. They focus on algorithmic problems. PSB curated 29 programming textbook-style exercises. For example, one problem is: “Given a vector of integers, at least one of which is 0, return the index of the last occurrence of 0 in the vector”. A set of input-output pairs for each problem was not originally provided but was provided later, in an attempt to reduce the implementation burden on authors (Helmuth and Kelly, 2021). PSB had a very positive impact on the field. It was quickly adopted by many other researchers, and although some problems remained difficult, on others there was steady progress over the years. The successor PSB2 is intended to increase difficulty, drawing problems from sources such as Advent of Code and Project Euler. It also improved on the original by mostly avoiding Boolean problems (too easy for systems to over-fit) and avoiding the combination of returning and printing results (unneeded implementation complexity).

New ML Benchmark Suites

Several new benchmark suites have been proposed in recent years. The OpenML (Vanschoren et al., 2014) and Penn Machine Learning Benchmark (PMLB; Olson et al., 2017) repositories provide access to hundreds of datasets culled from varied online sources. In addition, both provide convenient APIs for fetching data in a standardized

format in Python or R and using it quickly for benchmarking different algorithms. OpenML provides additional APIs to other languages, as well as several integrations for sharing ML pipelines themselves. The authors of OpenML have also created collections of datasets curated for specific benchmarking tasks, such as the OpenML Curated Classification benchmark (OpenMLCC-18). PMLB datasets are broadly categorized into classification and regression benchmarks, including some simulated physics datasets with known ground-truth solutions. These include physics and dynamical modeling problems sourced from textbooks by Richard Feynman and Steven Strogatz. The idea to use this large set of equations from physics textbooks is due to Udrescu and Tegmark (2020) in their paper on the AIFeynman SR algorithm.

SRBench

In addition to datasets, “living benchmarks” have been created such as SRBench (https://srbench.org) that provide results from prior experiments, the benchmarked methods themselves, and the code needed to reproduce or extend the experiments. SRBench began as a project that benchmarks symbolic regression methods and other ML algorithms on PMLB. It has since expanded to host an interpretable data science competition at GECCO 2022. It has several advantages: it supports a diversity of problems, integrates with the popular Scikit-Learn API, is reproducible, provides baseline results, and contains documentation and guides to contributing. Contributions of new methods are welcomed, and are managed using continuous integration testing through the GitHub repository.

GAMETES

The GAMETES datasets were defined a few months after the publication of our first paper (Urbanowicz et al., 2012), and recently used in several contributions by the GP community. The GAMETES datasets were generated by a tool for embedding epistatic gene-gene interactions into noisy genetic datasets. They incorporate a two-way epistatic interaction that is predictive of a disease classification, but this is masked by the presence of confounding features and noisy disease classifications. The number of features and

signal-to-noise ratio are tunable in order to vary difficulty. GAMETES is a clear example of a benchmark that, although synthetic, has many characteristics of real-world applications.

DIGEN

DIverse and GENerative Benchmark (DIGEN) is a newly proposed benchmark suite of 40 synthetic binary problems, each formulated with a different mathematical formula (Orzechowski and Moore, 2022). The equations were created in an optimization process using GP and Optuna (Akiba et al., 2019) in which some of the leading machine learning classifiers competed against each other in an attempt to maximise difference in accuracy. DIGEN comes with a Scikit-Learn interface, a Python package and a Docker script, which allows the user to reproduce the results on different machines. Additionally, for each of the 40 problems pre-computed results are provided for 8 ML methods, which underwent hyper-parameter tuning and runs from multiple starting points. DIGEN allows new methods to be added fairly easily and greatly simplifies the benchmarking process of any new methods.

In addition to these new benchmark suites, we have seen continued discussion on benchmark design. Nicolau et al. (2015) offered advice on choice of problems, noise, train-tests splits, and more, in the context of symbolic regression. Oliveira et al. (2018) defined a meta-space of benchmark properties with the goal of finding areas in the space not covered by any benchmarks. Woodward et al. (2014) discussed the issue of mismatch between benchmarks and real-world problems. We turn to this issue next.

Reflections on Mismatch

Woodward et al. (2014) wrote that “benchmarks should not be arbitrarily selected but should instead be drawn from an underlying probability distribution that reflects the problem instances which the algorithm is likely to be applied to in the real-world”. Even with a substantial benchmark suite, there can be a mismatch between the benchmark problems and the real-world problems for which they are a substitute. This mismatch is perhaps the fundamental issue in benchmarking.

With this perspective, it is worth mentioning the No Free Lunch (NFL) Theorem for optimisation (Wolpert and Macready, 1997) and its extension to Machine Learning (Wolpert, 2002), which tell us that no method can be better than any other on all existing problems. However, it has been argued that NFL does not truly constrain performance in a typical GP setting, where fitness is defined as a set of input-output pairs (McDermott, 2020).

Even without NFL, there is a consensus that no GP algorithm will be the best on all possible problems. Instead a new GP method, to be interesting (and, as such, worth being published) should at least out-perform the state of the art on *a set* of problems that are *interesting* from some viewpoint – for instance, a set of problems that are particularly relevant for a particular application area, or that have some particular characteristics, like a complex fitness landscape, or a too-large set of uninformative features. This is, in our vision, what benchmarking is all about. Ultimately, we are trying to answer the question: *what makes a problem interesting for GP*? This is a question that can be articulated in many different forms, and that can receive several different answers according to the context. However, in any case, the process of trying to find an answer to this question is the first step towards the definition of a good benchmark suite.

Synthetic problems

When answering the previous question, the *context* is of great importance. For instance, benchmark symbolic regression problems which consist of known equations can sometimes be criticised for being artificial, unrealistic, or too easy. In this section we consider some contexts where such problems may have valid uses, and some contexts where the criticism may be correct.

Synthetic problems can be useful when the goal is to *understand* our algorithm and its performance. For example, when a GP system fails, there are two possible reasons:

1. A good solution is not available in the search space.

2. A good solution is available, but the search failed to find it.

Using a synthetic problem we can eliminate (1).

Synthetic problems can serve as unit tests. Suppose we write a new GP system from scratch, perhaps as an exercise to try out a new language. It is very easy to make an error, such as sorting the population by ascending fitness instead of by descending fitness. Our first run of the system will hopefully reveal such an error. Using a synthetic problem for such a run is not bad practice! One way to view this is to say that such a usage is fine, but does not count as benchmarking *per se*, and need not be published. In the past such tests were more likely to be seen as adequate for publication.

Another benefit of synthetic problems is that their difficulty is easily tunable, which is useful for benchmark suites especially in the context of improvement over time (see Figure 1, later).

Finding readable equations is one of the central selling points of GP symbolic regression as an ML method. Some researchers use synthetic problems to test the ability of their algorithm to recover not only a readable equation but the *specific* equation which was used to generate the data. Often some artificial noise is added, as measurement noise is inevitable in real-world problems.

However, the main aim of benchmarks is to help us improve methods for real-world applications. It is difficult to define synthetic benchmark problems that are similar to real-world problems. Many functions from the "AI Feynman" regression database are very easy for GP and can be solved with only the initial population. Others, e.g. the Salustowicz or Salustowicz2d problems (Salustowicz and Schmidhuber, 1997, Vladislavleva et al., 2008) have a functional complexity that is unlikely to occur in practice (see also Nicolau et al., 2015).

Although some authors have used synthetic problems with the goal of recovering the original formula exactly (e.g. Schmidt and Lipson, 2009, Cranmer, 2020, Izzo et al., 2017), in the real world we never have more than a model. Even well-known physical equations have weak system boundaries, and unmeasured effects (Burham and Anderson, 2002). For example, the equation *F=ma* neglects the friction which would affect measurements.

From this point of view, regression is always about finding a model and not the “truth”, and finding synthetic formula exactly is misleading.

The separation of synthetic-only and real-world-only benchmark suites is a potential solution and has the benefit that it allows comparing methods for both cases. It is up to authors to state their assumptions and goals, and state how their experimental results lead to their broader conclusions.

Interpretability

GP researchers often claim that interpretability of models is an important advantage versus other non-GP methods – especially in SR. (GP researchers do not always follow through on such claims by publishing and interpreting their best models.) However, interpretability is difficult to quantify or measure. It is subjective and depends on domain knowledge. Consequently, one can argue that a benchmark should not try to measure interpretability per se. We may measure program length, number of parameters to be fixed, or another measure because minimizing these using parsimony pressure of some form may have advantages in ability to generalize or in computational effort, as well as tending to lead to interpretability as a by-product.

Real-world problems

Real-world problems have real-world problems. That is, problems that arise in the real world tend to present researchers with problems like missing values, outliers, contradictory labels, useless variables, missing variables, special cases, and so on. Part of the motivation for creating benchmarks is to make it easier for authors to try out multiple problems without dealing with all of these issues, which often seem mundane compared to the central goals of GP. A second motivation is to ensure that differences in pre-processing (e.g. removing some unneeded variables) do not make one GP method falsely appear better than another. Archetti et al. were in favour of this point of view, as they argued that the Bioavailability problem touched on too many of these issues, and ended up being too hard. They implicitly argued that the best benchmarks are in the “large important differences” area of Figure 1, below.

However, we also want to avoid a mismatch between our benchmark problems and real-world problems. Why are we “taming” or “sanitising” our benchmarks? We may resolve this tension by seeing that only a few researchers expect GP to become a “turn-key” solution and deal with all of these real-world issues, as opposed to removing them in pre-processing. This few is not none! It is an interesting and valid research area. For researchers who want GP to work robustly in the presence of such issues, of course benchmarks which include these issues are required. And some ML datasets, for example in SRBench, already contain such issues.

Looking forward

Sometimes benchmarks and benchmark suites grow organically, as multiple authors working in an area try their methods on each others’ real-world problems and gradually standardise. But proposing good benchmark suites seems to be recognised by the community as a contribution, so authors have an incentive to do this more deliberately.

It seems unlikely that a benchmark suite will be proposed which covers many GP areas and achieves wide acceptance. The issues and methods are just too different between different application areas (e.g., symbolic regression versus program synthesis). Instead we see benchmark suites focussed on specific areas. Some such suites have emerged in the past 10 years, as we have seen in the previous section. Are there other areas where researchers are currently using GP but lack an adequate benchmark suite? We can mention at least two possibilities as follows.

Image Processing Benchmarks

In 2012, when our first paper was published, Deep Learning (DL) was not yet used at such a large scale as it is today. Now, much more challenging problems are being addressed and some have become benchmarks in that community. This is surely the case in image processing, where several problems and datasets are today considered standard benchmarks. GP has been influenced by this trend, as testified by the large number of contributions using GP for image processing.

GP researchers do not always use the same benchmarks as are common in DL. This may give the impression that GP methods are less successful, so authors avoid direct comparison with the DL state-of-the-art. In some cases this may be true. However, there are also problems in image processing where GP is more than competitive, but they may not be the most well-known ones such as ImageNet. One example is in the field of image quality assessment. Datasets such as TID2013 (Ponomarenko et al., 2015) and VIDID2014 (Gu et al., 2015) could become “de facto” benchmarks for image quality assessment. These datasets consist of a rich set of reference images that account for diverse visual scenes and contain several different types of distortion with different degrees of intensity. The reference-distortion pairs were subjectively evaluated by involving hundreds of volunteers in a controlled experimental environment. The difference between these two datasets consists in the viewing distance of the images. This makes the two different but related, and so it could be appropriate to use one for training and one for testing: a stronger form of generalisation than is typical.

Boolean Function Learning

Since Koza (1992) described his application of GP to evolve multiplexer and parity functions, a diverse family of Boolean function benchmarks has emerged. One subgroup of these benchmarks are multiple-output problems such as digital adders, subtractors, or multipliers. The challenge to evolve expressions for this type of Boolean function resulted in the emergence of graph-based, rather than the traditional tree-based, representations. The digital multiplier benchmark was suggested as a suitable replacement for overused single-output benchmarks. Over the last decade, steps towards more challenging Boolean benchmarks have been taken. In the first place, high-order digital adder and multiplier functions were explored, as well as new types of benchmarks such as multi-function and cryptographic Boolean problems. Concerning cryptographic Boolean functions, *bent* and *resilient* functions have been evaluated with Cartesian GP (Miller, 2020). Multi-function benchmarks such as the arithmetic logical unit and the digital adder-subtractor were also attempted. Both cryptographic and multi-function Boolean benchmarks improved the evaluation diversity of experiments in the Boolean problem domain and have been found to be useful in exploring limitations of algorithms.

Moreover, new insights about the parametrization of Cartesian GP have been discovered with these benchmarks.Therefore, for the future of benchmarking in the Boolean problem domain it may be useful to further increase the diversity of benchmarks by exploring new Boolean problems and curating them into a benchmark suite.

Benchmark lifecycle

A problem which is easily solved by all methods, or which is totally unsolvable by all methods, will not be obviously useful as a benchmark. A useful benchmark must distinguish between methods (Orzechowski and Moore, 2022). However we can see a dynamic process here. Figure 1 below shows a stylised picture of progress on a given problem over time. Something like this has happened with the MNIST data, for example, in the field of neural networks (NNs), and arguably the same has happened for some problems in the PSB suite.

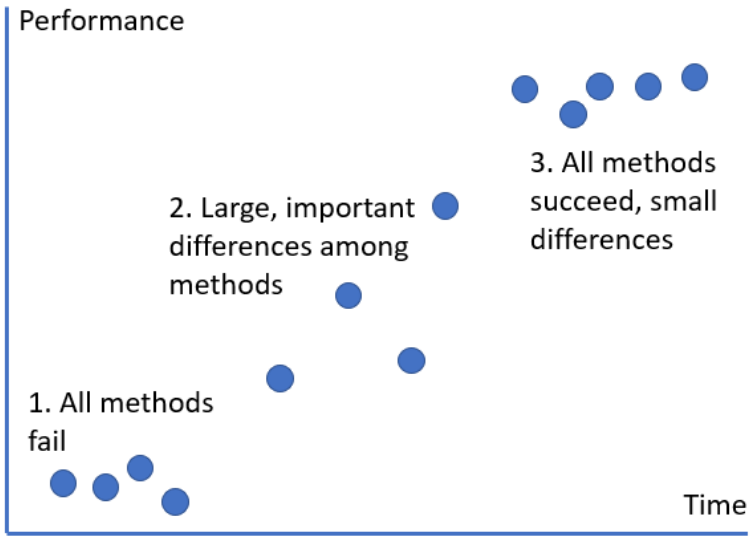

Fig 1: performance of various methods on a fixed problem, over time.

Perhaps we can say that the most useful phase for a benchmark problem to be in is the “large, important differences” phase. Problems which are in the “all methods fail” phase, if they are important, could be called “grand challenges” rather than benchmarks. They might tend to inspire totally new reformulations of the problem rather than incremental progress.

In the best case scenario, our methods eventually reach the saturation point. Then there are two possible reactions:

1. Great, we're done.
2. Ah, now we see why that benchmark was misleading (so progress on this benchmark doesn't represent real progress).

The latter is worth mentioning because it has indeed happened in other fields. In the field of AI, the Moravec paradox (Moravec, 1998) is the observation that tasks which appear difficult for humans (e.g. chess) turn out to be easy for computers or AI systems, and vice versa (e.g. vision). A recent example is the success of large language models in generating plausible text but without approaching true intelligence. Benchmarking can then be seen as an ongoing arms race between algorithm developers and benchmark developers. Perhaps in the future we will see some GP benchmarks conquered and then questioned in the same way.

References

Akiba, T., Sano, S., Yanase, T., Ohta, T. and Koyama, M., 2019, July. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 2623-2631).

Balog, M., Gaunt, A. L., Brockschmidt, M., Nowozin, S., & Tarlow, D. (2017). Deepcoder: Learning to write programs. In International Conference on Learning Representations. ICLR

Burke, E. K., et al. "Towards the decathlon challenge of search heuristics." Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers. 2009.

Burnham, K. P. Model selection and multimodel inference. A practical information-theoretic approach (1998).

Chen, X., Liu, C., & Song, D. (2018,). Execution-guided neural program synthesis. In International Conference on Learning Representations. ICLR

Christie, L.A., Brownlee, A.E. and Woodward, J.R., 2018, July. Investigating benchmark correlations when comparing algorithms with parameter tuning. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (pp. 209-210).

Cranmer, M., PySR: Fast & parallelized symbolic regression in Python/Julia, September 2020. URL http://doi.org/10.5281/zenodo.4041459.

Devlin, J., Uesato, J., Bhupatiraju, S., Singh, R., Mohamed, A. R., & Kohli, P. (2017). Robustfill: Neural program learning under noisy i/o. In International conference on machine learning (pp. 990-998). ICML.

Forsyth, R., 1981. BEAGLE – A Darwinian approach to pattern recognition. Kybernetes.

GitHub Team, 2022. GitHub Copilot. https://github.com/features/copilot

Gu, K., Liu, M., Zhai, G., Yang, X. and Zhang, W., 2015. Quality assessment considering viewing distance and image resolution. *IEEE Transactions on Broadcasting*, *61*(3), pp.520-531.

Helmuth, T. and Kelly, P., 2021, June. PSB2: the second program synthesis benchmark suite. In *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 785-794).

Helmuth, T. and Spector, L., 2015, July. General program synthesis benchmark suite. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* (pp. 1039-1046).

Izzo, D., Biscani, F. and Mereta, A., 2017, April. Differentiable genetic programming. In *European conference on genetic programming* (pp. 35-51). Springer, Cham.

Koza, J. R., 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press.

Lehman, J., et al. “The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities.” Artificial life 26.2 (2020): 274-306.

Liao, T., Taori, R., Raji, I.D. and Schmidt, L., 2021, August. Are we learning yet? a meta review of evaluation failures across machine learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

McDermott, J. “When and why metaheuristics researchers can ignore “No Free Lunch” theorems.” SN Computer Science 1.1 (2020): 1-18.

McDermott, J. and Forsyth, R.S., 2016. Diagnosing a disorder in a classification benchmark. Pattern Recognition Letters, 73, pp.41-43.

Miller, J.F., 2020. Cartesian genetic programming: its status and future. *Genetic Programming and Evolvable Machines*, *21*(1), pp.129-168.

Moravec, H (1988), Mind Children, Harvard University Press

Nicolau, M., Agapitos, A., O'Neill, M., & Brabazon, A. (2015, May). Guidelines for defining benchmark problems in genetic programming. In 2015 IEEE Congress on Evolutionary Computation (CEC) (pp. 1152-1159). IEEE.

Nicolau, M., 2017. Understanding grammatical evolution: initialisation. *Genetic Programming and Evolvable Machines*, *18*(4), pp.467-507.

Oliveira, L.O.V., Martins, J.F.B., Miranda, L.F. and Pappa, G.L., 2018, July. Analysing symbolic regression benchmarks under a meta-learning approach. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (pp. 1342-1349).

Olson, R. S., William La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, and Jason H. Moore. 2017. PMLB: a large benchmark suite for machine learning evaluation and comparison. BioData Mining 10, 36 (11 Dec 2017), 1–13. https://doi.org/10.1186/s13040-017-0154-4

Parisotto, E., Mohamed, A. R., Singh, R., Li, L., Zhou, D., & Kohli, P. (2017). In International Conference on Learning Representations. ICLR.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., 2011. Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, pp.2825-2830.

Ponomarenko, N., Jin, L., Ieremeiev, O., Lukin, V., Egiazarian, K., Astola, J., Vozel, B., Chehdi, K., Carli, M., Battisti, F. and Kuo, C.C.J., 2015. Image database TID2013: Peculiarities, results and perspectives. *Signal processing: Image communication*, *30*, pp.57-77.

Roelofs, R., Shankar, V., Recht, B., Fridovich-Keil, S., Hardt, M., Miller, J. and Schmidt, L., 2019. A meta-analysis of overfitting in machine learning. Advances in Neural Information Processing Systems, 32.

Ryan, C., Collins, J.J. and O'Neill, M., 1998, April. Grammatical evolution: Evolving programs for an arbitrary language. In *European conference on genetic programming* (pp. 83-96). Springer, Berlin, Heidelberg.

Salustowicz, R. and Schmidhuber, J., 1997. Probabilistic incremental program evolution. Evolutionary computation, 5(2), pp.123-141.

Schmidt, M. and Lipson, H., 2009. Distilling free-form natural laws from experimental data. *science*, *324*(5923), pp.81-85.

Sobania, D., M. Briesch, and F. Rothlauf. "Choose your programming copilot: a comparison of the program synthesis performance of github copilot and genetic programming." Proceedings of the Genetic and Evolutionary Computation Conference. 2022.

Stephens, T., 2016. Genetic Programming in Python With a Scikit-Learn Inspired API: gplearn. https://gplearn.readthedocs.io.

Udrescu, S.-M., and M. Tegmark. AI Feynman: A physics-inspired method for symbolic regression. Science Advances 6.16 (2020): eaay2631.

Urbanowicz, R. J., J. Kiralis, N. A. Sinnott-Armstrong, T. Heberling, J. M. Fisher, and J. H. Moore. “GAMETES: a fast, direct algorithm for generating pure, strict, epistatic models with random architectures.” BioData mining 5, no. 1 (2012): 1-14.

Vanschoren, J., Van Rijn, J.N., Bischl, B. and Torgo, L., 2014. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, *15*(2), pp.49-60.

Vladislavleva, E.J., Smits, G.F. and Den Hertog, D., 2008. Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. IEEE Transactions on Evolutionary Computation, 13(2), pp.333-349.