Note. *This is Version 3 of the paper. It has corrections in Table 3 compared to the original GECCO publication and an earlier unnumbered corrective version.*

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*

ABSTRACT

Genetic programming (GP) is not a field noted for the rigor of its benchmarking. Some of its benchmark problems are popular purely through historical contingency, and they can be criticized as too easy or as providing misleading information concerning real-world performance, but they persist largely because of inertia and the lack of good alternatives. Even where the problems themselves are impeccable, comparisons between studies are made more difficult by the lack of standardization. We argue that the definition of standard benchmarks is an essential step in the maturation of the field. We make several contributions towards this goal. We motivate the development of a benchmark suite and define its goals; we survey existing practice; we enumerate many candidate benchmarks; we report progress on reference implementations; and we set out a concrete plan for gathering feedback from the GP community that would, if adopted, lead to a standard set of benchmarks.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic programming—*Program synthesis*

General Terms

Algorithms, Experimentation, Measurement

Keywords

Genetic Programming, Benchmarks

1. INTRODUCTION

"I think GP has a toy problem problem."

This was the opening salvo of a debate on the Genetic Programming mailing list in March 2011 [19]. In this position paper we distill the ideas that emerged from this debate into an argument regarding the serious deficiencies in the GP community's benchmark and measurement approach, and suggest a way forward.

While much of the GP *application* literature focuses on nontrivial, domain-specific problems, the fundamental *comparison and analysis* literature typically uses benchmark problems that are often very simple. We argue that such benchmarks do little to move the field forward, and in fact may hold it back as they reward techniques that are effective at rapidly solving trivial problems rather than performing as well as possible on hard ones. As argued in [43], these two goals may be at odds. The problem is accentuated by the use of metrics that reinforce such rewards.

Devising a good benchmark suite has been recognized as an important issue for the next ten years of GP [49]. We believe that agreement on a set of benchmarks can be reached through selection based on community feedback, low barriers to participation, and deployment using a reference implementation. Establishing a benchmark suite with a community consensus has obvious advantages for individual authors and the field as a whole. A good, well-studied suite fosters well-grounded comparisons of techniques and makes it easier for researchers to understand the dynamics of algorithms as they are applied to known problems. Scalable benchmarks can also serve as goals to challenge the field.

In this paper, we review related work and discuss what constitutes a useful benchmark, the status quo of benchmarks in GP, and the statistical issues in evaluating performance on benchmarks. We will then list some benchmarks that may be suitable candidates to form a suite, with the goal of soliciting community feedback. Finally, we map out the necessary steps of gathering suitable benchmarks we may have omitted, and producing a benchmark suite.

GECCO'12, July 7–11, 2012, Philadelphia, Pennsylvania, USA.

*Computer Science and Artificial Intelligence Laboratory (CSAIL), Massachusetts Institute of Technology, USA, [jmmcd, unamay]@csail.mit.edu
†School of Computing Science, University of Glasgow, UK. david.r.white@glasgow.ac.uk
‡Department of Computer Science, George Mason University, USA, [sean, kdejong]@cs.gmu.edu
§Dipartimento di Informatica, Sistemistica e Comunicazione, University of Milano-Bicocca, Italy, [luca.manzoni, mauro.castelli]@disco.unimib.it
¶Instituto Superior de Estatística e Gestão de Informação (ISEGI), Universidade Nova de Lisboa, Portugal, lvanneschi@isegi.unl.pt
‖Institute of Computing Science, Poznan University of Technology, Poland, [wjaskowski, kkrawiec]@cs.put.poznan.pl
**Faculty of Science, University of Sydney, Australia, rharper2@bigpond.net.au

Though this paper is intended to be a significant step towards the goal of agreeing GP benchmarks, *it is not a list of benchmark recommendations*. It is intended as the start of a discussion, not the end of one. We do not attempt to dictate a set of benchmarks to researchers: rather the examples we provide are a vehicle for provoking further discussion and selection by the community. We invite discussion from practitioners in both academia and industry. We take the approach of suggesting many more candidate benchmarks than are necessary, in order that they be reduced to a more manageable size based on community feedback. We provide some reference implementations. Most importantly we suggest a plan for gathering community feedback through a web application, which we hope will lead to a community consensus and a standardized set of benchmarks.

2. PREVIOUS WORK

In the broader field of machine learning, benchmark suites are commonly used to measure progress on well-defined tasks. The UCI machine learning database[1] is a well-known example with many real-world datasets for tasks like classification, regression, and clustering.

Evolutionary computation (EC) has long seen the value of benchmark suites. The first widely used suite was the De Jong Test Suite [12]. It was designed for two purposes: to explore the effects that various design decisions and parameter settings had on the performance of early GA-based optimizers, and to facilitate comparisons between GA-based optimizers and more traditional mathematical optimization techniques. The goal was to have a small number of functions that served as representatives of different classes of landscapes, the features of which were believed to have a differential impact on optimization algorithms. In that same timeframe the Evolution Strategy (ES) community was developing their own set of real-valued test functions, initially chosen to be sufficiently mathematically tractable to allow convergence-related proofs [54].

Both the GA and ES communities continued to add functions to their test suites without any clearly articulated set of guidelines or principles. Cross-fertilization began to occur in the early 1990s as the various communities began to coalesce into the field of Evolutionary Computation. However, optimization benchmark suites consisted of functions chosen primarily to allow comparisons with earlier work. Some analysis and critiques of these benchmark suites began to appear [53], but there were no formal attempts to rationalize and organize them in a moderated repository for another 10 years. The first significant effort was the *Evolutionary Computation Benchmarking Repository*, (EvoCoBM) [55][2]. This project included descriptions of problems, rather than implementations. Unfortunately EvoCoBM is now unmaintained. More recently, the *Black-Box Optimization Benchmarking* project[3], which deals with real-valued optimization methods, is well-maintained and achieves community consensus through regular GECCO workshops and updates.

However, GP benchmarks face certain difficulties specific to them. GP fitness evaluation involves direct program execution, and this often demands a much higher degree of coding overhead and many more details, which in turn make standardization, repetition, and fair and significant comparisons between methods and implementations challenging.

One other previous effort is important to discuss. *GP-Beagle* [16] was an online benchmark database open to contributions, and which proposed links between the benchmark repository and the GP Bibliography. GP-Beagle was ambitious, and required significant extra work from those authors who chose to submit benchmarks to the repository. Notably, it defined a fixed template for the problems to be added, and it sought to specify every detail of the benchmarks. The repository, being machine-readable, was intended to open up the possibility of automated testing. Unfortunately, GP-Beagle was perhaps *overly* ambitious, and the original website[4] is now gone.

In the wake of GP-Beagle, other efforts have sprung up. For example, Widera et al. [70] have curated a collection of bioinformatics problems. There are also several ongoing competitions in the area of game-playing, such as car-racing and Mario AI controllers. The qualities and required of competition problems are largely the same as those required of benchmarks.

Our project differs from GP-Beagle in important ways. We recognize that it may be impractical to specify benchmark problems precisely, or that precise specifications may result in some problems being incompatible with some methods, languages, or existing software. Our initial aim is to describe current practice in the field, leading to recommendations, community discussion, and eventually consensus. This approach to community participation is rather different from that of GP-Beagle, and we hope it will prove more practical.

3. WHAT MAKES A GOOD BENCHMARK?

As with the remainder of this paper, our suggestions for benchmark criteria are intended to be open to improvements and additions. Several of the following criteria obviously conflict with each other.

Tunably Difficult. Any worthwhile benchmark is non-trivial. However an ideal benchmark is *tunable*: more and more difficult instances can be generated as required.

Varied. A benchmark suite should not be drawn from a single class of problem. Ideally, the benchmarks should be sufficiently orthogonal that they can uncover the strengths and weaknesses of different GP algorithms.

Relevant. A benchmark should be relevant to real applications and research in the GP field, and ideally to the wider machine learning, optimization, or search communities. Results should correlate well with results on real-world applications. A benchmark should not be vulnerable to paradigm shifts in the underlying application area that may render it irrelevant to the domain from which it was taken.

Fast. Because GP individuals are programs to be executed, often many times per individual, fitness evaluation can be slow. Bloat [56] exacerbates this problem. An ideal GP benchmark is fast enough to allow the large number of runs required for meaningful comparisons between approaches.

[1] http://archive.ics.uci.edu/ml/

[2] http://www.cs.bham.ac.uk/research/projects/ecb/

[3] http://coco.gforge.inria.fr/doku.php?id=bbob-2012

[4] http://www.gp-beagle.org

Accommodating to Implementors. Ideally an effective benchmark is widely implemented in a variety of toolkits. A benchmark must be *easy* to implement and reproduce; it must be self-contained; it must be open source and patent-unencumbered; and it must be accessible to and understandable by implementers without specialized domain expertise.

Easy to Interpret and Compare. The results produced by a benchmark should be easily comprehensible and statistically useful. Success rates in finding an ideal solution — the hallmark of toy problems — are discouraged in favor of measures such as average best fitness. Other features, such as optional multiple objectives, are welcome.

Representation-Independent. Nowadays GP is a community and a set of related algorithms, not a single technique. Common GP methods include Koza-style s-expressions, strongly-typed GP, Linear GP, Grammatical Evolution, Cartesian GP, Push GP, and others. In order to facilitate comparison across methods, a benchmark should not unnecessarily require a specific feature of a given representation. More weakly, though we cannot weed out a representation's particular optimization bias, a benchmark should not unduly favor a given representation's characteristics.

Precisely-Defined. The benchmark should be well documented and precisely defined. If appropriate, a benchmark should be accompanied by a set of recommendations on the resources typically made available, such as an upper limit on the number of fitness evaluations. However such recommendations should not be interpreted as permanent constraints.

Current. Over time, the field may move to new application areas, and abandon old ones. Researchers may find flaws in old benchmarks, or may discover that they are easily solved with state of the art methods or machines, making them less interesting in the future, or may discover that good performance on a particular synthetic problem fails to correlate with real-world applications. Even a good benchmark suite may need to be replaced after some years if it seems to be too strongly influencing the direction of research, i.e. if researchers seem to "teach to the test" [14].

Among the most common benchmark problems have historically been Even-N Parity, N-Multiplexer, Quartic Polynomial Symbolic Regression, and Artificial Ant. These problems were first used by Koza [32, 33], became common, and are still used more than 20 years later. They have constituted a useful *de facto* standard benchmark suite for many years. They represent several classes of problems. The Boolean problems are scalable in difficulty. They are self-contained, requiring no external data-set or domain knowledge.

However they are vulnerable to several criticisms. They are seen as trivial or "toy" problems, which are too easily solved and do not reflect performance in real-world applications. They have been in use for so long that they may have had a tendency to distort the field. The choice of the quartic polynomial function for symbolic regression seems to require an assumption that in the space of arithmetic functions, any target function is equally interesting. In fact, some targets are far easier to achieve than others. The N-Multiplexer's fitness values tend to chunk in powers of 2, so statistical treatment of mean best fitness is problematic. The extensive use of this limited set of benchmarks has resulted in some overfitting. Furthermore, we are expending research effort in solving problems that are already solved. For all these reasons, it seems desirable to revise and update the *de facto* suite with an improved set of problems.

Constructed problems have also been used in the GP literature (see Table 1). These are hand-written, defined by a language and an artificial procedure for assigning fitness to individuals. Constructed problems are an important feature of benchmarking in other fields of optimization. Some are designed to test performance, and others to test other algorithmic issues [61]: for example, Royal Tree problems are useful in testing building-block behaviour, while Order and Majority are useful in testing intron behaviour. They are generally easy to define (exceptions include the K-Landscapes and Multimodal Trap Functions), and self-contained. They are often tunable: their difficulty can be changed by modifying some parameters' values. This point is particularly important, since algorithmic properties should be tested over a complete range of different difficulties and landscape complexities in order to ensure that they are general and scalable. Constructed problems also have some disadvantages. Very small terminal and non-terminal alphabets are unrealistic. Some constructed problems have overly-simple fitness landscapes, e.g. Royal Tree and Unimodal Trap Functions have only a single global optimum. Also, the most common constructed problems are applicable only to tree-based GP.

4. CURRENT PRACTICE

To assess current practice, we surveyed the 183 articles presented in EuroGP and the GP track of GECCO from 2009 to 2011, of which 172 used benchmarks.

Symbolic Regression. The most widely used class of benchmarks (about one third of the papers) is symbolic regression. The implementation of these benchmarks is usually not difficult. The mean number of symbolic regression problems used in a paper was 2.4. There is a proliferation of different benchmarks, which makes a comparison between different approaches extremely difficult. The most-used function was the quartic polynomial, with 26.2% of the papers using it. However, papers used a wide variety of sampling, training, and testing approaches.

Classification. Classification problems were used in 20.2% of the surveyed papers. The use of a single classification problem is rare: the mean number of problems per paper is 3.5. In this area, the use of the UCI repository is prominent: two thirds of the problems are sourced from this repository. This is most likely a consequence of the difficulty and time required to obtain and curate new real-world data sets. Data is usually in a non-standard format or incomplete. Furthermore, for medical data there are governance issues, for example protecting patients' privacy.

Binary Functions. Binary functions were used in about 15% of the articles. The even-N parity (used in 67.9% of papers) and N-Bit Boolean Multiplexer (53.6%) problems were the most-used. Other problems, like the Digital Multiplier and the N-Majority were less used. Compared to symbolic regression, binary functions problems seem less diverse.

Predictive Modeling. Predictive modeling was included in about 12.6% of the papers. The number of problems used on average in an article is 1.48, a number lower than classification problems. There is no uniformity in the choice of benchmarks. Nonetheless, the Drug Bioavailability dataset is used by a little more than 30% of the papers, while the Median Oral Lethal Dose dataset and the Mackey-Glass time series are used in about 17.4% of the papers.

Path Finding and Planning. This class of problems has been used by 19.1% of the articles, with an average of 1.3 problems used in every paper. The Artificial Ant problem is the most-used example, with 54.3% of the papers.

Other Problems. The remaining problem classes were used in less than 10% of the papers. Among them, the traditional programming and constructed problems were the most common. In the latter class, the Max problem was most-used, by just over two fifths of the papers.

5. APPROPRIATE STATISTICS

Suggesting or adopting standard benchmarks in itself is not enough to produce meaningful results: the statistical procedure for comparing methods is crucial, and effective comparison benefits from consistency in the literature in this regard. How should we perform such comparisons?

A large portion of early (and indeed current) GP results were measured using *ideal solution counts:* whether or how often the optimum, or some threshold near the optimum, was reached. The most common approach [32] defined the *computational effort* measure as an estimate of the minimum number of individuals to be processed in a generational algorithm in order to achieve a $z = 99\%$ probability of discovering a solution. More formally, this was defined as $\min_i m \times (i+1) \times \left\lceil \frac{\ln(1-z)}{\ln(1-P(m,i))} \right\rceil$, where i was a generation number, m was the population size, and $P(m,i)$ was the *cumulative probability of success*, that is, the probability that an ideal solution would be found on or prior to generation i, as gathered through samples.

This measure has received significant criticism [51, 7, 43, 47, 4]. Critics have noted that ideal solution counts are really a measure of how well a method solves trivial problems, rather than the nontrivial ones found in real world applications. Attempts have since been made to address another central criticism: poor accuracy and statistical invalidity [8, 68].

We think that benchmark comparison measures should instead assume that techniques will be applied, ultimately, to problems where the optimum is not expected to be discovered, much less repeatedly and easily. For single objective problems, two obvious candidates are *best fitness of run* (appropriate for problems where the goal is optimization) and *generalizability measures* such as final testing against a withheld generalization set, or K-fold validation (appropriate for problems where the goal is to perform model-fitting).

6. TOWARDS A GP BENCHMARK SUITE

The goal of this position paper is not to propose a benchmark suite: that is a task that needs to be done via a wider community discussion and consensus. Our aim is to initiate that process and to facilitate it. Among the many issues to be decided through this process, we here emphasise two.

Problem/Category	Variables	Episodic	Known Solution	Deterministic	Discrete Fitness
Boolean Functions					
N-Multiplexer, Majority, Parity [32]	n	Y	Y	Y	Y
Generalized Boolean Circuits [23, 71]	n	N	Y	Y	Y
Digital Adder [67]	$2n+1$	Y	Y	Y	Y
Digital Multiplier [67]	$2n$	Y	Y	Y	Y
Classification					
mRNA Motif Classification [37]	-	Y	N	Y	Y
DNA Motif Discovery [38]	-	Y	N	Y	Y
Protein Structure Classification [70]	8	Y	N	Y	Y
Intrusion Detection [22]	41	Y	N	Y	Y
Protein Classification [34]	20	Y	N	Y	Y
Intertwined Spirals [32]	2	Y	N	Y	Y
Predictive Modelling					
Mackey-Glass Chaotic Time Series [35]	8	Y	N	Y	N
Sunspot Prediction [27]	12	Y	N	Y	N
Financial Trading [6, 13]	75–273	Y	N	Y	Y
Prime Number Prediction [66]	1	Y	N	Y	N
Drug Bioavailability [57]	241	Y	N	Y	N
Median Oral Lethal Dose [2]	626	Y	N	Y	N
Time Series Forecasting [65]	12–240	Y	N	Y	N
Path-finding and Planning					
Physical Travelling Salesman [41]	21–101	N	N	Y	Y
Artificial Ant [32]	0	N	Y	Y	Y
Lawnmower [33]	0	N	Y	Y	Y
Tartarus Problem [9]	7	N	N	Y	N
Circuit Design [44]	1	Y	N	N	N
Control Systems					
Chaotic Dynamic Systems Control [39]	3–4	N	N	Y	N
Pole Balancing [46]	4	N	N	Y	N
Truck Control [31]	4	N	N	Y	N
Game Playing					
TORCS Car Racing [15]	16	N	N	N	N
Ms. PacMan [17]	-	N	N	N	Y
Othello [40]	64	Y	N	Y	Y
Chessboard Evaluation [24]	32	N	N	N	Y
Backgammon [3]	-	N	N	N	Y
Mario [59]	21–101	N	N	N	Y
Robocode [58]	14	N	N	N	Y
Lose-Checkers [5]	267	N	N	N	Y
Dynamic Optimization					
Stochastic Dynamic Sym. Reg. [48]	5	Y	N	N	N
Periodic Dynamic Sym. Reg. [63]	4	Y	N	Y	N
Dynamic Scheduling [26]	10	N	N	Y	N
Traditional Programming					
Sorting [29]	2	N	Y	Y	Y
Recursive Sorting [1]	3	Y	Y	Y	Y
Bug-Fixing [69]	-	Y	N	Y	Y
Constructed Problems					
Royal Trees [52]	0	Y	Y	Y	Y
Max [18, 36]	0	Y	Y	Y	N
Lid [11]	0	Y	Y	Y	N
Order and Majority [20]	0	Y	Y	Y	Y
Trap functions [60]	0	Y	Y	Y	Y
K-landscapes [62]	0	Y	Y	N	Y
OrderTree problem [25]	0	Y	Y	Y	Y
TreeString problem [21]	0	Y	Y	Y	Y

Table 1: Some Candidate Benchmarks with Certain Features. For Symbolic Regression, see Table 3.

Name	Functions	Constants (ERC)
Koza	$+ - \times \% \sin \cos e^n \ln(\lvert n \rvert)$	None
Korns	$+ - \times \% \sin \cos e^n \ln(\lvert n \rvert)$ $n^2\ n^3 \sqrt{n} \tan \tanh$	Random finite 64-bit IEEE double
Keijzer	$+ \times \frac{1}{n} - n\ \sqrt{n}$	Random value from $N(\mu=0, \sigma=5)$
Vladislavleva-A	$+ - \times \% \ n^2$	$n^\epsilon \quad n+\epsilon \quad n\epsilon$
Vladislavleva-B	$+ - \times \% \ n^2\ e^n\ e^{-n}$	$n^\epsilon \quad n+\epsilon \quad n\epsilon$
Vladislavleva-C	$+ - \times \% \ n^2\ e^n\ e^{-n} \sin \cos$	$n^\epsilon \quad n+\epsilon \quad n\epsilon$

Table 2: Function Sets for Table 3. Terminals for variables (x, y, z, v, w) not shown. Vladislavleva's "constants" are functions with arguments, not terminals: ϵ is a uniform random value from $[-5, 5]$. Koza's function set traditionally has optional $[-1, 1)$ constants: but we assume no constants by default.

First is the choice of a benchmark suite. Which classes of problems, how many, and which ones?

Second, to what degree should the problems be specified? It is natural to answer that the specification should be as tight and complete as possible, with every detail specified *a priori*, even including limits on the number of fitness function calls allowed. Feldt et al. [16] supported such a policy by requiring a centralized repository of benchmark implementations. Further support is provided by Daida et al. [10], who show how innocuous changes in experimental details can produce wildly varying results. The advantage of such a policy would be to allow performance results to be compared *directly* across papers, without in-paper controls.

This policy also has disadvantages. Many packages differ in their treatment of various GP details in such a way that compatibility with a benchmark specification may be non-trivial. Most GP authors will not find it convenient to switch their development effort to a new software package or language. Some important GP advances would not be amenable to benchmarking if the benchmark suite is too restrictive: for example, the results of the linear-scaling approach to fitness advocated by Keijzer [28] were not directly comparable to previously-published results with the same functions. One can imagine similar hypothetical examples: a novel technique to improve performance in the context of GP random numerical constants might be "outlawed" by a benchmark that specified the workings of ERCs.

A possible alternative is to specify benchmarks less tightly, and ensure that they can be used universally. Authors should still aim for complete reporting of all implementation details. However, the possibility of differing representations and implementation details would mean that results on a particular benchmark cannot always be compared directly across papers. That is, authors would continue to use in-paper controls.

The community is therefore faced with a choice between a more ambitious (more tightly specified) or a more practical (less tightly specified) course of action. This is a difference of degree only: either course would still imply a real shift in community standards in the direction of increased rigor.

GP Benchmarks Wiki. We have developed a Wiki[5] collecting applications and problems in a wide variety of subjects and classifications, plus benchmark criteria. The intention is to provide a nexus for GP community discussion on this topic. Each candidate benchmark will be assigned a stable Wiki URL to allow automated tracking of future usage.

Candidate Benchmarks. In addition to the survey of recent GECCO and EuroGP papers, we have collected from the broader literature an initial set of candidate benchmarks to provide a starting point for discussion. It is set out in Table 1, with the symbolic regression examples separated out in Table 3. This collection casts a very wide net: it includes candidates we have previously criticized in this paper, as well as candidates that clearly violate one or more of the criteria set forth earlier: notably efficiency. However the candidates together satisfy all the criteria in Section 3.

In order to aid navigation in the tables and characterize problems in a systematic way, we use a set of common problem features. The features are: the number of true, data-fetching variables; whether or not the problem uses episodic, i.e. independent evaluations, in contrast to sequential evaluation as in the Artificial Ant; whether or not fitness is discrete; whether or not there is a known, ideal solution; and whether or not the problem is deterministic.

The 53 symbolic regression target functions in Table 3 (with function sets in Table 2) are drawn from several well-known sources in the literature [28, 30, 32, 45, 50, 64]. There is significant variance among them. The variance in difficulty is high: for example, Koza-1 ("Quartic") is well understood and relatively easy, while Pagie-1 has a reputation for being challenging [50]. The number of variables ranges from 1 to 5; the sample sizes vary from as few as 20 to as many as 10,000. Some benchmarks were meant for pure optimization, while others were intended for regression and so have separate training and testing sets. They also vary in their function sets and in how their sample points are generated.

Implementation. We have contributed all 53 symbolic regression benchmarks, including function sets and initialization procedures, to the ECJ toolkit [42]. Implementation of several constructed problems in the ECJ format is underway. ECJ is a popular GP library and one of the oldest and most stable in the field. Being written in Java makes it possible to guarantee a consistent interpretation of the benchmark code regardless of platform. A primary objective of this reference implementation is to provide clear, well commented, and clean code in order to make reuse as easy as possible. Authors wishing to use the benchmarks will have the option of using this code directly, using it as a guide to their own implementation, or continuing to use existing implementations if they believe them to be consistent with the reference implementation. Reference implementations in other languages and packages would be greatly welcomed.

7. NEXT STEPS

This paper is the beginning of a process that we hope will lead to a consensus and a standard set of benchmarks. The initial step will be an open discussion at GECCO 2012. Next, if GECCO delegates agree, the following plan will be carried out. A community poll and feedback mechanism will be run via the Wiki page and driven by community mailing lists. Key issues to be decided will be the choice of benchmarks and how tightly-specified they should be, as discussed above. Based on the results, a short publication will be submitted

[5] http://groups.csail.mit.edu/EVO-DesignOpt/GPBenchmarks

to a community venue such as the SigEVO newsletter or the Genetic Programming and Evolvable Machines journal, proposing a suite which we will term GPBenchmarks2012-α. It will encourage authors to begin to mention this term and the problems' stable Wiki URLs, if they wish to lend their weight to the effort. Based on a final round of feedback in response to the above publication, GPBenchmarks2012-*final* will be published as soon as possible. The issue of GP benchmarks will be revisited regularly, possibly via a conference workshop.

8. ACKNOWLEDGEMENTS

This work was supported in part by: NSF grant 0916870; IRCSET/Marie Curie; and NCN 2011/01/B/ST6/07318.

9. REFERENCES

[1] A. Agapitos and S. M. Lucas. Evolving Modular Recursive Sorting Algorithms. In *Proc. EuroGP*. 2007.

[2] F. Archetti, S. Lanzeni, E. Messina, and L. Vanneschi. Genetic Programming and Other Machine Learning Approaches to Predict Median Oral Lethal Dose (LD_{50}) and Plasma Protein Binding Levels (%PPB) of Drugs. In *Proc. EvoBIO*. 2007.

[3] Y. Azaria and M. Sipper. GP-Gammon: Genetically Programming Backgammon Players. *GPEM*, 6:283–300, 2005.

[4] D. F. Barrero, M. R-Moreno, B. Castano, and D. Camacho. An Empirical Study on the Accuracy of Computational Effort in Genetic Programming. In *Proc. CEC*. 2011.

[5] A. Benbassat and M. Sipper. Evolving Lose-Checkers Players using Genetic Programming. In *Proc. IEEE Computational Intelligence and Games*. 2010.

[6] R. Bradley, A. Brabazon, and M. O'Neill. Dynamic High Frequency Trading: A Neuro-Evolutionary Approach. In *Proc. EvoWorkshops*. 2009.

[7] S. Christensen and F. Oppacher. An Analysis of Koza's Computational Effort Statistic for Genetic Programming. In *Proc. EuroGP*. Springer-Verlag, 2002.

[8] S. Christensen and F. Oppacher. The Y-Test: Fairly Comparing Experimental Setups with Unequal Effort. In *Proc. CEC*. IEEE, 2006.

[9] G. Cuccu and F. Gomez. When novelty is not enough. In *Proc. EvoApplications*. 2011.

[10] J. M. Daida, et al. Challenges with Verification, Repeatability, and Meaningful Comparison in Genetic Programming: Gibson's Magic. In *Proc. GECCO*. 1999.

[11] J. M. Daida, et al. What Makes a Problem GP-Hard? Analysis of a Tunably Difficult Problem in Genetic Programming. *GPEM*, 2:165–191, 2001.

[12] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Ph.D. thesis, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, Michigan, 1975.

[13] I. Dempsey, M. O'Neill, and A. Brabazon. Adaptive Trading With Grammatical Evolution. In *Proc. CEC*. 2006.

[14] C. Drummond and N. Japkowicz. Warning: statistical benchmarking is addictive. Kicking the habit in machine learning. *Journal of Experimental & Theoretical Artificial Intelligence*, 22(1):67–80, 2010.

[15] M. Ebner and T. Tiede. Evolving Driving Controllers using Genetic Programming. In *Proc. IEEE Computational Intelligence and Games*. 2009.

[16] R. Feldt, et al. GP-Beagle: A benchmarking problem repository for the genetic programming community. In *Late Breaking Papers at GECCO*. 2000.

[17] E. Galván-López, J. Swafford, M. O'Neill, and A. Brabazon. Evolving a Ms. PacMan Controller Using Grammatical Evolution. In *Applications of Evolutionary Computation*. Springer, 2010.

[18] C. Gathercole and P. Ross. An Adverse Interaction between Crossover and Restricted Tree Depth in Genetic Programming. In *Proc. GECCO*. 1996.

[19] Genetic Programming Mailing List Discussion. http://tech.groups.yahoo.com/group/genetic_programming/message/5410, 2012. [Online: accessed 27–Jan-2012].

[20] D. E. Goldberg and U.-M. O'Reilly. Where does the Good Stuff Go, and Why? How Contextual Semantics Influence Program Structure in Simple Genetic Programming. In *Proc. EuroGP*. 1998.

[21] S. Gustafson, E. K. Burke, and N. Krasnogor. The Tree-String Problem: An Artificial Domain for Structure and Content Search. In *Proc. EuroGP*. 2005.

[22] J. V. Hansen, P. B. Lowry, R. D. Meservy, and D. M. McDonald. Genetic Programming for Prevention of Cyberterrorism through Dynamic and Evolving Intrusion Detection. *Decision Support Systems*, 43:1362–1374, 2007.

[23] S. Harding, J. F. Miller, and W. Banzhaf. Developments in Cartesian Genetic Programming: self-modifying CGP. *GPEM*, 11:397–439, 2010.

[24] A. Hauptman and M. Sipper. GP-EndChess: Using Genetic Programming to Evolve Chess Endgame Players. In *Proc. EuroGP*. Springer, 2005.

[25] T.-H. Hoang, et al. ORDERTREE: a New Test Problem for Genetic Programming. In *Proc. GECCO*. 2006.

[26] D. Jakobović and L. Budin. Dynamic Scheduling with Genetic Programming. In *Proc. EuroGP*. 2006.

[27] H. Jäske. Prediction of Sunspots by GP. In *Proc. Second Nordic Workshop on Genetic Algorithms*. Vaasa, Finland, 1996.

[28] M. Keijzer. Improving Symbolic Regression with Interval Arithmetic and Linear Scaling. In *Proc. EuroGP*. 2003.

[29] K. E. Kinnear, Jr. Evolving a Sort: Lessons in Genetic Programming. In *Proc. of the International Conference on Neural Networks*. 1993.

[30] M. F. Korns. Accuracy in Symbolic Regression. In *Proc. GPTP*. 2011.

[31] J. Koza. A Genetic Approach to the Truck Backer Upper Problem and the Inter-twined Spiral Problem. In *Proc. International Joint Conference on Neural Networks*. 1992.

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

[33] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.

[34] W. Langdon and W. Banzhaf. Repeated Patterns in Genetic Programming. *Natural Computing*, 7:589–613, 2008.

[35] W. B. Langdon and W. Banzhaf. Repeated Sequences in Linear Genetic Programming Genomes. *Complex Systems*, 15(4):285–306, 2005.

[36] W. B. Langdon and R. Poli. An Analysis of the MAX problem in Genetic Programming. In *Proc. GECCO*. 1997.

[37] W. B. Langdon, J. Rowsell, and A. P. Harrison. Creating Regular Expressions as mRNA Motifs with GP to Predict Human Exon Splitting. In *Proc. GECCO*. 2009.

[38] W. B. Langdon, O. Sanchez Graillet, and A. P. Harrison. Automated DNA Motif Discovery. arXiv.org, 2010.

[39] M. Lones, A. Tyrrell, S. Stepney, and L. Caves. Controlling Complex Dynamics with Artificial Biochemical Networks. In *Proc. EuroGP*, pp. 159–170. 2010.

[40] S. Lucas. Othello Competition. http://algoval.essex.ac.uk:8080/othello/html/Othello.html, 2012. [Online; accessed 27-Jan-2012].

[41] S. Lucas. The Physical Travelling Salesperson Problem. http://algoval.essex.ac.uk/ptsp/ptsp.html, 2012. [Online: accessed 27–Jan-2012].

[42] S. Luke. ECJ 20: An Evolutionary Computation Library in Java. http://cs.gmu.edu/∼eclab/projects/ecj/, 2012. [Online; accessed 27-Jan-2012].

[43] S. Luke and L. Panait. Is The Perfect The Enemy Of The Good? In *Proc. GECCO*. 2002.

[44] T. McConaghy. FFX: Fast, Scalable, Deterministic Symbolic Regression Technology. In *Proc. GPTP*. 2011.

[45] Q. U. Nguyen, et al. Semantically-Based Crossover in Genetic Programming: Application to Real-valued Symbolic Regression. *GPEM*, 12:91–119, 2011.

[46] M. Nicolau, M. Schoenauer, and W. Banzhaf. Evolving Genes to Balance a Pole. In *Proc. EuroGP*. 2010.

[47] J. Niehaus and W. Banzhaf. More on Computational Effort Statistics for Genetic Programming. In *Proc. EuroGP*. 2003.

[48] M. O'Neill, A. Brabazon, and E. Hemberg. Subtree Deactivation Control with Grammatical Genetic Programming in Dynamic Environments. In *Proc. CEC*. 2008.

[49] M. O'Neill, L. Vanneschi, S. Gustafson, and W. Banzhaf. Open Issues in Genetic Programming. *GPEM*, 11(3/4):339–363, 2010.

[50] L. Pagie and P. Hogeweg. Evolutionary Consequences of Coevolving Targets. *Evolutionary Computation*, 5:401–418, 1997.

[51] N. Paterson and M. Livesey. Performance Comparison in Genetic Programming. In *Late Breaking Papers at GECCO*. 2000.

[52] B. Punch, D. Zongker, and E. Goodman. The Royal Tree Problem, a Benchmark for Single and Multiple Population Genetic Programming. In *Advances in Genetic Programming 2*, pp. 299–316. MIT Press, 1996.

[53] R. Salomon. Performance degradation of genetic algorithms under coordinate rotation. In *Proc. EP*, pp. 153–161. 1996.

[54] H.-P. Schwefel. *Evolutionsstrategie und numerische Optimierung*. Ph.D. thesis, Technische Universität Berlin, Germany, 1975.

[55] B. Sendhoff, M. Roberts, and X. Yao. Evolutionary Computation Benchmarking Repository. *IEEE Computational Intelligence Magazine*, 1(4):50–60, 2006.

[56] S. Silva and E. Costa. Dynamic Limits for Bloat Control in Genetic Programming and a Review of Past and Current Bloat Theories. *GPEM*, 10:141–179, 2009.

[57] S. Silva and L. Vanneschi. State-of-the-Art Genetic Programming for Predicting Human Oral Bioavailability of Drugs. In *Proc. 4th International Workshop on Practical Applications of Computational Biology and Bioinformatics*. 2010.

[58] M. Sipper, Y. Azaria, A. Hauptman, and Y. Shichel. Designing an Evolutionary Strategizing Machine for Game Playing and Beyond. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(4):583 –593, 2007.

[59] J. Togelius, S. Karakovskiy, J. Koutnik, and J. Schmidhuber. Super Mario Evolution. In *Proc. IEEE Computational Intelligence and Games*. 2009.

[60] M. Tomassini, L. Vanneschi, P. Collard, and M. Clergue. A Study of Fitness Distance Correlation as a Difficulty Measure in Genetic Programming. *Evol. Comput.*, 13:213–239, June 2005.

[61] L. Vanneschi. *Theory and Practice for Efficient Genetic Programming*. Ph.D. thesis, Faculty of Sciences, University of Lausanne, Switzerland, 2004.

[62] L. Vanneschi, M. Castelli, and L. Manzoni. The K Landscapes: a Tunably Difficult Benchmark for Genetic Programming. In *Proc. GECCO*. 2011.

[63] L. Vanneschi and G. Cuccu. Variable Size Population for Dynamic Optimization with Genetic Programming. In *Proc. GECCO*. 2009.

[64] E. Vladislavleva, G. Smits, and D. Den Hertog. Order of Nonlinearity as a Complexity Measure for Models Generated by Symbolic Regression via Pareto Genetic Programming. *IEEE Trans EC*, 13(2):333–349, 2009.

[65] N. Wagner, Z. Michalewicz, M. Khouja, and R. McGregor. Time Series Forecasting for Dynamic Environments: The DyFor Genetic Program Model. *IEEE Trans EC*, 2007.

[66] J. Walker and J. Miller. Predicting Prime Numbers Using Cartesian Genetic Programming. In *Proc. EuroGP*. 2007.

[67] J. A. Walker, K. Völk, S. L. Smith, and J. F. Miller. Parallel Evolution using Multi-chromosome Cartesian Genetic Programming. *GPEM*, 10:417–445, 2009.

[68] M. Walker, H. Edwards, and C. Messom. The Reliability of Confidence Intervals for Computational Effort Comparisons. In *Proc. GECCO*. 2007.

[69] W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest. Automatically Finding Patches using Genetic Programming. In *Proc. 31st International Conference on Software Engineering*. 2009.

[70] P. Widera, J. Garibaldi, and N. Krasnogor. GP challenge: Evolving energy function for protein structure prediction. *GPEM*, 11:61–88, 2010.

[71] T. Yu. Hierarchical Processing for Evolving Recursive and Modular Programs Using Higher-Order Functions and Lambda Abstraction. *GPEM*, 2:345–380, 2001.

Name	Vars	Objective Function	Training Set	Testing Set	Function Set
Koza-1 [32]	1	$x^4 + x^3 + x^2 + x$	U[-1, 1, 20]	None	Koza
Koza-2 [33]	1	$x^5 - 2x^3 + x$	U[-1, 1, 20]	None	Koza
Koza-3 [33]	1	$x^6 - 2x^4 + x^2$	U[-1, 1, 20]	None	Koza
Nguyen-1 [45]	1	$x^3 + x^2 + x$	U[-1, 1, 20]	None	Koza
Nguyen-3 [45]	1	$x^5 + x^4 + x^3 + x^2 + x$	U[-1, 1, 20]	None	Koza
Nguyen-4 [45]	1	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	U[-1, 1, 20]	None	Koza
Nguyen-5 [45]	1	$\sin(x^2)\cos(x) - 1$	U[-1, 1, 20]	None	Koza
Nguyen-6 [45]	1	$\sin(x) + \sin(x + x^2)$	U[-1, 1, 20]	None	Koza
Nguyen-7 [45]	1	$\ln(x + 1) + \ln(x^2 + 1)$	U[0, 2, 20]	None	Koza
Nguyen-8 [45]	1	$\sqrt{x}$	U[0, 4, 20]	None	Koza
Nguyen-9 [45]	2	$\sin(x) + \sin(y^2)$	U[-1, 1, 100]	None	Koza
Nguyen-10 [45]	2	$2\sin(x)\cos(y)$	U[-1, 1, 100]	None	Koza
Pagie-1 [50]	2	$\frac{1}{1+x^{-4}} + \frac{1}{1+y^{-4}}$	E[-5, 5, 0.4]	None	Koza
Korns-1 [30]	5	$1.57 + (24.3\,v)$	U[-50, 50, 10000]	U[-50, 50, 10000]	Korns
Korns-2 [30]	5	$0.23 + 14.2\,\frac{v+y}{3\,w}$	U[-50, 50, 10000]	U[-50, 50, 10000]	Korns
Korns-3 [30]	5	$-5.41 + 4.9\,\frac{v-x+\frac{y}{w}}{3\,w}$	U[-50, 50, 10000]	U[-50, 50, 10000]	Korns
Korns-4 [30]	5	$-2.3 + 0.13\,\sin(z)$	U[-50, 50, 10000]	U[-50, 50, 10000]	Korns
Korns-5 [30]	5	$3 + 2.13\,\ln(w)$	U[-50, 50, 10000]	U[-50, 50, 10000]	Korns
Korns-6 [30]	5	$1.3 + 0.13\,\sqrt{x}$	U[-50, 50, 10000]	U[-50, 50, 10000]	Korns
Korns-7 [30]	5	$213.80940889(1 - e^{-0.54723748542\,x})$	U[-50, 50, 10000]	U[-50, 50, 10000]	Korns
Korns-8 [30]	5	$6.87 + 11\,\sqrt{7.23\,x\,v\,w}$	U[-50, 50, 10000]	U[-50, 50, 10000]	Korns
Korns-9 [30]	5	$\frac{\sqrt{x}}{\ln y}\,\frac{e^z}{v^2}$	U[-50, 50, 10000]	U[-50, 50, 10000]	Korns
Korns-10 [30]	5	$0.81 + 24.3\,\frac{2\,y+3\,z^2}{4\,(v)^3+5\,(w)^4}$	U[-50, 50, 10000]	U[-50, 50, 10000]	Korns
Korns-11 [30]	5	$6.87 + 11\cos(7.23\,x^3)$	U[-50, 50, 10000]	U[-50, 50, 10000]	Korns
Korns-12 [30]	5	$2 - 2.1\cos(9.8\,x)\,\sin(1.3\,w)$	U[-50, 50, 10000]	U[-50, 50, 10000]	Korns
Korns-13 [30]	5	$32 - 3\,\frac{\tan(x)}{\tan(y)}\,\frac{\tan(z)}{\tan(v)}$	U[-50, 50, 10000]	U[-50, 50, 10000]	Korns
Korns-14 [30]	5	$22 - 4.2\,(\cos(x) - \tan(y))\,\frac{\tanh(z)}{\sin(v)}$	U[-50, 50, 10000]	U[-50, 50, 10000]	Korns
Korns-15 [30]	5	$12 - 6\,\frac{\tan(x)}{e^y}(\ln(z) - \tan(v))$	U[-50, 50, 10000]	U[-50, 50, 10000]	Korns
Keijzer-1 [28]	1	$0.3\,x\sin(2\pi x)$	E[-1, 1, 0.1]	E[-1, 1, 0.001]	Keijzer
Keijzer-2 [28]	1	$0.3\,x\sin(2\pi x)$	E[-2, 2, 0.1]	E[-2, 2, 0.001]	Keijzer
Keijzer-3 [28]	1	$0.3\,x\sin(2\pi x)$	E[-3, 3, 0.1]	E[-3, 3, 0.001]	Keijzer
Keijzer-4 [28]	1	$x^3 e^{-x}\cos(x)\sin(x)(\sin^2(x)\cos(x) - 1)$	E[0, 10, 0.05]	E[0.05, 10.05, 0.05]	Keijzer
Keijzer-5 [28]	3	$\frac{30xz}{(x-10)y^2}$	x, z: U[-1, 1, 1000] y: U[1, 2, 1000]	x, z: U[-1, 1, 10000] y: U[1, 2, 10000]	Keijzer
Keijzer-6 [28]	1	$\sum_i^x \frac{1}{i}$	E[1, 50, 1]	E[1, 120, 1]	Keijzer
Keijzer-7 [28]	1	$\ln x$	E[1, 100, 1]	E[1, 100, 0.1]	Keijzer
Keijzer-8 [28]	1	$\sqrt{x}$	E[0, 100, 1]	E[0, 100, 0.1]	Keijzer
Keijzer-9 [28]	1	$\text{arcsinh}(x)$ *i.e.,* $\ln(x + \sqrt{x^2+1})$	E[0, 100, 1]	E[0, 100, 0.1]	Keijzer
Keijzer-10 [28]	2	x^y	U[0, 1, 100]	E[0, 1, 0.01]	Keijzer
Keijzer-11 [28]	2	$xy + \sin((x-1)(y-1))$	U[-3, 3, 20]	E[-3, 3, 0.01]	Keijzer
Keijzer-12 [28]	2	$x^4 - x^3 + \frac{y^2}{2} - y$	U[-3, 3, 20]	E[-3, 3, 0.01]	Keijzer
Keijzer-13 [28]	2	$6\sin(x)\cos(y)$	U[-3, 3, 20]	E[-3, 3, 0.01]	Keijzer
Keijzer-14 [28]	2	$\frac{8}{2+x^2+y^2}$	U[-3, 3, 20]	E[-3, 3, 0.01]	Keijzer
Keijzer-15 [28]	2	$\frac{x^3}{5} + \frac{y^3}{2} - y - x$	U[-3, 3, 20]	E[-3, 3, 0.01]	Keijzer
Vladislavleva-1 [64]	2	$\frac{e^{-(x-1)^2}}{1.2+(y-2.5)^2}$	U[0.3, 4, 100]	E[-0.2, 4.2, 0.1]	Vladislavleva-B
Vladislavleva-2 [64]	1	$e^{-x}x^3(\cos x\sin x)(\cos x\sin^2 x - 1)$	E[0.05, 10, 0.1]	E[-0.5, 10.5, 0.05]	Vladislavleva-C
Vladislavleva-3 [64]	2	$e^{-x}x^3(\cos x\sin x)(\cos x\sin^2 x - 1)(y-5)$	x: E[0.05, 10, 0.1] y: E[0.05, 10.05, 2]	x: E[-0.5, 10.5, 0.05] y: E[-0.5, 10.5, 0.5]	Vladislavleva-C
Vladislavleva-4 [64]	5	$\frac{10}{5+(x-3)^2+(y-3)^2+(z-3)^2+(v-3)^2+(w-3)^2}$	U[0.05, 6.05, 1024]	U[-0.25, 6.35, 5000]	Vladislavleva-A
Vladislavleva-5 [64]	3	$30\,\frac{(x-1)(z-1)}{y^2(x-10)}$	x: U[0.05, 2, 300] y: U[1, 2, 300] z: U[0.05, 2, 300]	x: E[-0.05, 2.1, 0.15] y: E[0.95, 2.05, 0.1] z: E[-0.05, 2.1, 0.15]	Vladislavleva-A
Vladislavleva-6 [64]	2	$6\sin(x)\cos(y)$	U[0.1, 5.9, 30]	E[-0.05, 6.05, 0.02]	Vladislavleva-B
Vladislavleva-7 [64]	2	$(x-3)(y-3) + 2\sin((x-4)(y-4))$	U[0.05, 6.05, 300]	U[-0.25, 6.35, 1000]	Vladislavleva-C
Vladislavleva-8 [64]	2	$\frac{(x-3)^4+(y-3)^3-(y-3)}{(y-2)^4+10}$	U[0.05, 6.05, 50]	E[-0.25, 6.35, 0.2]	Vladislavleva-A

Table 3: Symbolic Regression Benchmark Candidates. Variable names are, in order, *x, y, z, v, w*. Some benchmarks intentionally omit variables from the function. U[*a*,*b*,*c*] is *c* uniform random samples drawn from *a* to *b*, inclusive, for the variable. E[*a*,*b*,*c*] is a grid of points evenly spaced (for this variable) with an interval of *c*, from *a* to *b* inclusive. Testing and training sets are independent. See Table 2 for function sets.

Note. *This table has corrected values compared to the GECCO version. Nguyen-2, Nguyen-11, and Nguyen-12 have been omitted, and other corrections have been outlined in rectangular boxes.*