Benchmarking Grammar-Based Genetic Programming Algorithms

Christopher J. Headleand, Llyr Ap Cenydd and William J. Teahan

Abstract The publication of Grammatical Evolution (GE) led to the development of numerous variants of this Grammar-Based approach to Genetic Programming (GP). In order for these variants to be compared, the community requires a rigorous means for benchmarking the algorithms. However, GP as a field is not known for the quality of its benchmarking, with many identified problems, making direct comparisons either difficult or impossible in many cases. Aside from there not being a single, agreed-upon, benchmarking test, the tests currently utilised have shown a lack of standardisation. We motivate the following research by identifying some key issues with current benchmarking approaches. We then propose a standardised set of metrics for future benchmarking and demonstrate the use of these metrics by running a comparison of three Grammar-Based Genetic Programming methods. We conclude the work by discussing the results and proposing directions for future benchmarking.

1 Motivation

The benchmarking of Genetic Programming (GP) algorithms is a heavily debated topic. While there are standard problems which are typically used for comparison, the community has often discussed whether these challenges provide useful data.

One of the principle concerns raised is over a lack of standardisation which makes comparison across papers difficult or impossible [16]. Multiple papers describe different set-up conditions, and there has also been a lack of standard measures to evaluate the algorithms against.

With the development of Grammatical Evolution [9], its benchmarking raised additional concerns (discussed in section 2.2) as the use of a grammar adds further complications to the benchmarking process. As there are now several variants of

Christopher J. Headleand, Llyr Ap Cenydd, William J. Teahan
Bangor University, Wales e-mail: {c.headleand, eese60d, w.j.teahan}@bangor.ac.uk

this class of algorithm [4, 5, 3, 13, 10], the area is on the verge of being considered a discrete sub-field of GP which motivates us to consider some of the concerns with current benchmarking practice.

This research will explore the following objectives:

1. Discuss concerns identified in the literature along with the current practice of the benchmarking of Genetic Programming, focussing specifically on Grammar-Based variants.
2. Propose a standardised set of measures towards a new benchmarking strategy for future developments.
3. Run a benchmarking exercise of three comparative Grammar-Based algorithms using the proposed measures.

2 GP Benchmarking Concerns

2.1 Better Benchmarks Survey

At the GECCO conference in 2012 a position paper addressed the need to re-evaluate current benchmarking strategies [8]. The authors argued that this could be achieved through community feedback, an argument that led to a community survey later that year. We have selected some of the issues from that survey to highlight some of the GP communities feelings in regards to current benchmarking practices.

79 members of the GP community responded to the survey [16] with over 80% identifying that they run experiments to measure algorithm performance, and around 70% of practitioners responding that they performed well known GP benchmarks in their work. Furthermore, 94% of the GP community who responded to the questionnaire answered yes when asked whether they believed that the current experiment regime had disadvantages. Of that group, 85% responded that the lack of standardisation made it difficult/impossible to compare results across papers. Finally, 84% of the community members who answered the survey were in favour of the creation of a standardised GP benchmark suite.

While the authors were discouraged from pursuing the development of a new benchmarking suite, as an outcome of this work the authors propose a blacklist of problems to be avoided, and claim broad support for this blacklist from the community. This is based on the free-text response to question 23, “Can you suggest any existing benchmark problems you think should not be part of a benchmark suite?”. However, a close look at this dataset (available online [15]) indicates that there was very little consensus within the responses, with the most agreed upon response only receiving 6 nominations. This is a distinctly small number to claim “broad support” for blacklisting, and calls into question the validity of the proposed ‘blacklist’.

2.2 Grammar Search Space

There have been concerns over how various problems have been implemented in benchmarking exercises, preventing a fair comparison between algorithms. This has been proven to be the case in regards to the Santa Fe Trail, where studies have demonstrated key errors in the literature. In [12] the authors note that for the Santa Fe Trail for be used as a benchmarking test "the search space of programs must be semantically equivalent to the set of programs possible within the original Santa Fe Trail definition". The argument is presented that GE has not been fairly benchmarked against GP in the Santa Fe Trail problem as the grammar used is not equivalent to the original search space.

This was evaluated by Georgiou [2] who tested GE against both the grammar described in its original benchmarking paper [9] (BNF-Oneil) and a grammar based on Koza's original search space (BNF-Koza). The study demonstrated significant differences between the two visually similar grammars. Notably, while BNF-Oneil was more successful at finding a solution, the solutions generated were typically less efficient than the solutions generated using BNF-Koza. This supports the claim that the search spaces used were not semantically equivalent, and thus, direct comparisons cannot be made.

In addition [12], it has been noted that the documented limit on the number of steps allowed for an Artificial Ant to complete the trail has also been subjected to some error. Koza's original GP paper fixed the limit to 400 steps, which was assumed to be a mistake in later work [7] where the maximum limit was increased to 600 steps. A single GE publication [9] claims to have used the 600 step limit, which is assumed to be a mistype, as future publications discussed a 800 step limit. These claims of mistaken reporting are confirmed in [2] where, through simulations, it was proven that the best GP solution identified by Koza required 545 steps (exceeding the 400 steps quoted), and the best GE solution [9] required 615 steps (exceeding the 600 steps quoted).

Despite this, in the survey discussed in the previous section, the majority of participants voted that certain settings should be open to change during benchmarking which would include the number of steps. Yet it is an important consideration, as GP variants may not be comparable with identical settings; for example, a swarm based method may better suit a larger population than an evolution based approach.

3 Experimental Design

In the background to this work, we discussed some concerns over Grammar-Based Genetic Programming benchmarking. One of the main areas we identified was the lack of standardisation. In this work, we seek to address this concern by recommending a suite of five comparison metrics. It is worth noting that while we are focused on Grammar Based methods, these concerns have primarily come from the general GP field.

The metrics have been selected due to both there existence in the literature. For this study, three algorithms have been selected for comparison, Grammatical Evolution, Grammatical Herding and Seeded Grammatical Evolution.

3.1 Grammatical Evolution

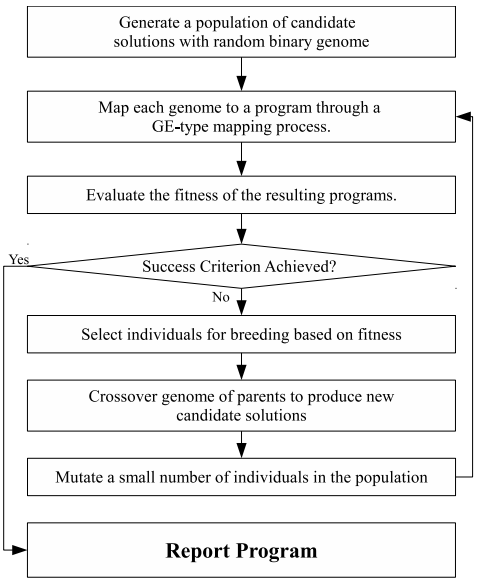

Fig. 1 An algorithmic representation of the Grammatical Evolution algorithm. A population of genomes consisting of a fixed length list of binary codons (typically 8 bits) is randomly generated. Each genome is then mapped via a grammar to produce a program. These programs are then evaluated and provided with a fitness score as with a typical genetic algorithm. Crossover and mutation are applied directly to the binary genomes (the genotype) not to the generated programs (phenotype).

Grammatical Evolution (GE) is a genetic algorithm based automatic programming method where candidate solutions are evolved over repeated generations. The system was designed in part to avoid some of the complications present in genetic programming (an alternate program synthesis method). By manipulating a bit string before mapping via usually a context free grammar, the candidate solution can be evolved without the need to directly manipulate code (see fig. 1).

3.2 Grammatical Herding

Grammatical Herding (GH) is a swarm-based automatic programming algorithm, which takes its inspiration from Particle Swarm Optimisation (PSO) and the grazing behaviour of horses; for a detailed explanation, see [4]. Each agent in GH is defined by a bit-string, which represents a location in the search space defined by a fixed number of 8-bit coordinates, equivalent to the genome of codons in Grammatical

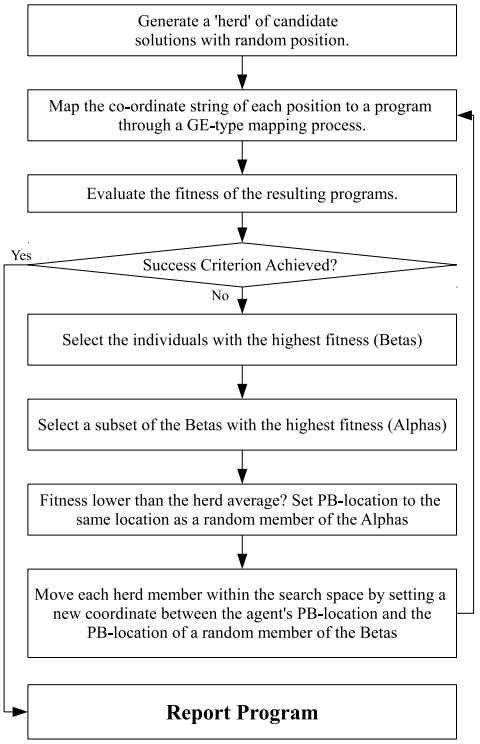

Fig. 2 Visual representation of the Grammatical herding algorithm. A herd of candidate solutions are generated with random initial positions. These co-ordinates within the search space are mapped through a GE-type mapping process to produce a program which is evaluated. The fittest agents are set as Betas, and the fittest members of the Betas as set as Alphas. If the agents fitness is lower than the herd average their position is set to that of one of the Alphas. The agents are then moved within the search space by moving between a position the agent has already visited, and the the best location found by a random Beta.

evolution. The agents move around the search space by selecting a position between themselves and a randomly selected individual with a high fitness. The position is weighted by the respective fitnesses of the agents.

3.3 Seeded Grammatical Evolution

Seeded Grammatical Evolution (sGE) is a hybrid algorithm which uses GH to pre-seed the population of GE. The purpose of the approach is to attempt to capitalise on the qualities of both algorithms, GH's ability to quickly evaluate a search space and the optimisation quality of GE [5]. In sGE, the population is transferred from one algorithm to the other at a point in the solution loop called 'handover' when a specific criterion was achieved. One criterion was a fitness threshold, a user defined fitness level (for problems with a known fitness), which, if achieved by an agent in the population, the handover event would be activated. Additionally an iteration threshold was included to provide a possible escape, if the seeding algorithm (GH) was unable to reach the desired fitness threshold. By including an iteration threshold, it ensured that if GH was unable to find a solution of a high enough fitness, the problem would be transferred to GE which may be more successful. It was also

discovered that the best solutions were produced when only a portion of the GH herd was transferred through handover to GE, with additional, randomly generated genomes added to complete the population. The percentage of agents passed from one algorithm to the other during handover is referred to as the 'seed ratio'.

3.4 Selected Problems

In this section, we will propose a set of measures for future benchmarking exercises. For the purpose of discussion, we have selected three benchmark tests which we believe to be comparable, as they are all problems that deal with agents with similar sensory-motor capabilities. These tests are the Santa Fe Trail, the Los Altos Hills Trail [6] and the Hampton Court Maze [14]. It is important to note that these challenges are not proposed as ideal benchmarking tests, but are used to demonstrate the proposed measures. The settings for GE, GH and sGE we have used in the benchmarking experiments described below are shown in Tables 1, 2 and 3.

3.4.1 Santa Fe Trail

The Santa Fe Trail (SFT), also called the artificial ant problem, is a common benchmark. The problem is to generate instructions for an artificial ant so that it can successfully traverse along a non-uniform path through a 32 by 32 toroidal grid (see figure 3). The generated program can use three movement operations, `move`, which moves the ant forward a single square, `turn-right` and `turn-left` which turn the ant 90 degrees to face a new direction, with each of these operations taking one time step. A sensing function`ifelse food-ahead` looks to the square directly ahead of the ant and executes one of two arguments based on whether the square contains food or not. These functions and operations are represented in the grammar detailed in listings 1. The fitness is calculated by counting the amount of food eaten by the agent before reaching the maximum number of time steps.

Listing. 1 Santa Fe Trail grammar

```
<expr>::=<line>|<expr><line>
<line>::=ifelse food-ahead[<expr>][<expr>]|<op>
<op>::=turn-left|turn-right|move
```

While there is debate on the quality and usefulness of the SFT benchmark [16, 12], there are also arguments in its favour. Firstly, it is one of the most commonly used benchmarks, providing historical results which span the entire history of Genetic Programming. Secondly, whilst the trail could be considered a "toy problem", new algorithms have continued to produce improved solutions to the problem, demonstrating that there is still room for continued research. A final argument is that despite the perceived simplicity of the problem, GP algorithms have failed to solve

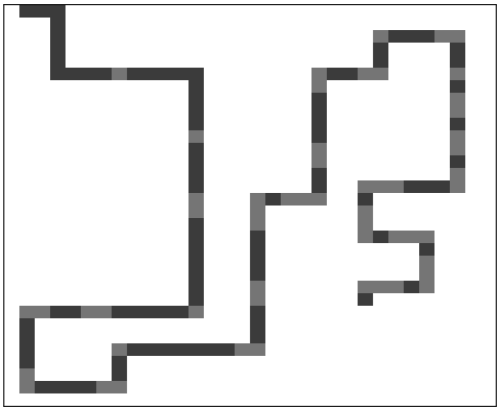

Fig. 3 The Santa Fe Trail benchmark environment, a 32 by 32 toroidal grid (a grid that wraps both vertically and horizontally). The the dark grey patches represent the 89 pieces of food along the trail, the light grey patches represent the gaps in the trail. The fitness is calculated by counting the amount of food eaten along the trail. Importantly food does not regenerate, so the food at each grey square can only be eaten once.

the benchmark reliably, for example, GE is only capable of an average success rate of 10% [2].

Algorithm	**Settings**
GE	Population = 1000; Max Generation = 50; Mutation Probability = 0.01; min Genome Length = 15 codons; max Genome Length = 30 codons; Wrap Limit = 10;
GH	Population = 1000; Max Generation = 50; Mutation = Alpha Wandering; Genome Length = 30 codons; numBetas = 25; numAlphas = 5; Wrap Limit = 10;
sGE	GH Settings = GH; GE Settings = GE; Fitness Threshold = 30; Iteration Threshold = 10; Seed Ratio = 25%

Table 1 Settings used for the Santa Fe Trail

3.4.2 Los Altos Hills Trail

The Los Altos Hills (LAH) benchmark is a more difficult variation of the Artificial ant problem introduced by Koza [6]. The objective is similar to the SFT, to generate a computer program capable of navigating an artificial ant to find all 157 pieces of food along a trail located in a 100x100 toroidal grid. The ant uses the same operations as with the Santa Fe Trail problem (see Listing 1). The LAH trail begins with the same irregularities, in the same order as the SFT. Then it introduces two new kinds of irregularity that appear toward the end of the trail. As with the SFT, the fitness is calculated based on the amount of food eaten before the maximum step limit is reached.

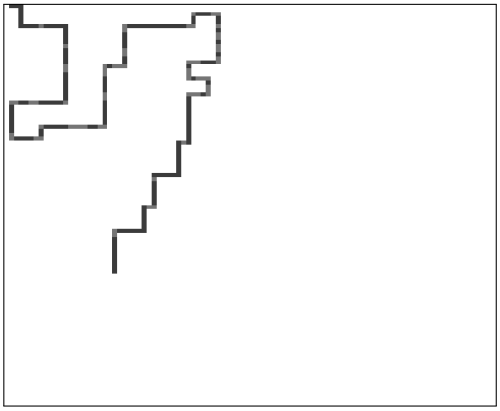

Fig. 4 The Los Altos Hills Trail benchmark environment, a 100 by 100 toroidal grid. The the dark grey patches represent the 157 pieces of food along the trail, the light grey represent the gaps in the trail.

Algorithm	Settings
GE	Population = 2000; Max Generation = 50; Mutation Probability = 0.01; min Genome Length = 50 codons; max Genome Length = 100 codons; Wrap Limit = 10;
GH	Population = 2000; Max Generation = 50; Mutation = Alpha Wandering; Genome Length = 100 codons; numBetas = 35; numAlphas = 10; Wrap Limit = 10;
sGE	GH Settings = GH; GE Settings = GE; Fitness Threshold = 30; Iteration Threshold = 10; Seed Ratio = 25%

Table 2 Settings used for the Los Altos Hills Trail

3.4.3 Hampton Court Maze

The Hampton Court Maze (HCM) problem, (first described in [14] and later proposed as a benchmark in [2]) is a simple, connected maze, designed on a 39 by 23 grid (see figure 5). The objective of the problem is to generate a program capable of guiding an agent from the entry square (light grey square at the bottom of the maze) to the centre (second light grey square identifies the entrance).

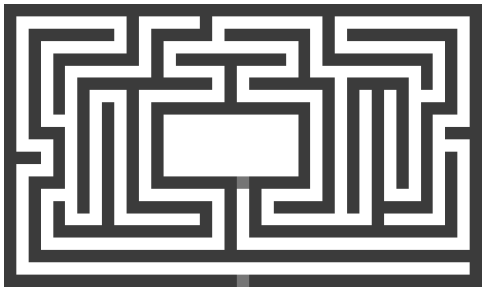

Fig. 5 The Hampton Court Maze benchmark environment, a 39 by 23 grid with walls. The dark grey patches represent the walls of the maze which are impassable by the agent. The light grey squares represent the entrance (bottom of the maze) and the goal (centre of the maze).

The grammar used is based on the SFT grammar (see Listing 2), with a key variation, the single sensing condition `food-ahead` is replaced with three possible sensing conditions. These conditions are `wall-ahead?`, `wall-left?` and`wall-right?`, which return true if a wall is detected in the sensed direction.

The fitness is calculated as a value between 0 and 1 using the following function.

$$F = 1/(1+(D/S))$$

where F is the fitness of the agent, D is the distance of the agent to the centre of the maze, and V is the number steps in the path traversed.

Listing. 2 Hampton Court grammar

```
<expr>::=<line>|<expr><line>
<line>::=ifelse<condition>[<expr>][<expr>]|<op>
<condition>::=wall-ahead?|wall-left?|wall-right?
<op>::=turn-left|turn-right|move
```

Algorithm	Settings
GE	Population = 100; Max Generation = 25; Mutation Probability = 0.01; min Genome Length = 15 codons; max Genome Length = 25 codons; Wrap Limit = 10;
GH	Population = 100; Max Generation = 25; Mutation = Alpha Wandering; Genome Length = 25 codons; numBetas = 10; numAlphas = 2; Wrap Limit = 10;
sGE	GH Settings = GH; GE Settings = GE; Fitness Threshold = 30; Iteration Threshold = 10; Seed Ratio = 25%

Table 3 Settings used for the Hampton Court Maze

4 Proposed Standardised Measures

In [1], four main categories of problem are discussed, notably, Black Art Problems, Programming the Unprogrammable (PTU), Commercially Useful New Inventions (CUNI) and Optimal Control. Different problems may also suit algorithms with different qualities which may not be considered "state of the art" by current measures.

We propose a standardised set of five metrics discussed over the following sections, and demonstrate the insights they provide by running a benchmark comparison of the three algorithms discussed in section 3. An experiment of 10,000 runs were completed for each of the three algorithms against each of the three benchmark tests discussed in the previous section.

4.1 Metric 1: Success Rate

This measure is a percentage, which represents how often the algorithm is capable of producing a full fitness result (on problems with a known full fitness). This metric is one of the most commonly used in the literature as it provides a good measure of the reliability of the algorithm. For problems with an unknown 'full fitness', we propose that a threshold level should be used, with all experiments capable of achieving or exceeding that threshold being considered successful.

	GH	GE	sGE
Santa Fe Trail	6.21%	32.95%	3.25%
Los Altos Hills Trail	0%	0%	0%
Hampton Court Maze	4.87%	2.41%	8.94%

Table 4 Metric 1: Success Rate for GE, GH and sGE

The data in Table 4 shows us that, using the settings defined in the previous section, GE achieves the strongest results on the SFT, though it under preforms on the HCM. Similarly, while sGE produced the weakest results on the SFT, it out performs GE and GH on the HCM. This result shows that multiple problems should be considered when benchmarking as certain algorithms may be better fitted for different domains. Also, none of the algorithms were capable of producing a full-fitness solution to the LAT.

4.2 Metric 2: Average Fitness

As demonstrated with our first metric, typically a full fitness solution (if full fitness is known) may only be found a relatively small amount of the time, even on problems that are perceived to be simple such as the Santa Fe Trail. However, there are examples where a full fitness solution may not be required, and in these cases an understanding of the typical "average fitness" may provide a better insight.

	GH	GE	sGE
Santa Fe Trail	54.50	74.98	59.51
Los Altos Hills Trail	42.16	46.75	62.44
Hampton Court Maze	99.23	98.18	99.25

Table 5 Metric 2: Average Fitness results for GE, GH and sGE

The average fitness results (table 5) demonstrate that on the SFT, sGE is capable of producing higher fitness results than GH, though GE is the most reliable overall. In addition, the results identify one possible flaw with the fitness function currently

defined for the HCM, notably that it biases towards large values, with little fidelity in the upper fitness values, possibly explaining the poor performance in Metric 1.

4.3 Metric 3: Solution Development

The fourth proposed metric examines how quickly an algorithm develops its solutions (as opposed to Metrics 1 and 2 which gauge overall success) by exploring how many generations/iterations it takes, on average, to achieve 25, 50, 75 and 100% fitness. However, this again relies upon knowing a full fitness value for the problem, so, as with Metric 1, we propose that a threshold system is used for problems where the full fitness is unknown. By examining this data, we can compare the number of evaluations required to produce high fitness solutions in order to gauge the relative "responsiveness" of the algorithm to the problem.

	Santa Fe Trail			Los Altos Hills Trail			Hampton Court Maze		
	GH	GE	sGE	GH	GE	sGE	GH	GE	sGE
25% of full Fitness	4	13	7	7	12	7	1	2	1
50%of full Fitness	12	31	21	-	-	47	1	3	2
75% of full Fitness	26	35	31	-	-	-	2	3	4
100% of full Fitness	45	46	42	-	-	-	3	5	5

Table 6 Metric 3: Solution Development results for GE, GH and sGE

Comparing the algorithms using the metric (see table 6 and figure 6), we see that GH produces high fitness solutions in less generations than sGE or GE . This trend can be seen across all three problems, although the small number of generations in the HCM makes this hard to observe. GE typically takes longer to develop solutions than GH (more than double the number of generations to reach 25% and 50% fitness) though it discovers full fitness solutions in a similar number of generations. In comparison, sGE is quicker than GE but less so than GH on the SFT and HCM, though it is the best performing on the LAH.

4.4 Metric 4: Best Solution (Highest Quality)

The final proposed metric is one of the most often used in the literature, where the best solution from all simulation runs is reported. This metric demonstrates the quality of solutions that an algorithm is capable of producing against a specific benchmark. For the problems we have evaluated, the highest quality solution is the one capable of achieving full fitness with the least number of steps.

From the results generated by this metric (table 7), we can observe that whilst GE produced the best solution for the SFT, GH and sGE produced better results in the

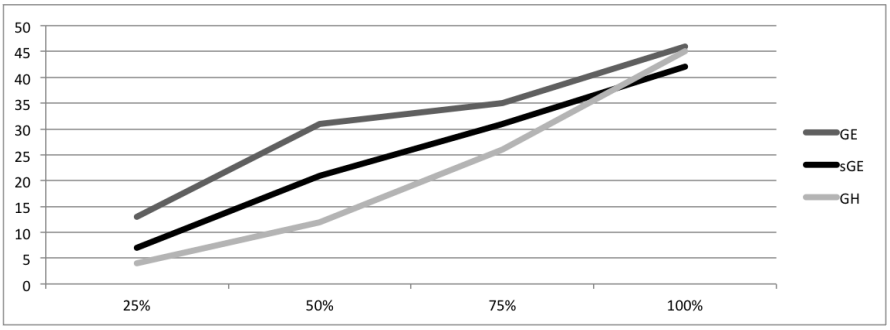

Fig. 6 The average number of generations each of the three algorithms require to achieve increasing levels of fitness for the Santa Fe Trail.

	GH	GE	sGE
Santa Fe Trail	405	347	386
Los Altos Hills Trail	-	-	-
Hampton Court Maze	602	690	602

Table 7 Metric 4: Best Solution results for GE, GH and sGE

HCM (the LAH not being solved by any of the algorithms).However, better results have been reported in the literature, for example, our sGE paper describes a solution capable of completing the SFT in 303 steps. Clearly, focusing too closely just on the best solution leads to the danger that other qualities of an algorithm are overlooked.

4.5 Metric 5: Average Quality of Successful Runs

This metric concerns the average 'quality' of successful solutions produced by the benchmarked algorithm, which may not be applicable for all benchmarking problems. For artificial ant problems, this value is simply the average number of steps taken for the ant to complete the problem.

	GH	GE	sGE
Santa Fe Trail	607.16	553.72	642.93
Los Altos Hills Trail	-	-	-
Hampton Court Maze	682.87	883.42	672.46

Table 8 Metric 5: Average Quality of Successful Runs results for GE, GH and sGE

The analysis of Metric 5 (see table 8) further supports the conclusions gained from Metric 1. Notably, that whilst GE achieves superior results on the SFT, it underperforms on the HCM in comparison to GH and sGE.

5 Conclusions and Next Steps

Three principle objectives have been addressed in this research: firstly, to identify some of the concerns which have been raised in regards to the benchmarking of three Grammar-Based Genetic Programming algorithms (Grammatical Evolution, Grammatical Herding and Seeded Grammatical Evolution); secondly, to propose a standardised set of metrics for future benchmarking to address one of the current concerns; and finally, to run a benchmarking exercise to demonstrate the insights that can be gained through these measures. An additional contribution has been the provision of further benchmarking data in order to compare the three algorithms.

The work has demonstrated that a standardised set of comparison metrics can aid in comparison between algorithms. Also, by including a more diverse set of metrics than those currently used, algorithms can be better matched to specific problems by understanding their unique qualities. For example, while sGE typically produced the poorest results in success rates and average quality, it was the only algorithm to achieve over 50% fitness in the Los Altos Hills.

Throughout this work, we have referred to the 'better benchmarks' survey [16] and the subsequent analysis. In this work, 84% of GP community members who answered the survey were in favour of the creation of a standardised GP benchmark suite, mainly because it would simplify comparisons between techniques. Additionally, devising high quality benchmarks has been identified as an important issue for the foreseeable future of the GP development [11]. However, the authors of the survey were discouraged from this course of action, instead, the research proposed a blacklist, which included the Artificial Ant Problem. However, this research has demonstrated even the simple version of this challenge (the SFT) is not solved reliably, and the more complex version (LAH) was significantly more difficult, as none of the evaluated algorithms were capable of producing a full-fitness solution.

During the benchmarking process, we did a large number of runs (a total of 90'000 experiments). This is significantly larger than the typical number of experiments reported in benchmarking studies (typically between 100 and 500 are usually run). We propose that a much larger sample should be run in future studies as by observing the averages we can avoid inconsistent results. For example, in the study describing sGE, we reported much stronger results based on this work we now believe that these were statistical anomalies, a risk of any stochastic search algorithms.

The next phase of this work will be to engage with the genetic programming community in the development of a standardised benchmarking suite. We propose that this should take the form of an open-source software package to be maintained by the GP community.

Acknowledgements: The authors would like to thank HPC Wales for providing their facilities and technical support during the running of the experiments described in this research. Chris Headleand would also like to thank Fujitsu for their ongoing financial support.

References

1. R Behera, BB Pati, BP Panigrahi, and S Misra. An application of genetic programming for power system planning and operation. *International Journal on Control System & Instrumentation*, 3(2), 2012.
2. Loukas Georgiou. *Constituent Grammatical Evolution*. PhD thesis, University of Wales, Bangor, 2004.
3. Loukas Georgiou and William J Teahan. Constituent grammatical evolution. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Two*, pages 1261–1268. AAAI Press, 2011.
4. Chris Headleand and William J Teahan. Grammatical herding. *J Comput Sci Syst Biol*, 6:043–047, 2013.
5. Chris Headleand and William J Teahan. Swarm based population seeding of grammatical evolution. *J Comput Sci Syst Biol*, 6:132–135, 2013.
6. John R Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.
7. William B Langdon and Riccardo Poli. Better trained ants for genetic programming. 1998.
8. James McDermott, David R White, Sean Luke, Luca Manzoni, Mauro Castelli, Leonardo Vanneschi, Wojciech Jaskowski, Krzysztof Krawiec, Robin Harper, Kenneth De Jong, et al. Genetic programming needs better benchmarks. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, pages 791–798. ACM, 2012.
9. Michael O'Neil and Conor Ryan. Grammatical evolution. In *Grammatical Evolution*, pages 33–47. Springer, 2003.
10. Michael O'Neill and Anthony Brabazon. Grammatical swarm: The generation of programs by social programming. *Natural Computing*, 5(4):443–462, 2006.
11. Michael O'Neill, Leonardo Vanneschi, Steven Gustafson, and Wolfgang Banzhaf. Open issues in genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):339–363, 2010.
12. Denis Robilliard, Sébastien Mahler, Dominique Verhaghe, and Cyril Fonlupt. Santa fe trail hazards. In *Artificial Evolution*, pages 1–12. Springer, 2006.
13. Tapas Si, Arunava De, and Anup Kumar Bhattacharjee. Grammatical bee colony. In *Swarm, Evolutionary, and Memetic Computing*, pages 436–445. Springer, 2013.
14. William John Teahan. *Artificial Intelligence–Agent Behaviour*. BookBoon, 2010.
15. David R White, James McDermott, Mauro Castelli, Luca Manzoni, Brian W Goldman, Gabriel Kronberger, Wojciech Jaśkowski, Una-May OReilly, and Sean Luke. Free text responses. Online, September 2012. http://gpbenchmarks.org/wp-content/uploads/2012/09/SurveyFreeTextResponses.txt.
16. David R White, James McDermott, Mauro Castelli, Luca Manzoni, Brian W Goldman, Gabriel Kronberger, Wojciech Jaśkowski, Una-May OReilly, and Sean Luke. Better gp benchmarks: community survey results and proposals. *Genetic Programming and Evolvable Machines*, 14(1):3–29, 2013.