# Modelling Evolvability in Genetic Programming

Benjamin Fowler(✉) and Wolfgang Banzhaf

Memorial University of Newfoundland, St. John's, Canada
{b.fowler,banzhaf}@mun.ca

**Abstract.** We develop a tree-based genetic programming system capable of modelling evolvability during evolution through machine learning algorithms, and exploiting those models to increase the efficiency and final fitness. Existing methods of determining evolvability require too much computational time to be effective in any practical sense. By being able to model evolvability instead, computational time may be reduced. This will be done first by demonstrating the effectiveness of modelling these properties *a priori*, before expanding the system to show its effectiveness as evolution occurs.

**Keywords:** Genetic programming · Evolvability · Meta-learning · Artificial neural networks

## 1 Introduction

Genetic Programming (GP) [17] would be more effective and efficient if we could select based on how individuals may contribute to evolutionary processes, not solely based on their fitness. In other words, it would be useful to select individuals that may contribute more to the fitness of future generations, individuals that are more evolvable. Evolvability indicates the capacity of an individual to improve its fitness [1]. We opt to define evolvability as the probability of a mutation operation resulting in a strictly positive fitness change, the reasoning for which is detailed in Sect. 3. However, it is expensive to measure; it is computationally impractical to measure evolvability for individuals and then use evolvability to aid selection processes.

Biologically, evolvability has been defined as the ability of a population to respond to selection [5]. In his review of other works, Pigliucci [23] comes the conclusions that evolvability, however it may be defined, itself evolves, but there is a lack of evidence to see if this is caused by natural selection or other evolutionary mechanisms. Wilder & Stanley [32] show adaptive processes in gene regulatory networks produce evolvability individuals, but divergent processes produce evolvable populations. Altenberg [2] notes that evolutionary computation brought about more biological-based evolutionary interest in evolvability; evolvability in organisms was simply presumed to exist. Altenberg further notes that there were 170 papers published in 2013 alone that mention the evolution of evolvability. Evolvability in genetic programming refers to the ability of an


M. Heywood et al. (Eds.): EuroGP 2016, LNCS 9594, pp. 215–229, 2016.
DOI: 10.1007/978-3-319-30668-1_14

individual or population of programs to produce higher fitness individuals [1]. To encourage more evolvable programs, it would be beneficial to quantify evolvability, and exploit these quantities when judging fitness. Kattan & Ong [16] use Bayesian inference to adjust fitness functions in order to encourage evolvability. Using genotype-phenotype or genotype-fitness mappings could also prove beneficial to the study of these properties [19]. Properties related to evolvability and robustness, such as self-repair, may emerge in artificial systems without modifying the underlying systems to encourage their emergence [22].

We model evolvability using GP properties that are computationally inexpensive to generate, and, once such models are developed, evolvability may be calculated and utilized in the GP selection process to improve evolution. This is accomplished by generating properties related to evolvability, as well as evolvability itself, *a priori* for a specific problem, then developing a machine learning model for evolvability. Evolvability may then be calculated during evolution. GP may be utilized to solve the problem while predicting evolvability values for individuals, which may then be used to influence selection. Section 2 reviews related literature, Sect. 3 describes the problem domain, the system to be used, and the risks in regard to the applicability of the method. Section 4 describes the specific problem parameters that are examined, the design of the experiments that are conducted, presents the results, and discusses the results and future work.

## 2 Related Work

In genetic programming, to maximize fitness, we favour programs that currently have greater fitness. There are various selection methods [17,24] that apply varying amounts of selection pressure. However, selection is inherently driven by differences in fitness. This process does not directly consider structural properties of programs, such as bloat [25]. Instead, selection is meant to allow more desirable structural properties, which allow greater fitness, to emerge [3]. The process further ignores how changes in genotype changes the phenotype [7], how this affects fitness, and how this might skip optima in the fitness search space [27]. Evolvability is related to these structural properties; by analyzing their interrelatedness, we should gain insight to improve genetic programming by accommodating them, in lieu of ignoring them, by measuring and selecting for evolvability. Basset *et al.* [4] postulate bloat occurs because offspring are not effectively inheriting the phenotype traits from their parents. The notion is that ideally, we want to perform a cross-over on the phenotype, not the genotype.

Altenberg [1] describes a method of measuring evolvability through a transmission function, deriving a formula that describes the probability that a population (not an individual) will produce an individual that has greater fitness than any in the existing population. Essentially, one considers all possible results of genetic operations on all individuals in a population, and computes the probability that an individual will be produced whose fitness surpasses that of the existing population. This is an intuitive method of measuring evolvability; an individual is highly evolvable if its potential offspring are more likely to be more fit.

It also requires extensive computations; instead of conduction one genetic operation on an individual, we need to conduct all possible operations, on each individual. Then, each fitness case needs to be evaluated for each such operation. Thus, measuring evolvability in this way would require many orders of magnitude more computational effort than standard GP.

Pragmatically, exhaustive searches to measure evolvability can be improved upon by using sampling or estimation [29]. This is much more computationally feasible, but the same question is posed when using exhaustive search; why not simply keep the resulting most fit individual? Sampling still adds a significant computational burden, as sufficient samples are required to estimate evolvability, but even adding a single sample doubles the computational time required in standard GP. As such, evolvability is too computationally expensive to measure directly. Instead, current literature efforts to exploit evolvability do so indirectly, without having to measure it, such as defining new evolvability metrics [28] and characterizing evolvability's relatedness to other properties [12]. There has been some success in determining how much to select for evolvability, but only under limited circumstances [30]. Li *et al.* [18] have had success balancing fitness selection with diversity metrics, using multi-objective optimization. Multi-objective approaches using Pareto dominance or hypervolume indicators, with various objective criteria, are well-studied in the literature, generally targeting concepts related to evolvability, such as diversity, rather than evolvability itself [9,26].

## 3 Approach

An extendible synthetic domain will be most useful for this work. White *et al.* [31] propose a set of benchmark problems to replace ageing, simple problems. Among the list of new synthetic, extendible problems is the order tree problem [13]. A synthetic, extendible problem such as the order tree problem allows for tunable problem difficulty, thus the conditions under which the use of evolvability is most beneficial may be more easily examined.

An order tree domain may be defined as having a size of $n$. Function nodes and terminal nodes take on values of whole numbers on a range of $[0, n-1]$. Function nodes all take two arguments. The fitness of a solution is calculated in a top-down fashion. A node will add 1 to the total fitness of the solution if its numeric value is strictly greater than its parent's numeric value, and, in the restricted version of the order tree problem, only if the parent is also adding to the total fitness of the solution. Thus, the optimal solution is an ordered tree, where the root is the functional node valued at 0, its children are valued at 1, and so on. The order tree problem is useful because the difficulty is tunable to $n$, where difficulty may be increased by increasing $n$, thus increasing our functional and terminal set. Furthermore, node dormancy is easily determined as a by-product of fitness evaluation. Problem difficulty may be further tuned by adjusting how much fitness is contributed by each node; by weighing higher-valued nodes more greatly (i.e., by increasing fitness greater than 1 for any given node) the fitness structure may be changed. This alters the fitness landscape, and

encourages higher-valued nodes to be selected, even though this interferes with finding the optimal solution. A more evolvable solution would still favour lower-valued nodes. This allows for tuning the desirability of evolvability. Tuning the order tree problem in these two ways will demonstrate the problem conditions for the effectiveness of the proposed system.

This work will focus on one representation of GP, tree-based representation, and the modelling of evolvability *a priori*. This will indicate if modelling evolvability is viable, and may be extended to dynamically built models, and data and model sharing between related problems. Eventually, the goal is apply the system to real-world problems where GP is known to excel relative to other algorithms or human efforts, in order to obtain better solutions more quickly. There are many ways to expand after initial efforts in controllable problems are shown to be functional; it may be certain classes of problems are more receptive to the methodology, alternate GP representations may be preferred, or certain structural properties are much more significant than others, and each of these may not be independent with another.

As we are concerned with modelling evolvability, structural properties of individuals in GP which may be easily measured (that is, without a significant increase in computational resources) are of interest. There can be significant sections of individuals in GP which, in addition to not affecting fitness, provide no change in output, regardless of input. These sections are referred to as introns in GP literature. Introns may be categorized by their behaviour; Nordin *et al.* [21] propose several categories. They are categorized based on whether their lack of contribution of fitness is due to the fitness cases themselves, or apply to the entire problem domain, and whether cross-over operations can introduce a change in fitness. Identifying all introns is computationally expensive. However, it is computationally inexpensive to identify a certain type of intron, that occurs when a code section is never executed for any fitness case; these are dormant sections [14]. In tree-based or cartesian GP, these nodes are referred to as dormant nodes, and can account for the majority of the nodes, around 90 %-95 % [14,20]. Despite the apparent uselessness of dormant sections of code, dormancy is helpful; if dormant nodes are detected and removed, performance actually suffers, and more generations are required to reach comparable solutions [14]. Locality is another structural property in GP, relating to evolvability, robustness, and genotype-phenotype mappings. A problem has high locality if neighbouring genotypes correspond to neighbouring phenotypes [7]. High locality problems are generally easier to solve. Low locality indicates a more rugged search space, which indicates a more difficult search. Furthermore, the ruggedness describes how robust and evolvable the search space is [8,15,27,29]. Neutral genetic operations represent plateaus on the search space. Evolvability and robustness act as counterparts; steep inclines indicate great fitness gains moving toward optima, but also great fitness losses moving away from optima. There is motivation to organize all the structural properties together, to analyze their interactions, for they all affect problem difficulty, the efficiency of the search, and the efficacy of the search.

We propose a narrower scope of tree-based GP. We further limit to algorithms available in the Waikato Environment for Knowledge Analysis (WEKA), an accessible machine learning software suite [10]. This would allow for rapid experimentation on a number of different algorithms, as well as some convenient visualization and analytical tools that may yield insight into the nature and relatedness of the structural properties of the individuals. The predicted evolvability of an individual shall be used to guide the selection process in various ways, to find the most beneficial usage of evolvability. Such a system would only indicate that, with enough data generated *a priori* for a specific problem, models for structural properties could be built which can benefit evolution.

## 4 Experimental Design and Results

We design and implement a tree-based GP system that records measurements of evolvability by sampling, along with records of other structural properties, such as dormancy, per every generation that occurs during evolution. Further, we model evolvability using these records, then exploit their predicted values during evolution, in order to develop a faster and more efficient GP system. This shall be accomplished by modifying an existing tree-based GP system to track and record additional structural elements, for specific problems. Once generated, evolvability will be modelled using machine learning algorithms. These models will be incorporated into the existing tree-based GP system, to predict evolvability without the need to sample them. Then, the predicted values will be used to guide selection beyond the standard fitness measurements. Initially, we consider the various structural properties of solutions generated by genetic programming for a small parity problem, and a contrived regression problem consisting of a single input variable. Once this system is verified to yield improvements in solution accuracy and efficiency for these simpler problems, the system will be modified further to develop models for evolvability as evolution actually occurs.

We modify a minimalist version of Open BEAGLE, referred to as BEAGLE Puppy [6]. Open BEAGLE is an evolutionary computation framework, developed in C++. BEAGLE Puppy utilizes the core GP algorithms of Open BEAGLE, but is simpler to modify for our purposes, since it is minimalist. It contains a tree-based GP implementation of simple parity and regression problems. Our methodology requires editing the selection process, additional tracking of various statistics (such as dormancy), sampling for evolvability, and eventually, dynamic modelling of evolvability. These are easier to implement by editing core GP algorithms. Furthermore, we are afforded more flexibility by working with a lesser amount of code. Another advantage is working with an efficient object-oriented language. To sample for evolvability, more fitness cases need to be evaluated, which is already the most computationally intensive part of GP. Object-oriented code allows for more easily reusable code; as we expand into more difficult problem domains, we can reuse our efforts building simpler ones.

Modelling evolvability, however, requires faster machine learning algorithms than evolutionary computation provides. These are provided by a machine learning suite, the Waikato Environment for Knowledge Analysis (WEKA) [10].

This implementation can be expanded to allow models to be generated by WEKA as evolution occurs. The accuracy of these models can be monitored until they are sufficiently accurate, in order to stop sampling evolvability, and instead, predict it. Sampling may still be interleaved to ensure the models remain accurate.

### 4.1 Sampling Accuracy

This subsection describes the effectiveness of altering the fitness mechanism of standard GP to consider evolvability in various ways. This will demonstrate the effectiveness of using sampled evolvability to improve GP. The significance of evolvability on selection will be monitored, so the optimal amount of selection can be used. Once the necessary conditions for the effectiveness of using evolvability in selection has been determined, it can be used to gauge the effectiveness of modelling evolvability.

Calculating precise evolvability is computationally infeasible for practical genetic programming. Instead of calculating all possible results of all possible genetic operations for any given individual genetic program, we elect to instead conduct sampling, where a random subset of all possible genetic operations are applied. Sampling can approximate the precise calculation of evolvability for a fraction of the computational cost. How many samples are necessary to produce a reasonable approximation of the correct evolvability, such that selection errors will occur less than 5 % of the time? How accurate must the approximation be to achieve an improvement when using evolvability to guide selection?

In order to answer these questions, we must first define more experimental parameters. Several evolvability metrics exist. We opt to define evolvability as the probability of a mutation operation resulting in a strictly positive fitness change. This may differ from other metrics in two ways: probability of change instead of magnitudes of change, and excluding neutral changes. Preliminary experiments indicated that selecting for the probability of a positive fitness change were more productive than when neutral changes were included. Similarly, they indicated that using probabilities instead of average magnitude of fitness change were more productive. Mutation operations are considered, in order to evaluate evolvability of individuals without considering how the gene pool of the population would affect measurements, as it would measuring evolvability using cross-over operations. More samples are required to achieve a good approximation if we consider the average magnitude of change of fitness. Furthermore, selecting for greater positive magnitude of fitness change will heavily bias evolution toward lower fitness individuals, as they have the greatest capacity for fitness improvement. We discount neutral changes, as this encourages a bias toward large trees in the order tree problem, as they have many possible neutral mutations. Considering neutral changes to be equivalent to positive ones encourages robustness, but not evolvability.

To determine how many samples are necessary to achieve a reasonable approximation of evolvability, we conduct the following experiment. We vary the number of samples while keeping other experimental conditions consistent, and compare the sampled evolvability to the strongest approximation (using the

**Table 1.** Evolutionary parameters for varying the number of samples.

| | | | |
|---|---|---|---|
| Population Size | 50 | Crossover Probability | 0.9 |
| Tournament Size | 3 | Probability of Non-Terminal Crossover | 0.9 |
| Min Initial Depth | 3 | Standard Mutation Probability | 0.05 |
| Max Initial Depth | 6 | Mutation Max Regen Depth | 2 |
| Max Depth | 6 | Swap Mutation Probability | 0.05 |
| Initial Grow Probability | 0.5 | Probability to Mutate a Function Node | 0.5 |

largest sample size). Even for a smaller order tree problem, it is still computationally infeasible to calculate the correct evolvability. By selecting for fitness, higher fitness individuals are more likely to occur. If we select for evolvability, more evolvable individuals are likely to occur.

The experimental parameters are shown in Table 1. 10000 runs with different random seeds are completed for standard GP, and 1000 runs for everything else. The max depth was raised for the 7th and 8th order tree problems. These parameters are consistent throughout the experiments in this work. We define the mean absolute error of evolvability as follows:

$$MAE = \frac{1}{n} * \sum_{k=1}^{n} |e'_k - e_k| \tag{1}$$

where $n$ is the number of runs, $e'_k$ is the measured evolvability for 1000 samples, and $e_k$ is the measured evolvability of the indicated number of samples.

Figure 1 shows that a reasonable approximation for evolvability occurs when the number of samples is about 100. Similar experiments for higher order tree problems show that holds true. Since the purpose of evolvability for this system is to be used with an altered fitness function in order to guide selection, the required accuracy of sampling and modelling evolvability is proportional to the actual influence evolvability has on selection. Therefore, it is necessary to choose precisely how evolvability will guide selection in order to determine how many samples are sufficient to ensure accurate selection. This can be evaluated by

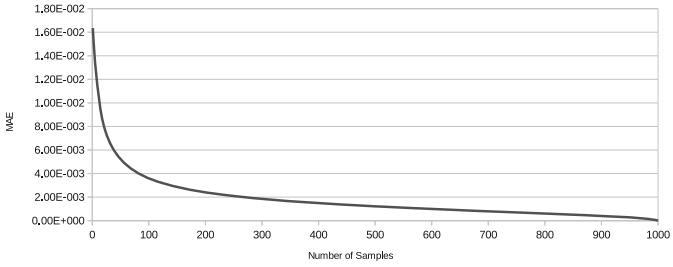


**Fig. 1.** Mean Absolute Error of Evolvability for number of samples compared to 1000 samples.

using the modified fitness function, and compare which individuals are selected when using a reduced number of samples (or a model) for evolvability with individuals selected using a large number of samples. Discrepancies indicate that an individual was incorrectly selected.

### 4.2 Selection of Evolvability

We need to determine how to select for evolvability. To determine the optimal selection amount, we conduct the following experiment. We vary the standard GP selection mechanism by using the sampled evolvability in various ways, while keeping other experimental conditions consistent. There are several methods to guide selection with evolvability. One is a threshold for fitness; if fitness of two individuals falls within a specific threshold, then we select the one with greater evolvability. Another is a weighted sum; we sum the fitness and evolvability, each weighted by a specified amount, and select individuals according to their weighted sum. We can allow a generational modifier for using a weighted sum; as the number of generations increase, we select less strongly for evolvability. Using a weighted sum and a generational modifier, we have, formally:

$$F' = \begin{bmatrix} (f + \frac{e*p(g_{max}-g)}{g_{max}}) \text{ if } & g < g_{max} \\ f & \text{otherwise} \end{bmatrix} \tag{2}$$

where $F'$ is the adjusted fitness function, $f$ is the standard fitness function, $e$ is evolvability, $p$ is the weight parameter, $g_{max}$ is the maximum generation parameter, and $g$ is the current generation. This translates to the fitness function being modified by the probability of a change being positive multiplied by the weight parameter for the initial population, and where this modifier linearly approaches zero as the generation increases. Upon reaching zero, the modifier becomes zero for the remaining generations, rendering evolvability uninfluential. This is desirable because evolvability should become less significant as the number of generations increase, as standard fitness approaches optimal values. Maximizing standard fitness becomes the only goal when evolution completes. Eventually, we would just want to select for standard fitness. We conduct experiments for different Order Tree problems under varying selection pressures (varying the weight and maximum generation parameters). The other experimental parameters are identical to the previous experiment, as shown in Table 1.

Figures 2, 3, and 4 shows the average maximum fitness as the generation increases, for a subset of the tested problems, for clarity. This indicates that using modified fitness functions that use evolvability in addition to standard fitness outperform using standard fitness functions alone. They indicate the general appropriate proportion of evolvability to use for selection, indicated by the better performing selection pressures. Furthermore, using a greater weight parameter is still useful, provided that a maximum generation parameter is specified, so fitness becomes more dominant as individuals approach higher fitness values. Using extreme values for a weight parameter, even tempered by small maximum

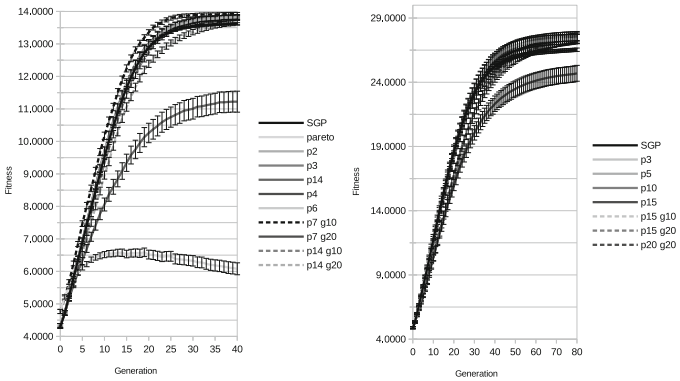


**Fig. 2.** Fitness over generation for varying selection pressures, for the Order Tree 4 (left) and 5 (right) problem. 'p' indicates the evolvability weight, and 'g' indicates the $g_{max}$ value. Error bars indicate the 95 % confidence interval of standard error of the mean. SGP refers to standard genetic programming.

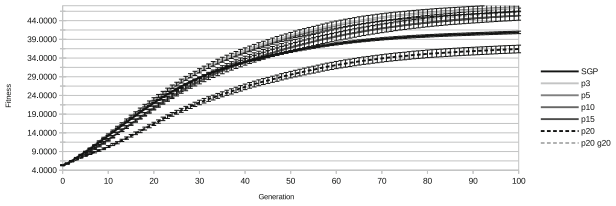


**Fig. 3.** Fitness over generation for varying selection pressures, for the Order Tree 6 problem.

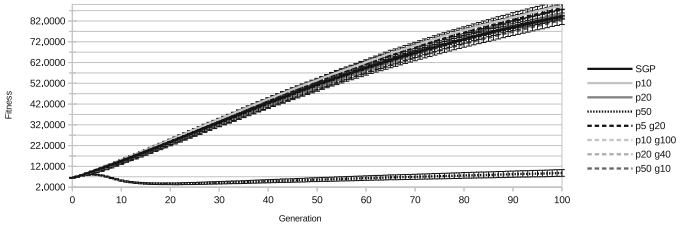


**Fig. 4.** Fitness over generation for varying selection pressures, for the Order Tree 8 problem.

generation parameter, did not produce fit results. For the higher difficulty problems, a greater emphasis on evolvability improves the results. We also note that selection based on pareto-dominance, where fitness and evolvability are the two

**Table 2.** Probability of an incorrect selection comparing 100 samples with 1000 samples under various modified fitness functions over 100 runs.

| Order | p | g | Mean selection error |
|---|---|---|---|
| 4 | 7 | 10 | 0.34495 % |
| 5 | 10 | N/A | 2.6304 % |
| 6 | 5 | N/A | 3.6288 % |
| 7 | 10 | N/A | 4.338 % |
| 8 | 20 | 40 | 0.34230 % |

objectives, produces worse results than standard GP. A subset of all the tested values for varying selection pressures are shown, for clarity.

We see in Table 2 that under the varying selection pressures, that using 100 samples for evolvability differs from using 1000 samples less than 5 % of the time. The tested selection pressures were some of the top performing selection methods for their order of problem, as shown in the previous experiments. Establishing a performance baseline for evolvability selection pressure allows us to proceed to modelling evolvability.

### 4.3 Modelling of Evolvability

Once the effectiveness of using sampled evolvability has been demonstrated and the evolvability selection methods have been evaluated, we must now build a model for evolvability and demonstrate its effectiveness. Firstly, we must describe the attributes we use to build the machine learning models for evolvability. We record a number of attributes associated with individuals. These include generation, tree height, tree size, functional & terminal frequency, number of dormant nodes, dormancy ratio, previous standard fitness, fitness change, and standard fitness. These may all be recorded for each individual without significant computational costs beyond standard fitness calculation. These attributes were subjected to attribute significance testing using WEKA, using the correlation-based filter method Correlation-Based Feature Selection [11], and further tests on WEKA classifiers. The most significant attributes were determined to be generation, size, function frequency, terminal frequency, number of dormant nodes, previous fitness, and fitness.

WEKA offers rapid use of many machine learning classifiers. In order to build a model, we must provide training data and choose a classifier. We can generate training data by running standard GP with the addition of evolvability sampling and selection; this will produce individuals which will be similar to those that will occur when using the model system, ensuring the models will be more accurate in practice. We can evaluate the effectiveness of the different models for evolvability by comparing the mean absolute error between them, also comparing this with the mean absolute error of the evolvability by varying number of samples. Various experiments indicated that a number of machine learning

**Table 3.** Probability of an incorrect selection comparing a multilayer perceptron model constructed with a varied number of training instances (themselves constructed under a varied number of evolvability samples) with 1000 evolability samples under the 4th Order Tree problem using the p7 g10 fitness function over 1000 runs.

| Samples | Training instances | Mean selection error |
|---|---|---|
| 1000 | 2000 | 0.487524 % |
| 1000 | 4000 | 0.488446 % |
| 1000 | 8000 | 0.483173 % |
| 1000 | 40000 | 0.460605 % |

models were appropriate for this task, having similar mean absolute error rates. We select the multilayer perceptron (an artificial neural network) for verifying the effect of the number of training instances and number of evolvability samples are required for acceptable mean absolute error rates. Acceptable mean absolute error rates are those which indicate that erroneous selection will occur less than 5 % of the time. We do this by varying the number of training instances, and the amount of evolvability samples used to generate those instances, and measuring the frequency of selection error compared with 1000 samples of evolvability.

Once the conditions required for acceptable selection error rates have been determined, we test the system by comparing the top performing selective conditions in each order tree problem, compared with standard GP and the improvements made by using sampled evolvability, to indicate that modelling evolvability and modifying the standard fitness function, we can improve GP. This will indicate that modelling evolvability is viable.

We see in Table 3 that relatively few training instances are required to build an accurate model of evolvability. Very few selection errors are made when the evolvability used to train the model is accurate; that is, when a large number of samples of evolvability are taken to generate the model. We see in Fig. 5 that the models perform sufficiently well in practice. They are a statistical improvement over standard GP, and fare about as well as sampled evolvability. Even as few as 1000 training instances can build a successful model. Since a training instance is generated for each individual in the population for each generation, a single run with these settings generates 5000 training instances.

In Figs. 6, 7, 8 and 9 we see that this trend holds in higher Order Tree problems; modelling evolvability offers a statistically significant improvement over standard GP, and performs about as well as using samples to calculate evolvability. Using models built 1000 samples of evolvability even performs better than continually sampling evolvability 100 times for each individual.

In conclusion, we have demonstrated the necessary amount of evolvability sampling to generate a sufficiently accurate calculation of evolvability. We have further demonstrated how evolvability may be used to modify the standard fitness function, in order to encourage the selection of evolvable individuals, and how this may generate an overall increase in standard fitness. We have

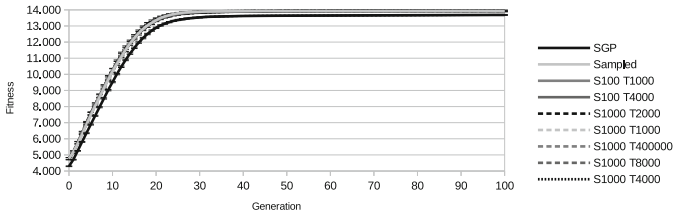


**Fig. 5.** Fitness over generation for ANN models built from various amounts of training instances and various amounts of evolvability samples for the Order Tree 4 problem. 's' indicates the number of evolvability samples, and 'I' indicates the number of training instances.

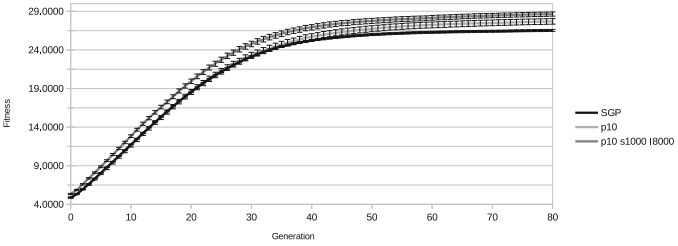


**Fig. 6.** Fitness over generation comparing standard GP, using sampled evolvability & modelled evolvability with a modified fitness function for the Order Tree 5 problem.

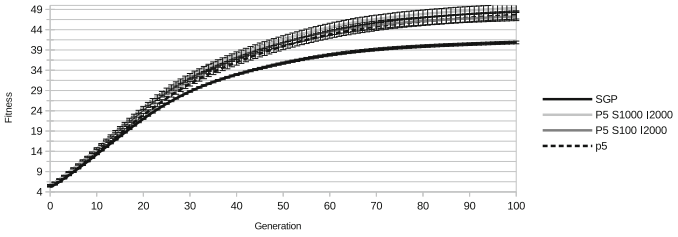


**Fig. 7.** Fitness over generation comparing standard GP, using sampled evolvability & modelled evolvability with a modified fitness function for the Order Tree 6 problem.

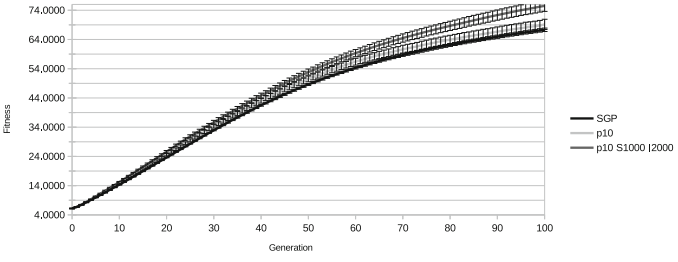


**Fig. 8.** Fitness over generation comparing standard GP, using sampled evolvability & modelled evolvability with a modified fitness function for the Order Tree 7 problem.

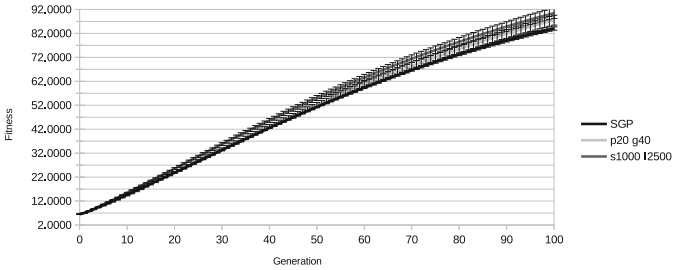


**Fig. 9.** Fitness over generation comparing standard GP, using sampled evolvability & modelled evolvability with a modified fitness function for the Order Tree 8 problem.

demonstrated how many instances and samples are required to build a sufficiently accurate model of evolvability, in order to predict it to guide selection. Finally, we have shown that modelling evolvability and using it in selection allows for a similar improvement in overall fitness than simply sampling for evolvability. The additional number of evaluations required to sample evolvability to guide selection is prohibitive. The extra computational time required to predict evolvability using an external program is prohibitively computationally expensive, as well. Furthermore, its use is limited in this experiment by gathering training instances *a priori*. However, the results demonstrate that predicting evolvability from relatively few training instances with a relatively few number of samples still leads to improved fitness. This indicates the viability of modelling evolvability in order to improve genetic programming. In future work, the system will be modified to generate training instances for evolvability periodically while evolution occurs, in order to build and update models for evolvability periodically, so that more performance gains can be achieved. This work lays a foundation for the success of such a system.

## References

1. Altenberg, L.: The evolution of evolvability in genetic programming. In: Advances in Genetic Programming, pp. 47–74 (1994)
2. Altenberg, L.: Evolvability and robustness in artificial evolving systems: three perturbations. Genet. Program. Evolvable Mach. **15**(3), 275–280 (2014)
3. Banzhaf, W.: Genetic Programming and Emergence. Genet. Program. Evolvable Mach. **15**(1), 63–73 (2013)
4. Bassett, J.K., Coletti, M., De Jong, K.A.: The relationship between evolvability and bloat. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation. GECCO 2009, NY, USA, pp. 1899–1900. ACM, New York (2009)
5. Flatt, T.: The evolutionary genetics of canalization. Q. Rev. Biol. **80**(3), 287–316 (2005)
6. Gagné, C., Parizeau, M.: Genericity in evolutionary computation software tools: principles and case study. Int. J. Artif. Intell. tools **15**(2), 173–194 (2006)
7. Galván-López, E., McDermott, J.: Defining locality as a problem difficulty measure in genetic programming. Genet. Program. Evolvable Mach. **12**(4), 365–401 (2011)

8. Galván-López, E., Poli, R., Kattan, A., ONeill, M., Brabazon, A.: Neutrality in evolutionary algorithms. What do we know? Evolving Syst. **2**(3), 145–163 (2011)
9. Hadka, D., Reed, P.: Borg: an auto-adaptive many-objective evolutionary computing framework. Evolutionary Comput. **21**(2), 231–259 (2013)
10. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. SIGKDD Explor. Newsl. **11**(1), 10–18 (2009)
11. Hall, M.A., Smith, L.A.: Practical feature subset selection for machine learning (1998)
12. Heywood, M.I.: Evolutionary model building under streaming data for classification tasks: opportunities and challenges. Genet. Program. Evolvable Mach. **16**(3), 283–326 (2015)
13. Hoang, T.H., Hoai, N.X., Hien, N.T., McKay, R.I., Essam, D.: ORDERTREE: a new test problem for genetic programming. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation. GECCO 2006, vol. 1, pp. 807–814 (2006)
14. Jackson, D.: The identification and exploitation of dormancy in genetic programming. Genet. Program. Evolvable Mach. **11**(1), 89–121 (2009)
15. Jones, T.: Evolutionary algorithms, fitness landscapes and search. Ph.D. thesis, The University of New Mexico (1995)
16. Kattan, A., Ong, Y.S.: Bayesian inference to sustain evolvability in genetic programming. In: Handa, H., Ishibuchi, H., Ong, Y.S., Tan, K.C. (eds.) Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems. Proceedings in Adaptation, Learning and Optimization, vol. 1, pp. 75–87. Springer, Heidelberg (2015)
17. Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
18. Li, K., Kwong, S., Cao, J., Li, M., Zheng, J., Shen, R.: Achieving balance between proximity and diversity in multi-objective evolutionary algorithm. Inf. Sci. **182**(1), 220–242 (2012)
19. Malan, K.M., Engelbrecht, A.P.: A survey of techniques for characterising fitness landscapes and some possible ways forward. Inf. Sci. **241**, 148–163 (2013)
20. Miller, J.F., Smith, S.L.: Redundancy and computational efficiency in cartesian genetic programming. IEEE Trans. Evol. Comput. **10**(2), 167–174 (2006)
21. Nordin, P., Francone, F., Banzhaf, W.: Explicitly defined introns and destructive crossover in genetic programming. In: Advances in Genetic Programming, pp. 111–134. MIT Press, Cambridge, MA, USA (1996)
22. Öztürkeri, C., Johnson, C.G.: Self-repair ability of evolved self-assembling systems in cellular automata. Genet. Program. Evolvable Mach. **15**(3), 313–341 (2014)
23. Pigliucci, M.: Is evolvability evolvable? Nat. Rev. Genet. **9**(1), 75–82 (2008)
24. Poli, R., Langdon, W., McPhee, N., Koza, J.: A field guide to genetic programming (2008)
25. Silva, S., Dignum, S., Vanneschi, L.: Operator equalisation for bloat free genetic programming and a survey of bloat control methods. Genet. Program. Evolvable Mach. **13**(2), 197–238 (2011)
26. Sindhya, K., Miettinen, K., Deb, K.: A hybrid framework for evolutionary multi-objective optimization. IEEE Trans. Evol. Comput. **17**(4), 495–511 (2013)
27. Smith, T., Husbands, P., Layzell, P., O'Shea, M.: Fitness landscapes and evolvability. Evol. comput. **10**(1), 1–34 (2002)
28. Tarapore, D., Mouret, J.B.: Evolvability signatures of generative encodings: beyond standard performance benchmarks. Inf. Sci. **313**, 43–61 (2015)

29. Wang, Y., Wineberg, M.: Estimation of evolvability genetic algorithm and dynamic environments. Genet. Program. Evolvable Mach. **7**(4), 355–382 (2006)
30. Webb, A.M., Handl, J., Knowles, J.: How much should you select for evolvability?. In: Proceedings of the 2015 European Conference on Artificial Life, pp. 487–494. MIT Press (2015)
31. White, D.R., McDermott, J., Castelli, M., Manzoni, L., Goldman, B.W., Kronberger, G., Jaśkowski, W., O'Reilly, U.M., Luke, S.: Better GP benchmarks: community survey results and proposals. Genet. Program. Evolvable Mach. **14**(1), 3–29 (2013)
32. Wilder, B., Stanley, K.: Reconciling explanations for the evolution of evolvability. Adapt. Behav. **23**(3), 171–179 (2015)