Comparing Methods for Module Identification in Grammatical Evolution

John Mark Swafford
School of Computer Science & Informatics
john-mark.swafford@ucdconnect.ie

Miguel Nicolau
School of Computer Science & Informatics
miguel.nicolau@ucd.ie

Erik Hemberg
School of Computer Science & Informatics
erik.hemberg@ucd.ie

Michael O'Neill
School of Computer Science & Informatics
m.oneill@ucd.ie

Anthony Brabazon
School of Business
anthony.brabazon@ucd.ie

Natural Computing Research & Applications Group
Complex and Adaptive Systems Laboratory
University College Dublin, Ireland

ABSTRACT

Modularity has been an important vein of research in evolutionary algorithms. Past research in evolutionary computation has shown that techniques able to decompose the benchmark problems examined in this work into smaller, more easily solved, sub-problems have an advantage over those which do not. This work describes and analyzes a number of approaches to discover sub-solutions (modules) in grammatical evolution. Data from the experiments carried out show that particular approaches to identifying modules are better suited to certain problem types, at varying levels of difficulty. The results presented here show that some of these approaches are able to significantly outperform standard grammatical evolution and grammatical evolution using automatically defined functions on a subset of the problems tested. The results also point toward a number of possibilities for extending this work to further enhance approaches to modularity.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming

General Terms

Algorithms, Performance

Keywords

Grammatical Evolution, Genetic Programming, Modularity

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

1. INTRODUCTION

Modularity is an important open issue in the field of genetic programming [16] and has been studied in a variety of contexts. These range from examining abstract principles taken from biology [20] to the empirical analysis of the performance of different approaches for enabling and exploiting modularity in evolutionary computation [2, 18]. This research is classified under the latter. As modularity has been shown to be extremely useful for the scalability of evolutionary algorithms (Koza [9] shows this for genetic programming), it is important to understand the effects of different methods of encapsulating and exploiting modules in these stochastic search methods.

The focus of this work is a popular form of grammar-based genetic programming, grammatical evolution (GE) [3, 13, 15]. Studying modularity in the context of GE is especially interesting because of its genotype-to-phenotype mapping. The nature of this process facilitates a simple mechanism for incorporating modules into the population via GE's grammar. For the purposes of this work, modules are encapsulated derivation sub-trees believed to be useful for the population. By adding modules to GE's grammar, they are accessible to the entire population and no special operators are required to make use of them. Over the course of the mapping process, a derivation tree is created, denoting which rules from the grammar are picked to create an individual's phenotype. One guaranteed way to find "good" modules, would be to examine and evaluate each of the derivation sub-trees in an individual. However, the computational overhead required for this is unreasonable, leading to the need for more efficient methods of discovering modules. By evaluating a subset of the sub-derivation trees, it is possible to approximate which ones improve the fitness of the individual. When identified, the portion of the phenotype represented by these derivation sub-trees are encapsulated and made available through GE's grammar.

When attempting to exploit modularity, two primary issues arise:

1. how modules should be identified,
2. and how modules should be used.

In previous work, Swafford et al. [23, 25] present different methods of incorporating modules into GE's grammar once they have been identified. However, they only consider one method of identifying modules, leaving the question of how modules should be identified open for examination. This paper defines multiple methods for identifying modules and reports the positive and negative characteristics of these methods.

The rest of this paper is structured as follows: Section 2 gives a review of related approaches to discovering modules in GAs, GP, and GE. Next, Section 3 defines the methods for module identification used in the experimental sections of this paper. Following that, Section 4 outlines the experimental setup used. Sections 5 and 6 present the results and a discussion of the experiments carried out. Finally, Section 7 gives the conclusions and avenues for future work.

2. PREVIOUS WORK IN MODULE IDENTIFICATION

Some of the earliest work in the area of module identification is that of Angeline and Pollack [1, 2]. They pick parent individuals for candidate modules based on their fitness. Once a parent has been picked, a module is randomly picked from that individual. Angeline and Pollack report positive results from this method of identifying modules, however their simplistic approach leads to the question if there are better, more "informed" methods for module identification. This paper attempts to address this question by implementing and evaluating more "informed" methods for identifying modules.

Rosca and Ballard [19] take a different approach. They only allow an individual to contribute a module if its fitness is better than its least fit parent. Once suitable individuals have been identified, all subtrees of a given depth are examined and the most frequently activated subtrees are used as modules. This has the advantage of taking information from good individuals and also ensuring that subtrees picked as modules are used by individuals. Rosca [17] found that the frequency of appearance of a sub-tree was not an especially valuable method. However, Dessi et al. [4] found that frequency-based module identification methods performed comparably well to the other approaches tested. This leaves the question open as to weather or not using frequency as a module identification technique is beneficial.

Krawiec and Bartosz [10, 11] introduce the idea of functional modularity. This centers around the notion that modules may be identified by examining their semantics. They propose testing modules on a subgoal or subgoals of the fitness function to obtain a value for the modules' semantics. This has the potential to be a powerful approach to modularity in GP, but Krawiec and Bartosz point out the problem of selecting an appropriate subgoal for the potential modules.

Majeed and Ryan [12] describe yet another method for identifying useful modules. They iterate each individual with a fitness better than the population's average fitness at the end of a run and calculate the fitness contribution of each subtree in those individuals. This is done by replacing the subtree with an identity node to cancel the effects of the subtree on the rest of the individual. A module list is created from the best subtrees and is made available to individuals in subsequent runs. They state that approaches which are able to make use of the discovered modules significantly outperform approaches which are not able to use modules. This method is the inspiration for the M-ID approach described in Section 3.

Walker and Miller's [26] approach to modularity in Embedded Cartesian Genetic Programming (ECGP) is similar to Angeline and Pollack's [2, 1]. They randomly identify modules from a linear genotype. Modules are compressed into a single primitive, stored in a module list, and made available in the function set used to create individuals. Modules may also be expanded back to their pre-compressed form. They also add operators to modify the modules' input, output, and contents. Walker and Miller report very positive results from this form of modularity.

The most popular approach to enabling and exploiting modularity in GP is the use of automatically defined functions (ADFs) [9]. ADFs are parameterized functions which are evolvable and reusable subtrees in a GP individual. GP equipped with this form of modularity is shown to outperform standard GP on problems of sufficient difficulty [9]. There also have been previous approaches to enabling ADFs, in GP [8] and GE [7] (as well as the similar Dynamically Defined Functions in GE [5]). Spector et al. [21] use a method similar to ADFs in Push GP [22]. They add functionality to tag particular instructions for later reuse. Tagged instructions may be reused or un-tagged. The tagging and untagging of information is completely up to evolution. Modularity in GE has also been studied by using grammars with different levels of modularity [24]. A more in-depth review of previous work relating to modularity in GP is given in work by Walker and Miller [26] and Hemberg [6].

3. MODULE IDENTIFICATION METHODS

It would be naïve to assume that one method for module identification is suitable for all problems, as this violates the no free lunch theorem. For this reason, a number of different module identification approaches were developed. For the current work, these approaches are as follows:

Mutation Identification (M-ID): An individual is taken from the population and a node on its derivation tree is randomly picked. The derivation sub-tree starting with this node is the candidate module, and the fitness of the individual, f_0 is recorded. This derivation subtree is replaced n times with randomly created subderivation tress of the same size. The value of n represents how many evaluations each candidate module undergoes. For each replacement, the entire individual is re-evaluated and the updated fitness is recorded, $f_{1\ldots n}$. The candidate module is saved, if f_0 is less than $n \times \rho$ of $f_{1\ldots n}$, where ρ is a parameter in the range $(0, 1]$. ρ corresponds to how many evaluations the candidate module must pass. This The fitness of this module is:

$$f_m = \frac{\sum_{i=1}^{n}(f_0 - f_i)}{n}.$$

Insertion Identification (I-ID): First, n test individuals are generated using the same initialization method as

the population. The fitness of each test individual is calculated. These fitness values are saved for later use: $f^b_{1...n}$. A candidate module is randomly picked from an individual and inserted into each of the test individuals to replace a random sub-derivation tree with the same depth as the candidate module. After the candidate module is inserted into a test individual, the fitness is recalculated and saved. The set of fitness values created by evaluating the candidate module in the test individuals is also saved $f^a_{1...n}$. If $n \times \rho$ of the values in $f^a_{1...n}$ show improvement over their corresponding value in $f^b_{1...n}$, the candidate module is saved. The fitness of this module is:

$$f_m = \frac{\sum_{i=1}^{n}(f^b_i - f^a_i)}{n}.$$

Frequency Identification (F-ID): This method counts the occurrence of every derivation sub-tree in the population, except for single non-terminals. The most common derivation sub-trees are used as modules and given fitness values based on their frequency:

$$f_m = \frac{\#\text{ of occurrences}}{\text{total }\#\text{ of subtrees}}.$$

Random Identification (R-ID): This method picks a random derivation sub-tree from each individual in the population and creates a module out of it. Since no evaluation of the module occurs, the fitness of the parent individual is used as the module's fitness:

$$f_m = \text{fitness of parent.}$$

Each of these methods was developed with a particular motive in mind. The M-ID approach samples how the picked derivation sub-tree contributes to the fitness of the individual it appears in. I-ID estimates how well a derivation subtree performs in multiple individuals, as opposed to only the individual it appears in. To address the conflicting results of Rosca [17] and Dessi et al [4], F-ID is included. The R-ID method is added as a control to examine how M-ID, I-ID, and F-ID compare to a random approach. Testing this set of module identification methods will give insight into what types of approaches perform best on certain problems. After a module is identified, the phenotype of that module is "locked," meaning it is not allowed to be modified by crossover or mutation events. Modules are unchangeable because their evaluation suggests that their current phenotypes are useful (except with the R-ID method which performs no evaluation of modules). It may be beneficial to allow modules to be evolved, but this is out of the scope of this work.

Once suitable modules have been identified, they must be made available to the population. This is accomplished by adding the 20 best identified modules to GE's grammar. Swafford et al. [25] show 20 as a reasonable value for the number of modules kept. These modules are picked based on the fitness values given to them at their creation. Figure 1 shows how modules are incorporated into the grammar. When the grammar already contains 20 modules and more have been identified, both the new and old modules are sorted based on their fitness values and the best 20 are kept. If a module is removed from the grammar, and there are still individuals using it, every occurrence of the module in each individual is expanded into the full subtree used to create that module. This prevents the phenotypes of individuals from changing when the grammar is modified, but has the potential to cause large genotypic bloat in individuals using many large modules.

Table 1: Experimental Parameters

Trials	50
Parameter	Value
Population Size	500
Generations	100
Fitness Evaluation Limit	50000
Crossover	Single point (90%)
Mutation	Int-Flip (1%)
Selection	Tournament (1% of pop. size)
Elite Size	50 (10% of pop. size)
Initialization	Ramped Half and Half

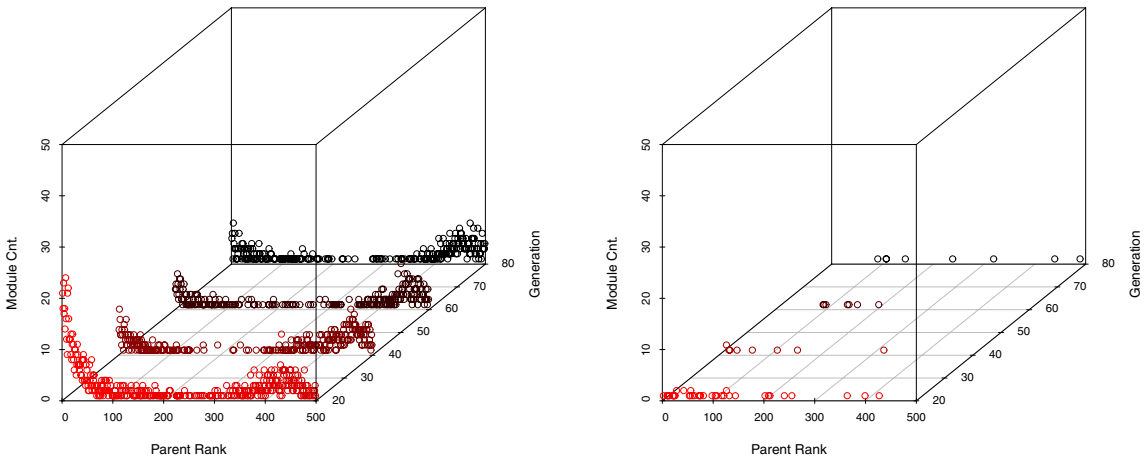
4. EXPERIMENTAL SETUP

The aim of this work is to examine different methods for identifying modules and to establish which of these different methods make them more, or less, beneficial for different problems. These approaches are used on the Santa Fe Ant Trail, $x^5 - 2x^3 + x$ Symbolic Regression, even 7 parity, and 8×8 Lawn mower problems. When evaluating candidate modules for the I-ID and M-ID approaches, 50 is used for the value of n and 0.75 is used for ρ. Duplicate modules are always thrown away. For all problems, modules are identified every 20 generations. This value is borrowed from Swafford et al. [25] who experimented with various module identification intervals. Experimentation with various intervals for identifying modules could be valuable but is out of the scope of this work.

As additional baselines for comparison, the above problems are also attempted with GE with ADFs. This is implemented by including two ADFs in GE's grammar which are defined by codons in individuals' chromosomes and can be used by the result producing branch of the grammar. Each individual in our implementation of GE (GEVA [14]) has its own local copy of the grammar, meaning, each ADF is local to a single individual. Because the different module identification approaches described here may use significantly more fitness evaluations than standard GE and GE with ADFs, the average number of fitness evaluations used by GE is calculated and used as a cutoff point. When any approach uses more fitness evaluations than the cutoff, data from the last full generation using less than or the same number of fitness evaluations as the cutoff is used for the results reported in Section 6. Table 1 lists the parameters used in setting up the following experiments.

5. INITIAL RESULTS WITH MODULE IDENTIFICATION

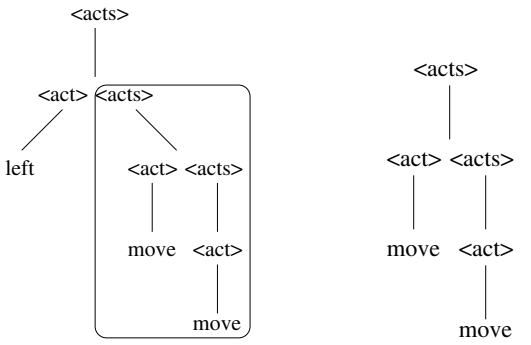
Preliminary experiments compare GE, GE with two ADFs, and the 4 approaches to modularity described in Section 3. The results from these experiments show that there is a discrepancy between how many generations each approach completes before the fitness evaluation limit is reached. Approaches that used no extra fitness evaluations during module identification tended to have better average best fitness values when they reached the fitness evaluation limit. The



(a) Mutation ID

(b) Insertion ID

Figure 2: This figure shows which section of the population contributes the most modules during the identification process. The x-axis represents an individual's rank in the population based on fitness, 0 being the best and 500 being the worst. The y-axis (height) represents how many times individuals contributed a module over the course of 50 runs. This value does not take into account the quality of the module discovered. The z-axis (depth) is the generation at which modules are identified. The problem used in these graphs is the Santa Fe Ant Trail.



(a) A sample individual

(b) A sample module

```
<acts> ::= <act> | <act> <acts>
<act>  ::= move | left | right
```

(c) Original Grammar

```
<acts>          ::= <act> | <act> <acts>
                  | <acts_mod_lib>
<act>           ::= move | left | right
<acts_mod_lib>  ::= <mod_0>
<mod_0>         ::= move move
```

(d) Updated Grammar

Figure 1: These figures show how the grammar is modified when a module is added to it. Figure 1(a) shows a sample individual with a candidate module circled. The removed module can be seen in Figure 1(b). Figure 1(c) shows the original grammar and Figure 1(d) shows how the grammar is modified when a module is added to it.

M-ID and I-ID approaches on average reached generation 40 before hitting fitness evaluation limit, where all other approaches regularly reached generation 100. Even if the M-ID and I-ID methods are finding better modules than the R-ID and F-ID methods, evolution is not given many generations to work with those modules to create better solutions. The source of this problem is that each individual in the population is given the chance to contribute a module. In an attempt to remedy this problem the individuals of the population from which the most modules are identified is given. If these individuals can be identified, focus can be given to them and the rest of the population can be ignored.

To determine which group of individuals focus should be given to, the number of modules that come from different ranked individuals at the module identification steps is plotted. Figure 2 shows a visualization of this. It shows that in the case of M-ID (Figure 2(a)) more modules come from approximately the best 7.5% of individuals in the first module identification step at generation 20. The I-ID approach (Figure 2(b)) finds fewer modules in total, but most modules are still coming from the top of the population at the first module identification step. The graphs in Figure 2 only show results from the Santa Fe Ant Trail problem. Data from the other problems vary slightly, but have a similar trend.

Another interesting trend in Figure 2(a) is the bump in the number of modules coming from the worst individuals in the population. One possible explanation for this may be due to the diversity in the different areas of the population. As evolution progresses, the population tends to be less diverse in terms of the phenotypes of the individuals. As modules are being taken from the top individuals, if they are discovered again in later individuals, they are rejected since duplicate modules are not kept. However, the

tail end of the population is likely to have individuals who are more recently created by crossover and mutation events. The modules discovered here may be different due to the variety of phenotypes in this area of the population.

After considering which individuals yield the most modules across all problems, the following was implemented:

1. Originally, candidate modules needed to "pass" 75% of their evaluations to be considered for use in the grammar. But, the I-ID approach rarely finds modules that passed 75% of the 50 evaluations attempted. One set of variations lowers the percentage of evaluations candidate modules needs to pass to 50% and 25%.

2. One trend that can be seen in Figure 2 is that more modules come from the best individuals in the population, especially at the first module identification step. To save fitness evaluations during module identification, modules are only identified from the top 7.5% of the population.

A third extension is also implemented based on data from the initial experiments. The largest improvements in fitness are made in the first 20 generations. Modules are only identified every 20 generations, meaning they are not available initially for evolution to build upon and take advantage of. The final variation identifies modules from the initial generation, in addition to every 20 generations.

These variations are applied to each of the original setups discussed in Section 3 when possible. The following section analyzes the experimental results of this.

6. EFFECTS OF MODULE IDENTIFICATION ON FITNESS AND SCALABILITY

This section discusses the results of applying the additional variations from Section 5 to the module identification methods. The first set of results to consider is given in Table 2. Due to space restrictions, only GE, GE with ADFs, and the best five approaches to modularity defined in this paper are shown in Tables 2 and 3.

Table 2 shows how at least one approach to modularity is able to significantly outperform GE on all problems and GE with ADFs on all but the Even 7 Parity and Lawn mower problems. It is not surprising that the best approaches for each problem are different. However, one surprising result is that the R-ID and F-ID-based approaches are often some of the best approaches. It was anticipated that modules identified under the M-ID and I-ID approaches would outperform the others due to the fact that they approximate a module's worth based on how much it contributes to an individual's fitness. The most likely explanation for the R-ID approaches performance is that runs using this approach are not cut short for hitting the fitness evaluation limit. Modules are being discovered and used by evolution from the first identification step until the final (100^{th}) generation. Attempting to discover modules with the M-ID and I-ID approaches has the potential to greatly reduce the total generations they are able to complete before reaching the fitness evaluation limit. This means that even if good modules are being discovered, little time is given to evolution to use them. This is where only identifying modules from the best individuals becomes useful. Any of the M-ID and I-ID approaches that are in the best approaches listed in Table 2 only identify modules from the top of the population. In fact, each of the approaches listed in both Tables 2 reach at least generation 80 before reaching the fitness evaluation cutoff.

The next question to ask is how scalable are the various methods presented above? This question in particular comes to mind because methods attempting to exploit modularity have been shown to be particularly useful when increasing problem difficulty [9]. To address the issue of scalability, even 8 and 9 parity, $x^6 - 2x^4 + x^2$ and $x^7 - 2x^5 + x^3$ symbolic regression, and 12×12 and 14×14 Lawn mower problems are tested with each of the previous approaches. The results of these experiments are shown in Table 3.

The first observation to be made about Table 3 is that only the Lawn mower and Even 8 Parity problems have the same number one ranked approach as their predecessor problems in Table 2. This indicates that for all the problems tackled, there is no single best approach, even within problem classes. The exception being the Lawn mower problem, where GE with ADFs dominates every other approached examined here. The most probable reason for this difference in performance is the constantly adaptive nature of ADFs. Since ADFs are built into GE's grammar, they are available to individuals from their initialization and are able to grow and evolve via mutation and crossover operations at every generation. This makes ADFs especially suited to the Lawn mower problem. The ability to cover as much of the lawn as possible is a valuable characteristic. Even if a solution is not particularly efficient, covering a large portion of the lawn is well rewarded. ADFs are able to grow and produce more phenotypic information at every generation. Not only are the ADFs themselves able to produce more information, the individuals are able to use them and more frequently at each generation. The approaches to modularity defined in Section 3 create static modules. Not only are modules unable to grow or change, new modules are collected at specified intervals, not every generation, due to the amount of extra fitness evaluations required to evaluate new modules.

To further this analysis, consider only the sets of symbolic regression and parity problems. In the easier two problems of each class, different variations of the R-ID method are the best approaches and are significantly better than GE. As the symbolic regression problems increased in difficulty, the same approaches to modularity introduced in this paper remained at or near the top of the best performing approaches while GE and GE with ADFs only moved down the list. The parity problems tell a different story. As they increased in difficulty, GE's performance actually improves relative to the other approaches examined, and GE with ADFs remains in the top three approaches. However an interesting trend in these problems is the presence of M-ID approaches in the top ranking methods. This suggests that as the parity problems increase in difficulty, they are able to better take advantage of the modules discovered by the more rigorous selection of modules used by M-ID approaches.

7. CONCLUSION AND FUTURE WORK

This work examines methods for identifying modules in GE and compares them to standard GE and GE with ADFs. The different approaches are tested on four different benchmark problems, three of which include varying levels of difficulty. The results reported in Sections 5 and 6 suggest that the problems tested can benefit from the approaches to mod-

Table 2: This table shows the best variations of all the approaches described in Sections 3 and 5, standard GE, and GE with 2 ADFs. The key to the approach column is as follows: `M-(num)` – M-ID (`num` denotes the percentage of evaluations a candidate module must pass), `I-(num)` – I-ID (`num` is the same as M-ID), `TP` – modules are only taken from the top 7.5% of the population, `G1` – modules are identified after the first generation. The ranks are based on the average best fitness out of 50 trials. If two approaches have the same average best fitness they are ranked based on the number of solutions found. The Last Generation column represents the last full generation before the fitness evaluation limit is reached. The `p-value` columns report the p-value given by a Wilcoxon signed rank test comparing the approach and GE or GE with ADFs (p-values < 0.05 denotes significance and have been underlined and are in **bold**). The best fitness values before the fitness evaluation cutoff of each run are used to calculate the p-value

Rank	Approach	Best Fitness ± Std. Error	Number Solved	Last Generation	p-value (GE)	p-value (GE-ADF)
Santa Fe Ant Trail						
1	F-ID G1	14.50 ± 1.84	12	100	**1.7×10^{-3}**	**4.3×10^{-3}**
2	R-ID G1	15.88 ± 2.03	13	100	**0.02**	**0.03**
3	F-ID	18.70 ± 1.92	10	100	0.20	0.24
4	I-25 TP	19.58 ± 1.92	7	92	0.36	0.34
5	R-ID TP G1	19.64 ± 1.93	7	100	0.33	0.48
15	GE-ADF	21.80 ± 1.54	2	100	0.64	NA
19	GE	22.42 ± 1.62	4	100	NA	0.64
8×8 Lawn mower						
1	GE-ADF	0.28 ± 0.28	42	100	**7.2×10^{-10}**	NA
2	M-75 TP	1.70 ± 0.61	20	85	**7.7×10^{-10}**	**8.1×10^{-4}**
3	M-50 TP	2.31 ± 0.71	20	85	**7.7×10^{-10}**	**3.0×10^{-4}**
4	M-50 TP G1	2.50 ± 0.71	17	81	**7.7×10^{-10}**	**3.6×10^{-5}**
5	M-25 TP	2.55 ± 0.73	21	85	**7.7×10^{-10}**	**2.2×10^{-5}**
6	R-ID TP	2.78 ± 0.75	20	100	**7.7×10^{-10}**	**2.2×10^{-4}**
21	GE	29.71 ± 0.17	0	100	NA	**7.2×10^{-10}**
$x^5 - 2x^3 + x$ Symbolic Regression						
1	R-ID	0.427 ± 0.054	6	100	**0.0085**	**0.01**
2	GE-ADF	0.468 ± 0.030	2	97	0.65	NA
3	F-ID	0.479 ± 0.48	7	100	0.11	0.21
4	I-75 TP	0.493 ± 0.068	5	88	0.08	0.14
5	GE	0.504 ± 0.024	1	100	NA	0.65
6	M-25 TP	0.507 ± 0.075	7	85	0.061	0.13
7	F-ID G1	0.519 ± 0.057	4	100	0.56	0.52
Even 7 Parity						
1	R-ID TP G1	0.82 ± 0.43	46	100	**0.01**	0.28
2	GE-ADF	1.72 ± 0.67	43	99	0.05	NA
3	I-75 TP G1	1.82 ± 0.89	45	86	0.08	1.00
4	I-50 TP G1	1.82 ± 0.89	45	86	0.09	0.93
5	I-25 TP G1	1.82 ± 0.89	45	86	0.10	1.00
6	M-75 TP G1	1.84 ± 0.71	43	80	0.06	0.83
20	GE	4.40 ± 1.26	38	100	NA	0.05

ularity examined in this paper. Unsurprisingly, how much each problem instance benefits is dependent on the nature of the problem and the approach to modularity used.

One constant result across all the problems tested is the best performing approaches to modularity require minimal additional fitness evaluations to find modules. This is likely due to the fact that the fitness evaluations used to identify modules detract from those available to evolve the population. On each of the problems, except the Lawn mower problem, approaches to modularity that significantly outperform GE use no additional fitness evaluations to identify modules. This suggests that if useful modules are discovered by using extra fitness evaluations, evolution still needs sufficient time to use these modules in a beneficial way.

The results from this work also point towards promising opportunities for future work. Considering how the various approaches performed on the different benchmark problems, one avenue with the great potential would be designing and implementing a hybrid/adaptive module identification operator which is able to automatically determine how and when modules should be identified. Criteria for these operators could come from modules usage, population fitness, and/or population diversity. Under the M-ID and I-ID approaches, if a candidate module is evaluated and found to be a "bad" module, the evaluations used have been essentially wasted. Another vein of future work could use the knowledge of which derivation sub-trees were found to be unfit to hinder or eliminate their usage. Modules could also be parameterized or given that ADFs performed reasonably well on some of the problems examined, combining ADFs with the approaches to modularity described in Section 3 would allow for both an adaptive and static means for exploiting modularity. Similarly, experimenting with modules that are parameterized, behaving in a more ADF-like fashion, may

Table 3: This table shows the results of testing the approaches in Sections 3 and 5 on more difficult instances of the problems from Table 2. Also recall the key for the *Approach* column in Table 2.

Rank	Approach	Best Fitness ± Std. Err.	Number Solved	Last Generation	p-value (GE)	p-value (GE-ADF)
12×12 Lawn mower						
1	GE-ADF	15.00 ± 2.54	23	99	$\mathbf{7.8{\times}10^{-10}}$	NA
2	M-75 TP	30.37 ± 1.18	0	84	$\mathbf{7.8{\times}10^{-10}}$	$\mathbf{2.0{\times}10^{-4}}$
3	M-25 TP	32.47 ± 1.10	0	84	$\mathbf{8.3{\times}10^{-10}}$	$\mathbf{1.4{\times}10^{-5}}$
4	M-50 TP	32.65 ± 1.10	0	84	$\mathbf{8.3{\times}10^{-10}}$	$\mathbf{1.4{\times}10^{-5}}$
5	M-75 TP G1	33.46 ± 0.91	1	80	$\mathbf{7.8{\times}10^{-10}}$	$\mathbf{6.1{\times}10^{-6}}$
6	M-25 TP G1	34.95 ± 0.72	0	80	$\mathbf{1.8{\times}10^{-9}}$	$\mathbf{5.0{\times}10^{-6}}$
13	GE	51.12 ± 0.62	0	100	NA	$\mathbf{7.8{\times}10^{-10}}$
14×14 Lawn mower						
1	GE-ADF	33.93 ± 3.41	10	99	$\mathbf{7.8{\times}10^{-10}}$	NA
2	M-75 TP	55.30 ± 1.25	0	83	$\mathbf{3.9{\times}10^{-9}}$	$\mathbf{5.7{\times}10^{-7}}$
3	M-25 TP	56.44 ± 1.07	0	83	$\mathbf{1.8{\times}10^{-9}}$	$\mathbf{1.7{\times}10^{-6}}$
4	M-50 TP	57.96 ± 1.03	0	83	$\mathbf{1.5{\times}10^{-9}}$	$\mathbf{2.5{\times}10^{-8}}$
5	M-75 TP G1	61.10 ± 1.05	0	80	$\mathbf{1.83{\times}10^{-8}}$	$\mathbf{9.81{\times}10^{-9}}$
6	M-50 TP G1	63.25 ± 1.46	0	80	$\mathbf{4.64{\times}10^{-6}}$	$\mathbf{3.9{\times}10^{-9}}$
12	GE	72.48 ± 0.81	0	100	NA	$\mathbf{7.8{\times}10^{-10}}$
$x^6 - 2x^4 + x^2$ Symbolic Regression						
1	F-ID	0.255 ± 0.022	4	100	0.20	0.12
2	R-ID	0.260 ± 0.023	6	100	0.24	0.21
3	M-75 TP	0.274 ± 0.022	5	86	0.69	0.72
4	I-25 TP	0.278 ± 0.020	4	92	0.57	0.65
5	I-75 TP G1	0.287 ± 0.019	2	85	0.56	0.98
9	GE	0.291 ± 0.017	2	100	NA	0.84
10	GE-ADF	0.293 ± 0.016	1	97	0.84	NA
$x^7 - 2x^5 + x^3$ Symbolic Regression						
1	M-25 TP	0.566 ± 0.20	2	86	0.66	0.06
2	R-ID	0.581 ± 0.21	1	100	**0.0066**	$\mathbf{6.4{\times}10^{-4}}$
3	F-ID	0.642 ± 0.20	0	100	0.18	**0.02**
4	R-ID TP	0.676 ± 0.21	1	100	0.21	**0.03**
5	M-50 TP G1	0.690 ± 0.22	0	84	0.78	0.15
10	GE	0.859 ± 0.24	0	100	NA	0.38
19	GE-ADF	1.248 ± 0.29	0	97	0.38	NA
Even 8 Parity						
1	R-ID TP G1	2.24 ± 1.02	45	100	**0.01**	0.12
2	F-ID G1	4.24 ± 1.38	41	100	0.08	0.43
3	GE-ADF	6.80 ± 2.39	41	99	0.27	NA
4	M-ID TP G1	7.56 ± 2.44	39	80	0.32	0.92
5	R-ID G1	7.64 ± 2.71	40	100	0.23	0.78
6	M-50 TP G1	7.76 ± 2.46	38	80	0.38	0.80
15	GE	10.76 ± 2.85	36	100	NA	0.27
Even 9 Parity						
1	R-ID TP	15.32 ± 4.22	37	100	0.24	0.72
2	M-75 TP G1	17.76 ± 4.50	36	80	0.36	0.78
3	GE-ADF	18.40 ± 4.75	36	99	0.40	NA
4	R-ID	20.66 ± 5.31	36	100	0.62	0.80
5	M-25 TP G1	21.12 ± 6.44	38	80	0.54	0.73
6	M-50 TP G1	21.12 ± 6.19	35	80	0.38	0.96
13	GE	25.48 ± 5.81	32	100	NA	0.40

also be beneficial for future work. Another profitable extension of this research could be experimenting with dynamic and incremental learning problems.

8. ACKNOWLEDGMENTS

The authors would like to thank members of the Natural Computing Research and Applications group for their support, comments, and discussions. This work is funded by the University College Dublin School of Computer Science & Informatics and Science Foundation Ireland under Grant No. 08/IN.1/I1868.

9. REFERENCES

[1] P. J. Angeline and J. Pollack. Evolutionary module

acquisition. In D. Fogel and W. Atmar, editors, *Proceedings of the Second Annual Conference on Evolutionary Programming*, pages 154–163, La Jolla, CA, USA, 25-26 Feb. 1993.

[2] P. J. Angeline and J. B. Pollack. The evolutionary induction of subroutines. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pages 236–241, Bloomington, Indiana, USA, 1992. Lawrence Erlbaum.

[3] I. Dempsey, M. O'Neill, and A. Brabazon. *Foundations in Grammatical Evolution for Dynamic Environments*. Springer, 2009.

[4] A. Dessi, A. Giani, and A. Starita. An analysis of automatic subroutine discovery in genetic programming. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 996–1001, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

[5] R. Harper and A. Blair. Dynamically defined functions in grammatical evolution. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 9188–9195, Vancouver, 6-21 July 2006. IEEE Press.

[6] E. Hemberg. *An Exploration of Grammars in Grammatical Evolution*. PhD thesis, University College Dublin, 2010.

[7] E. Hemberg, M. O'Neill, and A. Brabazon. An investigation into automatically defined function representations in grammatical evolution. In R. Matousek and L. Nolle, editors, *15th International Conference on Soft Computing, Mendel'09*, Brno, Czech Republic, 24-26 June 2009.

[8] J. R. Koza. Architecture-altering operations for evolving the architecture of a multi-part program in genetic programming. Technical report, Stanford, CA, USA, 1994.

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

[10] K. Krawiec and B. Wieloch. Analysis of semantic modularity for genetic programming. *Foundations of Computing and Decision Sciences*, 34(4):265–285, 2009.

[11] K. Krawiec and B. Wieloch. Functional modularity for genetic programming. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 995–1002, New York, NY, USA, 2009. ACM.

[12] H. Majeed and C. Ryan. Context-aware mutation: a modular, context aware mutation operator for genetic programming. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1651–1658, New York, NY, USA, 2007. ACM.

[13] R. I. McKay, N. X. Hoai, P. A. Whigham, Y. Shan, and M. O'Neill. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3/4):365–396, Sept. 2010. Tenth Anniversary Issue: Progress in Genetic Programming and Evolvable Machines.

[14] M. O'Neill, E. Hemberg, C. Gilligan, E. Bartley, J. McDermott, and A. Brabazon. GEVA: grammatical evolution in Java. *ACM SIGEVOlution*, 3(2):17–22, 2008.

[15] M. O'Neill and C. Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, 2003.

[16] M. O'Neill, L. Vanneschi, S. Gustafson, and W. Banzhaf. Open issues in genetic programming. *Genetic Programming and Evolvable Machines*, 11:339–363, 2010. 10.1007/s10710-010-9113-2.

[17] J. Rosca. *Hierarchical Learning with Procedural Abstraction Mechanisms*. PhD thesis, University of Rochester, 1997.

[18] J. Rosca and D. Ballard. Learning by adapting representations in genetic programming. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 407–412 vol.1, Jun 1994.

[19] J. P. Rosca and D. H. Ballard. *Discovery of subroutines in genetic programming*, pages 177–201. MIT Press, Cambridge, MA, USA, 1996.

[20] H. A. Simon. *The sciences of the artificial*. MIT Press, 3 edition, 1996.

[21] L. Spector, B. Martin, K. Harrington, and T. Helmuth. Tag-based modules in genetic programming. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 1419–1426, New York, NY, USA, 2011. ACM.

[22] L. Spector and A. Robinson. Genetic programming and autoconstructive evolution with the push programming language. *Genetic Programming and Evolvable Machines*, 3:7–40, March 2002.

[23] J. M. Swafford, E. Hemberg, M. O'Neill, M. Nicolau, and A. Brabazon. A non-destructive grammar modification approach to modularity in grammatical evolution. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 1411–1418, New York, NY, USA, 2011. ACM.

[24] J. M. Swafford and M. O'Neill. An examination on the modularity of grammars in grammatical evolutionary design. In *IEEE Congress on Evolutionary Computation*. IEEE, Jul 2010.

[25] J. M. Swafford, M. O'Neill, M. Nicolau, and A. Brabazon. Exploring grammatical modification with modules in grammatical evolution. In S. Silva, J. A. Foster, M. Nicolau, P. Machado, and M. Giacobini, editors, *EuroGP*, volume 6621 of *Lecture Notes in Computer Science*, pages 310–321. Springer, 2011.

[26] J. Walker and J. Miller. The automatic acquisition, evolution and reuse of modules in cartesian genetic programming. *Evolutionary Computation, IEEE Transactions on*, 12(4):397 –417, August 2008.