Analyzing Module Usage in Grammatical Evolution

John Mark Swafford, Erik Hemberg, Michael O'Neill, and Anthony Brabazon

Natural Computing Research & Applications Group
Complex and Adaptive Systems Laboratory
University College Dublin, Ireland
john-mark.swafford@ucdconnect.ie
{erik.hemberg,m.oneill,anthony.brabazon}@ucd.ie

Abstract. Being able to exploit modularity in genetic programming (GP) is an open issue and a promising vein of research. Previous work has identified a variety of methods of finding and using modules, but little is reported on how the modules are being used in order to yield the observed performance gains. In this work, multiple methods for identifying modules are applied to some common, dynamic benchmark problems. Results show there is little difference in the performance of the approaches. However, trends in how modules are used and how "good" individuals use these modules are seen. These trends indicate that discovered modules can be used frequently and by good individuals. Further examination of the modules uncovers that useful as well as unhelpful modules are discovered and used frequently. The results suggest directions for future work in improving module manipulation via crossover and mutation and module usage in the population.

1 Introduction

A recent survey by O'Neill et al. [13], cites modularity as an important open topic in genetic programming (GP) [7]. Some of the earliest work in this area shows how GP representations which enable and/or exploit some form of modularity may outperform and scale better than standard GP on certain benchmark problems [8]. Since the early 1990s, numerous methods have been implemented with varying levels of success. While these methods can be valuable and yield significant performance improvements over standard GP, little has been reported on how the discovered modules are used and contribute to the population's fitness.

Studies of modularity in dynamic environments (also an open issue [13]) are also sparse. Dynamic environments provide a testbed that more resembles real-world problems which are rarely static, like most problems GP researchers tackle. Recent work by Kashtan et al. [6] and O'Neill et al. [11] shows dynamic environments which vary the fitness function over time can even speed up evolution. While dynamic environments is a large topic in GP, no work has been published, to the author's knowledge, examining how incorporating modularity changes GP's performance in dynamic environments.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 347–356, 2012.

The experiments carried out for this work use the popular grammar-based form of GP, grammatical evolution (GE) [12]. To study how enabling and exploiting different forms of modularity impact GE's search in dynamic environments, methods for identifying and making modules available to the population described by Swafford et al. [16] are used. For this work, modules are defined as encapsulated sub-derivation trees taken from the full derivation trees of GE individuals.

In the rest of this work, the effects of modules on GE's search will be presented. First, Sect. 2 describes some of the relevant previous work in dynamic environments and identifying and using modules. Next, Sect. 3 explains how modules are identified and made available for use by the population. Section 4 outlines the various experimental setups used. Following the explanation of the experimental design, Sect. 5 details the results of this study and discusses their meaning. Finally, Sect. 6 draws together some conclusions and proposes ideas for more future work.

2 Previous Work

To this day, there have been numerous approaches for identifying and using modules. Some of the best known of these are Koza's automatically defined functions [8], Angeline and Pollack's Genetic Library Builder [1], Rosca and Ballard's Adaptive Representation [14], and Walker and Miller's Embedded Cartesian GP [17]. Each of these are valuable for defining how modules may be identified and the benefits they give to the evolving population. However, little is said about the modules are actually used by the population. Harper and Blair [3] touch on this issue on their work with Dynamically Defined Functions (DDFs). They give an example how individuals which do not use any DDFs are quickly weeded out of the population and replaced with individuals using one or two DDFs. While this is useful, a more in-depth examination of how modules (in this case DDFs) are used could provide insight into how they could be better exploited to maximize the performance gain they provide. Miller et al. [9] also spend some time examining the frequency of use of certain sub-programs in their study of evolutionary design of digital circuits.

Little work has gone into understanding how modularity enhances or inhibits evolutionary search in dynamic environments. Some of the earliest approaches to modularity are tested on a dynamic problem (the Pac-man game) [8,14] and outperform standard GP on this problem. But no attention is given to how the addition of modules actually encourage the discovery of better solutions. More related work by Kashtan et al. [4,5,6] examine how evolution in dynamic environments can lead to more modular solutions and speed up evolution under certain conditions. They provide a useful analysis of their findings, but do not incorporate any mechanism for identifying and promoting the use of modules. For a more comprehensive survey of work in dynamic environments, see Dempsey et al. [2].

3 Module Identification Methods

Numerous methods for identifying modules and making them available to individuals during evolution have been developed by previous researchers. The experiments presented here use the methods described by Swafford et al. [15,16]. For the proceeding methods for discovering modules, a module is defined as encapsulated sub-derivation trees taken from the full derivation trees of GE individuals. These approaches for finding modules are briefly summarized as follows:

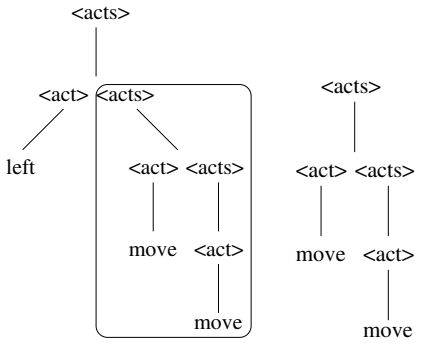
Mutation Identification (M-ID): An individual is taken from the population and a node on its derivation tree is randomly picked. This is the candidate module. It is replaced 50 times with randomly created sub-derivation trees of the same size. For each replacement, the entire individual is re-evaluated and the updated fitness is recorded. For each random sub-derivation tree inserted into the individual, the difference between the individual's original fitness and updated fitness is saved. If the original fitness is better than 75% of the updated fitness values, the candidate module is saved and the mean of the fitness differences is used as the module's fitness.

Insertion Identification (I-ID): First, 50 test individuals are generated using the same initialization method as the population. Next, the fitness of each is calculated. A candidate module is randomly picked from an individual and is inserted into each of the test individuals to replace a random sub-derivation tree with the same depth. Then, the test individuals are re-evaluated. When the candidate module is inserted into each of the test individuals, the difference between the original and updated fitness is saved. If the candidate module improves the fitness of 75% of the test individuals, it is saved and the mean of the fitness differences is used as the module's fitness.

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

Random Identification (R-ID): A random sub-derivation tree is picked from each individual in the population and a module is created out of it. Because the module is not evaluated, the parent individual's fitness is used as the module's fitness.

For the following experiments modules are only selected from the top 15% of individuals. Once the modules have been identified a mechanism for making them available to the population is needed. In GE, adding them to the grammar used to create individuals in the population is a simple and effective method to resolve this issue. As with the methods for identifying modules, the manner in which modules are inserted into and removed from the grammar is also borrowed from Swafford et al. [16]. This is summarized in Fig. 1. The number of modules allowed in the grammar at one time has been limited to 20 based on results reported by Swafford et al. [15]. If more than 20 modules have been identified,



(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

Fig. 1. These figures show how the grammar is modified when a module is added to it

they are ranked based on their fitness values assigned upon their creation and the best 20 of these are kept. If a module is removed from the grammar, and individuals still use it, every occurrence of the module is expanded into the full sub-tree used to create that module. This prevents the phenotypes of individuals from changing when the grammar is modified.

4 Experimental Setup

This work examines the hypothesis that modules can be useful in dynamic environments as they may be able to find and encapsulate sub-solutions that are useful across multiple fitness scenarios. In order to do this, an easy, medium, and hard instance of each of the following common benchmark problems are used: Symbolic Regression (x^5-2x^3+x, $x^6-2x^4+x^2$, $x^7-2x^5+x^3$), Even Parity (7, 8, 9), and Lawn Mower (8×8, 12×12, 14×14). Parameters for specifying how often and the manner in which the fitness function changes are borrowed from Murphy et al. [10]. The number of generations between fitness function changes are 5 and 20. Each of these period lengths was used with random and cyclic changes. Modules identification also occurs every 5 and 20 generations, meaning modules are identified at the beginning of each fitness period. However, this does not allow time for evolution to adjust to the new fitness function before modules are selected from the population. In response to this, staggered module identification steps are also used. Instead of searching for modules at the beginning of each fitness period, the initial module identification starts 2 generations into the first fitness period and then continues every 5. A similar staggered approach starting at generation 10 and continuing every 20 generations is also employed. More frequent and random changes in the fitness function mean GE has little time to adjust to the new environment. Longer fitness periods allow GE more time to adjust to the new target. This variety of fitness and module identification steps allows for testing how well the discovered modules helps GE recover from changes in the target functions.

5 Results and Discussion

This section answers the question of how modules alter (or not) the behavior of GE in dynamic environments. First, modules' frequency, lifetimes, and the fitness of individuals using them are examined. Hand picked modules are also shown for the purpose of understanding what kinds of modules are being discovered and used. Then, a short explanation of how the various methods for identifying modules helps or hinders GE's search capabilities in the dynamic environments.

5.1 Frequency and Lifetime of Modules in the Population

To begin the analysis of the various modular approaches, the frequency of module usage in the population is shown in a heatmap in Fig. 2. This reveals trends in how many individuals are using modules and how long modules are used. This figure shows the results of a Symbolic Regression instance where the fitness function target changes randomly every 20 generations and modules are identified every 20 generations starting at generation 10. Modules are identified using the M-ID approach. Out of the 1519 modules discovered across all 50 runs of this variation, only 442 modules are present in the heatmap. These modules appear in at least 50% of the population. Out of all the modules in Fig. 2 only a small portion of the modules identified in early generations are heavily present throughout multiple module identification and replacements. This suggests these modules contain information that is useful in all of the fitness instances of this particular problem. Through crossover, mutation, and selection operations, some of these modules are able to move from being used by less then 10% of the population to over 90%. Figure 2 also shows that the majority of modules are identified, used frequently, and then are used rarely, if at all, after one or two fitness periods. It also shows a scattered few modules are being used by a large portion of the population for only 5–25 generations before their usage plummets. This indicates that being used by a large percentage of the population does not ensure longevity across many generations.

Only examining the frequency and longevity of use of modules does not paint a full picture of how they are used. To further understand this, Fig. 3 shows the average fitness of every individual each module appears in. The modules in Fig. 3 and Fig. 2 are the same. An interesting characteristic is the similarity between Fig. 2 and Fig. 3. This similarity shows a strong correlation between modules getting used frequently and modules being used by highly fit individuals. A comparison of Figs. 2 and 3 also shows that better individuals are often the only individuals that use modules for long periods of time before and after they are used more widely in the population. Another notable trait of Fig. 3 is the darker colored cells immediately after the fitness period changes. This signifies a drop in the population's fitness as they are adjusting to a new target.

However, Figs. 2 and 3 only show results for a single variation on one problem. Changing the length of the fitness periods and frequency of identifying modules yields no significant changes in the correlations noted above. Using the R-ID and F-ID module identification methods also show similar correlations between

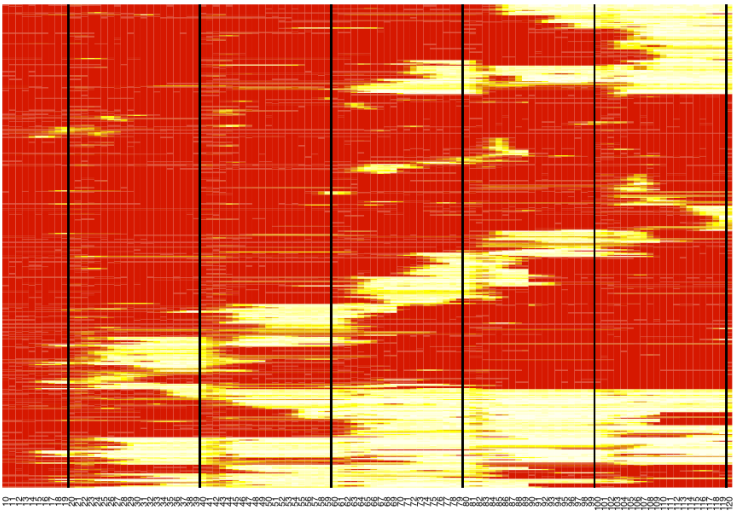


Fig. 2. This figure show module usage across all runs on the Symbolic Regression problem using the M-ID approach, identifying modules every 20 generations starting from generation 10 with the fitness function changing randomly every 20 generations. Each row (the y axis) represents a module that is used by at least 50% of the population at any point in a run and each column (the x axis) corresponds to a single generation. The cells represent the percentage of the population that uses the module in each generation. The black vertical lines indicate generations when the fitness function changes. Dark red colored cells indicate that modules are not being used by any individuals. White colors denote that a module is used by a large percentage or all of the population. The modules have been clustered together with modules that were used similarly to make the graph easier to read. Generations 0–9 have been omitted because module identification has not yet occurred in those generations.

the frequency of module usage and fitness of individuals using those modules. The exception to this is the insertion (I-ID) method for finding modules. It finds very few modules in general. Many runs using the I-ID method find very few modules, and even fewer of these are used largely by the population. The small number of modules found by I-ID is due to the fact that it requires modules to be beneficial in multiple individuals, not only one. In many instances, no modules are used by more than 50% of the population. This suggests that this approach is inappropriate for the instances of the Symbolic Regression problem examined.

Analyzing the Even Parity problem in the same manner, Figs. 4(a) and 4(b) show similar behavior to the Symbolic Regression problem. Modules that are used more frequently tend to also be used in individuals with better fitness. But there are also modules being used in good individuals and few other individuals in the population. Another similarity is that the I-ID approach finds remarkably fewer modules than M-ID, R-ID, and F-ID. In the case of the Lawn Mower problem, Figs. 4(c) and 4(d) tell a different story. Any correlation between how frequently the modules are used and the fitness of individuals using them appears to be absent. Figure 4(c) shows modules being used by both large and small

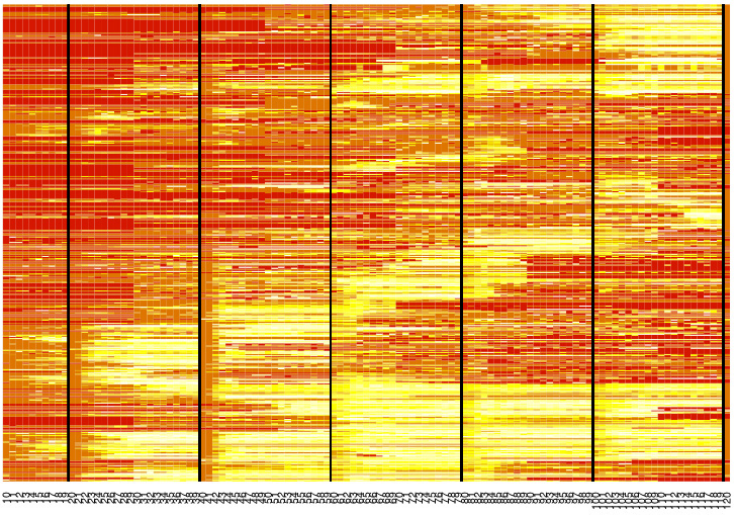


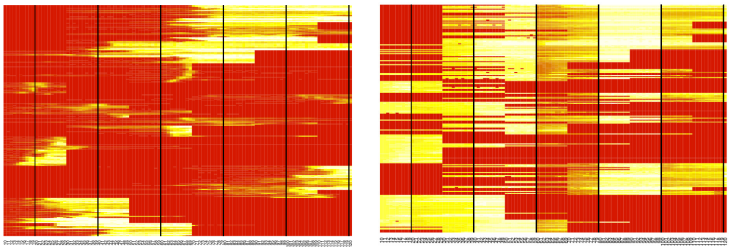
Fig. 3. This figure shows the average fitness of individuals that use a given module. The module identification and fitness period parameters are used in this figure are the same those in Fig. 2. Fitness values have all been normalized between 0 and 1. White and pale yellow colored cells represent modules being used by individuals with good fitness values. Orange cells mean modules get used in individuals with worse fitness. Red cells indicate that modules are not used at all at that generation.

percentages of the population. But the same modules in Fig. 4 are constantly used by the best individuals in the population, regardless of how frequently they are used. The most likely reason for this is the nature of the Lawn Mower problem itself. As modules are able to encapsulate multiple mowing instructions into a single production, individuals using modules can more easily cover larger areas of the lawn than individuals that must combine single terminal symbols to cover the same area. Taking this under consideration, it is easy to understand why modules are frequently being used by the best individuals.

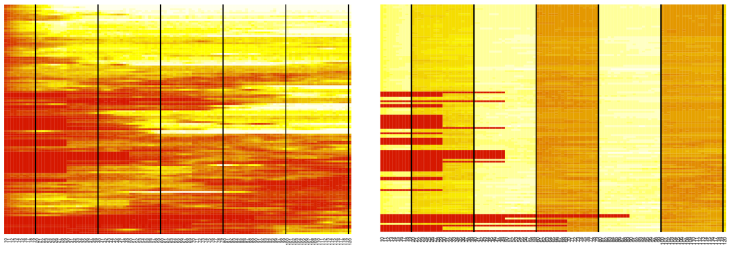
Examining how modules are used in this way naturally leads to two questions:

1. What kinds of modules are being discovered?
2. Do the modules being discovered and used improve GE's fitness?

To answer the first question, a small selection of modules from the Symbolic Regression runs shown above can be examined. The following modules are described in detail because their clear utility and their longevity of use. The first two modules to be examined are `*--xx+11/*1x/11` and `*+11x`, which simplify to $-2x$ and $2x$ respectively. These two modules can be discussed together as their stories are similar and it is easy to see how they could potentially be used as building blocks for the target solutions. Both of these modules were discovered at generation 10. Neither of these modules are immediately adopted by large amounts of individuals. For many generations, they fluctuate in usage percentage from at 0.008% to 70% of the population. By generation 28, both modules



(a) Parity - Usage (b) Parity - Fitness



(c) Lawn Mower - Usage (d) Lawn Mower - Fitness

Fig. 4. This figure shows how frequently modules are used and the average fitness of individuals modules are used in for the Even Parity and Lawn Mower problems. These problem instances use the same parameters as Figs. 2 and 3

are being used by at least 90% of the population. They also experience lulls in usage at different generations where they are used by as little as 5% or 10% of individuals before becoming more prominent in the population again. A third module, `*xx`, or x^2, was also discovered and used in a similar manner. The difference in this particular module is that it never experiences the drop in usage the others do. Once more than 90% of the population uses this module, never again do less than 94% of individuals use it.

When examining the modules used by large portions of the population, another notable trend was seen. Many popular modules simply reduced to 0. Another observation about the modules discovered is that some of the frequently used ones have no apparent usefulness. A possibility for this is that they do not damage or change the fitness of individuals. More investigation is needed to give a definitive reason why these modules are used frequently. A possible cause of this is that it is very difficult to find good modules and the module identification methods sometimes find bad modules.

5.2 Modularity and Fitness

Different approaches to modularity lead to differences in GE's performance. These approaches are compared using the metrics presented by Murphy et al. [10] are used (*draw down*, *area under the curve*, and *fall off*). The methods for identifying modules from Sect. 3 are compared to standard GE and GE with ADFs. The observed data shows that among the methods examined, there is no single

best approach on any of the problems except the Lawn Mower, where GE with ADFs is the best performing approach. As seen in Sect. 5.1, modules are being identified and often used. This suggests that even though modules are being used frequently, they are being used sub-optimally or are not good modules.

6 Conclusion and Future Work

This work examined the effects of incorporating four approaches to modularity on three common benchmark problems in GP: Symbolic Regression, Lawn Mower, and Even Parity. Each of these problems was studied as a dynamic problem, where targets of increasing levels of difficulty were used. The results observed show no large difference between any of the approaches in terms of increasing performance over standard GE. However, the Symbolic Regression and Even Parity instances did show interesting trends in how often modules were used by the population and the fitness of individuals using those modules. The data suggests that modules used by a large percentage of the population also tend to be used by individuals with high fitness. Further examining a selection of the modules being used frequently also shows how potentially helpful modules are being found, used frequently, and used by highly fit individuals. On the other hand, a number of useless modules are being used in the same way. These results point towards a number of possibilities for future work.

One potentially extension from this work would be ensuring that modules are being used in contexts where they can be the most useful. Two of the approaches for identifying modules estimate how well sub-derivation trees perform in particular contexts. If those sub-derivation trees become modules and are used in other contexts, they may be harmful instead of helpful. When estimating the worth of sub-derivation trees, many are passed over as they are deemed unworthy of becoming a module. These may even be harmful sub-derivation trees. Another interesting vein of research could be considering these "bad" sub-derivation trees as anti or taboo modules. It may be beneficial to guide search away from these structures as they are not considered to contain helpful information.

Acknowledgments. 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.

References

1. Angeline, P.J., Pollack, J.B.: The evolutionary induction of subroutines. In: Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society, pp. 236–241. Lawrence Erlbaum, Bloomington (1992)
2. Dempsey, I., O'Neill, M., Brabazon, A.: Foundations in Grammatical Evolution for Dynamic Environments. Springer (2009)
3. Harper, R., Blair, A.: Dynamically defined functions in grammatical evolution. In: Proceedings of the 2006 IEEE Congress on Evolutionary Computation, pp. 9188–9195. IEEE Press, Vancouver (2006)

4. Kashtan, N., Parter, M., Dekel, E., Mayo, A.E., Alon, U.: Extinctions in heterogeneous environments and the evolution of modularity. Evolution 63, 1964–1975 (2009)
5. Kashtan, N., Mayo, A.E., Kalisky, T., Alon, U.: An analytically solvable model for rapid evolution of modular structure. PLoS Comput. Biol. 5(4), e1000355 (2009)
6. Kashtan, N., Noor, E., Alon, U.: Varying environments can speed up evolution. Proceedings of the National Academy of Sciences 104(34), 13711–13716 (2007)
7. Koza, J.R.: Genetic Programming: on the Programming of Computers by Means of Natural Selection. MIT Press (1992)
8. Koza, J.R.: Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, Cambridge (1994)
9. Miller, J.F., Job, D., Vassilev, V.K.: Principles in the evolutionary design of digital circuitspart ii. Genetic Programming and Evolvable Machines 1, 259–288 (2000), doi:10.1023/A:1010066330916
10. Murphy, E., O'Neill, M., Brabazon, A.: A comparison of ge and tage in dynamic environments. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011, pp. 1387–1394. ACM, New York (2011)
11. O'Neill, M., Nicolau, M., Brabazon, A.: Dynamic environments can speed up evolution with genetic programming. In: Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO 2011, pp. 191–192. ACM, New York (2011)
12. O'Neill, M., Ryan, C.: Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Kluwer Academic Publishers (2003)
13. O'Neill, M., Vanneschi, L., Gustafson, S., Banzhaf, W.: Open issues in genetic programming. Genetic Programming and Evolvable Machines 11, 339–363 (2010), doi:10.1007/s10710-010-9113-2
14. Rosca, J.P., Ballard, D.H.: Discovery of subroutines in genetic programming, pp. 177–201. MIT Press, Cambridge (1996)
15. Swafford, J.M., Hemberg, E., O'Neill, M., Nicolau, M., Brabazon, A.: A non-destructive grammar modification approach to modularity in grammatical evolution. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011, pp. 1411–1418. ACM, Dublin (2011)
16. Swafford, J.M., Nicolau, M., Hemberg, E., O'Neill, M., Brabazon, A.: Comparing methods for module identification in grammatical evolution. In: Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO 2012. ACM, Philadelphia (2012)
17. Walker, J., Miller, J.: The automatic acquisition, evolution and reuse of modules in cartesian genetic programming. IEEE Transactions on Evolutionary Computation 12(4), 397–417 (2008)