Zipper-based Meta-Genetic Programming

Kyle I. Harrington
Jordan B. Pollack

Brandeis University, Waltham, MA 02453
kyleh@cs.brandeis.edu

August 16, 2011

Abstract

We present a zipper-based instruction set for constructing genetic programming variation operators. We study the effects of such variation operators on the lawnmower problem. Operators in this language possess the ability to outperform standard mutation and crossover in terms of both decreasing the average population error as well as the best population error. Furthermore, the expression of standard mutation and crossover in this language is trivial, allowing evolution to re-discover these operators when necessary. We conclude by using these operators in a meta-genetic programming context, showing that it is possible for zipper-based meta-genetic programming to expedite evolutionary search.

1 Introduction

The first instance of the meta-evolution of programs was presented in (Schmidhuber, 1987). However, the algorithm is presented as a thought experiment as opposed to a usable tool. The suggestion of a pure meta-evolutionary system composed of a hierarchy of meta-populations also served to dissuade the algorithm in practice; as Schmidhuber notes,one should expect that as the order of meta-evolution increases, the rate of adaptation of the highest order meta-operator should slow. This cost may be one of the reasons that meta-evolutionary techniques have received minimal attention.

Since this initial proposal of meta-evolution, few researchers have chosen to pursue the study of meta-evolutionary techniques; of those, a smaller fraction have demonstrated improvements over classical techniques. The majority of these improvements are outside of the field of genetic programming (i.e. evolutionary programming (EP) (Fogel et al., 1991), genetic algorithms (GA) (Grefenstette, 1986; Tuson, 1995), etc.). A complete review of these meta-evolutionary techniques is out of the scope of this work, we will focus on a few meta-evolutionary techniques that has been applied to tree-based GP. For

the interested reader we note some meta-evolutionary approaches that operate on alternative representations (Teller, 1996; Kantschik et al., 1999; Baum and Durdanovic, 1998; MacCallum, 2003; ONeill and Brabazon, 2005).

Some of the initial promising results of adaptive evolution within GP were heavily inspired by EP. In brief, meta-evolutionary programming associates perturbation values with components. These perturbation values are used when varying components. This idea of associating perturbation values with components was incorporated into GP by associating probabilities of mutation and crossover with nodes (Angeline, 1995). This form of self-adaptation allows programs to adapt the probability of variation at specific points within the program tree. One of the appealing features of this algorithm is that the parameters of variation are unique to individuals, a property we will see is not always present in meta-evolutionary algorithms.

The first application of meta-GP to a standard GP population was presented by Edmonds (2001). The meta-population is composed of tree manipulating operators. Both the solution and meta populations are subject to the general framework of evolutionary algorithms, where solutions receive fitness based on their ability to solve problems while variation operators receive fitness based upon their ability to vary solutions. Although a number of nuances are required in order to improve the meta-GP algorithm, the ultimate version of the algorithm is shown to perform comparably to traditional GP. In closing, Edmonds discusses methods for evolving the population of variation operators. The most appealing of these suggestions is the recursive application of variation operators through a bucket-brigade algorithm (Holland, 1985). However, the bucket-brigade algorithm is another form of the the meta-meta-... problem presented in the original meta-evolution algorithm. Though Edmonds demonstrated that meta-evolution can work even with one meta-level, the technique does not allow for individualistic self-adaptations like those presented in (Angeline, 1995).

A novel alternative to the previously mentioned meta-evolutionary algorithms is autoconstruction (Spector and Robinson, 2002). In autoconstruction, individuals are responsible for their own reproduction and variation. The meta-meta-... problem is remedied by intermingling the code of variation with problem specific code. Although the appeal of variation operators customized to each individual is clear, autoconstruction has yet to be demonstrated to be comparable to traditional GP. Furthermore in recent work revisiting autoconstruction the size of the instruction set has led to a combinatoric explosion (Spector, 2010). The vast majority of instructions considered in this revisitation were included to assist in autoconstructive mechanisms. We pose the question of whether this design decision sacrifices the evolvability of the system.

Before describing our study, we highlight a number of theoretical points about variation and evolvability of importance to meta-GP. Fitness functions are mappings of individual programs onto real numbers. This mapping is often many-to-one, meaning that many individuals may be seen as "equivalent" with respect to the fitness function. However, as is noted in (Altenberg, 1994) the evolvability of these "equivalent" individuals may be quite different. The idea of evolvability is of crucial importance in a discussion of variation.

Instruction	Function
ZIP.UP	Move Z_1 up 1 node if there will be a left neighbor
ZIP.DOWN	Move Z_1 down 1 node
ZIP.LEFT	Move Z_1 left 1 node if there will be a left neighbor
ZIP.RIGHT	Move Z_1 right 1 node
ZIP1.ROOT	Move Z_1 to the root
ZIP2.ROOT	Move Z_2 to the root
ZIP.RAND	Replace the subtree at Z_1 with a random subtree
ZIP.RLOC	Move Z_1 to a random location below it
ZIP.RRLOC	Move Z_1 to a random location anywhere in the tree
ZIP.SWAP	Rename Z_1 to Z_2 and vice versa
ZIP.SWAP.ST	Replace the subtree at Z_1 with the subtree at Z_2 and vice versa
ZIP1.REP.ST	Replace the subtree at Z_1 with the subtree at Z_2
ZIP2.REP.ST	Replace the subtree at Z_2 with the subtree at Z_1

Table 1: Instruction set for variation operators.

While the simple study we present does not account for properties such as constructional selection, the transmission function (Altenberg and Feldman, 1987) is central to any study of variation. The transmission function is the function which maps sets of individuals to new individuals; that is, the transmission function describes the distribution of children that a variation operator can construct from a set of parents. Within GP, the mathematical understanding of the transmission functions for a given variation operator with respect to a population draws upon ideas from GP schema theory (Poli, 2001). Thus far, GP schema theory is limited to a subset of possible variation operators, and each operator generally requires a separate mathematical formulation. As an alternative to mathematically extending GP schema theory, we present empirical results for the effect of variation operators on a real GP population.

We design a set of variation operators that allow for the concise representation of a wide range of variation operations. We continue to implement a meta-GP system in order to evaluate the performance of these operators in the wild. Our variation operators are expressed in a simple instruction set that not only draws on observations of optimality in (Edmonds, 2001), but are also designed such that they may be introduced into alternative meta-evolutionary frameworks such as autoconstruction and meta-GP.

2 Evolutionary System

Our framework contains two instruction sets: one for variation and the other for specific problems. Both types of instructions are based on LISP-style S-expressions, while the framework is written in the Lisp-like language, Clojure (Hickey, 2008). The problem-specific instructions are presented later. We now

introduce the instruction set for variation operators.

As the representation of our problems are traditional tree based genetic programs our variation operators necessitate the ability to manipulate trees. We choose to use zippers (Huet and France, 1997). A zipper is a simple idea drawn from functional programming which represents a location within a tree. Lists of neighboring, parent, and child subtrees relative to the current location are maintained in a single data structure, along with the tree itself. Much of the impetus for selecting this representation for variation instructions is drawn from the observations of (Edmonds, 2001) about the performance of certain instructions in meta-GP, such as tree traversal and the utility of root-access functions. In addition to the traditional zipper functions we add GP-type instructions. These include random movements within trees and subtrees, replacement of random subtrees, and the swapping of subtrees.

Recombination is a highly desirable feature within GP. We design all variation instructions to take two inputs and produce two outputs. Unary operators simply ignore the second argument, passing it unchanged. This uniformity allows us to easily compose instructions. We refer to the mathematical form of composition (i.e. $f(g)$, f is composed with g). The result of composing a sequence of variation operators is a single function that, when given two program trees, produces two program trees. When applying a zipper function to two trees, Z_1 and Z_2, we can write a function

$$\Upsilon(Z_1, Z_2) = (Z_1', Z_2') \tag{1}$$

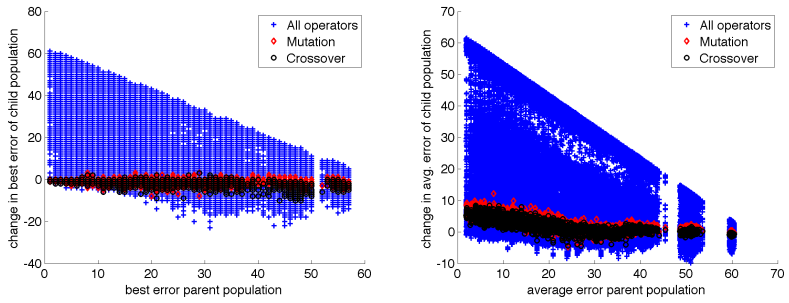
where Z_1' and Z_2' are the trees as modified by Υ. In all the presented experiments the input zippers are initially located at the root, and to produce a child we use the first of the two resulting trees. The full instruction set is shown in table 1.

The creation of standard GP variation operators with these instructions is easy. Note that the compositional nature of the instruction set leads to reading the variation operators from right to left, where the rightmost operators are applied first. A standard mutation operator can be represented as **(ZIP.RAND ZIP.RRLOC)**. In order to apply the operator two individuals are selected from the population, Z_1 and Z_2. **ZIP.RRLOC** moves Z_1 to a random location within its tree and leaves Z_2 unchanged. **ZIP.RAND** then replaces the subtree at Z_1 with a random subtree, and again Z_2 is unchanged. The modified root of Z_1 is then returned as the child program. Crossover can be expressed as **(ZIP.SWAP.ST ZIP.RRLOC ZIP.SWAP ZIP.RRLOC)**. The evaluation of this operator begins similarly with Z_1 being moved to a random location. **ZIP.SWAP** then renames the two zippers, with Z_2 becoming Z_1 and vice versa. Z_1 (formerly Z_2) is moved to a random location by **ZIP.RRLOC**. The subtrees rooted at Z_1 and Z_2 are then swapped by **ZIP.SWAP.ST**. The tree rooted at Z_1 is then returned as the child. We now continue to describe the results of applying other operators to real GP populations.

2.1 Problem: Lawnmower

The "lawnmower" problem was initially presented by (Dickmanns et al., 1987) as the problem of a robot traversing as many squares of a checkerboard as possible. Optimal solutions in this initial study were facilitated by the use of loops, a primitive form of modularity. In later work by Koza (1994) the problem was used as a tool for studying the performance of automatically defined functions (ADFs), a functional form of modularity in GP. Since these studies, the problem has been used to study modularity in a variety of GP systems (i.e. (Spector and Luke, 1996; Bruce, 1997; Walker and Miller, 2006; Spector et al., 2011)). It has been observed within GAs that crossover facilitates solutions to modular problems ((Mills and Watson, 2007)). Although such a claim does not appear to have been made within GP, (Kashtan and Alon, 2005) have observed that structural modularity can emerge in response to dynamic environments. Is it possible that recombinatory variation operators also facilitate stationary modular problems within GP?

3 Distribution of Operator Performance



(a) Best error (b) Average error

Figure 1: Performance of variation operators on the 8 by 8 lawnmower problem.

The goal of this initial portion of the study is to explore the capability of our zipper-based instruction set for performing variations during the evolution of a population of programs. However, it will not always true that an operator will be able to improve an initial random population, nor will it always be true that an operator will be able to improve a population later in evolution. Our primary goal is to understand how a variation operator can *change* a given population. This leads us to use a significantly modified evolutionary algorithm.

For each trial we initialize a fixed set of variation operators and a random GP population. The previously described mutation and crossover operators are ensured to be contained within every set of variation operators. This was done

Parent population error	Variation operator
55	(ZIP2.REP.ST ZIP.SWAP.ST ZIP.RIGHT ZIP.DOWN ZIP.DOWN ZIP.DOWN ZIP2.REP.ST ZIP.SWAP.ST ZIP.UP ZIP2.REP.ST ZIP1.ROOT ZIP1.REP.ST ZIP.DOWN ZIP2.ROOT)
Simplified	(ZIP2.REP.ST ZIP.SWAP.ST ZIP.RIGHT ZIP.DOWN ZIP.DOWN ZIP.DOWN ZIP2.REP.ST ZIP1.ROOT ZIP1.REP.ST ZIP.DOWN)
45	(ZIP.RRLOC ZIP.LEFT ZIP2.REP.ST ZIP1.ROOT ZIP2.ROOT ZIP1.REP.ST ZIP2.ROOT ZIP.RIGHT ZIP1.REP.ST ZIP.RLOC ZIP.SWAP.ST ZIP.LEFT)
Simplified	(ZIP2.REP.ST ZIP1.ROOT ZIP1.REP.ST ZIP.RIGHT ZIP1.REP.ST ZIP.RLOC ZIP.SWAP.ST)
31	(ZIP1.ROOT ZIP2.ROOT ZIP.RLOC ZIP1.REP.ST ZIP.DOWN ZIP.SWAP ZIP1.ROOT ZIP.SWAP ZIP.DOWN)
Simplified	(ZIP1.REP.ST ZIP.DOWN ZIP.DOWN)
25	(ZIP.RIGHT ZIP.RAND ZIP.RLOC ZIP.LEFT ZIP.SWAP.ST ZIP.RRLOC ZIP.RRLOC ZIP.RLOC ZIP.RAND ZIP.UP)
Simplified	(ZIP.RAND ZIP.RLOC ZIP.LEFT ZIP.SWAP.ST ZIP.RRLOC ZIP.RAND)
21	(ZIP2.ROOT ZIP.RAND ZIP1.REP.ST ZIP.RRLOC ZIP.LEFT ZIP.LEFT ZIP2.ROOT ZIP.SWAP.ST ZIP.RLOC ZIP.RIGHT)
Simplified	(ZIP.RAND ZIP.RRLOC ZIP.SWAP.ST ZIP.RLOC)
15	(ZIP1.REP.ST ZIP.DOWN ZIP.RAND ZIP.RRLOC ZIP1.REP.ST)
Simplified	(ZIP1.REP.ST ZIP.DOWN ZIP.RAND ZIP.RRLOC ZIP1.REP.ST)
4	(ZIP.RIGHT ZIP.DOWN ZIP.RRLOC ZIP.RIGHT ZIP.SWAP.ST ZIP.RRLOC ZIP.DOWN ZIP1.ROOT ZIP.RLOC ZIP.SWAP ZIP2.ROOT ZIP.RAND)
Simplified	(ZIP.SWAP.ST ZIP.RRLOC ZIP.SWAP ZIP.RAND)

Table 2: Selected best variation operators from the lawnmower problem and their simplified versions.

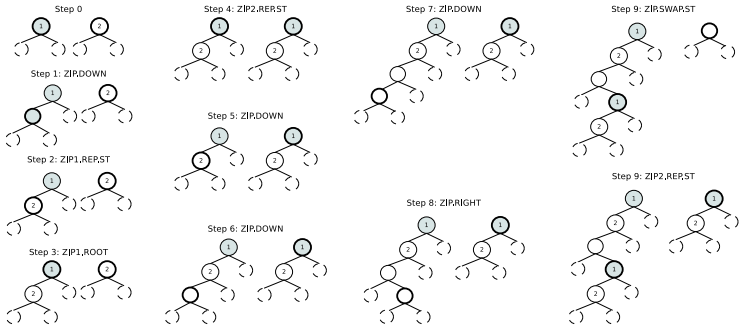


Figure 2: Best variation operator for lawnmower parent population of error 55: **(ZIP2.REP.ST ZIP.SWAP.ST ZIP.RIGHT ZIP.DOWN ZIP.DOWN ZIP.DOWN ZIP2.REP.ST ZIP1.ROOT ZIP1.REP.ST ZIP.DOWN)**. Shading indicates source tree. Bold outline indicates current zipper location. Dashed outline indicates phantom nodes.

both for the sake of comparison as well as to ensure that the population would continue to improve, although as we will see, the latter justification may very well have been unfounded. Every variation operator generates an equal-sized child population with each pair of parents chosen by fitness proportional selection. We compute statistics for the performance of each variation operator by measuring fitness changes between the parent and its respective child population. All child populations, as well as the parent population, are then combined into a single significantly larger population. The new parent population is generated by taking the best individuals from the merged populations. The process is then repeated until an individual of maximal fitness is present in the parent population. The goal of this algorithm is not to increase the optimality of evolutionary search, but to ensure that statistics are computed for variation operators with populations of a variety of fitness values.

Small population sizes are used maximize the range of error levels for parent populations. Initial populations are composed of full trees of depth 4. The maximum length of any variation operator is 15. Randomly generated operators are ensured to contain at least one tree manipulation operator. In total, 500 trials of the lawnmower problem are performed.

We examine the 8 by 8 lawnmower problem with turn and move limits of 100. The results for this problem are shown in figure 1. Figure 1a shows the change in best (lowest) error of parent population to child population for variation operators. The best operators are below the X-axis, while the operators along the upper diagonal bring the child population to maximal error. It is clear that crossover and mutation are useful operators as they generally produce some of the best individuals; however, there are a number of alternative operators that outperform both of these traditional operators. The results for changes in average population error (Figure 1b) tell a slightly different story.

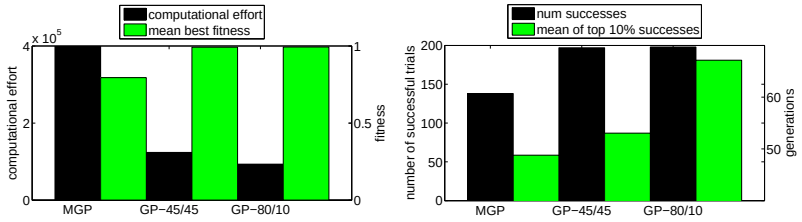
There are always variation operators that outperform mutation and crossover. Furthermore, as the average population error approaches 0 both standard GP operators appear to lose their ability to improve the average population error. Some alternative operators possess the remarkable ability to reduce the average population error to nearly 0.

Given the ease of expressing traditional GP operators with this instruction set, one might suspect that the better variation operators are simply reformulated versions of crossover and mutation where additional code that has no effect on produced child obfuscates their detection. For example, such instances might perform crossover, and apply a number of movement operators without performing additional variations. Although such reformulations do occur, that is not the predominant case. In table 2 we show some of the best performing variation operators on the lawnmower problem. The performance of these variation operators can be found in figure 1a by finding the parent population error on the x-axis and then examining the minimum point on the y-axis (with the exception of parent population error of 7, where one of three operators was chosen). Henceforth, operators will be referred to by their respective parent population errors. Operator 55 can be simplified to **(ZIP2.REP.ST ZIP.SWAP.ST ZIP.RIGHT ZIP.DOWN ZIP.DOWN ZIP.DOWN ZIP2.REP.ST ZIP1.ROOT ZIP1.REP.ST ZIP.DOWN)** by removing instructions that have no effect. A graphical representation of the effect of this operator is shown in figure 2. This operator begins by inserting Z_2 as a subtree of Z_1, followed by inserting a duplicate of itself deep into itself. This operator duplicates the left deep version of itself, effectively duplicating its left "module." Operator 4 can be simplified to **(ZIP.SWAP.ST ZIP.RRLOC ZIP.SWAP ZIP.RAND)** which is an alternative representation of mutation. Operator 15 possesses a particularly unique form of variation. A random subtree is inserted into a candidate solution, and a copy of of the same candidate solution is then inserted in place of the left branch of the random tree. Many other operators have been tested in this study, a few of which have been selected for their ability to make significant jumps in maximum fitness (Table 2).

While investigating these interesting alternative operators we have avoided highlighting the large number of variation operators that increase error. The upper portion of both plots in figure 1 contain a large number of such variation operators. These are the operators of worry when performing meta-GP. Is the risk of injecting these detrimental variation operators worth the potential gains? We answer this question in the following section.

4 Meta-GP

We use the simplest evolutionary algorithm possible in our comparison. As opposed to Koza who uses overselection with fitness proportional selection to expedite his search for solutions of the lawnmower problem, we use only fitness proportional selection. This leads us to study the 8x6 lawnmower problem with move and turn limits of 75. Both the GP and meta-GP systems use configu-



(a) Computational effort and mean best fitness.

(b) Total and fastest successes.

Figure 3: Comparison of meta-GP and GP performance.

ration parameters matching (Koza, 1994) with the exceptions: no overselection is used, variation parameters (which are discussed later), generation limit is 151 generations, and population size is 500. The meta-population, composed of zipper-based variation operators, uses the mean over a sliding window (size 3) of winner-take-all (WTA) scores. The WTA score is the fitness of the best program produced by an operator within a given generation. During the reproduction phase of the meta-population, variation operators inherit the score history of the primary parent operator. 5 forms of variation are used on the meta-population: 80% one-point crossover, 3.3% mutation, 3.3% shuffle (complete permutation), 3.3% deletion, and 10% reproduction. Three constraints are imposed on variation operators: the maximum size is 20, the minimum size is 2, and they must contain at least 1 recombinatory instruction that modifies the first zipper. The limit on minimum size is imposed to resolve the introduction of **(ZIP.RAND)** which may achieve a high fitness, but is not a "productive" variation operator. The constraint on recombinatory instructions that modify Z_1 reduces the number of invalid programs that are considered. **ZIP.RAND** is not included within this constraint because in practice this often leads to programs that simplify to **(ZIP.RAND)**. In the meta-GP study, we remove the instructions **ZIP.UP** and **ZIP2.ROOT** which preliminary experiments revealed to be used infrequently. These results are generated from 200 trials of each parameter configuration.

We compare 3 configurations: first, standard GP with 80% crossover, 10% mutation, and 10% reproduction, second, standard GP with 45% crossover, 45% mutation, and 10% reproduction, and third, meta-GP. The meta-population is composed of 50 variation operators, and is subject to the previously described selection and variation processes.

The results of the meta-GP/GP comparative trials are presented in figure 3. We begin with figure 3a. In terms of computational effort (for details see Koza (1990)) meta-GP lags behind standard GP. However, this is not surprising as meta-GP must first sort through a random population of variation operators, many of which have no meaning. We note that the mean best fitness (MBF)

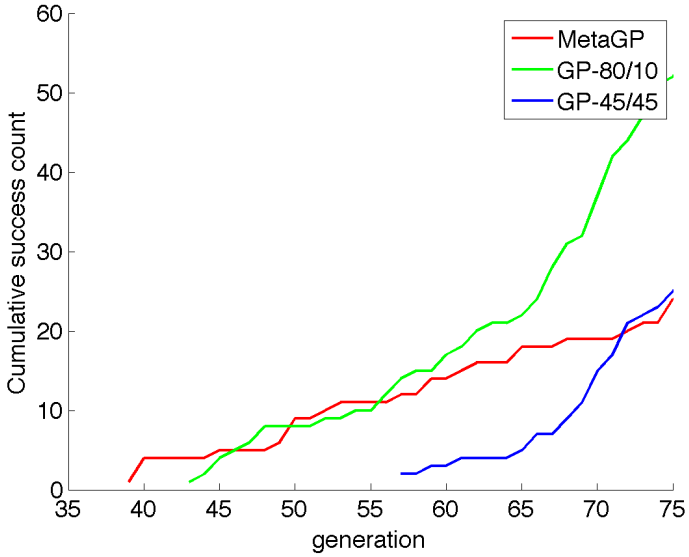


Figure 4: Comparison of GP's and Meta-GP's initial cumulative successes with respect to generation.

in figure 3a shows meta-GP nearly on par with GP. The relative similarity of MBF values support the notion that meta-GP has an initial cost to overcome but recovers, because the terminal fitness values are generally near maximal fitness. In figures 3b and **??** we can see the benefits of meta-GP more clearly. While there are fewer successes in total for meta-GP, the top performing meta-GP runs are faster than the top performing GP runs. The value of shorter runs with larger population sizes under certain circumstances has been quantified Luke (2001). This supports the observations seen in the distribution of operator performance which suggest that there are alternative operators capable of outperforming traditional GP operators.

5 Discussion

We present an analysis of zipper-based variation operators and show that they can outperform traditional GP operators. In using these operators in a meta-GP context we discover a "Babe Ruth" type phenomenon. When meta-GP performs well, it outperforms GP. Yet in many cases there is an initial cost that meta-GP must overcome. We can explain some of the difficulties of meta-GP by noting that meta-GP must deal with an initial set of operators of unknown quality, while GP starts out with hand-picked operators known to perform well.

Genetic programming has often referred to as a dark art. When using GP

to solve problems the practicioner must be aware of not only the properties of function sets but the effects of numerous parameters. Not only does the manual selection of parameters introduce bias, but the operators themselves (crossover and mutation) introduce bias. We have shown that it is possible to avoid this bias at the level of the problem solving population by using a population of zipper-based variation operators.

The utility of our language is clear in these results. Although the performance of crossover and mutation are consistent, alternative operators possess the ability to make larger jumps in the solution space at various stages of a population's evolution. This consistent performance of crossover and mutation supports the historical decisions to use these operators within the GP community. However, we note that the performance of alternative variation operators can easily double that of crossover and mutation. In particular the ability to both improve average population error throughout the entire evolution as well as make large jumps in best fitness suggest that it may be time to move beyond the traditional variation operator bias of genetic programming.

In light of our observations that there are many alternative variation operators capable of outperforming traditional crossover and mutation, we suspect that further study should be able to improve the consistency of meta-GP's performance. Seeding the meta-population with standard GP operators may allow meta-GP to overcome the initial cost of discovering useful variation operators de nouveau; however, we encourage focus on techniques for variation operator selection and ranking as the primary approach for improving meta-GP.

We have presented and measured the performance of a language for the concise representation of variation operators. By studying the lawnmower problem we have seen that some of the best operators function by increasing the structural modularity of programs. Emperical evidence of structural modularity (repeated patterns) can be seen in Langdon and Banzhaf (2008). The ability of our zipper-based language to naturally facilitate structurally modular solutions is primarily in the form of recombinatory modularity. That is, code from other individuals in the population, and occasionally the same individual, is incorporated as sub-programs within new individuals. The advantage of evolution via recombination is often explained as a response to changing environments (Maynard Smith and Szathmáry, 1992). One of the most important ideas from coevolutionary theory is that the evolving population itself is a dynamic environment. As such we suggest that meta-evolution and coevolution are not only linked via the coevolving variation operators, but even through the coevolution amongst members the problem solving population.

Future work should explore the selection and ranking mechanisms of the meta- population, as well as the coevolutionary consequences of meta-GP[1].

[1]Additional work explores zipper-based autoconstructive evolution (Harrington et al., 2011).

6 Acknowledgements

The authors thank Lee Spector for encouraging the use of zippers, Emma Tosch for many insightful discussions, the DEMO lab, the Hampshire College CI lab, and Carolyn Earnest. This material is based upon work supported by the National Science Foundation under Grant No. 0757452. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Altenberg, L. (1994). The evolution of evolvability in genetic programming. *Advances in genetic programming*, pages 47–74.

Altenberg, L. and Feldman, M. (1987). Selection, generalized transmission and the evolution of modifier genes. I. The reduction principle. *Genetics*, 117(3):559.

Angeline, P. (1995). Two self-adaptive crossover operations for genetic programming. In *Advances in genetic programming 2*, pages 89–110.

Baum, E. and Durdanovic, I. (1998). Toward code evolution by artificial economies. In *Late Breaking Papers at the Genetic Programming*, pages 22–25.

Bruce, W. (1997). The lawnmower problem revisited: Stack-based genetic programming and automatically defined functions. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 52–57.

Dickmanns, D., Schmidhuber, J., and Winklhofer, A. (1987). Der genetische algorithmus: Eine implementierung in prolog. *Fortgeschrittenenpraktikum, Institut f ur Informatik, Lehrstuhl Prof. Radig, Technische Universit at M unchen*.

Edmonds, B. (2001). Meta-genetic programming: Co-evolving the operators of variation. *Turk J. Elec. Engin.*

Fogel, D., Fogel, L., and Atmar, J. (1991). Meta-evolutionary programming. In *Signals, Systems and Computers, 1991. 1991 Conference Record of the Twenty-Fifth Asilomar Conference on*, pages 540–545.

Grefenstette, J. (1986). Optimization of control parameters for genetic algorithms. *Systems, Man and Cybernetics, IEEE Transactions on*, 16(1):122–128.

Harrington, K., Tosch, E., Spector, L., and Pollack, J. (2011). Compositional Autoconstructive Dynamics. In *Proc. of the 8th Intl. Conf. on Complex Systems*.

Hickey, R. (2008). The Clojure programming language. In *Proc. of the 2008 Symp. on Dynamic Languages*, page 1.

Holland, J. (1985). Properties of the bucket brigade. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 1–7.

Huet, G. and France, I. (1997). Functional pearl: The zipper. In *J. Functional Programming*, pages 549–554.

Kantschik, W., Dittrich, P., Brameier, M., and Banzhaf, W. (1999). Meta-evolution in graph GP. *Genetic Programming*, pages 652–652.

Kashtan, N. and Alon, U. (2005). Spontaneous evolution of modularity and network motifs. *Proceedings of the National Academy of Sciences of the United States of America*, 102(39):13773.

Koza, J. (1990). Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Technical report, Stanford University Computer Science Department.

Koza, J. (1994). *Genetic programming II: automatic discovery of reusable programs*.

Langdon, W. and Banzhaf, W. (2008). Repeated patterns in genetic programming. *Natural Computing*, 7(4):589–613.

Luke, S. (2001). When short runs beat long runs. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 74–80.

MacCallum, R. (2003). Introducing a Perl genetic programming system-and can meta-evolution solve the bloat problem? *Genetic Programming*, pages 17–49.

Maynard Smith, J. and Szathmáry, E. (1992). The origin of life.

Mills, R. and Watson, R. (2007). Variable discrimination of crossover versus mutation using parameterized modular structure. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1312–1319.

ONeill, M. and Brabazon, A. (2005). mGGA: The meta-grammar genetic algorithm. *Genetic Programming*, pages 311–320.

Poli, R. (2001). Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover. *Genetic Programming and Evolvable Machines*, 2(2):123–163.

Schmidhuber, J. (1987). *Evolutionary principles in self-referential learning. (On learning how to learn: The meta-meta-... hook.)*. PhD thesis, Institut für Informatik, Technische Universität München.

Spector, L. (2010). Towards Practical Autoconstructive Evolution: Self-Evolution of Problem-Solving Genetic Programming Systems. *Genetic Programming Theory and Practice VIII*, pages 17–33.

Spector, L. and Luke, S. (1996). Cultural transmission of information in genetic programming. In *Proceedings of the First Annual Conference on Genetic Programming*, pages 209–214.

Spector, L., Martin, B., Harrington, K., and Helmuth, T. (2011). Tag-based modules in genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2011)*.

Spector, L. and Robinson, A. (2002). Genetic programming and autoconstructive evolution with the push programming language. *Genetic Programming and Evolvable Machines*, 3(1):7–40.

Teller, A. (1996). Evolving programmers: The co-evolution of intelligent recombination operators. In *Advances in genetic programming 2*, pages 45–68.

Tuson, A. (1995). Adapting operator probabilities in genetic algorithms. *Master's thesis, Department of Artificial Intelligence, Univeristy of Edinburgh*.

Walker, J. and Miller, J. (2006). Embedded cartesian genetic programming and the lawnmower and hierarchical-if-and-only-if problems. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 911–918.