A Probabilistic Functional Crossover Operator for Genetic Programming

Josh C. Bongard
Department of Computer Science
University of Vermont
Burlington, VT 05405
josh.bongard@uvm.edu

ABSTRACT

The original mechanism by which evolutionary algorithms were to solve problems was to allow for the gradual discovery of sub-solutions to sub-problems, and the automated combination of these sub-solutions into larger solutions. This latter property is particularly challenging when recombination is performed on genomes encoded as trees, as crossover events tend to greatly alter the original genomes and therefore greatly reduce the chance of the crossover event being beneficial. A number of crossover operators designed for tree-based genetic encodings have been proposed, but most consider crossing genetic components based on their structural similarity. In this work we introduce a tree-based crossover operator that probabilistically crosses branches based on the behavioral similarity between the branches. It is shown that this method outperforms genetic programming without crossover, random crossover, and a deterministic form of the crossover operator in the symbolic regression domain.

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: Artificial Intelligence

General Terms

Experimentation, Algorithms, Performance

Keywords

homologous crossover, crossover operators, schema theory

1. INTRODUCTION

Genetic programming [10] refers to a family of algorithms that employ various data structures to represent candidate solutions to a given problem. These genotypes either produce behavior directly that is then selected, or are directly or indirectly transformed into a phenotype that in turn exhibits behavior which is subjugated to selection pressure. The choice of genetic encoding, the genotype to phenotype mapping, and the variation operators have a significant impact on the system's evolvability [18], or ability to continually improve solutions.

The choice of variation operators is of particular interest in that they significantly affect how the population moves through the search space. Mutation operators are designed to discover better variants of a single genotype; crossover operators on the other hand should, when implemented properly, combine useful genetic substructure from multiple genotypes. Because most genetic programming instantiations are tree-based, crossover typically involves swapping subtrees between two parent trees, and this structural change often has a large phenotypic effect on the resulting genotypes. As originally articulated by Fisher [5], the magnitude of the phenotypic effect of a genetic perturbation is inversely proportional to the probability of that perturbation being beneficial. For this reason it is often observed that random subtree crossover can adversely affect the performance of a genetic programming system. It may favor gradual increase in the size of genotypes over evolutionary time without providing any fitness benefit, a problem known as bloat [11], and/or it may slow search by producing offspring that are less fit than their parents.

Several crossover operators have been proposed in the GP literature to improve their ability to combine useful genetic substructure from several parent genotypes. Headless chicken crossover [9] crosses subtrees between two GP trees in which one tree has survived selection while the second is created randomly in an attempt to introduce fresh genetic material into the population. Size fair crossover [12] crosses subtrees between parent trees with a probability that is proportional to the size similarity between the selected subtrees. Homologous crossover refers to a family of crossover operators that attempt to preserve the context of the two crossed subtrees within their parent trees. D'haeseleer [4] has described deterministic and Langdon [12] probabilistic homologous crossover operators that swap subtrees based on the similarity of their positions within their parent trees. Other homologous crossover operators based on syntactic similarity [16, 14] have met with limited success.

Several researchers have argued that genetic material should be combined based on its semantic, rather than syntactic or structural similarity. Semantic crossover [1] uses standard (random) crossover between two trees and then retains the resulting trees only if they differ semantically from their parents. In enzyme genetic programming [13], genotypes are composed of independent elements that attach to

GECCO'10, July 7–11, 2010, Portland, Oregon, USA.

one another based on their input and output characteristics. Crossover is accomplished by injecting elements from a donor into an existing genotype; the donated components will only be incorporated into the new genotype if they can connect to existing components.

In this paper we introduce a probabilistic crossover operator that swaps subtrees based on their functional (semantic) rather than structural (syntactic) similarity, in an attempt to reduce the magnitude of the phenotypic effect of the cross. A deterministic version of this operator was reported in [3]. The next section describes both a deterministic and probabilistic form of this functional crossover operator and its application to symbolic regression. Section 3 contrasts these two crossover operators with standard crossover and no crossover, section 4 analyzes why the probabilistic form of the operator out-competes these other algorithm variants, and section 5 provides some concluding remarks.

2. METHODS

This section describes the functional crossover (FXO) concept as it relates to genetic programming, in both its previously-published deterministic form (D-FXO, [3]) and its probabilistic form, the latter which is explored in this paper.

2.1 Deterministic Functional Crossover

The primary difference between functional crossover and previously-described recombination operators is that functional crossover swaps genetic material between parents based on the *behavioral* rather than *structural* similarity of the crossed material. In the symbolic regression task explored here, genetic programming is used to automatically evolve a model of a hidden function. Thus, the behavior of any node in a GP tree can be defined as the range of values that it propagates upward while the tree is evaluated.

As a tree is evaluated against each datum in a training set, the minimum and maximum value computed by each node is recorded at the node. If a terminal node encodes a floating-point value: the minimum and maximum values are equal, and are equal to the constant. If the node encodes a dependent variable, the node records the minimum and maximum value of that dependent variable in the training set. If a non-terminal node roots the branch $(\times)(2)(x_0)$, then that node's minimum and maximum value will be equal to twice the minimum and maximum value returned by the dependent variable x_0 during evaluation, and so on.

Thus after evaluation, each node in a tree has a range associated with it. In deterministic functional crossover (D-FXO), when two parent trees are to be crossed, a node i is chosen at random in the first parent. The second parent is then scanned to find the node k that has the closest range to that of the chosen node in the first parent using

$$d_{ij} = \frac{1}{2}(|\max_i - \max_j| + |\min_i - \min_j|), \tag{1}$$

where d_{ij} is the behavioral distance between nodes i and j, and $\min_i$ and $\max_i$ is the minimum and maximum value computed by node i during evaluation, respectively. Once both nodes have been determined, the branches rooted at those nodes (along with the nodes themselves) are swapped as in standard GP crossover, the ranges are erased from each node in the two trees, and the two new trees are evaluated.

The results of using this operator for evolving sets of ordinary differential equations to describe both simulated and physical systems was reported in [3]. Although the results were promising, this operator suffers from the limitation that many crosses are neutral: if the selected node in the first parent roots a branch that is structurally identical to a branch that exists in the second parent (or can be algebraically re-arranged into such a structure), those branches will be swapped and the overall behavioral effect on the new trees will be zero.

Such neutral crosses become more common as optimization proceeds and the trees become larger, because the probability of choosing a terminal node (or a non-terminal node high depth) increases for larger trees, and there is thus a greater probability of finding a structurally identical branch in the second parent because the branch itself contains one or only a few nodes. Such a neutral cross triggered by the selection of the first node often occurs even though there is a behaviorally similar but not behaviorally identical branch in the second parent that would be reduced the error of the first parent. By choosing the second node deterministically, this useful other branch will never be chosen.

2.2 Probabilistic Functional Crossover

In order to combat this pathology, a probabilistic form of the functional crossover operator (P-FXO) was developed, and is explored in this paper. In P-FXO each node in a tree is labeled with its minimum and maximum value as in D-FXO. Also, when two trees are chosen for crossing, a node is chosen at random in the first parent. The distance between the selected node and each node j in the second parent is then computed using eqn. 1. The resulting distances are then normalized so that the distances as a whole represent a probabilistic density function:

$$d'_{ij} = \frac{d_{ij}}{\sum_{k=1}^{s} d_{ik}} \tag{2}$$

where d'_{ij} is the normalized distance and s is the number of nodes in the second parent. These normalized values are then inverted and re-normalized so that the greater the distance between node i and j, the less chance there is of node j being chosen as the second cross point:

$$p_{ij} = \frac{1 - d'_{ij}}{\sum_{k=1}^{s}(1 - d'_{ik})} \tag{3}$$

where p_{ij} now represents the probability that node j will be selected as the second cross point. Once node j in the second parent is selected, the branches are swapped in the normal manner, the range at each node for both new trees is erased, and the two new trees are evaluated.

3. RESULTS

Five hundred independent runs were conducted for four experimental regimes. Each run was initially seeded with random trees grown by adding a terminal or non-terminal node if the current depth is less than five; otherwise, a terminal node is added. Throughout the run, if a mutation and crossover event creates a tree with a depth greater than five, the new tree is discarded and the event is repeated until a tree with depth of at most five is created. An additional $500 \times 4 = 2000$ runs were performed in which trees had a maximum depth of six, and a third set of 2000 runs were performed with a maximum depth of seven.

Valid operator nodes were four algebraic operators (+, −, ×, div), two trigonometric operators (sin(), cos()) and

a null operator with an arity of one (values arriving from the child are passed up unchanged to the node's parent). Operand nodes could refer one of the two dependent variables x_0 or x_1; an additional binary parameter encoded at the node indicates which dependent variable that node refers to. An operand node may also encode a constant value, represented as a random floating-point value in $[-1000, 1000]$. During initial tree construction, each operator and operand type had an equal probability of being selected.

At the outset of each run, the target function was created as follows. A randomly-generated tree was created (not exceeding the maximum depth) and used as the hidden function to model. $21 \times 21 = 441$ values for the dependent variables x_0 and x_1 were generated in a uniform sequence across the interval $[-1, 1]$. This produced tuples in the sequence $(-1, -1)$, $(-1, -0.9)$, ... $(-1, +1)$, $(-0.9, +1)$, ... $(+1, +1)$, which were stored in a vector $\mathbf{x}$. Each tuple was used to evaluate the target function, resulting a results vector $\mathbf{y}$ of length 441. The tuple $(\mathbf{x}, \mathbf{y})$ is therefore the training set for the run.

A second set of test data was generated by creating $20 \times 20 = 400$ test cases with x_0 and x_1 varying uniformly over the interval $[-0.95, -0.85, \ldots, +0.95]$. This ensured that each test datum differed from any point in the training set. The resulting 400 tuples were stored in the vector $\mathbf{x}'$. The target function was again run against each test point to generate a test results vector $\mathbf{y}'$; the tuple $(\mathbf{x}', \mathbf{y}')$ therefore forms the testing set. No noise was added to either the training or testing set.

Each run was initially seeded with 1000 randomly-generated trees, and was run for 10000 generations. In the first generation, each tree i is evaluated on both the training set and the testing set using the error function

$$e = \frac{\sum_{i=1}^{441|400} |y_i^{(t)} - y_i^{(m)}|}{441|400},$$

where $y_i^{(t)}$ indicates the result of training (or testing) point i returned by the target function, and $y_i^{(m)}$ indicates the result of the model for the same point. e_r is henceforth used to denote the tree's error on the training set, and e_t on the testing set.

The solutions in the population were then sorted in decreasing order according to e_r, and the lowest 1% (100 trees) were discarded. This very high level of elitism was adopted so that there would a large amount of different genetic material for the crossover operators to draw on. The empty 100 slots were then filled by randomly selecting and copying from the remaining 990 trees, with replacement (i.e. a tree could be chosen twice).

Each of the newly-created trees underwent mutation with a probability of 50%. When a tree was mutated, one of its nodes was chosen at random. This node and its child branches were deleted, and replaced with a random node. If the targeted node had less than the maximum depth one of the operator or operand node types was selected at random with each type being selected with a uniform probability. If the node was at the maximum depth one of the operand node types was selected at random. If an operator node was created, its descendent nodes were created in the same manner. When a dependent variable node was created it was set to either x_0 or x_1 with equal probability. If a constant-value node was created its value was selected from $[-1000, 1000]$ with uniform probability. It is noted that this is a simple yet weak mutation operator: As the focus of this work was on the crossover operators introduced here, no effort was put into improving the mutation operator.

In the first of the four experimental regimes, no crossover operator was employed; evolution proceeds solely through mutation. In the second through four regimes, crossover is applied: in the second regime random crossover is applied; in the third regime D-FXO is applied; and in the fourth regime P-FXO is applied. At the end of each generation in those regimes that employ crossover, each of the 100 newly-generated trees are paired up randomly, and each pair is crossed with a probability of 50%. In the next generation the 100 possibly mutated, crossed, or mutated and crossed trees are evaluated, and the run continues in this manner to completion.

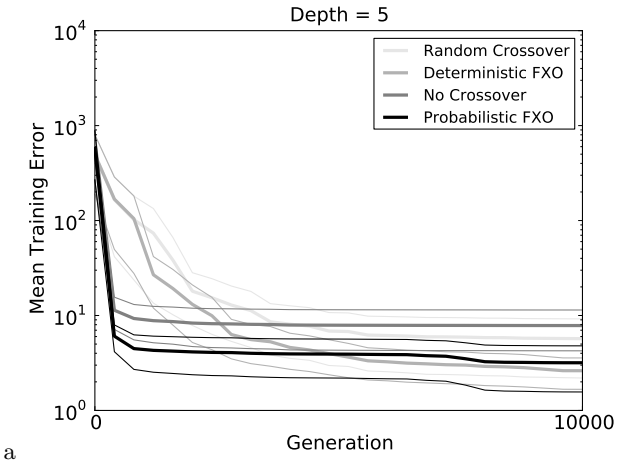
Fig. 1 reports the mean error of the trees in the population for all four experimental regimes. As can be seen, there is relatively little difference in error reduction across the four regimes, with the possible exception of the higher rate of error reduction at the outset of the run for the P-FXO regime when the maximum tree depth is five or six. Also, the D-FXO regime performs significantly better than the other regimes when the maximum depth is seven.

Fig. 2 reports the mean performance of the four regimes, but measures their mean errors on the unseen testing set (e_t). This figure depicts a very different story. It can be seen that the regimes with random or D-FXO perform very poorly on the unseen data, indicating that these evolved solutions have overfit the data. In fact these two crossover operators are deleterious in these experiments, as they perform significantly worse than an equivalent experimental regime in which no crossover is employed (dark gray lines). However, the regime that employs P-FXO performs significantly better than the other three regimes when maximum depth is six, and is substantially (yet not significantly better) better than other regimes for the two maximum depths.

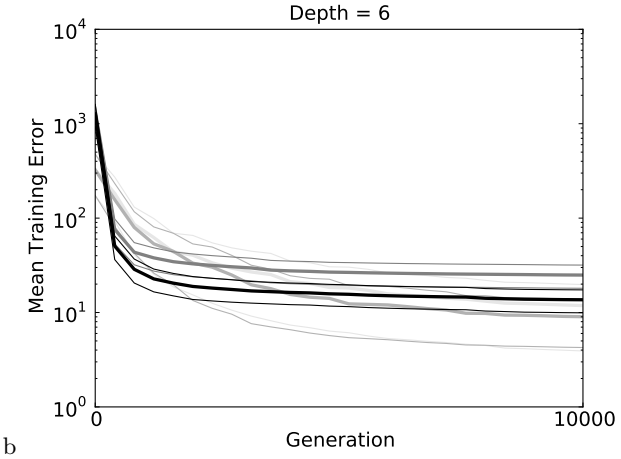
Fig. 3 reports the mean sizes of the evolved trees across the three maximum depths and four experimental regimes. As can be seen, despite the fact that the regimes with random crossover and D-FXO overfit the data such that they perform poorly on the unseen data, these size of the solutions in these regimes do not grow rapidly. Indeed, they are generally smaller than the actual size of the target function (as evidenced by the light gray lines falling far below the dotted lines in Fig. 3). Conversely, evolved solutions in the regimes that employ no crossover or P-FXO tend to grow much larger than the target function. Despite this however, they encode accurate, robust models of the target function, especially when P-FXO is employed. This suggests that in the no crossover and P-FXO regimes, significant compression could be performed on these solutions, or mutation operators designed to reduce the size yet maintain the accuracy of the solutions could be employed [2].

4. ANALYSIS

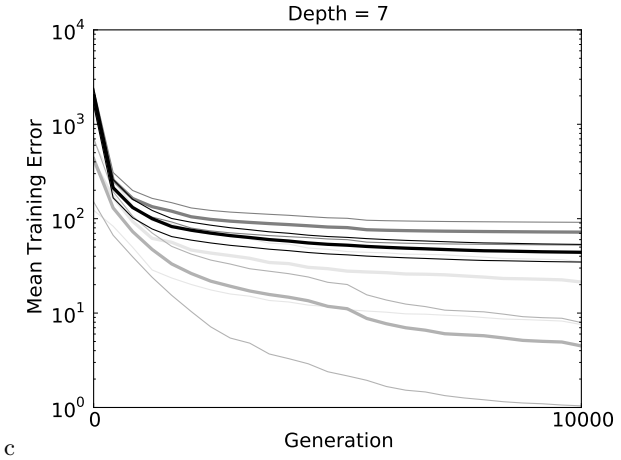
The observation that only the probabilistic functional crossover operator does better than using no crossover at all warrants some analysis. To do this, data was collected from each solution that underwent crossover. The testing error (e_t) of each parent solution that imported external genetic material was recorded, as was the testing error of the resulting solution after evaluation. Also, the size of importing par-



a

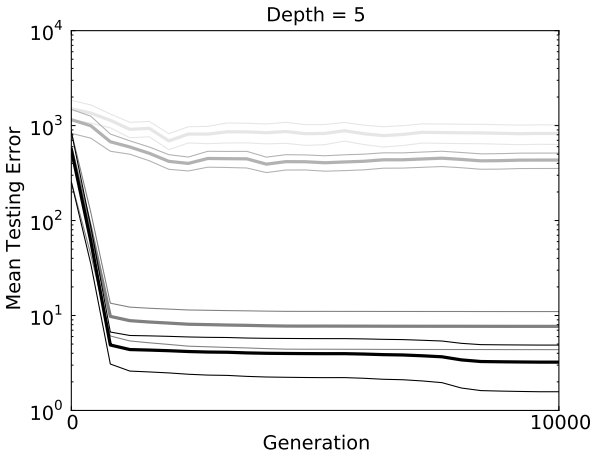


b

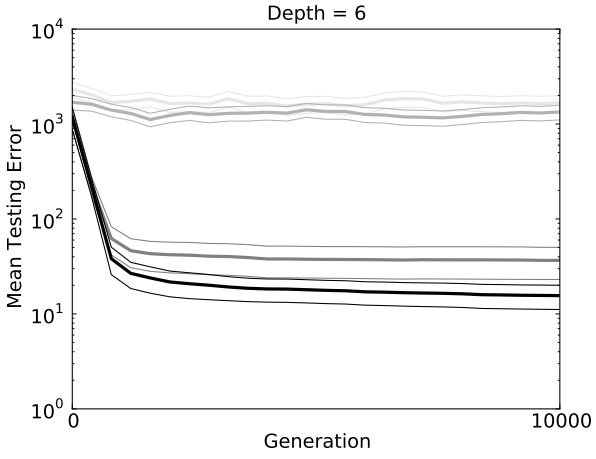


c

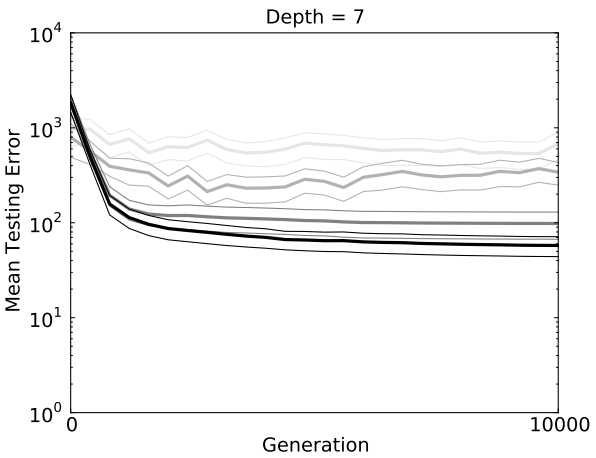
Figure 1: Training errors averaged across the four experimental regimes. Thick lines indicate the mean for that experimental regime. The thin lines bracketing each mean report one unit of standard error from the mean.



a



b

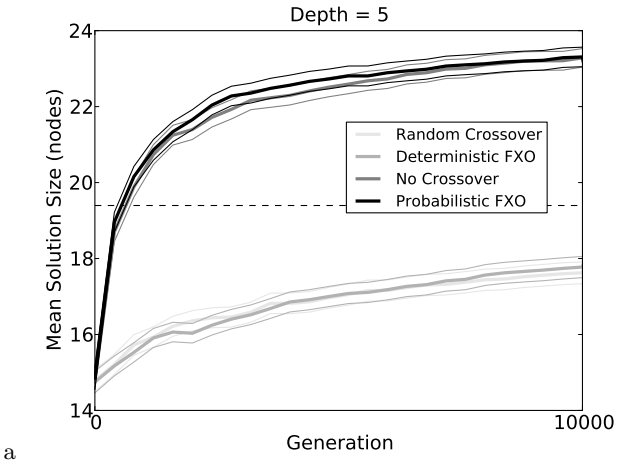


c

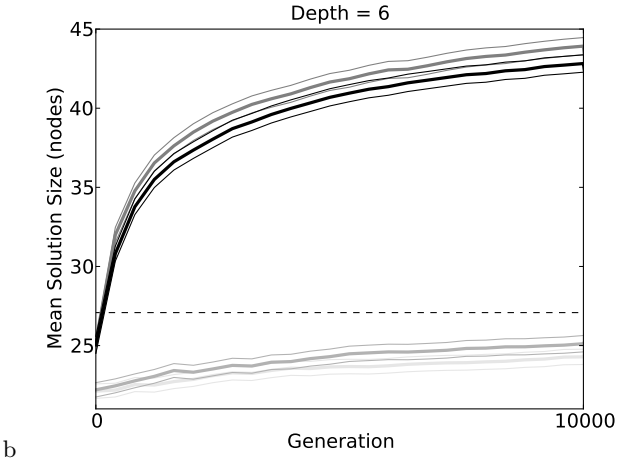
Figure 2: Testing errors averaged across the four experimental regimes.

ent solution was recorded. For each such crossover event, it was determined whether the crossover event was beneficial in the sense that the child's error was lower than its parent ($E_c < E_p$); the event was phenotypically neutral in the sense that both errors are equal ($E_c = E_p$); or the event was detrimental to the child ($E_c > E_p$).

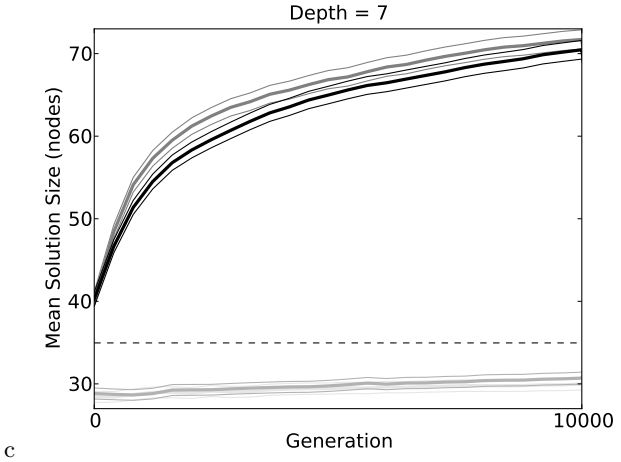
It should be noted that newly-generated trees may have undergone both mutation and crossover, so that a change in error from parent to child may not be due to the crossover event alone. However, the analysis only compares the means rates of beneficial, neutral and detrimental crossover be-



a

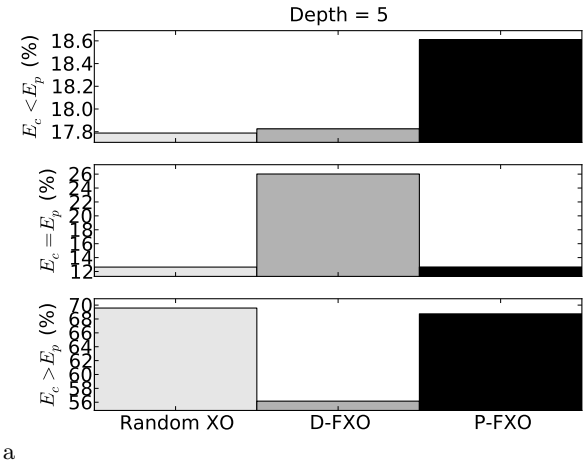


b

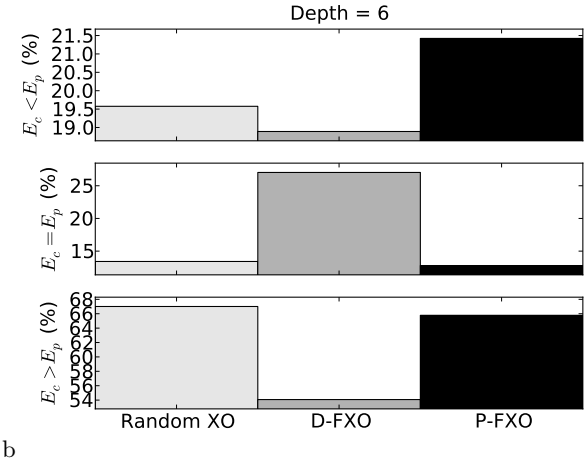


c

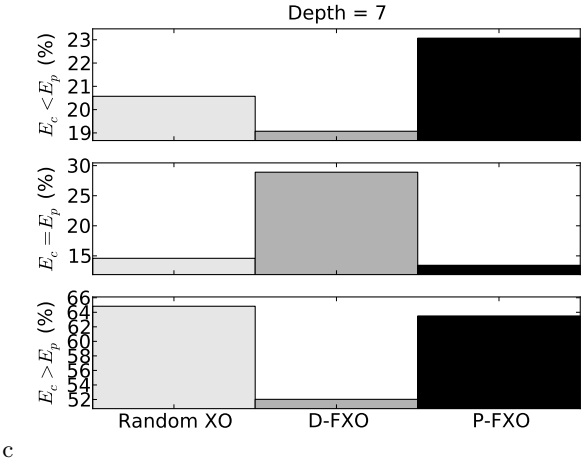
Figure 3: Size of evolved solutions averaged across the four experimental regimes. The dotted line reports the mean size of the target functions for each of the three maximum tree depths.



a



b



c

Figure 4: Percentage of solutions that underwent crossover that achieved less error (top row of each panel), the same error (middle row), or greater error (bottom row). Percentages compared across the three regimes that employed crossover.

tween experimental regimes, all of which have a constant rate of mutation. Therefore, any observed differences between these rates across regimes can be attributed to the effect of the different kinds of crossover operators employed in those regimes.

Fig. 4 reports the percentage of beneficial, neutral and detrimental effects caused by all of the crossover events in the regimes that used random crossover (left column), deterministic functional crossover (middle column) and prob-

abilistic functional crossover (right column). As expected, the majority of crossover events are detrimental (lowest row in each panel). However, the regime that employed D-FXO tends to have a significantly[1] lower rate of detrimental crossover events, but a much higher rate of neutral crossover events. This can be explained by the fact that if a D-FXO event selects a branch for crossover that is semantically identical to a branch that exists in the second parent (for example x_0 or $\times(x_0)(x_1)$), this other branch will be selected for crossover by definition. Because both branches compute the same function, the crossover event will have a neutral effect on the new solutions' error.

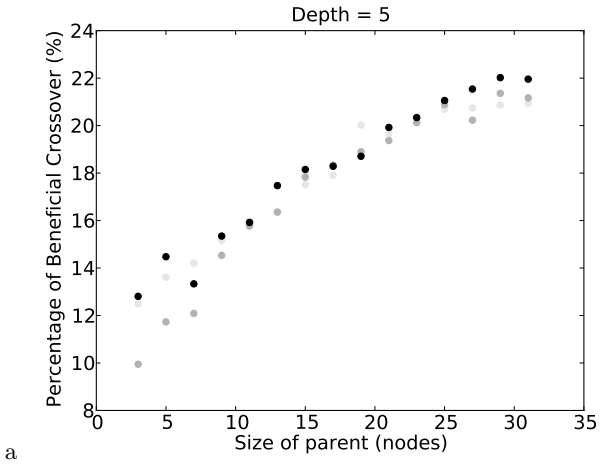
The regime employing probabilistic functional crossover however demonstrates a rate of beneficial crossover events that is significantly higher than the other two regimes (compare the black bar in the upper row of each panel against the other two bars). Despite this significant difference however, the total percentage difference is relatively low: about 2% more of the crossover events in the P-FXO regime are beneficial compared to the other two regimes.

Nevertheless this slight difference is sufficient to ensure consistently better performance in this regime; fig. 5 provides evidence to suggest why this is the case. It reports the percentage of beneficial crossover events as a function of size (in nodes) of the parent tree when it imported genetic material (its new size after the crossover event is ignored). It can be seen that there is a correlation between the size of the importing tree and the percentage of beneficial crossover events, across the three experimental regimes that employed crossover, and for all three of the maximum tree depth experiments.

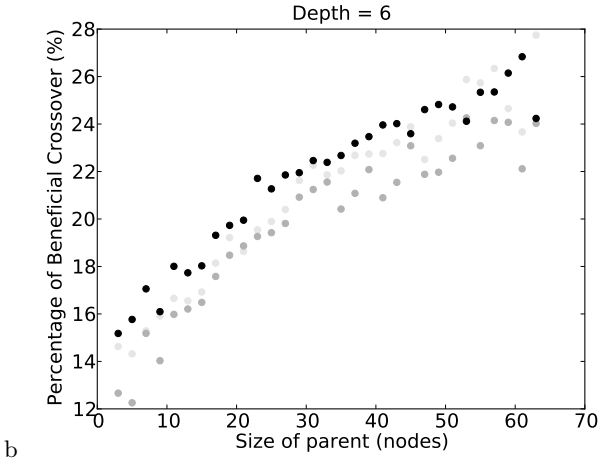
This is to be expected, as in essence larger trees are buffered against genetic alteration. The ratio of terminal nodes to internal nodes increases in larger trees, so that terminal nodes (or at least deeper nodes) are selected more often than internal (or shallower nodes). Crosses at leaf nodes or deep nodes tend to have less overall behavioral effect on the new tree, and it has often been observed in biological [5] and computational [2] studies that the probability of a genetic perturbation being beneficial is inversely proportional to its phenotypic effect.

For smaller trees, the percentage of beneficial crossover events is therefore quite low. However, when the maximum tree depth is six or seven (Fig. 5b,c), the regime that employs probabilistic functional crossover achieves significantly higher rates than the other two regimes. For larger trees, there is little difference in the beneficial crossover rates across the three regimes. As can be seen in Fig. 3, smaller trees are rapidly replaced by larger trees during the early stage of optimization. Together with Fig. 5 this suggests that in the regime that employs probabilistic functional crossover, there are many crosses between small, relatively inaccurate trees that give rise to larger, more accurate trees.

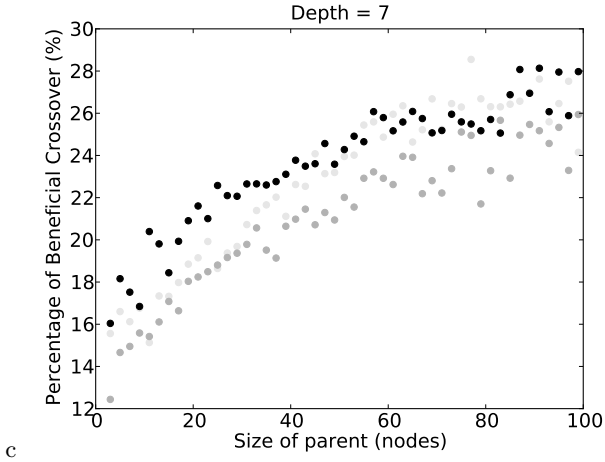
This suggests that P-FXO is achieving one of the original and (remaining) primary goals of evolutionary computation (at least in this domain), which is to support the automated combination of genetic building blocks [8, 6, 16, 15]: P-FXO combines complementary genetic material to create larger, more accurate trees. Further work is warranted however to prove this result in more rigor, and to demonstrate it on other domains besides symbolic regression.



a



b



c

Figure 5: The percentage of beneficial crossover events as a function of the size of solution undergoing crossover. Sizes report the original size of the solution before crossover. Light gray, dark gray and black dots represent crossover event results from the experimental regime employing random crossover, D-FXO, and P-FXO, respectively.

[1]Fig. 4 does not report any error bars because the error margins are infinitesimal due to the large sample size: 0.5 probability of mutation × 100 possible mutation events per generation × 10000 generations gives $n = 500000$.

5. CONCLUSIONS

This paper has introduced a new recombination operator that significantly increases the probability of a beneficial recombination event. This is accomplished by probabilistically weighting the nodes in the genome based on their *behavioral* rather than *structural* similarity. In the symbolic regression task demonstrated here, the minimal and maximal values propagated up through each node in a genetic programming tree are a proxy for the behavior of the branch rooted at that node. Behavioral similarity is thus regarded as the similarities between the ranges defined by these minimal and maximal values.

It was shown that this recombination operator, referred to as *probabilistic functional crossover (P-FXO)*, consistently out-competes random crossover on this task, as well as an equivalent regime in which no crossover is employed, and another regime in which nodes are exchanged deterministically based on their range similarities.

It is noted that comparing ranges between nodes is a crude approximation of the branch behaviors rooted at those nodes. In future work we plan to create probability density functions (PDFs) at each node during the evaluation of a tree, and probabilistically compare the resulting PDFs when choosing crossover points. In this work a large elitism percentage was used (90%) so that there was a lot of variation in the population to draw on during crossover. We also plan to investigate whether considering behavioral similarities across nodes and over multiple trees further improves the recombination power of P-FXO.

Finally, by altering the kurtosis of the PDF at each node using a single parameter such that low values of the parameter produces a more uniform distribution than the original PDF and a high value produces a more peaky distribution than the original PDF, it would become possible to tune the magnitude of the behavioral effect of a probabilistic functional crossover event. A peakier PDF would ensure that there is a greater probability of behaviorally similar (or identical) branches being crossed to compared to more dissimilar branches, thereby tuning down the behavioral effect of a crossover event. Conversely, a shallower PDF would increase the probability that behaviorally dissimilar branches were crossed, therefore increasing the average behavioral effect of a crossover event.

These strategy parameters could then travel with the nodes they are associated with during evolution, and in turn be mutated in the same manner that strategy parameters in evolutionary strategies [17, 7] control the mutation step size, and allow for adaptive mutation rates. This approach, ported to genetic programming via P-FXO, could pave the way to a more principled (i.e. less random) approach to altering data structures that do not take the form of a vector during evolution.

6. REFERENCES

[1] L. Beadle and C. Johnson. Semantically driven crossover in genetic programming. In J. Wang, editor, *Proceedings of the IEEE World Congress on Computational Intelligence*, pages 111–116. IEEE Press, 2008.

[2] J. Bongard and H. Lipson. Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Science*, 104(24):9943–9948, 2007.

[3] J. C. Bongard. A functional crossover operator for genetic programming. In R. Riolo, U.-M. O'Reilly, and T. McConaghy, editors, *Genetic Programming Theory and Practice VII*, pages 195–210. Springer, 2010.

[4] P. D'haeseleer. Context preserving crossover in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 256–261. IEEE Press, 1994.

[5] R. Fisher and J. Bennett. *The genetical theory of natural selection: a complete variorum edition*. Oxford University Press, USA, 1999.

[6] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[7] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[8] J. H. Holland. *Adaptation in Natural and Artificial Systems*. Michigan Press, Ann Arbor, MI, 1975.

[9] T. Jones. Crossover, macromutation, and population-based search. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 73–80. Morgan Kaufmann, 1995.

[10] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Boston, MA, 1992.

[11] W. Langdon, T. Soule, R. Poli, and J. Foster. The Evolution of Size and Shape. *Advances in Genetic Programming*, page 163, 1999.

[12] W. B. Langdon. Size fair and homologous tree genetic programming crossovers. 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 1092–1097. Morgan Kaufmann, 1999.

[13] M. A. Lones and A. M. Tyrrell. Enzyme genetic programming. In *Proceedings of the Congress on Evolutionary Computation*, pages 1183–1190. IEEE Press, 2001.

[14] P. Nordin, W. Banzhaf, and F. D. Francone. Efficient evolution of machine code for CISC architectures using instruction blocks and homologous crossover. In L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. J. Angeline, editors, *Advances in Genetic Programming 3*, pages 275–299. MIT Press, 1999.

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

[16] R. Poli and W. Langdon. Schema theory for genetic programming with one-point crossover and point mutation. *Evolutionary Computation*, 6(3):231–252, 1998.

[17] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer System nach Prinzipien der biologischen Evolution*. Fromann-Holzboog, Stuttgart, DE, 1994.

[18] G. Wagner and L. Altenberg. Perspective: Complex adaptations and the evolution of evolvability. *Evolution*, 50(3):967–976, 1996.