Subtree Crossover: Building Block Engine or Macromutation?

Peter J. Angeline
Natural Selection, Inc.
509 Colgate Street
Vestal, NY 13850
angeline@natural-selection.com

ABSTRACT

In genetic programming, crossover swaps randomly selected subtrees between parents. Recent work in genetic algorithms (Jones 1995) demonstrates that when one of the parents selected for crossover is replaced with a randomly generated parent, the algorithm performs as well or better than crossover for some problems. Jones (1995) termed this form of macromutation *headless chicken crossover*. The following paper investigates two forms of headless chicken crossover for manipulating parse trees and shows that both types of macromutation perform as well or better than standard subtree crossover. It is argued that these experiments support the hypothesis that the building block hypothesis is not descriptive of the operation of subtree crossover and that subtree crossover is better modeled as a macromutation restricted by population content.

1 Introduction

In genetics-based forms of evolutionary computation, which include genetic algorithms (Holland 1992; Goldberg 1989) and genetic programming (Koza 1992), crossover is regarded as the principal operation that keeps the evolutionary process in motion. The prominent role of crossover in creating solutions in these methods is supported by an operational principle called the *building block hypothesis* (Holland 1992; Goldberg 1989) which states that crossover juxtaposes smaller fit building blocks of parents to create larger fit building blocks in offspring. Mutation is often regarded as a minor operator in genetic-based systems and in some genetic programs is even excluded altogether (Koza 1992; and many others). Recent work in genetic algorithms and evolutionary programming has questioned the efficacy of crossover (Fogel and Atmar 1990; Eshelman and Shafer 1993; Fogel and Stayton 1994; and others).

Jones (1995) demonstrates that a macromutation with the mechanical form of crossover that substitutes a randomly constructed parent as one of the recombinants, performs as well as and occasionally better than crossover when clearly defined building blocks are not present. This macromutation, termed *headless chicken crossover*, had the mechanical form of crossover, i.e. the transfer of genetic material between two parents, but paired each population member with a randomly generated parent rather than a parent chosen from the population. Jones (1995) shows that on problems where well defined building blocks do not exist the macromutation performs better than the GA with crossover. The effectiveness of this operator further calls into question the range of problems for which crossover is well suited and suggests that macromutations are often sufficient to solve difficult problems.

Recently, a similar conclusion has been reached for genetic programs using subtree crossover. Angeline (1997) demonstrated that two headless chicken crossovers defined for subtrees performed equivalently to standard subtree crossover when compared on three problems. These experiments used a modified version of a non-standard genetic program defined in prior work (Angeline 1996a; 1996b). This genetic program included several subtree mutation operations that were previously shown to assist in solution discovery but are not widely used in the field. In addition, the size of the randomly generated subtrees in this study were limited to a range smaller than what would normally exist in the population, potentially providing an advantage to the macromutations, since those parents were consistently recombined with smaller trees. Angeline (1997) concludes that mutation operations can perform on par with subtree crossover and suggests that the building block hypothesis may not accurately describe the operational characteristics of subtree crossover.

The focus of the present study is to validate the experiments of Angeline (1997) without using subtree mutation operations and with randomly generated parents that are more reflective of the actual population contents. It is shown that both of the defined subtree macromutations consistently perform on par with subtree crossover for the tested problems and can perform significantly better for some problems. These experiments corroborate those of Angeline (1997) and demonstrate that the comparison of subtree crossover to macromutation is robust.

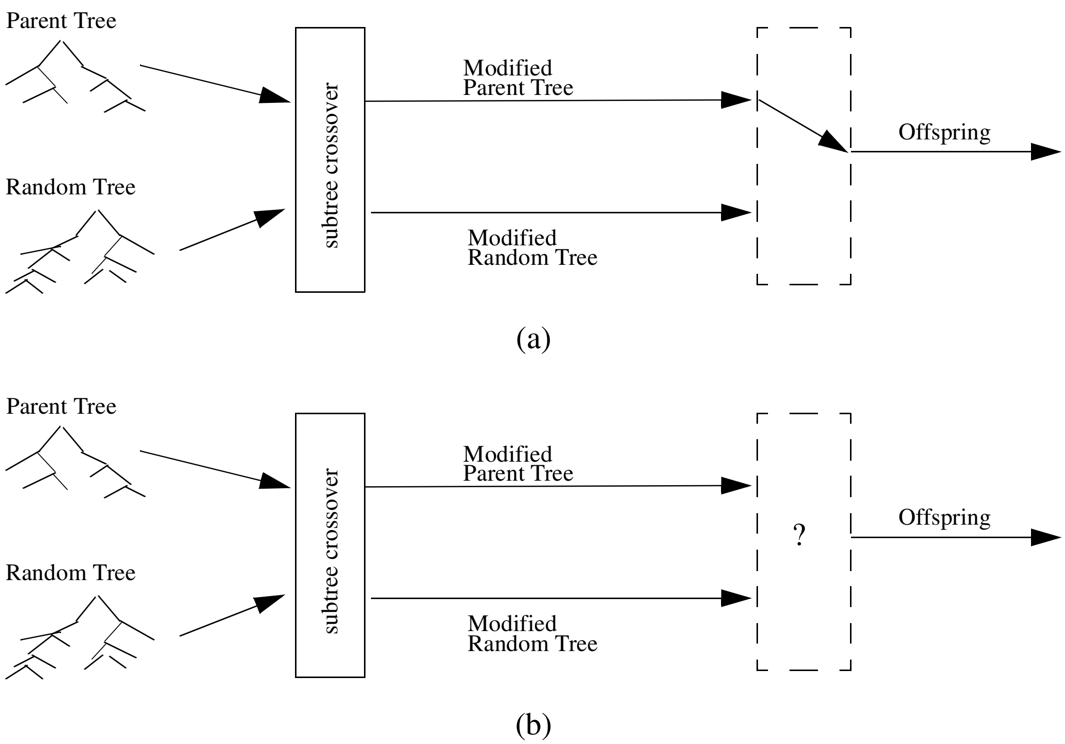


Figure 1: (a) Diagram of strong headless chicken crossover (SHCC). After subtree crossover with a randomly generated tree, the modified parent tree is selected to be the offspring for this parent. This is done twice per "mating", once for each selected parent. (b) Diagram of weak headless chicken crossover (WHCC). After subtree crossover, the modified parent tree and the modified random tree selected with equal probability for entering the population. This is done twice per "mating", once for each selected parent.

This paper begins with a review of the operators and experimental results described in Angeline (1997) in the next section. Section 3 describes the details of the two experiments and their results with Section 4 providing a discussion and the implications for genetic programming. An argument for the characterization of subtree crossover as a macromutation is also offered.

2 Strong and Weak Headless Chicken Crossovers

Subtree crossover was first described in Cramer (1989) and subsequently refined in Koza (1992). Given two parse trees as parents, subtree crossover selects a subtree at random from each parents and replace the selected subtree in one parent with the selected subtree from the other. In the spirit of Jones (1995), and following Angeline (1997), this paper defines two forms of macromutation that are mechanically identical to subtree crossover: *strong headless chicken crossover* (SHCC) and *weak headless chicken crossover* (WHCC). Schematics of both operations are shown in Figure 1.

Strong headless chicken crossover works as follows. For each of the two parents trees, SHCC generates a random tree to be its mate for the operation. Once the random parent is constructed, standard subtree crossover is performed on the given parent and its corresponding randomly generated counterpart. The operation is repeated for the second parent and its randomly generated tree. Once both subtree crossovers are completed, both modified parent trees, rather than the modified randomly generated trees, are returned.

Note that in most situations when SHCC is applied the modified parent tree will contain a relatively small amount of randomly generated material. This is the sense of the "strength" of this operation, that is, SHCC tends to preserve the non-randomly generated material from the parent tree in the offspring. Weak headless chicken crossover does not. WHCC proceeds exactly as SHCC except it has an even probability of returning either the modified parent tree or the modified randomly generated tree for each of the two offspring. This operation is considered "weak" since for almost half of the produced offspring, a relatively small amount of non-random material from a parent will be returned to the population.

It should be noted that SHCC is nearly semantically identical, but not mechanically identical, to the form of subtree mutation described in Koza (1992), an operator that has been discounted and even discarded in the genetic programming field. Much of the lack of interest in subtree mutation is due to the number of example problems solved in Koza (1992) that do not use this operator. Koza (1992, pp. 607-608) also performs an experiment that compares the performance of subtree crossover against this form of subtree mutation on 200 runs of the 6-multiplexor problem using a population size of 1000. In the runs using only subtree crossover 90% of the offspring were manipulated compared to only 10% of the offspring were manipulated in the runs using only subtree mutation. The remaining offspring were duplicates of the selected parents. Not surprisingly, the results show a significant advantage for subtree crossover alone over mutation alone with crossover solving the problem for all runs and mutation successful for only 8% of the runs. Koza (1992, pp. 608) concludes that crossover is very important for solving this problem and argues elsewhere (Koza 1995) that this experiment constitutes evidence counter to the hypothesis that subtree mutation can perform on par with subtree crossover. Obviously, the unequal frequencies in the application of mutation and crossover used in this comparison provides a significant advantage for subtree crossover. Any operator if only applied to 10% of the offspring in the course of a run would perform poorly against a comparable operator applied to 90% of the offspring. Consequently, this experiment does not adequately reflect the relative performance of subtree crossover and subtree mutation.

Using the headless chicken crossovers described above, Angeline (1997) demonstrated that when the frequency of application of subtree crossover and macromutation are equal, macromutation consistently performs as well or better than subtree crossover. Angeline (1997) investigated three problems: the 6-multiplexor problem, the intertwined spiral problem and the sunspot prediction problem. For each of these problems, both SHCC and WHCC performed on par with standard subtree crossover while SHCC was shown to be statistically significantly better than standard subtree crossover on the sunspot prediction problem. These experiments used a non-standard genetic program that includes a number of subtree mutations (distinct from the macromutation operations) that have been shown in previous work to provide advantages when evolving programs for complex problems (Angeline 1996a; 1996b).

There are two potential complaints with the experiments performed in Angeline (1997). First, some genetic programming purists may protest the comparison of standard subtree crossover against the headless chicken crossovers in a genetic program that includes a substantial commitment to subtree mutations, as is the case in the genetic program used in Angeline (1997). Most genetic programming aficionados rely only on crossover when evolving parse trees. They may feel that the introduction of subtree mutation operations may have hindered subtree crossover's performance. Second, In Angeline (1997), the randomly generated trees used in these operations were limited to no more than 15 nodes. By using only relatively small random trees for crossover partners in the macromutations, the dynamics of the evolving population, for instance the number of crossover operations rejected due to creating offspring that exceed size constraints, potentially favor the macromutations.

In the following experiments, both of the above potential criticisms are addressed. First, a genetic program that uses only subtree crossover is compared to genetic programs that rely only on one of the two headless chicken crossovers without the assistance of subtree mutation operations. Second, care was taken in this study to generate each random parent tree to have same number of nodes as the other parent that would have been the crossover partner if using standard subtree crossover. For instance, if the two parents passed into either the SHCC or WHCC operator contained 47 and 75 nodes respectively then the parent having 47 nodes would be crossed with a random tree having 75 nodes and the parent having 75 nodes would be crossed with a random tree having 47 nodes. This was done specifically to ensure that the experimental results were not affected by anomalies due to distinct distributions of parent tree sizes between the runs using the macromutations and those using subtree crossover.

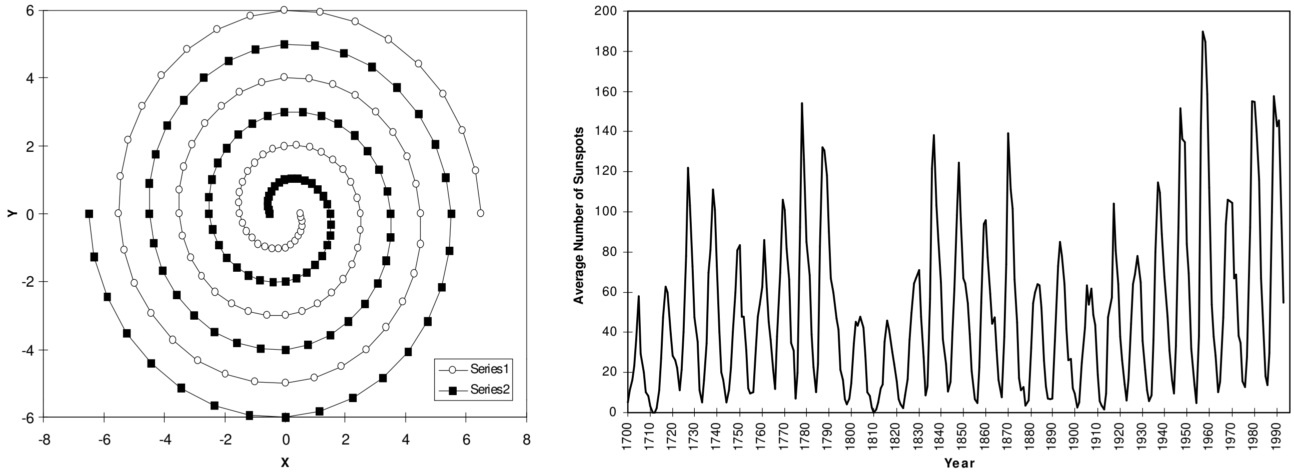
The selection and reproduction methods used in the experiments below are typical in evolutionary programming (Fogel 1995). In this method, the fitness of each population member is compared to *k* other population members selected at random and a point is assigned for each that is smaller. The population is then sorted by their point values with the top 50% reserved as parents for the next generation. Each offspring is generated by copying a parent and then applying the crossover operation associated with that experiment. Each parent is used to create exactly one offspring per generation.

3 Experiments

The following experiment compares the performance of standard subtree crossover against the two defined macromutations over three selected problems: 6-Bit Even Parity problem, the Intertwined Spiral problem and the Sunspot Prediction problem. These problems were selected to give a reasonable coverage of the various types of problems encountered in genetic programming. The 6-bit even parity problem is a boolean discrete problem while the intertwined spiral problem is discrete but requires a numerical language. Lastly, the sunspot prediction problem is a fully numeric problem. For each problem, a total of 30 trials per technique was performed. All resulting means were compared using Student's t-test to determine if there was a significance to the differences in mean performance.[1]

Jones (1995) was able to distinguish between standard crossover and headless chicken crossover for problems that contained explicit building blocks and those that did not. Headless chicken crossover performed as well or better on problems without building blocks but performed worse than standard crossover when building blocks were available. A

1. See Angeline (1996b) for an argument as to why this method should be preferred to a direct comparison of the computational effort statistic defined in Koza (1992).



(a) (b)

Figure 2: Data sets for two of the problems investigated in the study. (a) Data sets of the intertwined spiral problem. (b) Data set for the Sunspot prediction problem.

similar distinction in the problem set used in this experiment is not attempted since it is unclear what makes a problem difficult for a genetic program or even if the notion of a building block is valid (O'Reilly and Oppacher 1995). [1]

3.1 Problem Descriptions

The first problem investigated is the boolean 6-Bit Even Parity problem, a version of which is defined in [Koza 1992]. The function set used here for this problem is {*not, and, or*} and the terminal set is {*d0, d1, d2, d3, d4, d5*}. The object of the problem is to induce a function that returns TRUE when an even number of the terminals are TRUE and returns FALSE otherwise. This problem is known to be very difficult for genetic programming (Koza 1992; 1994). In the experiment described below, the number of generations is 500 as was the population size. The fitness of an individual is the number of correct responses for the 64 possible inputs. Due to the symmetry in this problem, both the actual function represented by the population member as well as the inverse of this function are rated with the individual's fitness assigned to be the greater of the two. Preliminary experiments suggested exploiting this symmetry improved the search results for this function set.

It is not expected that a population size of 500 run for 500 generations is sufficient to solve this problem for a reasonable percentage of attempts. A relatively small population is used to ensure that the problem is not completed to allow an in-progress comparison of the techniques. In Angeline (1997), the 6-multiplexor problem completed a significant number of times out of the 30 runs for each of the tested methods. This forced a comparison of the mean time to completion between the three techniques which is a statistic that is subject to high variance making statistical comparison difficult. Solving a harder discrete problem with a relatively small population to ensure the runs do not complete allows the mean performance to be compared which is a better behaved statistic.

The interlocking spirals problem is the second problem investigated. In this problem, a function must be learned that separates two classes of data that form interlocking spirals in the coordinate plane. Figure 2a shows the two data sets for the problem. Koza (1992) also investigated this problem and the form of this experiment follows the experiment described there. The function set is defined to be {*+, -, *, /, iflte, sin, cos*} and the terminal set is defined as {*x, y, RNT*} where *x* and *y* are the coordinates of the point to be classified and RNT is a *random numerical terminal*, which is identical to an ephemeral random constant (Koza 1992). For this experiment, the Gaussian noise typically added to RNTs (Angeline 1996a; 1996b) was turned off. The fitness of an individual is calculated as the number of points from the 194 given points in the two data sets correctly classified. The population size for this problem is 300 which is 50 more than that used in Angeline (1997). [2]

The third problem used in the experiments is a time series modeling problem. Since the year 1700, the average number of sunspots observed each month have been recorded. These values are referred to as the Wolfe Sunspot Data and are available on-line.[3] Figure 2b shows the average number of sunspots per year for the time period from 1700 to 1993. The

1. See O'Reilly and Oppacher (1995) for a definition of GP-schema and an attempt at a building block hypothesis for GP and its corresponding problems. See Punch, Zoingker and Goodman (1996) for a definition of a Royal Tree function similar to the Royal Road functions (Mitchell, Forrest and Holland 1992) used to investigate building blocks in GAs.

2. This was an oversight in the experimental setup. The intent was to make the experiments identical. This difference does not alter the validity of the experiment or the associated conclusions.

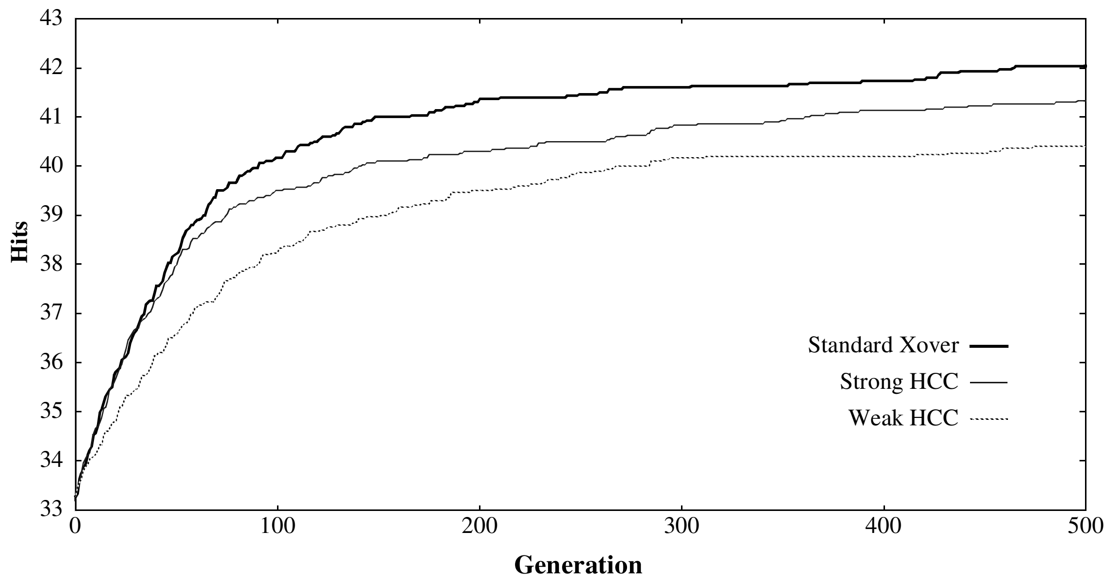


Figure 3: Graphs showing the performance of the three methods on the 6-Bit Even Parity problem. The graph plots the generation against the mean score of the best evolved individual over the 30 runs.

objective of this problem is to predict the number of sunspots that will be observed in the following year given previous data points.

Following (Oakley 1994), a function and terminal set were selected that befit a time series prediction problem. The functions used were {+, -, *, %, *sin, cos*} and the terminal set was {*x1, x2, x4, x8, RNT*} where *x1* is the number of sunspots observed in the year before the predicted year, *x2* is the number of sunspots observed two years before the predicted year and so on. It should be noted that the data was not separated into training and test sets making this a time series modeling task rather than a time series prediction task. Given that the objective is to minimize the prediction error of the evolved function, the fitness function was the sum of the squared prediction error for each of the data points (years 1708 to 1993). The actual fitness function negates the error to make this a maximization problem.

3.2 Experiment Results

Figure 3 graphs the performance of the three operators on the 6-Bit Even Parity problem. Each graph shows the mean best-of-generation score averaged over the 30 runs plotted against generation. It appears that early on SHCC and standard subtree crossover perform equivalently with WHCC lagging slightly behind. By generation 100, standard subtree crossover is slightly better than both SHCC and WHCC, with WHCC performing the worst. This relative ordering continues through to the end of the run. Table 1 shows the mean

Table 1: Mean Best-of-Run Results for the 6-Bit Even Parity Problem

Method	Mean Hits	Std
Subtree Xover	42.03	2.108
Strong HCC	41.33	2.073
Weak HCC	40.39	2.268

best-of-run results over the 30 trials for each of the methods investigated along with the corresponding standard deviations for the means. The table shows that each of the methods performed within a standard deviation of all others. A t-test between all pairings of techniques was performed to determine which if any of the differences in best-of-runs means were significant. While the differences in the mean generation of completion shown in the graph of Figure 3 and the values of Table 1 appear to slightly favor standard subtree crossover, none of the differences in performance were found to be statistically significant ($p > 0.05$ for all pairings).

Figure 4 graphs the mean best-of-generation score over 30 runs per generation for the various methods on the spiral problem. Once again, early in the runs, SHCC performs competitively with standard subtree crossover while WHCC lags behind. After about generation 250, all techniques appear to be performing equivalently with similar perfor-

3. The World Wide Web address for this data is http://www.ngdc.noaa.gov/stp/SOLAR/SSN/ssn.html.

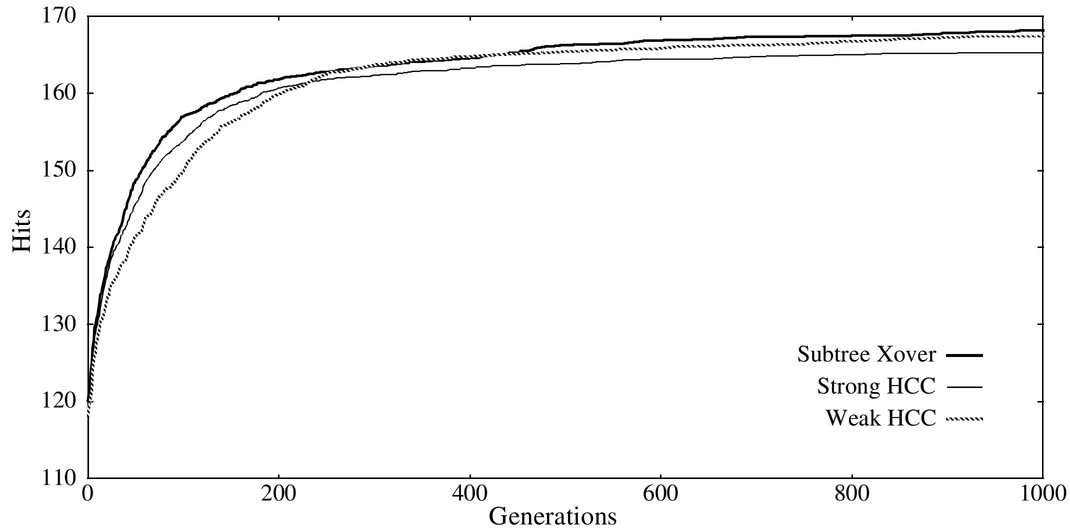


Figure 4: Graphs showing the performance of the three methods on the spiral problem. The graph plots the generation against the mean score of the best evolved individual over the 30 runs.

mance continuing through the end of the run. Table 2 shows

Table 2: Mean Best-Of-Run Scores for the Spiral Problem

Method	Mean Score	Std
Subtree Xover	168.13	8.939
Strong HCC	165.26	9.620
Weak HCC	167.43	10.190

the mean best-of-run score over the 30 trials for the spiral problem for each of the three methods along with the standard deviation for each mean. The mean scores are all fairly close after 1000 generations and well within a standard deviation of the other techniques. Consequently, t-tests performed between all pairings showed no statistical significance for the differences in the means ($p > 0.05$ for all pairings.)

Figure 5graphs the mean prediction error for the best-of-generation individual for the various methods investigated on the sunspot problem. As in the previous two problems, standard subtree crossover and SHCC perform equivalently early in the runs and then SHCC consistently out performs subtree crossover after generation 150. Likewise, WHCC performs worse than both of the other techniques early on and then surpasses subtree crossover at about generation 300 and performs consistently better for the remainder of the generations. Note that the performance of subtree crossover on this problems is relatively flat after generation 200 while each of the macromutations continue to improve. Table 3

Table 3: Mean Best-Of-Run Results for Sunspot Problem

Method	Mean Squared Prediction Error	Std
Subtree Xover	-84230.212	21343.375
Strong HCC	-64450.795	10384.571
Weak HCC	-60097.610	10214.841

lists the mean squared prediction error for the best-of-generation individuals for each of the methods along with the associated standard deviation. Note that the performance of subtree crossover is more than a standard deviation worse than the performance of both macromutations for this problem. T-tests performed between each pairing of methods showed that there was a highly statistically significant difference between the mean performance of subtree crossover and both macromutations favoring the macromutations ($p \ll 0.01$). T-tests performed between the macromutations showed no statistically significant difference.

4 Discussion

First, it should be evident from the experiments above that when the testing conditions are equivalent, macromutation of subtrees consistently creates solutions that are comparable to subtree crossover. In each of the experiments, the mean performance of both SHCC and WHCC was consistent with that of subtree crossover with SHCC and WHCC performing significantly better for one problem. These results, along with those of Angeline (1997) are a strong counter-example to the comparison experiment described in Koza (1992) and definitively demonstrates that under identical conditions, subtree macromutations can perform at least as well as subtree crossover and potentially better on some problems.

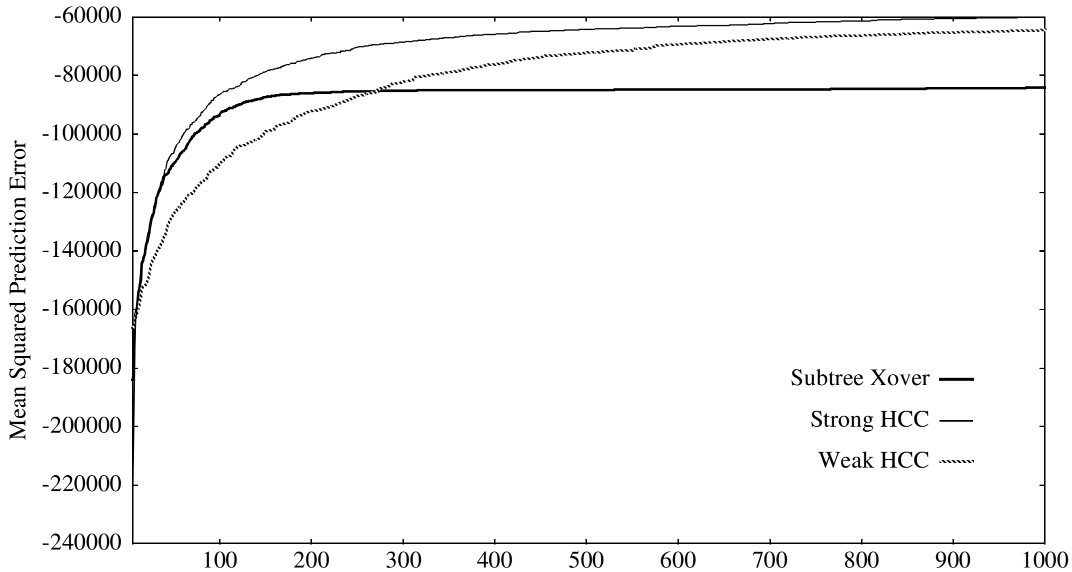


Figure 5: Graphs showing the performance of the three methods on the sunspot prediction problem. The graph plots the generation against the mean score of the best evolved individual over the 30 runs. The first 5 generations are omitted from the graph.

A second important observation is that WHCC, which introduces significantly more randomly generated material into a population than SHCC, performed well on all tested problems, both here and in Angeline (1997). This suggests that the continual infusion of random material into the population associated with this operator does not impair progress in the population. The characteristic lag in performance for this method at the beginning of a run compared to the other methods suggests a less exploitative search of program space, which is a reasonable conclusion given its operation.

It is not surprising that both macromutations performed so much better than subtree crossover on the fully numeric sunspot prediction problem. Finding good solutions for this problem is highly dependent on having appropriate numeric terminals in the population's trees. Without some form of mutation, subtree crossover is doomed to substandard performance on this problem since it is forced to use only a subset of the numeric constants introduced in the initial population. As the population evolves, the appropriateness and usefulness of the particular numeric constants in the population will change, often for the worse, forcing the genetic program to compensate by applying more functions to massage the value of the numerical terminals. At some point the inadequacy of the numeric terminals is so great that progress on the problem is halted, hence the flat performance after generation 200 for standard subtree crossover in Figure 5. The macromutations are not susceptible to this problem since they have the opportunity to introduce new numeric terminals with each application that may be more beneficial than the current numeric terminals. This effect of the waning usefulness of numeric terminals prompted the definition of the random numerical mutation used in Angeline (1996a; 1996b). Why this effect does not appear in the results for the spiral problem is most likely due to the fact that only the sign of the values produced by the evolved functions are scored in the calculation of fitness rather than the magnitude of the computed values.

The results here agree strongly with those of Angeline (1997) in spite of the differences in the experimental setup. This not only validates the results of the previous study but also shows the effect of macromutation performing competitively with subtree crossover as being robust across not only different problems but different genetic programs. This suggests that subtree crossover consistently performs like subtree macromutations. This raises the main question of this paper, namely is subtree crossover an engine for combining building blocks or really a macromutation that is limited by the content of the population? If it were a building block engine, then SHCC and WHCC should perform considerably worse than subtree crossover since they consistently introduce random material into the population rather than promote the proliferation of building blocks. In reality, as shown both here and in Angeline (1997), these macromutations consistently perform as well or better than subtree crossover. If the operation of selecting a crossover partner from the population is no better than the generation of a random crossover partner, then it may be that the population is just a source of random trees rather than a reservoir for parse tree building blocks. And in the case that randomly generating crossover partners is better than choosing partners from the population, then the population must be inhibiting the progress of the genetic program.

It is hard to imagine that subtree crossover could be anything other than a macromutation when the dynamics of the operator are scrutinized. Consider the operation itself and what would constitute a successful "building block." During subtree crossover, any subtree in the population can be inserted into any position of any other parent tree. Consequently, genetic programming "building blocks" would need to be reasonably effective in any number of distinct functional contexts in order to survive long enough and be prevalent enough in the population to be used effectively. In genetic algorithms, a particular building block fits into one and only one position in the genome. Its semantics are associated with the semantics of the positions it occupies in the string. Its presence in the population displaces competing building blocks by physically excluding them from those individuals it occupies. The reliance of position is so important in genetic algorithms that even when the ordered fixed-length restriction of the genome is removed, the intended position of a building block is still carried around to identify its semantics (e.g. Goldberg, Korb and Deb 1989; and others). In genetic programs, any cohesive subtree is unrestricted by position while its semantics are not fully defined until the context in which it is applied is known. Consequently, any subtree is a potential competing building block for any other subtree in the population, which is a recipe for anarchy rather than a ordered progression of building block composition. The lack of positional semantics is one of the distinguishing features of genetic programs (Angeline 1996c) and is the most compelling reason to discount subtree crossover's ability to be a building block engine.

One counter argument to this conclusion is that there may exist problems as yet untested where standard subtree crossover outperforms these macromutations. While this may be true, the mere existence of such a collection of problems does not necessarily suggest that subtree crossover operates as per a building block hypothesis. At best such a set of problems would only signify problems where the population restricts the distribution of the random subtrees available for crossover rather than maximize the presence of building blocks. The best class of problems to look for potential building blocks would be the boolean functions. Complex boolean functions often contain subfunctions that can be composed into the larger solutions. Such is the case with the 6-bit even parity function investigated here. Parity functions over smaller collections of bits can be composed into successful 6-bit even parity solutions. If there are parse tree building blocks, they would likely be evident while solving these problems.

5 Conclusions

Subtree crossover is not as uniquely suited to evolving parse trees as originally thought by practitioners of genetic programming. This paper investigated two forms of macromutation, mechanically similar to subtree crossover but semantically just variations of subtree mutation, that consistently performed as well or better than subtree crossover under identical experimental conditions. This suggests that subtree crossover may in effect be more accurately described as a population limited macromutation operation rather than a compositional operation consistent with the building block hypothesis.

At its core, the building block hypothesis for genetic algorithms characterizes schema processing by crossover as a linear progression from small building blocks to complete solutions. But the dynamics of a genetic program are much more opportunistic and non-linear while the definition of GP-schema, and hence GP building blocks, is problematic at best. Consequently, it maybe more accurate to operationally describe subtree crossover as a population limited macromutation rather than an engine for composing building blocks. Through additional experimentation, a model of subtree crossover's operation more accurate than the building block hypothesis should become evident.

Acknowledgments

Thanks are due to my former manager Rodger Fritz at Lockheed Martin Federal Systems in Owego NY who provided an excellent environment for this work to occur.

References

Angeline, P. J. (1994). Genetic Programming and Emergent Intelligence. In *Advances in Genetic Programming*, K. Kinnear (ed.), Cambridge, MA: MIT Press, pp. 75-96.

Angeline, P. J. (1996a). Two Self-Adaptive Crossover Operations for Genetic Programming. In *Advances in Genetic Programming: Volume II*, P. Angeline and K. Kinnear (eds.), Cambridge MA: MIT Press, pp. 89-110.

Angeline, P. J. (1996b). An Investigation into the Sensitivity of Genetic Programming to the Frequency of Leaf Selection During Subtree Crossover, In *Genetic Programming 1996: Proceedings of the First Annual Conference*, Koza, J., Goldberg, D., Fogel, D., and Riolo, R., (eds.), Cambridge MA: MIT Press, pp. 21-29.

Angeline, P. J. (1996c). Genetic Programming's Continued Evolution. In *Advances in Genetic Programming: Volume II*, P. Angeline and K. Kinnear (eds.), Cambridge MA: MIT Press, pp. 1-20.

Angeline, P. J. (1997). Comparing Subtree Crossover with Macro Mutation, In *The Sixth Conference on Evolutionary Programming*, P. Angeline, R. Reynolds, J. McDonnell and R. Eberhart (eds.), Springer-Verlag, In Press.

Bäck, T. and Schwefel, H.-P. (1993). An Overview of Evolutionary Algorithms for Parameter Optimization, *Evolutionary Computation*, 1 (1), pp. 1-24.

Cramer, N. (1985) A Representation for the Adaptive Generation of Simple Sequential Programs. In Proceedings of an International Conference on Genetic Algorithms and their Applications, J. Grefensttete (ed.), Hillsdale, NJ: Lawrence Erlbaum.

Eshelman, L. J. and Shaffer, J. D. (1993) Crossover's Niche, In *Proceedings of the Fifth International Conference on Genetic Algorithms*, S. Forrest (ed.), San Mateo, CA: Morgan Kaufmann, pp. 9-14.

Fogel, D.B. (1995). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, Piscataway, NJ: IEEE Press.

Fogel, D. B. and Atmar, J. W. (1990) Comparing Genetic Operators with Gaussian Mutation in Simulated Evolutionary Processes Using Linear Systems, Biological Cybernetics, 63, pp 111-114.

Fogel, D. B. and Stayton, L. C. (1994). On the Effectiveness of Crossover in Simulated Evolutionary Optimization, BioSystems, 32, pp. 171-182.

Fogel, L.J., Owens, A.J. and Walsh, M.J. (1966). *Artificial Intelligence through Simulated Evolution*, New York: John Wiley.

Goldberg, D.E. (1989) Genetic Algorithms in Search, Optimization and Machine Learning, Reading, MA: Addison Wesley.

Goldberg, D.E., Korb, B. and Deb, K. (1989). Messy genetic algorithms: Motivation, analysis and first results. *Complex Systems*, **3**, pp. 493-530.

Holland, J.H. (1992). *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: University of Michigan Press.

Jones, T. (1995) Crossover, Macromutation, and Population-based Search, In *Proceedings of the Sixth International Conference on Genetic Algorithms*, L. Eshelman (ed.), San Francisco, CA: Morgan Kaufmann, pp 73-80.

Koza, J. R. (1992). *Genetic Programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.

Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*, Cambridge, MA: MIT Press.

Koza, J. R. (1995). A Response to the ML-95 Paper Entitled "Hill Climbing Beats Genetic Search on a Boolean Circuit Synthesis Problem of Koza's," Unpublished manuscript.

Mitchell, M., Forrest, S. and Holland, J. (1992) The Royal Road Functions for genetic algorithms: Fitness landscapes and GA performance, In *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*. F. Verala and P. Bourgine (eds.), Cambridge, MA: MIT Press.

Oakley, E. H. N., (1994). Two scientific applications of genetic programming: stack filters and non-linear equation fitting to chaotic data. In *Advances in Genetic Programming*, K. Kinnear (ed.), Cambridge, MA: MIT Press, pp. 369-390.

O'Reilly and Oppacher (1995) The Troubling Aspects of a Building Block Hypothesis for Genetic Programming, In *Foundations of Genetic Algorithms 3*, L. D. Whitley and M. D. Vose (eds.), San Francisco, CA: Morgan Kaufmann, pp. 73-90.

Punch, W., Zognker, D. and Goodman, E. (1996) The Royal Tree Problem, A Benchmark for Single and Multiple Population Genetic Programming, In *Advances in Genetic Programming: Volume 2*, Angeline, P. and Kinnear, K. (eds.), Cambridge MA: MIT Press, pp. 299-316.

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*. Chichester, U.K: John Wiley.