

An Investigation Into the Selection Method of Crossover Points in a Genetic Program

John M. Hall, *Student, University of Idaho*

Abstract—This research assesses the use of six properties for use in crossover point selection methods. In order to do this, exhaustive crossover was performed between pairs of individuals after selected generations. This showed that the selected of crossover points is significant to the effectiveness of the crossover operation. It also showed that current methods used to select crossover points are not optimal. The results show strong correlations between several metrics and the percent of beneficial crossovers. These results provide a strong foundation for designing future crossover point selection methods.

Index Terms—Evolutionary Computation, Genetic Program, Crossover, Crossover Point Selection Method.

I. INTRODUCTION

GENETIC programming (GP) relies heavily on the crossover operation to improve the program trees. Unfortunately, no research has been done to investigate the effectiveness of the selection method used to choose crossover points. The current state of the art is somewhat arbitrary.

This research is important in order to improve the state of the art in crossover point selection methods. It is believed, but will not be addressed in this paper that this should extend to allowing genetic programs to be completed using fewer generations. This would allow larger problems to be solved using fewer computer resources. It is also believed that a good crossover algorithm will at the same time improve the diversity of the population.

The primary null hypothesis that will be investigated by this paper is that there is no benefit to other methods of selecting crossover points. In particular this is extended to include the six additional null hypothesis that there is no relationship between depth from the top of the tree, depth from the bottom of the tree, depth below, shortest depth below, percent of the tree depth, and percent of the tree contents from the selected crossover point.

II. BACKGROUND

There are several crossover point selection methods currently under research. This research looks at several of these methods. Random selection chooses nodes at random [3]. The 90/10 rule favors internal nodes. Depth fair crossover exchanges nodes of similar height [5]. Uniform crossover [15].

Random selection chooses crossover points completely at random. In a binary tree, this results in approximately 50% of the crossover points consisting of only a single leaf node. As the fan-out of the tree increases, so does this percentage.

Random selection makes it unlikely for the top nodes in a tree to be changed by the crossover operation. This is significant and may hinder evolution [18].

The current standard method for determining crossover points is known as the 90/10 rule. The 90/10 rule improves on the random crossover point selection method by requiring that 90% of the crossover points chosen must contain internal nodes and that only 10% of the crossover points consist of only a single leaf node. This results in 81% of crossovers having at least one internal node in each branch, 18% of crossovers where only one branch has internal nodes, and 1% of crossovers where both branches consist of only single leaf nodes. While this method does improve crossover by reducing the number of crossovers performed on the leaf level of the tree, it also does not make crossover selections toward the top nodes in the trees likely. As a result, the top of the program trees often become fixed.

Depth fair crossover selects branches of approximately the same size [5]. This is done in order to control code growth. The form of depth fair crossover selection discussed in this paper will only choose branches of exactly the same height. This is a slight simplification of the traditional depth fair algorithm. This simplification is not recommended for actual generational use. It is important to note that depth fair crossover is typically used not to improve the effectiveness of crossover, but instead to control growth.

In order to analyze crossover point selection methods, it is important to consider what defines a good crossover point selection method. The criteria presented here are not standard or rigorous. They are presented to understand the experimental criteria used below as well as to present limitations of this study. Crossover should

- 1) Improve the quality of the best individuals.
- 2) Maintain the overall quality of the population.
- 3) Be uniform over generations.
- 4) Be uniform over problems.
- 5) Maintain diversity.
- 6) Allow all possible recombinations.

The reasoning behind most of these criteria is simple. The first two specify that the population should in general improve over generations. The second two specify that the crossover operator should be general and not need specific domain knowledge. The final two criteria require crossover to allow the genetic program to assess a significant region of the fitness landscape. This paper only addresses the first three of these criteria.

III. EXPERIMENT

One genetic programming problem was used in this experiment. This program was the Santa Fe Trail. In general, this research used the standard definitions used for this problem. The Santa Fe Trail problem involves evolving programs to guide ants through a landscape containing food.

Each program was represented by an expression tree. The terminal nodes on this tree represented actions that a simulated ant could perform. There were three terminals, Forward, Left, and Right. As implied by their names, these actions would move or rotate the simulated ant. The non-terminals in the tree were the structure of a program. There were three non-terminals, If Food Ahead, Prog2, and Prog3. If Food Ahead will execute the commands on the left branch if there is food immediately in front of the simulated ant, and will execute the commands on the right branch if there is not food immediately in front of the simulated ant. Prog2 and Prog3 allow compound statements. Prog2 will execute the commands on the left branch followed by the commands on the right branch. Prog3 will execute the commands on the left branch, then the commands on the center branch, and then finally, the commands on the right branch. All non-terminals had three branches. If Food Ahead and Prog2 ignore the commands on the center branch. All valid solutions are constrained in that they must contain 500 or fewer nodes. They were also constrained to a maximum height of 20. These limitations were imposed to limit code growth as well as to effectively measure possible crossovers. Any solution that did not meet both requirements was eliminated from the population. It is interesting to note that the combination of both of these requirements makes it possible for both offspring of a crossover to be removed from the population. However, the probability of such an event is sufficiently small as to allow us to ignore it.

The fitness of each solution is calculated by actually running the program for the simulated ant. The ant is limited to 600 actions including Forward, Left, and Right. The simulated ant will immediately stop moving once this limit is reached. The fitness of each program is represented by an integer. The fitness is the number of food squares visited. Parsimony pressure is applied as a tiebreaker if two programs visit the same number of food locations.

The initial population of solutions is created randomly using a process of ramped half and half. Using this method, half of the trees are full and half are randomly grown. The maximum size of the tree cycles between 4, 5, and 6. The grown trees all have a non-terminal for the root node. At all other positions in the tree, there is an 80% chance of a non-terminal, and a 20% chance of a terminal. In all trees, both terminals and non-terminals are chosen uniformly from the appropriate node types.

The genetic program uses a generational model to update the population. This means, that in each generation, a new population is created from the current population. The population size was 200. The new population then replaces the current population. In order to guarantee that the best solution is never lost, a technique known as elitism is used. Two copies of the

best solution in a generation are added to the next generation. Offspring from the current generation are added until the generation is full. The offspring are created by selecting two parents. This selection is done using a process known as tournament selection. In this system, tournament selection chooses a parent by choosing the most fit of three randomly chosen individuals. Crossover and mutation are done to copies of the two selected parents. If either of these individuals contains more nodes than the maximum discussed above, it is discarded. The surviving offspring are added to the next population.

Mutation will always change a terminal to a terminal, and a non-terminal to a non-terminal. Since all non-terminals have three branches, this process may cause operative branches to become inoperative and vice-versa. Unfortunately, since there is a 1/3 chance that the mutation will operate on an unused branch, and there is a 1/3 chance that the mutation will change an operator to the same type, there is only a 4/9 probability of each mutation actually modifying a program.

The crossover selection method used is random. This was chosen to minimize the chance that the population at any given generation is biased due to the evolutionary process.

In previous experiments, a simple linear convergence random number generator was shown ineffective. As a result, the Mersenne Twister random number generator [11] was used to generate all random numbers.

The entire experiment was run for 50 trials. In each trial, a population of 200 individuals was evolved for 80 generations. From the initial population and from populations 1, 5, 10, 20, 40, and 80, twenty pairs of individuals were selected using the same selection method as used for the evolutionary process. On these twenty pairs of individuals, all possible pairs of crossover points were tried. The offspring from each pair of crossover points was evaluated. It should be noted that this large number of possible crossover points resulted in a large number of fitness evaluations. Mutation on the offspring was ignored. In addition, these individuals were not compared to the size or depth constraints used while running the generations. Each offspring program was evaluated as discussed above. Its resulting fitness was compared to its parent's fitnesses and was determined to be either a beneficial crossover, a neutral crossover, or a detrimental crossover. Those results were stored according to the following metrics about the crossover points selected:

- 1) *Depth* - distance to the top of the tree
- 2) *Reverse Depth* - distance to the lowest part of the tree
- 3) *Maximum Child Depth* - distance to the lowest part of the subtree.
- 4) *Minimum Child Depth* - shortest distance in the subtree to a leaf node.
- 5) *Percent of Tree Height* - the percentage down the tree of this selected point.
- 6) *Percent of Tree Count* - the percentage of the tree's nodes that are within this subtree.

These metrics were chosen because they are relatively easy to calculate so could be used by a crossover selection method. It also seems reasonable to believe that they may be related to effectiveness of crossover. The results of each crossover were

also stored based on which other methods could produce the crossover. Results for random, 90x10, and depth fair selection methods were stored.

Both offspring results for each crossover were viewed together. This allowed assessing how well the crossover impacted the population since all instances of fitness shifting were removed. This form of assessment is especially relevant for some forms of crossover such as depth fair crossover where the second point selected is limited by the first selection. Since depths were limited to 20 nodes, the first four tables were 20x20 results. The percentages were also scaled to be from 1 to 20. The first column represented 0%-5% and so forth. The tables created will allow statistically determining the number of successful crossovers that may be expected for every pair of crossover points matching the metric. Mathematically, this definition for a successful crossover, a neutral crossover, and a destructive crossover respectively is

$$f(o) > \max(f(p_1), f(p_2)) \quad (1)$$

$$f(o) = \max(f(p_1), f(p_2)) \quad (2)$$

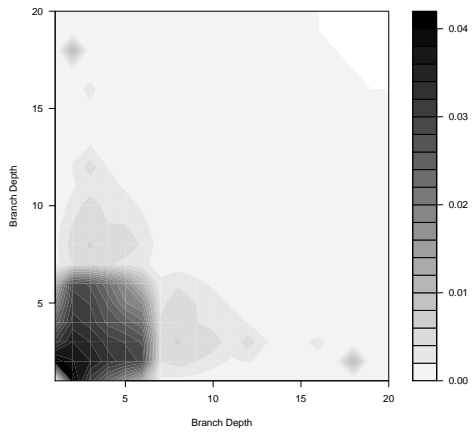
$$f(o) < \max(f(p_1), f(p_2)) \quad (3)$$

These definitions allow us to successfully assess the first crossover selection method criteria listed in the introduction. Specifically, in order to improve the population, the crossover must create an individual that is better than both parents.

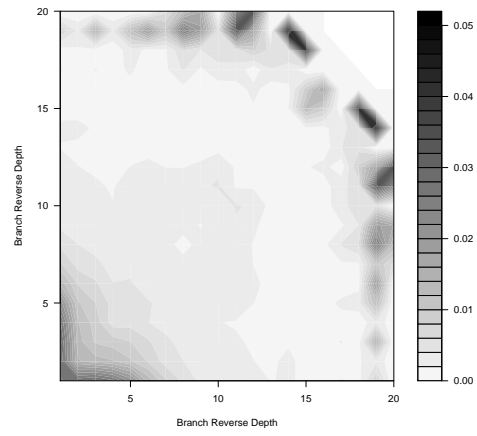
The code was optimized in several ways to allow the above test to complete in a reasonable time. Without these optimizations, the experiment was expected to take over one week on a 2 GHz workstation. The simplest and most beneficial optimization was to modify the parents themselves while testing the crossover. After evaluating both offspring, the crossover branches were moved back to their appropriate trees. Another optimization involved improving the performance of the evaluation routine. This optimization is described in [3].

IV. RESULTS

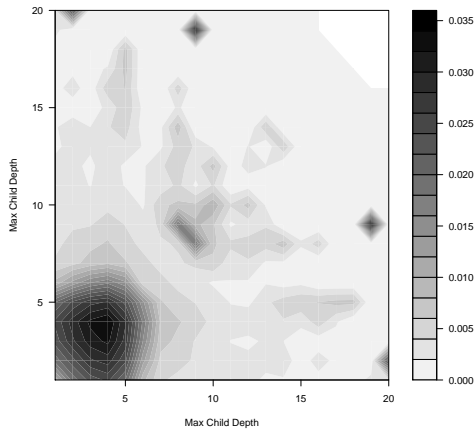
ame4(whe-355(er)-120ce)therJ 9.9s of -4(f91)-305(ran92c5(v)19r)-4119r 92c4.4119ragu(Sin)9rf9rallsdn39rair crossovro i.48 -19.5(The



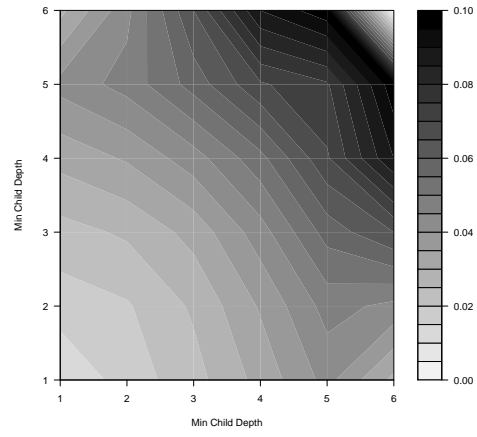
(a) Depth



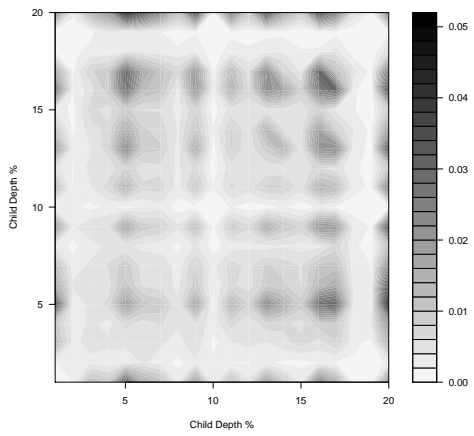
(b) Reverse Depth



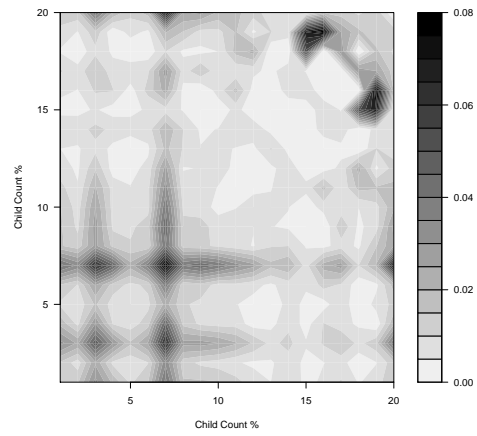
(c) Maximum Child Depth



(d) Minimum Child Depth

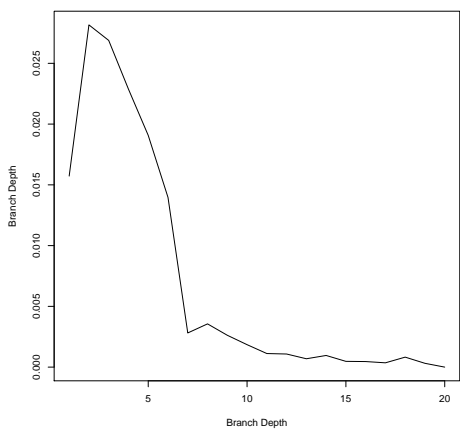


(e) Branch Depth Percent

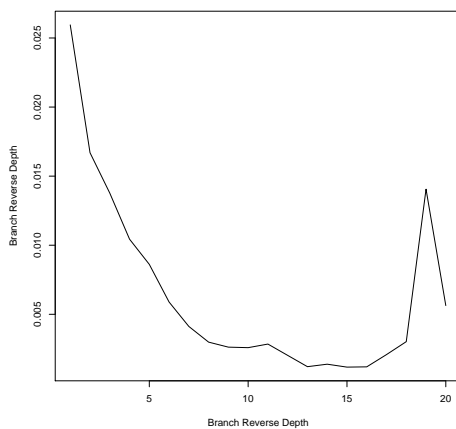


(f) Branch Count Percent

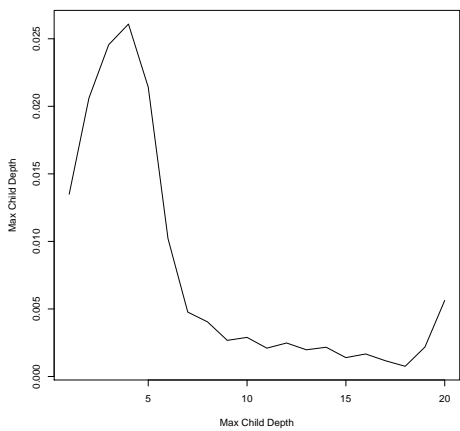
Fig. 1. Percentage of Beneficial Crossovers Over All Generations



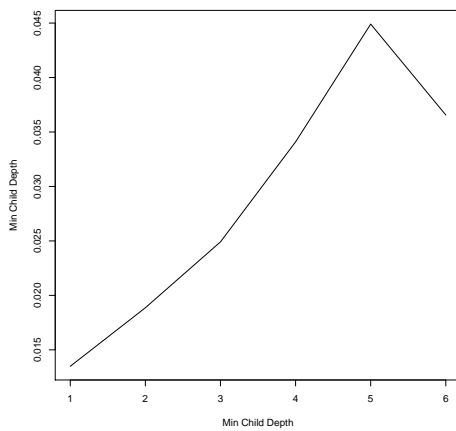
(a) Depth



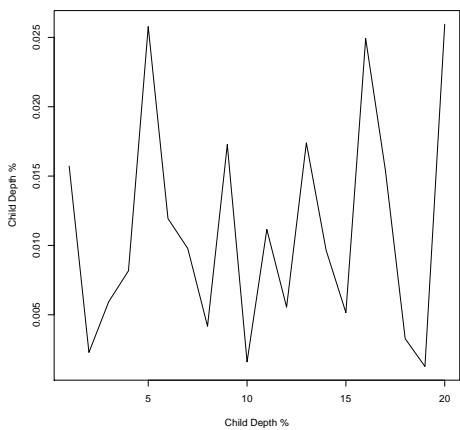
(b) Reverse Depth



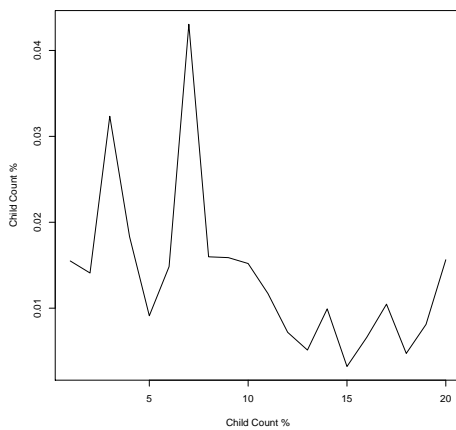
(c) Maximum Child Depth



(d) Minimum Child Depth

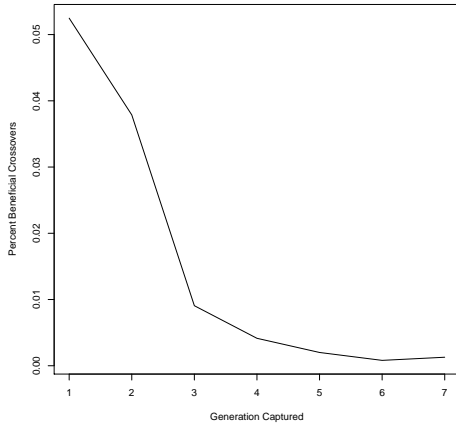


(e) Branch Depth Percent

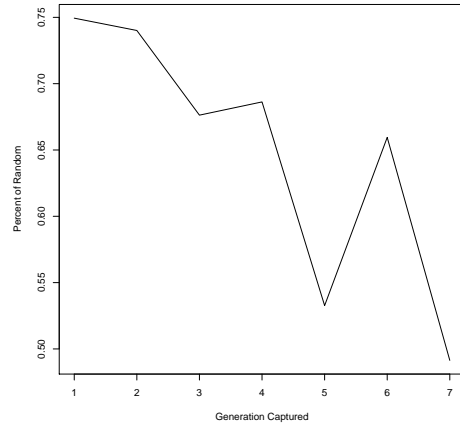


(f) Branch Count Percent

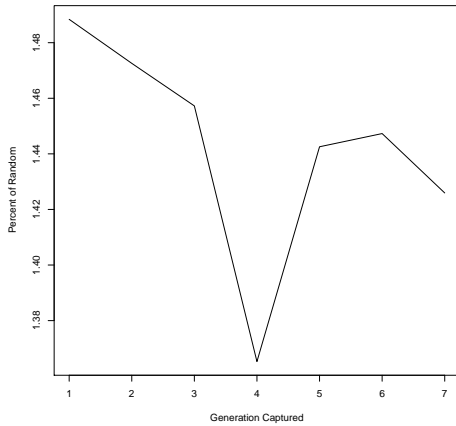
Fig. 2. Percentage of Beneficial Crossovers Over All Generations



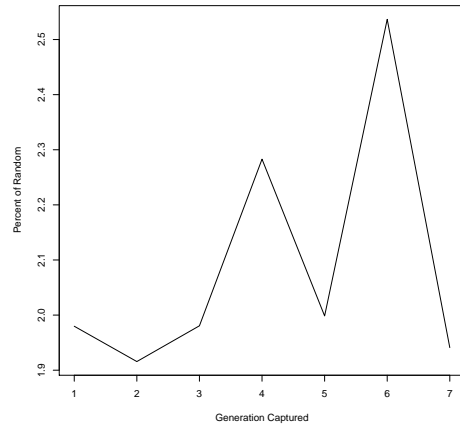
(a) Average Crossover



(b) Depth Fair Crossover (Scaled to Average)



(c) Ninety Ten Crossover Method (Scaled to Average)



(d) Test Crossover Method (Scaled to Average)

Fig. 3. Comparison of Crossover Selection Methods

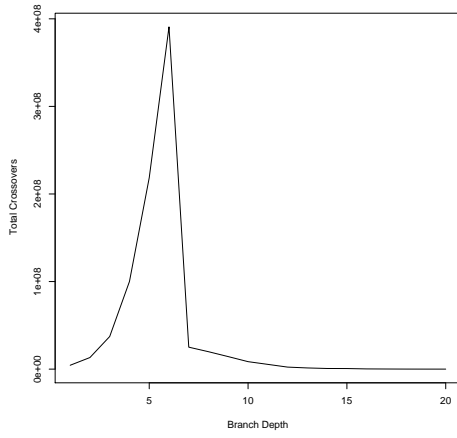
generational algorithm. If the drift hypothesis is true, then those programs that were too large may have skewed the results to increase our percentages.

The analysis so far has not dealt with criteria 2 from our introduction. The problem is in defining non-destructive. If the selection pressure is high, than disrupting a large portion of the population may be acceptable. However, if the selection pressure is low, then it may be easy to destroy the population. Without precise studies, it will be difficult to assess what percentages of crossovers must be non-destructive for the population to survive. In general, our results showed that especially in later generations a vast majority of crossovers were non-destructive. This would be explained well by the protective hypothesis regarding code growth. Though the results are not presented here, we believe any selection method considered above would not be overly destructive to the population.

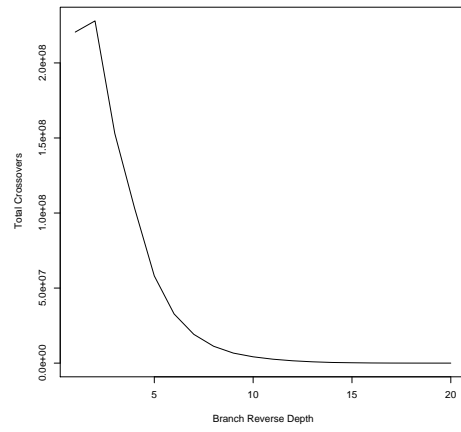
This exhaustive crossovers technique appears to be unique. It would appear to present some experimental methods of testing other genetic programming concepts. For example, the drift hypothesis could be tested by comparing the fitness to the size of each offspring reachable through crossover at any generation.

REFERENCES

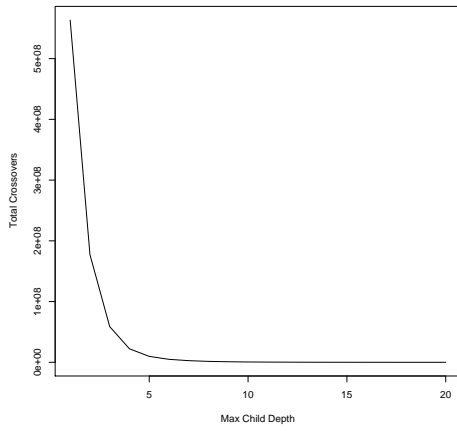
- [1] Bersano-Begey, Tommaso F., *Controlling Exploration, Diversity and Escaping Local Optima in GP: Adapting Weights of Training Sets to Model Resource Consumption*, Late Breaking Papers at the 1997 Genetic Programming Conference, John R. Koza ed, available at <http://citeseer.nj.nec.com/254220.html>.
- [2] Burke, Edmund, Steven Gustafson, Graham Kendall, Natalio Krasnogor, *Advanced Population Diversity Measures in Genetic Programming*, Seventh International Conference on Parallel Problem Solving from Nature, 2002, available at <http://citeseer.nj.nec.com/529057.html>.



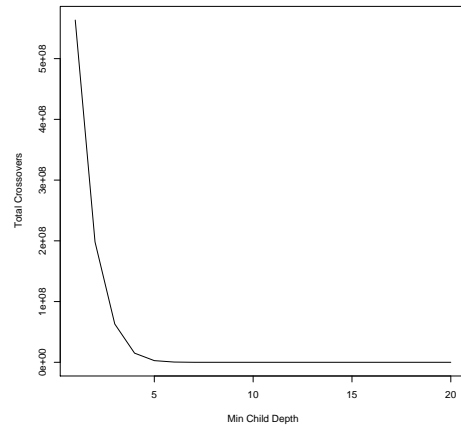
(a) Depth



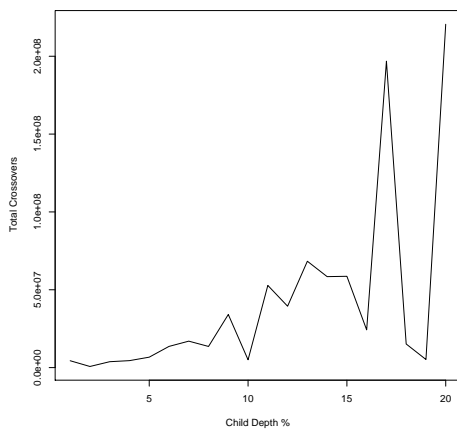
(b) Reverse Depth



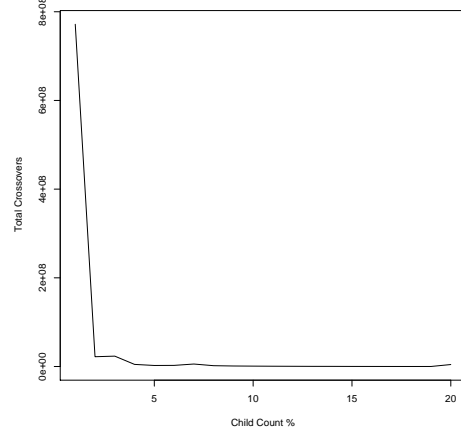
(c) Maximum Child Depth



(d) Minimum Child Depth

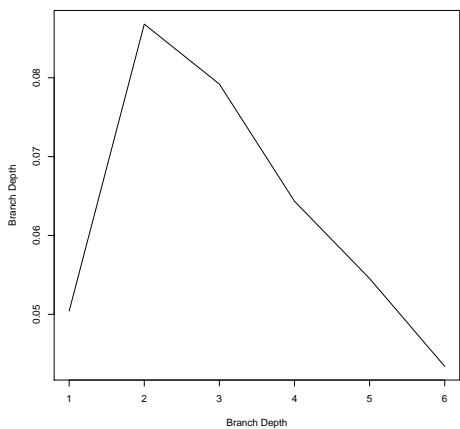


(e) Branch Depth Percent

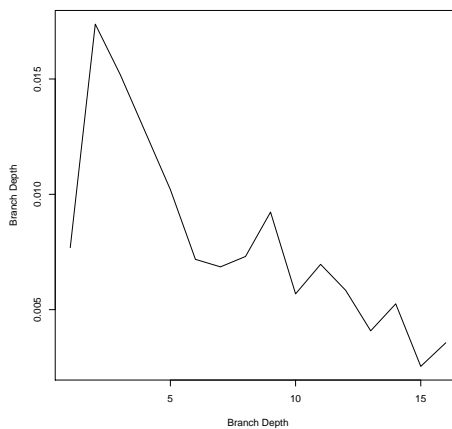


(f) Branch Count Percent

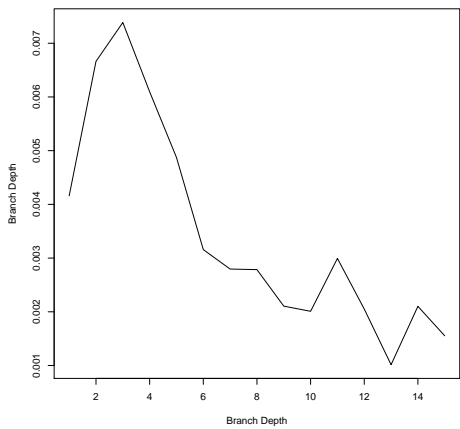
Fig. 4. Number of Possible Crossovers



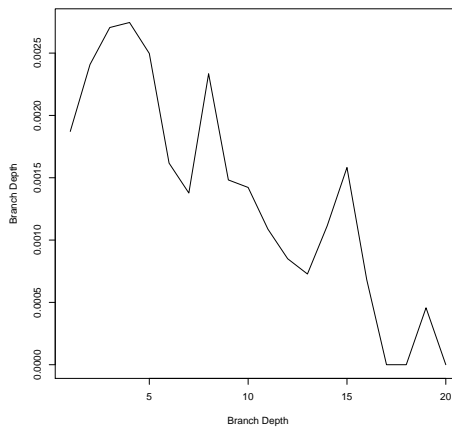
(a) Generation 0



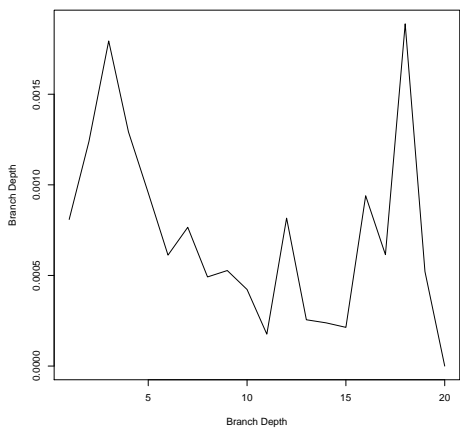
(b) Generation 5



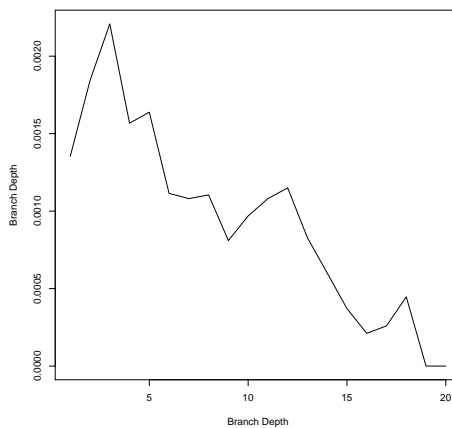
(c) Generation 10



(d) Generation 20



(e) Generation 40



(f) Generation 80

Fig. 5. Percentage of Beneficial Crossovers Based on Depth Over Time

- [3] Hall, John, *Using Genetic Programming to Create Smart Ants*, not published, 2003, available at <http://www.johnmhall.net/classes/evolution/Project2.tar.gz>.
- [4] Kantschik, Wolfgang, Peter Dittrich, Markus Brameier, Wolfgang Banzhaf, *Meta-Evolution in Graph GP*, 2nd European Workshop on Genetic Programming, May 1999, available at <http://citeseer.nj.nec.com/kantschik99metaevolution.html>.
- [5] Kessler, Matthew, and Thomas D. Haynes, *Avoiding Two-Bit Crossovers in Genetic Programming*, Proceedings of the 1999 ACM Symposium on Applied Computing, 1999, available at <http://citeseer.nj.nec.com/13724.html>.
- [6] Luke, Sean, *Code Growth Is Not Caused by Introns*, Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference, 2000, available at <http://citeseer.nj.nec.com/luke00code.html>.
- [7] Luke, Sean, *Two Fast Tree-Creation Algorithms for Genetic Programming*, IEEE Transactions on Evolutionary Computation, 4(3), 2000, available at <http://citeseer.nj.nec.com/luke00two.html>.
- [8] Luke, Sean, Liviu Panait, *A Survey and Comparison of Tree Generation Algorithms*, Proceedings of the Genetic and Evolutionary Computation Conference, 2001, available at <http://citeseer.nj.nec.com/440598.html>.
- [9] Mernik, Marjan, Matej Crepinsek, Viljem Zumer, *A Metaevolutionary Approach in Searching of the Best Combination of Crossover Operators for the TSP*, Proceedings of the IASTED international convergence Neural networks, May 2000, available at <http://citeseer.nj.nec.com/mernik00metaevolutionary.html>.
- [10] Mitchell, Melanie, *An Introduction to Genetic Algorithms*, The MIT Press, Cambridge Massachusetts, 1996.
- [11] Narasimhan, B., *The Mersenne Twister in Java*, February 2003, available at <http://citeseer.nj.nec.com/313273.html>.
- [12] Poli, Riccardo, Nicholas F. McPhee, *Exact GP Schema Theory for Headless Chicken Crossover and Subtree Mutation*, Proceedings of the 2001 Congress on Evolutionary Computation, 2001, available at <http://citeseer.nj.nec.com/502078.html>.
- [13] Poli, Riccardo, *General Schema Theory for Genetic Programming with Subtree-Swapping Crossover*, Proceedings of EuroGP, 2001, available at <http://citeseer.nj.nec.com/riccardo01general.html>.
- [14] Poli, Riccardo, W. B. Langdon, *On the Search Properties of Different Crossover Operators in Genetic Programming*, Genetic Programming 1998: Proceedings of the Third Annual Conference, 1998, available at <http://citeseer.nj.nec.com/poli98search.html>.
- [15] Poli, Riccardo, Jonathan Page, W. B. Langdon, *Smooth Uniform Crossover, Sub-Machine Code GP and Demes: A Recipe For Solving High-Order Boolean Parity Problems*, Proceedings of the Genetic and Evolutionary Computation Conference, 1999, available at <http://citeseer.nj.nec.com/501541.html>.
- [16] Soule, Terry, *CS472/572 GA Project*, April 2003, available at <http://www.cs.uidaho.edu/~tsoule/cs472/GPproj.html>.
- [17] Soule, Terence, James A. Foster, *Effects of code growth and parsimony pressure on populations in genetic Z. programming*, Evolutionary Computation, 6(4), 1998, available at <http://citeseer.nj.nec.com/296629.html>.
- [18] Soule, Terence, James A. Foster, John Dickinson, *Code growth in genetic programming*, Genetic Programming 1996: Proceedings of the First Annual Conference, July 1996, available at <http://citeseer.nj.nec.com/soule98code.html>.
- [19] Wang, Gang, Erik D. Goodman, William F. Punch III, *Simultaneous Multi-Level Evolution*, 2nd Online Workshop on Evolutionary Computation (WEC2), March 1996, available at <http://citeseer.nj.nec.com/wang96simultaneous.html>.

John M. Hall John graduated summa cum laude from Washington State University with a Bachelor's of Science in computer science in 1999. After graduating, John took a job in Boise with Hewlett-Packard. In the fall of 2002, Hewlett-Packard offered John a fellowship to continue his education at the University of Idaho. John expects to graduate with a Master's of Science in December of 2003. John's research interests include network security and evolutionary computing. John looks forward to returning to Boise to enjoy the ample rock-climbing and kayaking opportunities as well as to apply his recent education.