Using Context-aware Crossover to Improve the Performance of GP

Hammad Majeed
Department of Computer Sciences and Information Systems,
University of Limerick, Ireland.
hammad.majeed@ul.ie

Conor Ryan
Department of Computer Sciences and Information Systems,
University of Limerick, Ireland.
conor.ryan@ul.ie

ABSTRACT

This paper describes the use of a recently introduced crossover operator for GP, *context-aware crossover*. Given a randomly selected subtree from one parent, context-aware crossover will *always* find the best location to place the subtree in the other parent.

We examine the performance of GP when context-aware crossover is used as an extra crossover operator, and show that standard crossover is far more destructive, and that performance is better when only context-aware crossover is used.

There is still a place for standard crossover, however, and results suggest that using standard crossover in the initial part of the run and then switching to context-aware crossover yields the best performance.

We show that, across a range of standard GP benchmark problems, context-aware crossover produces a higher best fitness as well as a higher mean fitness, and even manages to solve the 11-bit multiplexer problem *without* ADFs. Furthermore, the individuals produced this way are much smaller than standard GP, and far fewer individual evaluations are required, so GP achieves a higher fitness by evaluating fewer and smaller individuals.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Performance

Keywords

Context Aware crossover, context, tree context, performance, destructive effects, standard crossover, one point crossover

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.

1. INTRODUCTION

Standard crossover for GP is widely accepted as being a destructive operator [2]. It is destructive because it exchanges sub trees without respect for their context, so, a subtree behaving constructively in one parent tree can have an adverse effect towards the fitness of the second parent. Many theories are put forth to combat this inherent problem of standard crossover. Most of these work by *preserving* the context of the exchanged subtree in the two parents, although Majeed and Ryan [7] have recently introduced a new *context-aware* crossover for GP which works by placing the subtrees-to-be-exchanged in the best context in the parents. It exhaustively explores all the possible positions at which the subtree-to-be-exchanged can be placed in the parent tree and then places it at the best position. Although this approach may sound quite expensive, it was shown that it resulted in dramatic fitness gains and evaluated considerably fewer individuals than standard crossover.

In the initial implementation of context-aware crossover no intention was given towards optimising its use, rather the emphasis was on demonstrating that using it in conjunction with standard crossover improved the performance of GP. We hypothesize that the adjustment of the *order* and the *time* of its application during a run can have a huge effect on the outcome of the run. This paper discusses different ways to employ it to get the best performance out of GP. The proposed methods are tested on different problem domains and the results obtained are very encouraging.

2. BACKGROUND

GP, in its traditional implementation (high crossover and low mutation rate), is driven by a combination of exploration and exploitation. Exploration happens during crossover, in which two parents are recombined to produce one or two offspring, while exploitation is achieved through a combination of selection and mutation. Selection ensures that the best newly produced offspring survive, while mutation allows for (relatively) minor tweaks of individuals, and a proper balance of parameters will get a population to evolve increasingly more fit individuals.

In a sense, GP has to make the best of a bad lot. Offsprings are randomly created, and there is no guarantee that they will even be as fit as their parents. However, the fact that GP manipulates a population of individuals means that there are usually enough offsprings produced in a generation to at least maintain fitness.

In the early stages of a run, GP is very productive, pro-

ducing increasingly more fit individuals, but, as time progresses, this exploration becomes more difficult. The trees tend to bloat, an effective measure against being disrupted by crossover, so minor changes become increasingly more difficult, and, as noted in [2], the majority of crossovers result in offspring that are inferior to their parents.

There have been a number of previous attempts to improve the performance of GP by improving crossover. This is a reasonable approach, motivated by the hypothesis that if selection is given a better (in the sense of more fit) set of individuals to choose from, the performance of GP will improve. The question, of course, is how to produce these better individuals. [7] contains a detailed description of several approaches. Most of these approaches [3][4][5] were context preserving, in that they tried to ensure that a sub tree being swapped into a parent was being used in the way it was in the original parent. The reasoning behind this is that changing the context of a sub tree is likely to be more disruptive.

Quite apart from the issue of how to define the context of a sub tree, these methods are hamstrung by the fact that (especially early on in a run) there may exist a different and *better* context for a subtree.

Altenberg in his "soft brood selection" method [1] generated a brood randomly and introduced the best of the brood in the next generation by holding a tournament. Tackett refined this idea and used the cheap "culling function" [9] to identify the best of the brood. He introduced the best two offsprings in the next generation.

Other approaches [8][10] try to choose good subtrees to swap, or good crossover points, depending on the approach, by measuring the contribution of the subtree that is to be replaced to the overall fitness of an individual. However, it isn't straightforward how to measure this contribution, as it isn't always clear how to replace a subtree. [6] demonstrated a way to neutralise individual subtrees, so that their contribution could be accurately measured. However, this approach could only be applied to problems with functions that can be canceled out using *identity* functions, such as Boolean problems and problems with function sets such as simple binary arithmetic.

3. CONTEXT-AWARE CROSSOVER FOR GP

In general, the above methods reduce the exploratory power of GP. The search is constrained either by reducing the different contexts into which a subtree can be swapped, or by reducing the number of subtrees that can be chosen for exchange. A different approach, termed *Context-aware crossover* was introduced in [7]. Context-aware crossover operates by finding the best possible context for a randomly chosen subtree in the new child. This is achieved by swapping the subtree into every possible position in the second parent, and evaluating each resulting child.

Although intuitively, this may sound expensive, it was shown that the resulting increase in performance permits the use of dramatically smaller populations, so that the total number of individuals evaluated decreased. Furthermore, because it is far more difficult for individuals to bloat with this type of crossover, the individuals tend to be much smaller.

Figure 1 shows the operation of this crossover. To make the figure more readable we have set tree-depth to five for this example.

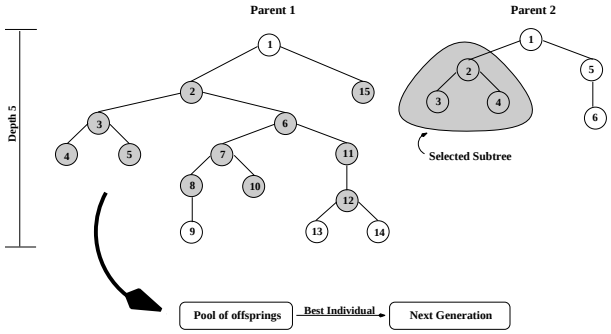

Figure 1: Context aware crossover. The shaded nodes in parent-1 are possible crossover points where selected-subtree from parent-2 can go in. All generated children are evaluated and the best selected

Two parents are selected for crossover as normal, and crossover cannot take place at the root node. In this example, node 2 of parent 2 is selected *randomly* as a crossover point (the subtree is shown shaded in parent 2). Next, all possible valid offspring are generated, that is, all those individuals that are within depth limits, etc. Each of these individuals is then evaluated, and the best one introduced to the next generation. Only *one* individual per crossover is entered into the following generation.

The selection of the crossover points is followed by generation of a pool of offsprings and their evaluation. The best individual among them is introduced into the next generation.

In the initial paper on this work, the probability of using standard crossover and context-aware crossover was varied as follows:

$$context\_aware\_prob = curr\_gen/max\_gen \quad (1)$$

$$standard\_xover\_prob = 1 - context\_aware\_prob \quad (2)$$

The motivation behind this is that it has been shown that standard crossover is most effective in the early part of runs [2], and is least effective in the later stages. As noted in the initial paper, no effort was made to optimise the rate at which the probabilities varied, although it was shown that, when compared to standard crossover, this mixed method tended to evaluate *fewer* individuals to get the same fitness.

This paper is concerned with examining how one should vary these probabilities, as well as determining whether it is better to only make available one operator at a time, rather than a combination of them as in the previous work. We will examine performance in terms of fitness attained, the total number of individuals evaluated, and the size of these individuals.

4. OPTIMISING GP'S PERFORMANCE

We performed two initial tests to analyse the settings used initially. First, the rate at which the probabilities change over the course of a run is examined, and second, the assumption that having both crossover operators available is tested. We hypothesise that standard crossover can actually

undo the work done by context-aware crossover because it is inherently more destructive, so the order of application of crossover operators (throughout a run) could be crucial.

For this paper we have selected the same set of problems used in the initial introductory paper of context-aware crossover [7] to have a better performance comparison.

4.1 Varying order and rates of crossovers

To find the optimal setting for the probabilities of the crossovers, a number of experiments were conducted. Due to space restrictions we will only discuss the standard Koza's Quartic Polynomial Symbolic Regression problem in this section. Other problems showed the same trend, and the results presented here are representative of the others.

For these experiments, a population size of 200 was allowed to evolve for 50 generations. No mutation was used, and the only variation operators were the standard and context-aware crossover operators. The initial population was generated using ramped half and half method with initial tree sizes varying from 2-6, and the maximum tree depth was set to 17.

$$exp^{\frac{curr\_gen^{slope}}{curr\_gen - max\_gen}} \quad (3)$$

After a careful study equation 3 was generated for varying the crossovers rates. The behavior of this exponential is highly dependent on the `slope` variable. By adjusting it, it can behave as a linear, exponential or combination of exponential and linear functions. For this paper, the slope variable was set to 0.3, 0.6 and 0.9 to vary the rate of change of crossover probabilities. The slope with a value of 0.3 has an exponential rise at the latter part of a run whereas slope with a value of 0.9 shows behavior similar to a linear polynomial. Slope with a value of 0.6 lies between the two extremes. The fitness graphs using only standard and context-aware crossovers are also shown to compare the destructive effects of standard crossover.

In the first set of experiments, the probability of context-aware crossover was varied from one to zero using eq 3, referred to as `context_aware_prob`, while the probability of standard crossover varied from zero to one using equation "$1 - context\_aware\_prob$".

In the second set of experiments, the probabilities of the crossovers were reversed, i.e., the probability of the standard crossover, `standard_prob` was varied from one to zero using equation 3, and the context-aware crossover was varied from zero to one using equation "$1 - standard\_prob$".

Apart from varying the crossover rates these setups show the importance of the order of the application of these crossovers. In the first setup, context-aware crossover was applied before standard crossover while in the second setup it was applied after standard crossover.

Results are averaged over 50 runs and plotted in Figure 2. It is clear from the graphs that activation of context-aware crossover during a run results in a significant improvement in mean fitness of the populations. The graphs using standard crossover with high probability and after the use of context-aware crossover (`P_0.3_inv`, `P_0.6_inv` and `P_0.9_inv`) show performance lower than the performance of standard and context-aware crossovers alone. It is a clear indication of the fact that standard crossover has lost most of the syntactic and semantic improvements done previously by the context-aware crossover.

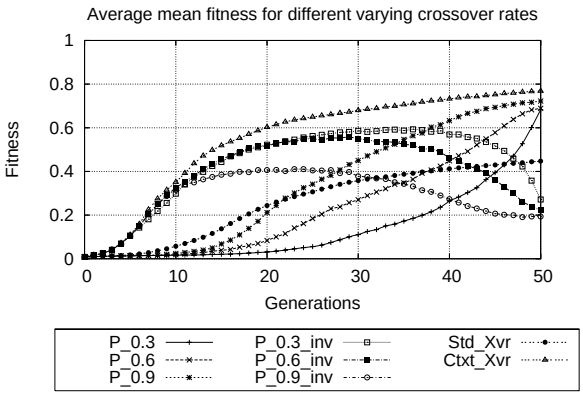

Figure 2: Mean average fitness of the quartic polynomial symbolic regression problem with varying crossover rates of context-aware and standard crossover operators. `Std_Xvr` and `Ctxt_Xvr` show the performance of only standard and context-aware crossovers.

These experiments give us enough evidence to conclude that standard crossover has to be applied before (if at all) the application of the context-aware crossover to get better performance.

4.2 Effect of standard crossover on performance

Although it was established in the previous section that the use of standard crossover after context-aware crossover lowers the mean population fitness, there is still a need to quantify this destruction along with the proof that this drop is *only* due to standard crossover, and not some unforeseen interaction between the two crossovers. To prove this another set of experiments was conducted. The standard Koza's Quartic Polynomial Symbolic Regression, the 11-bit multiplexer and the lawnmower problems were used for these experiments. Population sizes of 200, 50 and 100 were used for the regression, the multiplexer and the lawnmower problems respectively. For all experiments, the maximum number of generations was set to 50, the initial population was generated using ramped half and half method with initial tree sizes varying from 2-6, and the maximum tree depth was set to 17. The populations during a run were generated using either context-aware or standard crossover, not *both* at a time. Standard crossover was only applied in regular intervals of five generations, while the rest of the time context-aware crossover was used.

Results were averaged over 50 runs and the mean population fitness is plotted in figure 3. The standard crossover operator is consistently destructive in all the problems barring the multiplexer problem. The multiplexer problem is an exception because of two reasons. i) it has an inherent property of producing relatively fitter random populations compared to other two problems used. ii) It is difficult to drop below the fitness of the random population during a run. Notice, although there are huge drops in the graph, context-aware crossover has still managed to improve the mean population fitness by evaluating comparatively more number of individuals. Notice that `Ctxt_Xvr` (Figure 2) and `Regression` (Figure 3) converges to the same final fitness

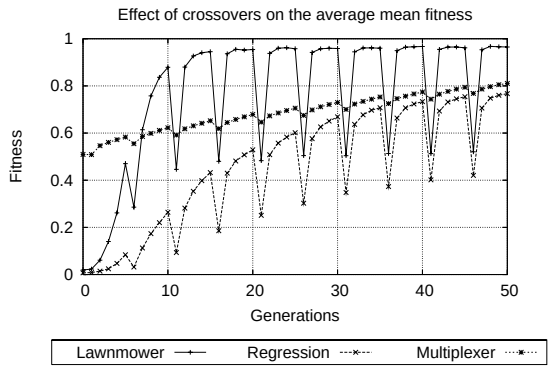

Figure 3: Destructive effects of standard crossover on GP performance when applied at regular intervals.

value but `Cxt_Xvr` evaluates 23% less individuals and generates 50% smaller individuals (not shown) to achieve it. Same argument holds true for the best fitness values of the two setups (not shown due to space restrictions). This is enough evidence to deduce that standard crossover is acting as a liability and context-aware crossover has to put in an extra effort to undo the damages done by standard crossover before imporving the fitness of the popualtion.

These experiments along with the experiments conducted in section 4.1 tell us that only one crossover operator should be used for population generation, and that standard crossover should be used (if any) *before* context-aware crossover during a run.

5. EXPERIMENTAL SETUP

The results from the previous section address the question of when context-aware crossover should be turned on. Clearly, once it is turned on, it should stay on, and be the only crossover operator available, but it isn't clear that standard crossover should never be used. To investigate this, same problems and setups as used in the previous section are employed here.

These are also the problems employed by [7], and here we use the same experimental setups. All the experiments were run for 50 generations and averaged over 50 runs. For each problem, two sets of experiments were conducted.

The first set comprised of ten experiments, each conducted by turning on context-aware crossover after completion of certain percentage of a run. For this paper intervals of 10% were used starting from 0% to 90%. For example, if the context-aware crossover was turned on after 10% completion of a run (labeled `P_10` in the figures to indicate the **P**ercentage of the run completed), then the first 10% of the run was completed by using only standard crossover and the rest of the run was generated by only using context-aware crossover. This setting is in accordance with the rules laid out in the section 4. These experiments will tell us the best point to turn on context-aware crossover to get optimal performance with minimal effort (evaluations) and how sensitive GP is to when it gets turned on.

The second set of experiments was a control group, comprising of the experiments presented in [7]. These involved the solving of the above problems using standard GP (labeled as `StdGP` in the figures) and context-aware crossover with varying rates of crossovers using equations 1 and 2 (labeled as `P_var` in the figures to indicate the probability varies throughout the run). For this paper, the maximum number of generations are reduced to 50 which was 100 in [7]. As equations 1 and 2 show that P_var experiments are dependent on the maximum number of generations therefore these experiments were rerun with the new generation value.

As in [7], we used oversized population for the standard GP experiments, which helped us to better compare the evaluations count for standard and context-aware crossovers. In addition, to complete this study we also checked the performance of context-aware and standard crossovers with small and normal sized populations respectively. Even with small populations, context-aware crossover performed better than standard crossover where they both had the same evaluations count, and it is well known that standard GP performs better with an oversized population. Therefore, the use of oversized population for standard GP is an additional help to its performance. For the sake of brevity we will only discuss the results obtained with normal and oversized populations for context-aware and standard crossovers respectively.

Note, to facilitate reading, in the following sections the term `context-aware` crossover will be used to refer to the new way of using context-aware crossover, i.e. switching on context-aware crossover after x% completion of a run, not the way it was used in the original implementation [7]. The original implementation of context-aware crossover is only used for `P_var` experiments.

The number of individuals evaluated by each `P_var`, `StdGP` and `P_x` can vary dramatically at different stages of a run, therefore, it seems reasonable to compare the performance of the setups in terms of number of evaluations done by each setup.

For each problem, four different graphs were plotted after completion of the runs. A brief description of each is as follows:

The Average Best and Average Mean Fitness graphs show the comparison of the best and average performances of the different setups.

The Running Evaluation Count graph keeps track of the cumulative sum of the evaluations done per generation. This is a good measure for calculating the expensiveness of a setup, because the context-aware crossover operator produces a variable number of offspring each generation.

The Program Size graph shows the average number of nodes per tree generated by each setup. The expensiveness of a setup is directly proportional to the tree sizes generated by it. Smaller trees need less effort to evaluate due to fewer nodes.

5.1 Quartic Polynomial Symbolic Regression Problem

To show the performance of context-aware crossover on the symbolic regression problem domain, Koza's Quartic Polynomial Symbolic Regression problem ($x^4 + x^3 + x^2 + x$) was employed, and a population size of 200 with fitness proportionate was used. To make comparisons fair between the standard setup and others an oversized population of 4000 was used for the standard setup. The function and terminal sets used were $\{+, -, \div, \times, \sin, \cos, \log, \exp\}$ and $\{x, \Re\}$

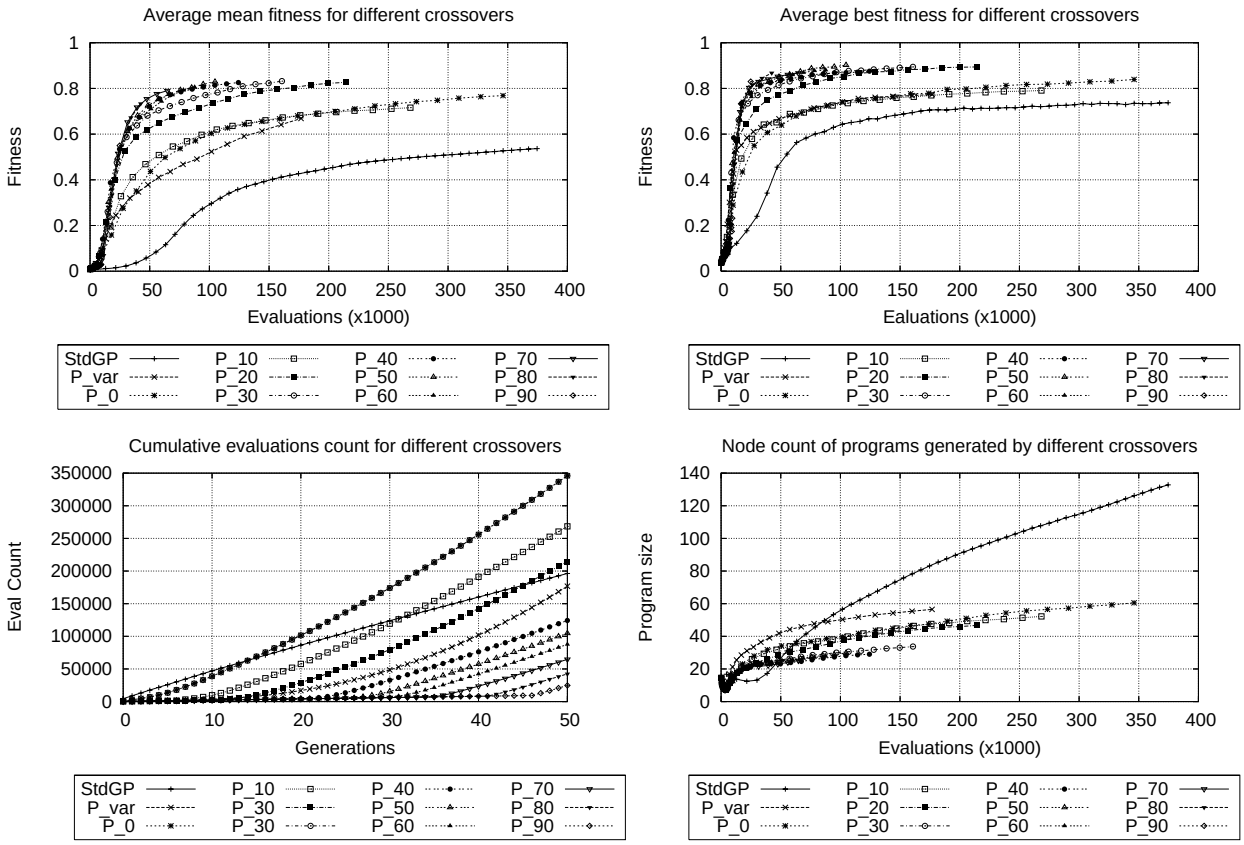

Figure 4: Performance comparison for the Quartic Polynomial Symbolic Regression problem. "StdGP" and "P_var" show the performance of standard GP and varying the probabilties of the two crossover operators, while the P_x curves show the performance of the system when context-aware crossover is turned on after completion of x% of the run.

respectively. $\Re$ is the commonly used ephemeral random constant. The averaged results are shown in figure 4.

Figure 4 (Top-Left) shows the average mean population fitness. Context-aware crossover, in general, has shown an exponential gain over `P_var` and `StdGP` due to the absence of standard crossover. Surprisingly, early use of context-aware crossover results in the low final fitness value (`P_0` and `P_10`). One plausible reason for this behavior is the presence of unhelpful genetic material in the initial generations. This is because context aware crossover is greedy, in that, although, the best possible use is made of the subtrees in the population, there is no guarantee that the best subtrees will get a chance to emerge as quickly as with standard crossover. The delayed use of context-aware crossover improves the fitness instantly and relatively cheaper. For example, `P_70` after 50,000 evaluations shows 100% and 860% performance gain over `P_var` and `StdGP` respectively.

Figure 4 (Top-Right) shows the average best population fitness. The trends shown by all the plots are similar to the ones shown by the plots of the figure 4 (Top-Left), and they indicate that the system shows an improvement in best fitness as well as mean fitness. By examining figure 4 (Top-Right) and figure 4 (Top-Left), one can identify the best point to turn on context-aware crossover for this problem. As the plots suggest the use of it should be delayed as much as possible. Note, although `P_50` attains the best final fitness (0.87) but it turns out to be significantly expensive, therefore we will not advocate its use and think that the setups with the delayed use of context-aware crossover can attain the same performance cheaply if are allowed to evolve for a few more generations.

Figure 4 (Bottom-Left) compares the cumulative evaluations counts for all the setups. The cumulative counts of `P_30` and beyond are much less than the cumulative counts of `StdGP` and `P_var`. These dramatically smaller counts have also resulted in significantly higher fitness values.

Figure 4 (Bottom-Right) compares the program sizes generated by different setups. The use of context-aware crossover has not only increased the system's performance cheaply but managed to *contain code bloat* to a great extent. Again, context-aware crossover has generated significantly smaller individuals than `P_var` and `StdGP` setups, showing that not only are there fewer trees evaluated, but those that are evaluated are substantially smaller than with standard GP.

The peformance and program size graphs for this problem tell us that context-aware crossover should be used after 80% completion of a run.

5.2 11-Bit Multiplexer Problem

To check the performance of context-aware crossover on hard problems, we tried to solve the 11-bit multiplexer problem *without* using ADFs. This is known to be a very difficult

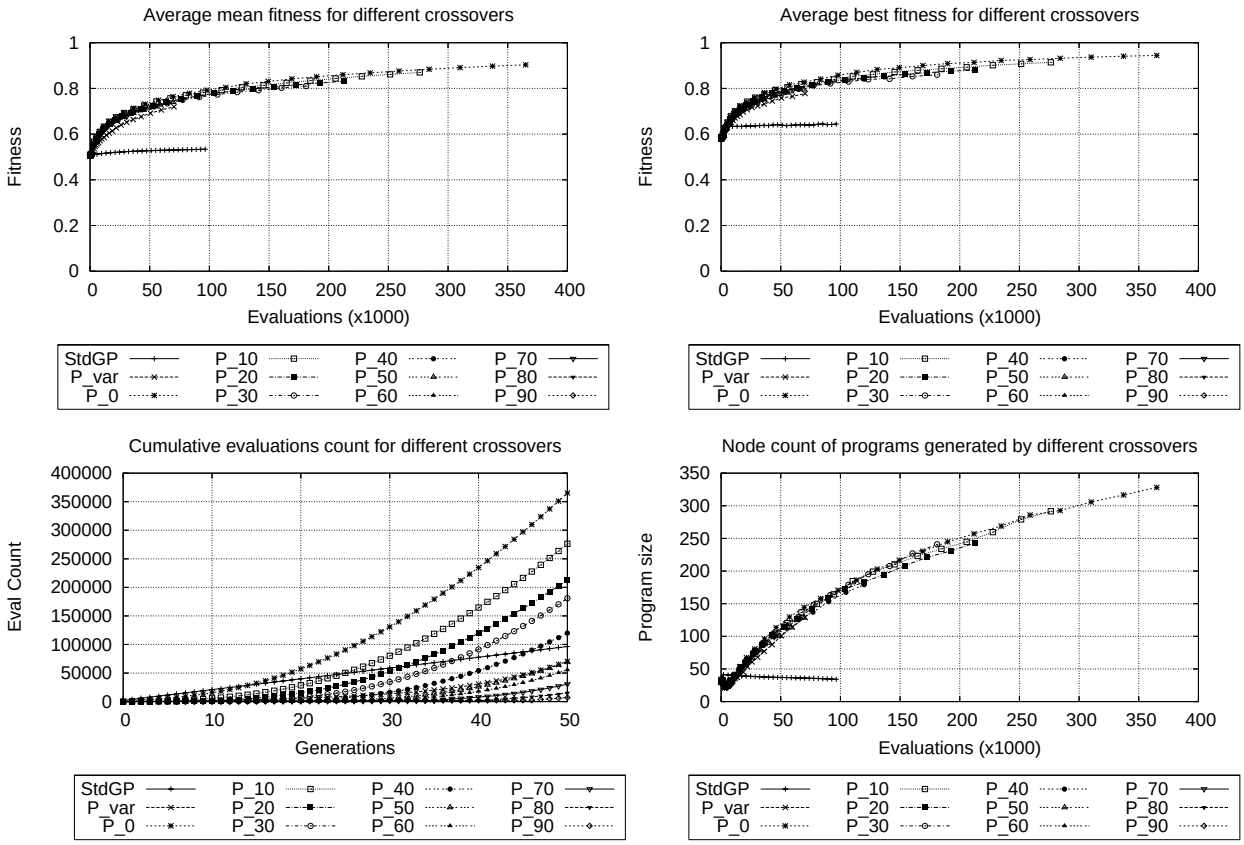

Figure 5: Performance comparison for the 11-bit multiplexer problem.

problem for GP when ADFs are not available, and standard non-ADF based GP implementation could not solve the 6-bit and higher multiplexer problems. A population size of 50 and fitness proportionate selection method for selecting parents were used. To make comparisons fair between the standard setup and others, an oversized population of 2000 was used for the standard setup. The function and the terminal sets were defined as $\{and, or, nand, nor, not, if\}$ and $\{a0, a1, a2, d0, d1, d2, d3, d4, d5, d6, d7\}$ respectively. The averaged results are shown in figure 5.

Figure 5 (Top-Left) compares the average mean population fitness values of different setups. The results obtained by using context-aware crossover are slightly better than `P_var` and `P_0` converges to the maximum fitness value 0.9 after an extended evolution. Only plausible reason behind difference in the performance of `P_var` is the use of standard crossover as it is mostly acting as a liability. For this problem, only use of context-aware crossover throughout the run results in the best performance as standard crossover has nothing significant to contribute towards the fitness of the population, this fact is also evident from the flat plot of `StdGP`. As in [7], `StdGP` failed to show any sign of improvement throughout the run.

Figure 5 (Top-Right) shows the comparison of the average best fitness values. We were able to *solve* this problem using context-aware crossover, even though the system did not employ ADFs, while standard GP is unable to solve a simpler version of the problem (6-bit) without ADFs. The perfect solution was found on *three* different independent runs and most of the runs with the `P_0` setup finish with fitness values higher than *0.95*. For this problem it is a good strategy to *always* and *only* use context-aware crossover through out a run.

Figure 5 (Bottom-Left) compares the cumulative evaluations count generated by the different setups. The runs using context-aware crossover, in general, have achieved the maximum fitness relatively cheaply compared to standard GP. Consider figure 5 (Top-Right), after 70,000 evaluations, `P_0` is showing 7% and 31% performance gain over `P_var` and `StdGP` respectively..

Figure 5 (Bottom-Right) shows the program sizes generated by each setup. Interestingly, all the setups using context-aware crossover generated huge trees when attempting to improve the population fitness. This seems to be a requirement for the improvement in the absence of ADFs, as all the setups with relatively high fitness values generated relatively larger trees. This indicates that, although context-aware crossover promotes small trees in general, it is still capable of producing large ones when required. The programs generated by standard crossover diminish in size with time. We believe that ramped half and half initialization method generated relatively larger trees with many subtrees acting adversely on the fitness, standard crossover got rid of them with time but couldn't improve upon them due to the complexity of the search landscape.

5.3 Lawnmower Problem

The Lawnmower problem is an example of a scalable prob-

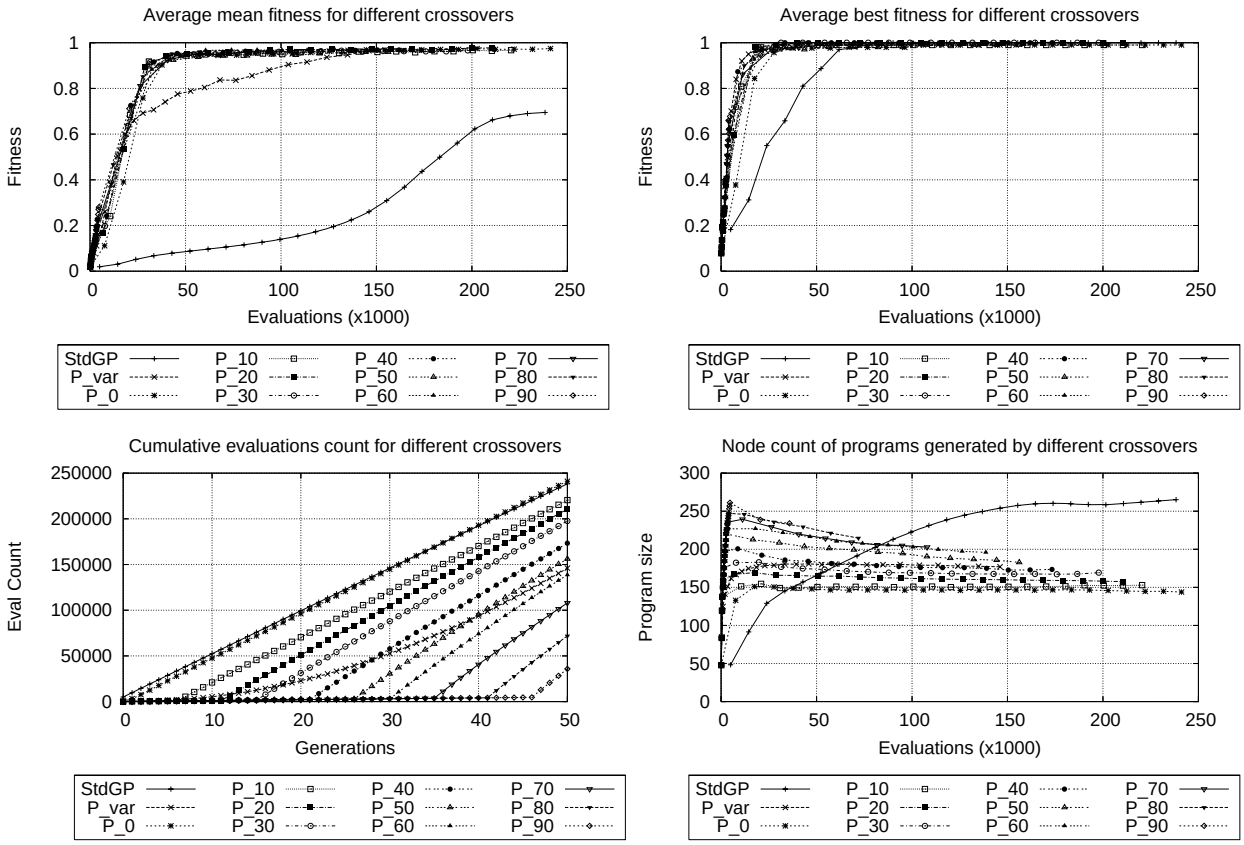

Figure 6: Performance comparison for the Lawnmower problem.

lem, which has been solved both with and without using ADFs by Koza. In this paper, to show the performance of context aware crossover on ADF related problems, we tried to solve it using ADFs. For this problem, a population size of 100 and tournament selection with size seven were used. For a fair comparison with the standard setup, a population size of 5000 was employed for the standard setup. Using Koza's standard implementation, two function defining branches (ADF0 and ADF1) and one result producing branch (RPB1) were used. The results are shown in figure 6.

Figure 6 (Top-Left) shows the fitness comparison of the setups employed. This is clearly a relatively easy problem, as all the setups perform fairly well. Barring `StdGP`, they all attain the same maximum fitness value by the end. The context aware crossover setups show an *exponential* rise in fitness and reach the maximum value as soon as the context-aware crossover is turned on.

Figure 6 (Top-Right) shows the plots for the best performance of the setups. All the context-aware setups attain the maximum fitness value by evaluating less than 20,000 individuals, while `StdGP` turns out to be quite slow and evaluates 75,000 (3.75 times more) individuals to achieve the same fitness value.

Figure 6 (Bottom-Left) compares the cumulative evaluations count generated by the different setups. Interestingly, the behavior of the cumulative counts is different from the ones shown by the aforementioned problems. In the previous problems the cumulative counts were growing with a positive derivative while for this problem they are growing with a constant derivative value. The change in this behavior can be understood by inspecting the size of the individuals generated over time (look at Figure 6 (Bottom-Right)). Unlike other problems, the use of standard crossover for this problem in the early stages of a run has generated bloated trees and context-aware crossover after activation used these bloated parent trees to generate significantly high number of offsprings due to availability of increased number of the crossover points. The evaluation of these individuals resulted in higher number of evaluations count. The drop in the tree sizes in the latter part of the run results in sluggish growth of the evaluations count over time.

Figure 6 (Bottom-Right) shows the program size generated by the various setups. The use of standard crossover in the early stages of the run results in bloated tree, while the use of context-aware crossover in the latter stages of the run results in deletion of the dead code (inactive nodes) and in drop of the program size. The use of context-aware crossover in the early stages of the run does not result in code bloat which is evident from the *flatness* of the `pre-P_30` plots.

Although, all the setups using context-aware crossover are performing quite well but the setups with its delayed use are performing slightly better than the rest.

6. DISCUSSION

The proposed context-aware crossover has performed consistently well on different problem domains. The experimental results have confirmed the effectiveness of context-aware

crossover. It has performed significantly better than the P_var and the StdGP setups on all the problems examined and was able to reduce the destructive effect of the standard crossover. Additionally, it improved the performance dramatically without incurring high costs in terms of numbers of individuals evaluated.

On the Quartic Polynomial Symbolic Regression problem the delayed use of context-aware crossover has not only resulted in the best performance but did so quite cheaply. This is an evidence of the fact that context-aware crossover acts as a local-search operator and improves upon the individuals evolved by standard crossover.

In the case of the 11-bit multiplexer problem we were able to solve it on *three* different occasions. This is an impressive result on a very hard problem, particularly as GP fails to solve a simpler version of the problem (the 6-bit multiplexer) when using a similar configuration, that is, without ADFs. The use of standard crossover had an insignificant effect on the out come of the run and most of the improvements were done by employing context-aware crossover.

Finally, the Lawnmower problem is proven to be an easy one due to the use of ADFs and all the experiments, including those using standard crossover, have shown good performance. Although, all the setups involving context-aware crossover have performed quite well but the ones using it bit late are performing marginally better than the others.

To summarize, the new setup has shown a dramatic gain in the performance on all the problems using relatively less computational effort and by generating quite small programs. The results also show that any use of context-aware crossover is good for the system as long it is used after the use of standard crossover.

7. CONCLUSION & FUTURE WORK

This paper discusses different ways to use context-aware crossover to get an optimal performance. We have recorded an exponential gain in the performance and a dramatic reduction in the computational effort during a run by *only* adjusting its *order* and *time* of application.

We have shown that using both standard and context-aware crossover at the same time during a run can be counterproductive, because most of the time standard crossover acts destructively and requires an extra effort from the context-aware crossover to improve the fitness of the population. This extra effort results in additional evaluations. In most cases, the use of context-aware crossover only has resulted in a huge performance gain using less computational effort.

We have also shown that context-aware crossover should be used *after* the use of standard crossover. As the use of standard crossover after the use of context-aware crossover can results in the loss of all the semantic and syntactic improvements previously achieved by context-aware crossover.

In many cases the delayed use of context-aware crossover during a run can result in a dramatic drop in the evaluations count without effecting the performance adversely. This is because GP benefits from having some access to standard crossover in the initial generations, which is then later exploited by context aware crossover.

In general, we would advise practioners wishing to use context-aware crossover to turn it on as late as possible for the problem on which standard crossover shows a performance improvement, for others, only context-aware crossover should be used throughout the run.

Currently we only introduce a single offspring for each two parents chosen for crossover, even though a substantial number of them may be produced. Introducing more will certainly lead to faster convergence, but a balanced approach will be required to prevent this convergence from being premature.

We are also looking at the ways to select a subtree intelligently from the first parent tree, which will help us to increase the performance of the system.

8. REFERENCES

[1] Lee Altenberg. The evolution of evolvability in genetic programming. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 3, pages 47–74. MIT Press, 1994.

[2] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, dpunkt.verlag, January 1998.

[3] Patrik D'haeseleer. Context preserving crossover in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 256–261, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[4] S. Hengproprohm and P. Chongstitvatana. Selective crossover in genetic programming. In *ISCIT International Symposium on Communications and Information Technologies*, ChiangMai Orchid, ChiangMai Thailand, 14-16 November 2001.

[5] Takuya Ito, Hitoshi Iba, and Satoshi Sato. Non-destructive depth-dependent crossover for genetic programming. In, *Proceedings of the First European Workshop on Genetic Programming*, volume 1391 of *LNCS*, pages 71–82, Paris, 14-15 April 1998. Springer-Verlag.

[6] Hammad Majeed. A new approach to evaluate GP schema in context. In *Genetic and Evolutionary Computation Conference (GECCO2005) workshop program*, pages 378–381, Washington, D.C., USA, 25-29 June 2005. ACM Press.

[7] Hammad Majeed and Conor Ryan. A less destructive, context-aware crossover operator for GP. In *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 36–48, Budapest, Hungary, 10 - 12 April 2006. Springer.

[8] Riccardo Poli and William B. Langdon. On the search properties of different crossover operators in genetic programming. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 293–301, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[9] Walter Alden Tackett. *Recombination, Selection, and the Genetic Construction of Computer Programs*. PhD thesis, University of Southern California, Department of Electrical Engineering Systems, USA, 1994.

[10] Chi Chung Yuen. Selective crossover using gene dominance as an adaptive strategy for genetic programming. Msc intelligent systems, University College, London, UK, September 2004.