# On the Constructiveness of Context-Aware Crossover

Hammad Majeed
Computer Science & Information Systems
University of Limerick,
Limerick, Ireland
hammad.majeed@ul.ie

Conor Ryan
Computer Science & Information Systems
University of Limerick,
Limerick, Ireland
conor.ryan@ul.ie

## ABSTRACT

Crossover in Genetic Programming is mostly a *destructive* operator, generally producing children worse than the parents and occasionally producing those who are better. A recently introduced operator, *Context-Aware Crossover*, which implicitly discovers the best possible crossover site for a subtree has been shown to consistently attain higher fitnesses while processing fewer individuals.

It has been observed that context-aware crossover is similar to *Brood Crossover* in that multiple children are produced during each crossover event. This paper performs a thorough analysis of these crossover operators and compares the performance of the two and demonstrates that, although they do work similarly, context-aware crossover performs a far better sampling of the search space and thus performs much better.

We also demonstrate that context-aware crossover benefits from a speed up of almost an order of magnitude when using a simple and very small cache, which is over two orders of magnitute *smaller* than caches typically used.

## Categories and Subject Descriptors

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

## General Terms

Performance

## Keywords

context-aware crossover, context, constructive, crossover, cache, fitness

## 1. INTRODUCTION

In the standard implementation of GP, the recombination operator is considered to be a major driving force behind the success or otherwise of a GP run. Many variants of the recombination operator are introduced and used by different researchers, but the most commonly used one is the one point crossover operator due to its simplicity and ease of implementation. This simplicity has a huge toll on its performance and it acts destructively most of the time during the run.

Many researches have tried to improve its performance. The most commonly adpated one is the "context preserving" approach [2][3][4]. In it, the order and number of the parent nodes of a swapped subtree in its container parent tree are preserved to the best possible extent in the other parent. The reasoning behind this is that changing the (syntactic) context of a subtree is likely to be more disruptive.

Altenberg, in his "soft brood selection" method [1], first generated a brood by crossing over the selected patents N times and then introduced the best of the brood in the next generation by holding a tournament. Tackett refined this idea and used the cheap "culling function" [10] to identify the best of the brood. He introduced the best two offspring in the next generation.

Other approaches [9][11] 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.

Majeed and Ryan in their context-aware crossover [6] tried to minimize the destructive effects of standard crossover by placing the selected subtree in its best context in the parent tree. It implicitly calculates the best context by using the effect of the placement of the selected subtree on the overall fitness of the parent tree and then selecting the placement resulting in the maximum final fitness.

This paper discusses the constructiveness of context-aware crossover and compares its performace with standard one point crossover and with a tuned version of Brood Recombination. The results reported demonstrate that context-aware crossover is much more constructive than the other crossover operators examined, and produces consierably more fit individuals. We also demonstrate that the use of a very simple cache, which is *two orders of magnitude* smaller than standard caches, can dramatically reduce the number of evaluations that need to be carried out with context-aware crossover, because context-aware crossover can exploit a cache very efficiently.

### 1.1 Use of caching in GP

In GP, the evaluation of a generation is widely accepted to be the most expensive process. Many approaches have been put forth to make it less expensive and more efficient. The most promiment ones are subtree caching, vectorized evaluation [5] and removal of dead code [8]. Keijzer in [5], introduced two types of caching, *bottom-up* and *top-down* caching. The top-down approach encourages the caching of big subtrees while the bottom-up approach encourages the caching of small subtrees. The caching of big subtrees makes the evaluation process more efficient (due to the number of nodes saved) but the larger the size of the cached subtree the less likely it is to be matched and used again during the evaluation process. In his implementation the cache size was varied from 20 to 5000 subtrees.

One possible implementation of the use of cache in tree based GP is shown in figure 1.

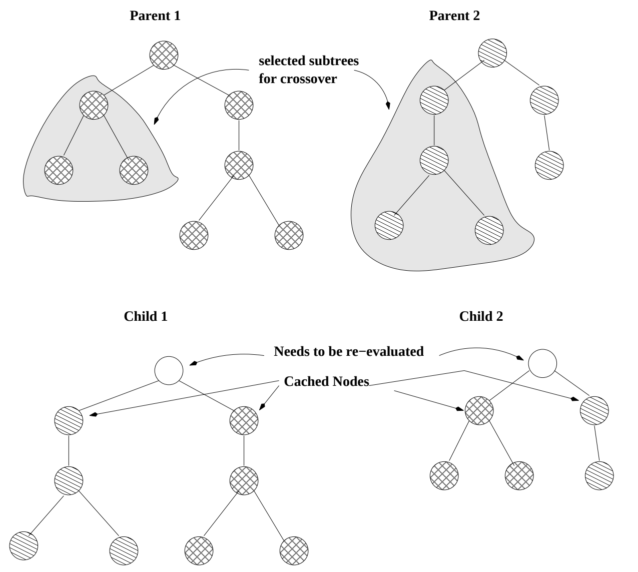

**Figure 1: Use of cache in GP.**

In figure 1, two parents along with the selected subtrees for crossover are shown. The subtrees of both the parents are cached during evaluation and these cached values are reused while evaluating the generated children and results in 9 and 6 times less nodes evalautions, respectively, than the standard GP implementation without the use of cache.

The use of this approach in our implementation of context-aware crossover has resulted in a dramatic improvement in the peformance. These results are presented and discussed in the following sections.

The paper is laid out as follows. Section 2 describes context-aware crossover, while section 2 details the experiments in the paper. Section 4 contains the the results of the experiments and section 5, while section 6 draws some conclusions on the performance of the systems examined in the paper.

## 2. CONTEXT-AWARE CROSSOVER

Another study [6] into the destructive nature of standard crossover introduced *context-aware crossover*. This works by first selecting a random subtree from the first selected parent and then tries to place it, exhaustively, at all the possible locations in the second selected parent in an effort to find its best place. After this, the generated pool of offsprings is evaluated and the best of the lot is introduced in the next generation. The same procedure is repeated for the second parent. The main focus of this crossover is on minimizing the randomness in the operation of standard one point crossover which is believed to be the major cause of its destructiveness.

They showed that context-aware crossover performed best when introduced after about 80% of a run was complete, as it benefited from the mixing provided by standard crossover. This is because the randomness inherent in standard crossover is useful in the initial stages of a run to generate building blocks which are then exploited by context-aware crossver later in the run.

### 2.1 Performance boost by using caching

In context-aware crossover the evaluation of the generated pool of offsprings is the most expensive process and the number of evaluations increases exponentially with the increase in the average tree size of the population and results in extended run time. Caching the evaluated values of each subtree of the selected parents for all the fitness cases and then using them during the re-evaluation phase of the generated pool can result in a dramatic decrease in the number of nodes evaluations.

The number of evaluated nodes strictly depends on the placement of the selected subtree in the parent tree. The number of evaluations will be quite small if a subtree is placed high up in the selected parent because the number of the parent that have to be evaluated are small. Conversely, if the evaluated node count will be quite high if it is placed down below in the parent tree.

The performance of a cache depends on its size. The larger a cache is, the more likely a cache look up is produce a hit (the node or item is present in the cache). However, the larger the cache is, the more expensive it is to search for items in it.

Context-aware crossover is particularly suited to the use of a cache, because the evaluation of multiple children from the same crossover involves the re-evaluation of the same group of nodes. This means that the cache size can be tiny, as it only needs to contain the nodes of the new subtree and the existing parent, and can be emptied at the start of each crossover.

### 2.2 Constructiveness

The constructive nature of the crossover operator(s) used during a GP run has a big impact on the final outcome of the run. Standard one point crossover is widely accepted as a destructive operator due to its ignorance of the context of the exchanged subtrees and the randomness in its operation (because it only samples a tiny fraction of the space of possible offspring for each crossover). Context-aware crossover on the other hand, implicitly searches for the best context of the exchanged subtree before placing it.

One way to look at the constructiveness of a crossover is to compare the fitness of the generated children compared to their parents. A constructive crossover should produce fit children more frequently compared to a destructive crossover. The fitness of the overall population in general and the fitness of the selected parents in particular, has a huge impact on the constructiveness or otherwise of a crossover. The selection of highly fit parents makes it difficult for the crossover operator to generate children of the same fitness due to the availability of the small room for improvement. We believe that context-aware crossover has the power to improve over highly fit parents due to its high exploratory power and we shall investigate this in the rest of the paper.

## 3. EXPERIMENTAL SETUP

In this study, five different sets of experiments were conducted and for all the experiments Koza's Quartic Polynomial Symbolic Regression problem was examined. All the runs were allowed to complete 50 generations. Fitness proportionate selection was used for all the studied crossover operators and the initial population was generated using ramped half and half method with tree size varying from 2-6, with the maximum tree depth was set to 17. Results are averaged over 50 runs and presented in the following section. The reproduction operator was also used in the extended-brood and standard crossover setups, and the probability of its usage was set to 0.1. For the context-aware crossover setup, the initial generations were generated by using only standard one point crossover and context-aware crossover was switched on after 80% completion of the run.

The first experiment was designed to check the constructiveness of each crossover operator. In this experiment, the percent gain or loss in the fitness of the generated children compared to its parents was noted.

The second experiment keeps the generational count of the children better than both, better than one or worse than both the parents. This gives us an insight into the genetic pool generated by brood and context-aware crossover operators and in their consistency of re-generating them. Recall that brood and context-aware crossover operators first generate a pool of offsprings and then select the best of them for the introduction into the next generation, and that the difference in the way they operate is that context-aware crossover systematically evaluates every possible crossover point, and so generates many offspring.

Although the previous test keeps the count of the generated children on the basis of their fitness it tells us little about the fate of the each **crossover event**. A crossover event has different meaning for each crossover studied. In the case of standard crossover it is the generation of two children, while in case of context-aware and brood crossovers it is the generation of the whole brood or pool of offsprings. In this study, one crossover event for context-aware and brood crossover operators was compared against multiple standard crossover events to make the evaluations count same and comparison fair. This is because the total number of evaluations varies from run to run, as it depends on the size of the trees.

Note that one very fit crossover event can generate a high percentage of the total count of the better-than-both-parents individuals generated during a generation. Therefore, the count of the fit individuals generated during a generation alone does not tell us the complete picture and needs further investigation. To investigate this, we labeled each crossover event and then counted its frequency. This tells us the frequency of the successful crossover events generated by each crossover operator during a generation. The crossover event was labeled as constructive if it generated at least one individual fitter than both the parents, semi-constructive if it generated at least one individual fitter than either of the parents and otherwise labeled as destructive.

The fourth experiment looked into the average fitness of the selected parents of each generation. This is helpful in fully understanding the first experiment, as only the gain or loss in the fitness of the generated child hides its absolute fitness and poor parents can show positive results by generating marginally better children than them.

The fifth experiment recorded the average and best fitness values of the generated population. This was to compare the performance of each crossover.

## 4. RESULTS

As the crossover operators used in this study work differently and generate different numbers of indivduals during each crossover event the total number of individuals processed varies across runs and generations. Thus, the graphs below count the total number of evaluations rather than generations.

For context-aware crossover, a population size of 200 was used and the initial 80% of the run was completed using only standard one point crossover and rest of the time only context-aware crossover was used. This setting was in conformance with the strategy devised in [7] and helped us to understand the interactions between the standard and context-aware crossover operators during a run. The results generated by the use of standard one point crossover can be easily identified in all the graphs (prior to 9,000 evaluations) as there is a dramatic change in performance from that point on.

For brood crossover, the population size was also set to 200. In the original implementation of brood crossover, the brood size varied from 2 to 5 and was set before starting the runs, but in our implementation it varied with the parent size and was set to the 20% of the parent size. The same procedure was repeated for the second parent. This makes brood crossover behave more like context-aware crossover and improves its chances of producing more fit offspring by increasing the size of the sample it takes from the space of possible children. Due to this alteration in the implementation, we shall refer to it as "extended-brood crossover" in this paper.

An oversized population size of 2000 was used for experiments involving standard one point crossover. This was to make the evaluations count comparison fair with the other crossover operators. All results are shown below.

### 4.1 Fitness of the generated children

Figures 2, 3 and 4 show the number of the children generated depending on their fitness compared to their parents for the studied crossover operators. A generated child can be labeled as "better-than-both", "better-than-one" or "worse-than-both", if it is better than both the parents, better than one or worse than both the parents, respectively. The y-axis shows the percent count of the generated children in the three categories. In figure 2, both the initial phase of context-aware and the phase where context-aware crossover is used show a drop in the better-than-both and better-than-one categories throughout the run barring a brief initial period in which standard crossover is shows an improvement in the better-than-one parent count. It is widely accepted that in the initial stages of a run it is easy for a crossover to generate better children than its parents than in the later stages, this is due to the poor initial fitness of the selected parents. Later on, once context-aware crossover is turned on, we also see a drop in the better-than-both and better-than-one counts. We believe this is due to the production of the fit parents which makes it difficult for the context-aware

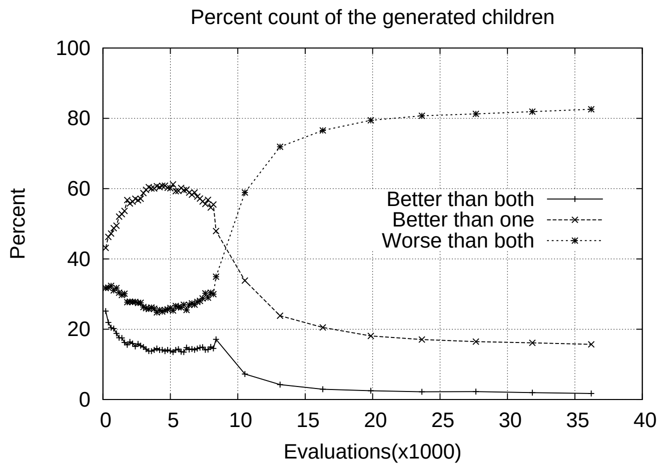

**Figure 2: Fitness plots of the generated children relative to their parents for context-aware crossover.**

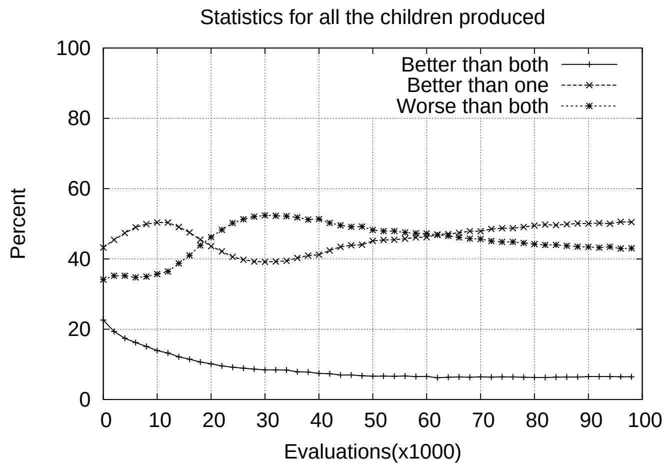

**Figure 4: Fitness plots of the generated children relative to their parents for standard crossover.**

crossover to generate better children than their fit parents.

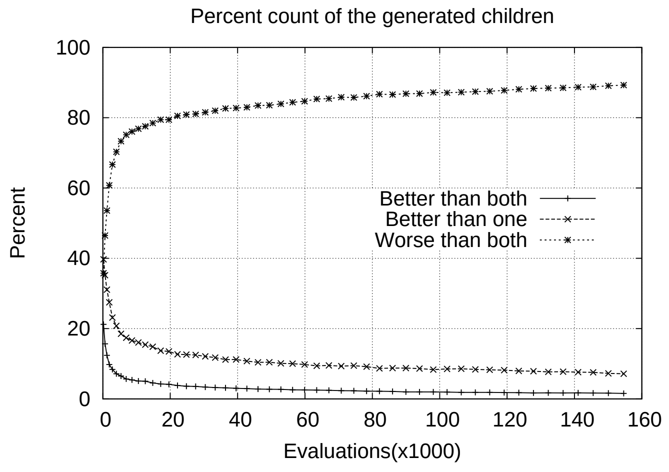

**Figure 3: Fitness plots of the generated children relative to their parents for extended-brood crossover.**

Figure 3 shows the results for extended-brood crossover. The counts of children better than both the parents and children better than one of the parents drops with time and this drop is significant in the initial stages of the run.

In case of standard crossover (see figure 4), there is a drop in the count of the better-than-both the parents in the initial stages of the run and remains above the zero mark in the latter part of the run. The other two counts remain arround 50% mark with little deviation.

## 4.2 Fitness of the selected parents

Figures 5, 6 and 7 show the fitness of the selected parents and the mean average and best fitness of the population during the run for the studied crossover operators. These plots help us to interpret the figures shown in the section 4.1 better. Figure 5 shows the results for context-aware crossover.

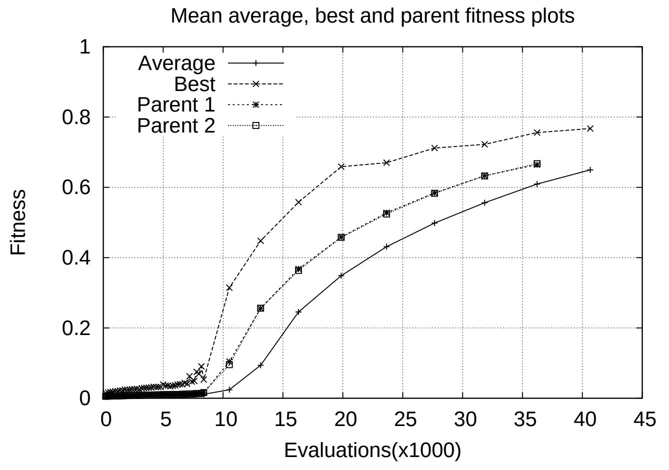

**Figure 5: The fitness of the selected parents along with the mean average and best fitness plots for context-aware crossover.**

The selected parents become increasingly more fit over time making it difficult for both the standard and context-aware crossovers to improve upon them and generate fitter children than them. The best and average fitness plots show a dramatic improvement in the fitness after switching on of context-aware crossover and at the end of the run the mean best fitness and mean average fitness are as high as 0.78 and 0.66, respectively.

Figure 6 shows the results for extended-brood crossover. The parents selected are inferior in fitness to the ones selected by context-aware crossover, and this makes it easier to generate children that are more fit than their parents. Extended-brood recombination does not show any improvement over context-aware crossover and the count for worse-than-both the parents is higher and count for the better-than-either of the parents is significantly lower than the corresponding counts for context-aware crossover.

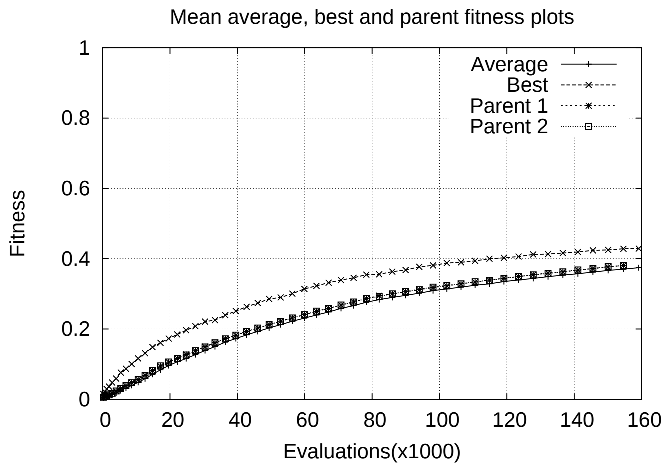

**Figure 6: The fitness of the selected parents along with the mean average and best fitness plots for extended-brood crossover.**

There is a dramatic difference in the best and average fitness values of extended-brood and context-aware crossovers at the end of 35,000 evaluations. Extended-brood crossover turns out to be more than three times less fit and much less contrutive after 35,000 evaluations.

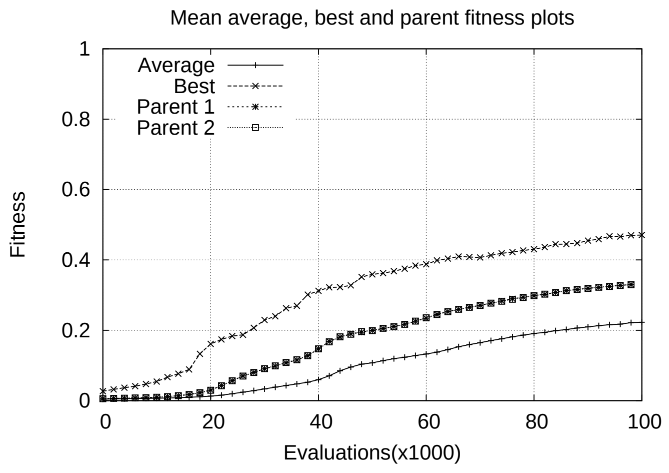

**Figure 7: The fitness of the selected parents along with the mean average and best fitness plots for standard crossover.**

Figure7 shows the results for standard crossover. The selected parents are not very fit in the initial stage of the run and leaves much room for improvement. Surprisingly, standard crossover performs marginally better than extended-brood crossover but is way behind the performance of context-aware crossover.

## 4.3 Statistics of the generated crossover events

Figures 8, 9 and 10 show the statistics of the crossover events take place during the run for the studied crossover operators. Recall that an event is called a constructive one if it generates at least one child better than both the parents, and its called semi-constructive if it generates at least one child better than either of the parents, otherwise it is termed as a destructive. In figure 8, standard one point crossover events are mostly destructive and beyond 7,000 evaluations, their number increases over time while the constructive and semi-constructive events are decreasing with time. As soon as context-aware crossover is switched on the plots change dramatically. The percent count of destructive crossovers drops from 58% to 2% while the percent counts of constructive and semi-constructive crossovers improve to 50% and 48% from 4% and 42% respectively. Interestingly, after a slight increase in the number of destructive crossovers, it shows a downward trend and remains below 10% mark. The number of constructive crossovers drops over time due to the presence of the fit selected parents. Surprisingly, at the latter part of the run the semi-constructive count is as high as 80%, which is a good indication of the constructiveness of context-aware crossover even in the face of a highly fit population.

Figure 9 shows the results for extended-brood crossover. The destructive events are 5% higher than the destructive events reported by context-aware crossover and semi-constructive events are 25% less than the ones generated by context-aware crossover. The constructive count for extended-brood crossover is signigicantly higher than context-aware crossover at the end of 35,000 evaluations but context-aware crossover is far more constructive throughout the run than extended-brood crossover.

As expected, standard crossover (see figure 10) generates very few constructive crossover events during the run and most of the crossover turns out to be destructive due to its randomness.

## 4.4 Constructiveness of the crossovers

Figures 11, 12 and 13 show the percent gain of the children over their parents for the studied crossover operators. In figure 11, the children generated by standard crossover are not much better than their parents and have the same

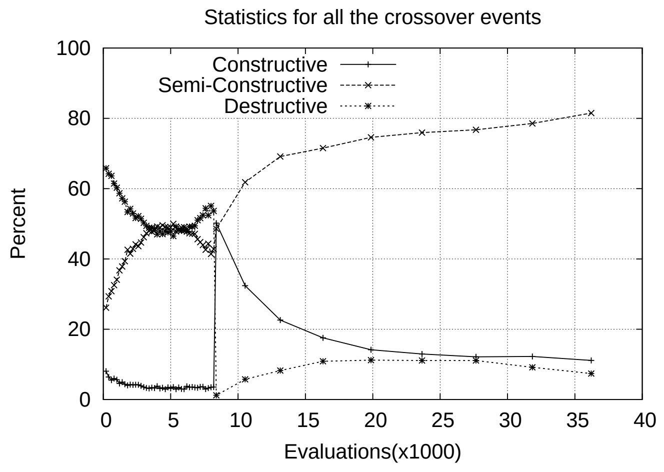

**Figure 8: The percent count of constructive, semi-constructive and destructive crossover events for context-aware crossover.**

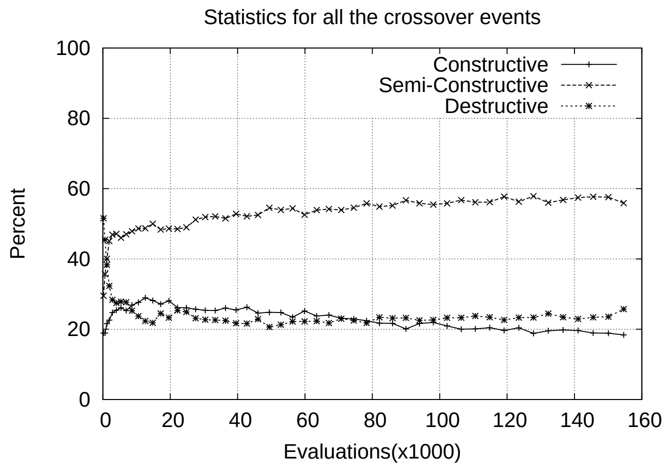

**Figure 9: The percent count of constructive, semi-constructive and destructive crossover events for extended-brood crossover**

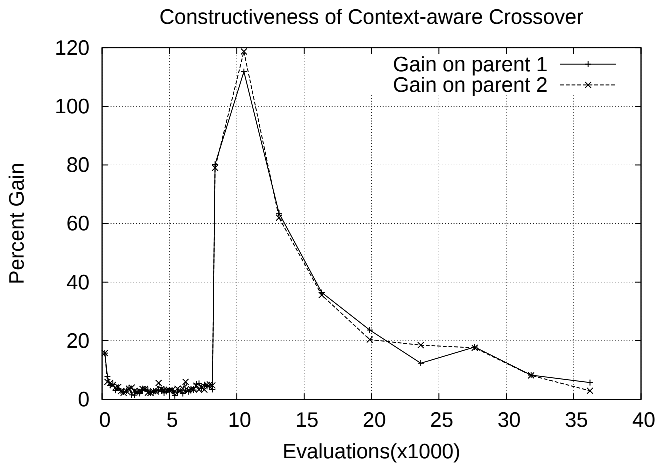

**Figure 11: The percent count of constructive, semi-constructive and destructive crossover events plots for context-aware crossover.**

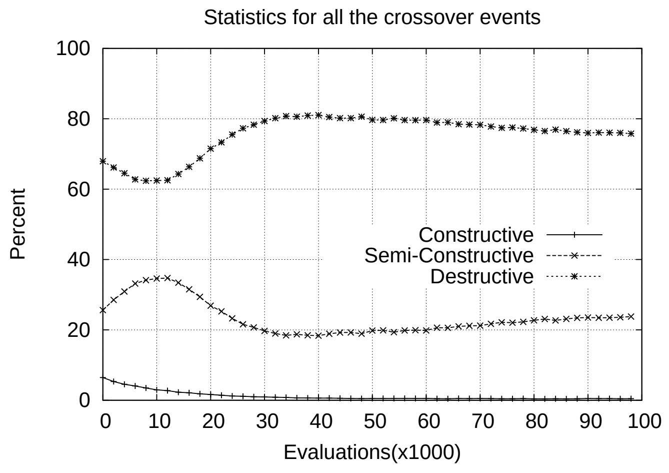

**Figure 10: The percent count of constructive, semi-constructive and destructive crossover events plots for standard crossover.**

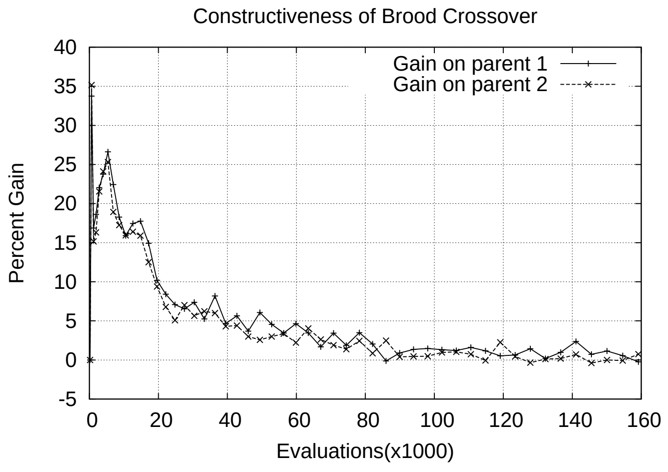

**Figure 12: The percent count of constructive, semi-constructive and destructive crossover events plots extended-brood crossover**

fitness as their parents. Note that at this early stage of the run, there is much room for improvement and it is easy for the selected parents to generate children that are fitter than them. When started, context-aware crossover on the other hand shows a huge gain in the fitness of the generated children compared to their parents and keeps on generating fit children for some time before falling down to zero mark. Interestingly, even in the later stages of the run it never becomes destructive and acts neutrally at the worst.

In figure 12, the behavior shown by extended-brood crossover is similar to context-aware crossover, earlier in the run it about 25% of offspring are more fit than their parents, and this gradually falls, but, like context-aware crossover, never falls below the 0% mark and at the worst remains neutral.

Most of the time, the children generated by standard crossover are inferior to their parents (see figure 13). The main reason for its destructiveness is considered to be its random placement of subtrees.

## 4.5 Use of cache in context-aware crossover

The use of cache in context-aware crossover results in a considerable improvement in the system's performance by reducing the number of evaluated nodes. The presented results in figure 14 show the difference in the number of evaluated nodes by using cache after the activation of context-aware crossover during the run.

The non-cached node plot shows a significant increase in the count of the evaluated nodes with time, mostly due to the generation of larger trees in the later generations. The use of a cache reduces the evaluated nodes count significantly and keeps it steady around 20,000 mark. At the end of 35,000 evaluations the evaluated nodes count using

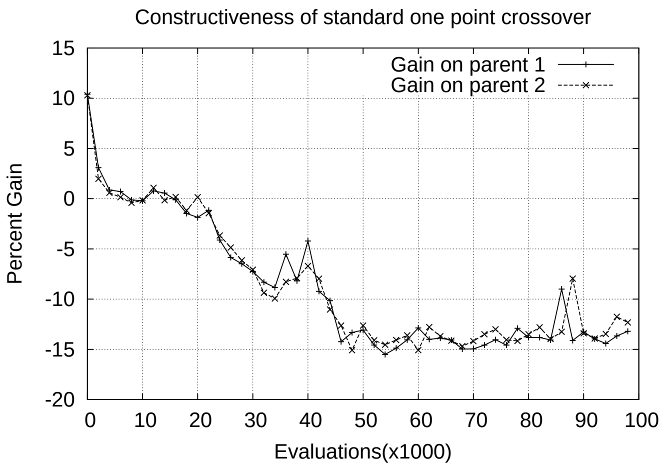

**Figure 13: The percent count of constructive, semi-constructive and destructive crossover events plots for standard crossover.**

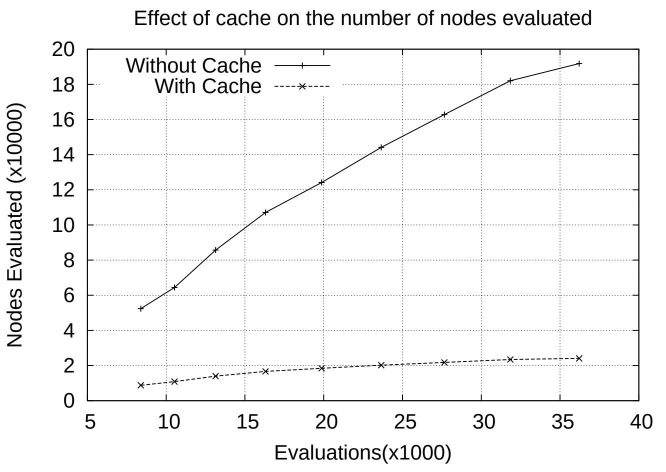

**Figure 14: Number of nodes evaluated by context-aware crossover with and without the use of cache.**

cache is 10 times less than its counterpart. This performance gain makes context-aware crossover a lot more efficient and the increased number of evaluations, due to generation of a brood of offsprings, less noticeable.

The cache size used for context-aware crossover was variable (the size depends on the size of the selected parents) and much smaller than the cache size used by the commonly used generation level caching techniques, varying in size from 8 to 25 during the run. This is much smaller than that used in [5], where it varied from 25 to 10,000. The use of a small sized cache reduces the evaluated node count, memory usage and lookup time during the evaluation phase and is possible beacuse of the way individuals are created and evaluated repeatedly.

## 5. DISCUSSION

The experimental results demonstrate that there is a clear difference between context-aware and brood crossovers. Context-aware crossover, through its systematic search through the space of possible children, consistently operates more constructively and produces runs that are considerably more fit than brood (and standard crossover) both in terms of the best individuals found and in terms of the quality of individuals in general.

The number of children better-than-both and better-than-one parents drops over time for context-aware crossover and this is due to the generation of increasingly more fit parents. At the end of the run, even though the fitness of the parents is as high as .78, there is still the production of 20% of the population involving individuals better than at least one of the parents and the percent generation of the worst-than-both the parents individuals becomes steady at 80%. This shows the exploratory and constructive power of context-aware crossover and its ability to generate fit individuals even in the presence of very fit and converged population. Extended-brood crossover generates the same number of better-than-both, better-than-one and worse-than-both individuals but does so in the presence of quite inferior parents and more room for improvement, which makes it less constructive.

The use of context-aware crossover during the run has a dramatic effect on the fitness and the number of constructive, semi-constructive and destructive crossovers previously shown by standard crossover. The instant it was switched on, it resulted in an exponential gain for the fitness of the population and made all the crossover events constructive, this is evident from the drop of destructive count to zero in figure 8. On most of the occasions it has resulted in the generation of children better than at least one of the parents, again showing its ability to generate fit individuals in the presence of fit parents. Extended-brood crossover maintains the number of constructive, semi-constructive and destructive crossovers in the presence of increasingly more fit parents. This makes it consistent in its performance but less constructive than context-aware crossover. Standard crossover generates mostly destructive crossover events throughout the run and managed to improve the fitness of the population due to small number of constructive and semi-constructive crossover events.

The use of context-aware crossover during the run has resulted in the generation of the children with twice the fitness of their parents, although with time this gain has reduced due to the convergence of the population and selection of the fit parents but has never fallen below zero mark. Extended-brood crossover has shown similar behavior but for this crossover the fitness gain of the generated over their parents is significantly low, which is due to the less constructive nature of this crossover, nonetheless, like context-aware crossover it also never has become destructive throughout the run. Standard crossover on the other hand has proved to be the most destructive as pointed out in [7], due to tiny sample size it takes from the space of children.

The overall constructiveness of context-aware crossover has resulted in a dramatic improvement in the final fitness of the population and generated a very fit population as compared to other two crossovers.

The use of cache has reduced many times the expensiveness of the evaluation process of context-aware crossover as awell as having reduced its run time. The use of individual level cache in context-aware crossover has turned out to be very effective by making each crossover event less resource hungry and efficient as context-aware crossover evaluates the same individual many times with a little variation.

## 6. CONCLUSION

We have compared context-aware crossover to standard GP crossover as well as to an extended version of brood crossover. We have demonstrated the context-aware crossover is consistently more *constructive* than the others, and that, even when the population is very fit, it can still find individuals better than their parents.

Clearly, finding the best context for the incoming sub-tree dramatically improves the performance of GP. In the absence of a method that explicitly discovers this context, an exhaustive search is the only alternative. However, because context-aware crossover is so constructive, it is much cheaper than the other crossover methods, and so far smaller populations can be used. Not only that, we have demonstrated that it can be optimized further through the use of a very simple cache. The cache is simple because it is very small, two orders of magnitude smaller than a standard cache, and is reset for each crossover event. The manner in which context-aware crossover repeatedly evaluates the same sub-trees makes this possible, and we demonstrate that the simple cache gives almost an order of magnitude speed up.

## 7. 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] 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.

[3] S. Hengproprohm and P. Chongstitvatana. Selective crossover in genetic programming. In *ISCIT* , ChiangMai Orchid, ChiangMai Thailand, 14-16 November 2001.

[4] 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.

[5] Maarten Keijzer. Alternatives in subtree caching for genetic programming. In *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 328–337, Coimbra, Portugal, 5-7 April 2004. Springer-Verlag.

[6] Hammad Majeed and Conor Ryan. A less destructive, context-aware crossover operator for GP. In *Proceedings of the 9th EuroGP*, volume 3905 of *LNCS*, pages 36–48, Budapest, Hungary, 10 - 12 April 2006. Springer.

[7] Hammad Majeed and Conor Ryan. Using context-aware crossover to improve the performance of GP. In *GECCO 2006*, volume 1, pages 847–854, Seattle, Washington, USA, 8-12 July 2006. ACM Press.

[8] Hammad Majeed, Conor Ryan, and R. Muhammad Atif Azad. Evaluating GP schema in context. In *GECCO 2005*, volume 2, pages 1773–1774, Washington DC, USA, 25-29 June 2005. ACM Press.

[9] 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.

[10] 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.

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