An Analysis of Constructive Crossover and Selection Pressure in Genetic Programming

Huayang Xie
Victoria University of Wellington
Wellington, New Zealand
hxie@mcs.vuw.ac.nz

Mengjie Zhang
Victoria University of Wellington
Wellington, New Zealand
mengjie@mcs.vuw.ac.nz

Peter Andreae
Victoria University of Wellington
Wellington, New Zealand
pondy@mcs.vuw.ac.nz

ABSTRACT

A common problem in genetic programming search algorithms is destructive crossover in which the offspring of good parents generally has worse performance than the parents. Designing constructive crossover operators and integrating some local search techniques into the breeding process have been suggested as solutions. This paper reports on experiments demonstrating that premature convergence may happen more often when using these techniques in combination with standard parent selection. It shows that modifying the selection pressure in the parent selection process is necessary to obtain a significant performance improvement.

Categories and Subject Descriptors

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

General Terms

Performance

Keywords

Genetic Programming, Crossover, Stochastic Elements, Selection Pressure

1. INTRODUCTION

Crossover (sexual recombination) is considered to be the primary genetic operator for modifying program structures in Genetic Programming (GP) [6]. It plays a critical role in deriving optimal solutions as shown by the large number of attempts since the 1990s to develop new crossover operators, especially *constructive* crossover operators.

Selection pressure in the selection of parent programs is intended to improve the average problem solving quality of the population. It gives individuals of higher quality a higher probability of being used to create the next generation so that the search algorithm focuses on promising regions in the search space [1].

In the standard breeding process of the GP algorithm, crossover operators produce two offspring from a pair of parents programs. With the standard breeding process, exploring new states in the neighbourhood search space of current states can be viewed as a set of random walks. If parents are randomly selected for mating, the GP algorithm will effectively act like a random (beam) search algorithm. Therefore selection pressure is applied to the parent selection process to reduce the stochastic element of the search and to provide individuals having good fitness with more chances to be chosen than others. Good genetic material in the chosen individuals is expected to be propagated along evolution in order to speed up population convergence.

With the standard breeding process, applying selection pressure on the parent selection has been recognised as more effective than selecting parents randomly. Individuals with good fitness are used with increased frequency to produce offspring. However, the number of possible offspring in the immediate neighbourhood of any chosen pair of parents is very large, and a very large fraction of these offspring will not constitute improvement over the parents. Therefore, even an increased number of matings of good parents is still insufficient to provide a good chance of finding good offspring. For example, Nordin, *et al.* reported that most crossover events in the standard breeding process produce offspring with less than half of the fitness of their parents [11, 12].

One approach to overcoming this problem is to increase the chance of generating improved offspring from a pair of parents by using a customised *constructive* crossover operator that avoids generating worse offspring altogether. However, designing such operators can be difficult and is likely to be very domain dependent. An open question is whether this approach would result in more effective GP — it is possible that reducing the stochastic element of GP in this way may result in premature convergence or other undesirable restrictions on the GP search.

An alternative, simpler, and domain independent approach is to integrate variants of local search techniques into the breeding process to search for good offspring. This can achieve the same effect as constructive crossover operators, though at the cost of a possibly expensive local search. We use this approach to explore the consequences for GP of using constructive crossover operators, and also explore the technique in its own right as modification of the standard GP search.

The next section reviews related work that integrates local search techniques into GP, and the following section presents our research questions in detail.

1.1 Related Work on Crossover

Tackett [14] deliberately increased the amount of genetic material used from a selected individual regardless of the frequency of the individual being selected during the process of generating the next generation. He designed a *brood recombination* operator. The operator is inspired by the fact that animal species produce far more offspring than are expected to live. It *randomly* applies crossover N times to two chosen programs to produce $2N$ offspring. After evaluating all offspring, it puts the best two into the next generation and discards rest of the offspring.

The brood recombination operator can be categorised as a partial local search operator because it looks for the best state in available states but only looks at $2N$ possible successor states. Tackett asked whether the brood recombination operator reduces the diversity of subtrees, eliminating ones which are unfit in the current generation but which might be useful at a later time. He compared brood recombination using a tournament size of 6 with a random parent selection using the same set of initial populations. The results suggest that the brood recombination operator is effective. However, a difficulty with his conclusion is that the number of random crossover operations (the brood size factor) is chosen without sufficient regard to parent program sizes so that the degree of intensive search within all possible successor states of chosen parents has not been well investigated with regard to the population diversity concern.

Lang [8] introduced a *headless chicken crossover* operator which is applied to a chosen program P and a newly (and randomly) generated program R. The operator repeatedly produces offspring from P by replacing a sub-tree of P with a replaced sub-tree from R until it finds an offspring with greater or equal fitness (problem solving quality) to P's.

The headless chicken crossover operator can be categorised as a hill-climbing local search [13]. It is only a partial local search because it randomly looks for a state better than or equal to the current state and stops once it finds such a state rather than looking at all possible successors. Note that Lang's method is really a mutation rather than crossover, since only one "parent" is chosen from the current generation[7].

Majeed and Ryan [10] introduced a *context-aware crossover* operator which identifies all possible contexts in one parent for a randomly-chosen sub-tree from the other parent, then evaluates each of them. The context that generates an offspring with the highest fitness is used and the offspring generated is then passed into the next generation. Fitness proportionate selection and tournament selection with size 7 are used to select parents in different problems. The authors claimed that the operator improves both mean best fitness and mean average fitness, reduces bloat in most of their experiments, and produces significantly smaller individuals in most cases.

The context-aware crossover operator can be also categorised as a partial local search operator. From their discussion of future work, it seems that they experienced a fast population convergence problem and their temporary solution was to permit only one offspring per crossover.

Harries and Smith [5] evaluated more children but only accept new programs whose fitness values are greater than or equal to their parents in a study of depth-based crossovers. Their search algorithm is a type of stochastic hill-climbing algorithm because not all possible children are evaluated and the fittest child is not necessarily chosen. Mahfoud [9] illustrated the interaction between directed crossover operators and selection pressure in a context of genetic algorithms. Terrio and Heywood [15] investigated a family of directed crossover operators under a steady state selection model.

1.2 Research Questions and Goals

A common feature of the related work is the integration of variants of local search techniques in the breeding process, done by allowing parents to produce many offspring and applying a selection pressure to choose high performing offspring. The effect of the modification is to create a constructive or at least "less destructive" crossover operator [10]. The more intensive the local search, the more constructive the operator: a very intensive search can generate the best possible offspring of two parents; a less intensive search that only considers part of the neighbourhood will have a greater stochastic element and may generate offspring that are good but not the best possible.

The use of constructive operators in the breeding process alters the standard GP search algorithm by reducing the stochastic element in the process of generating the next generation. The more intensive the local search, and the greater the selection pressure in the choice of offspring, the smaller the stochastic element in the breeding process.

An important question is how intensive the local search should be. The approaches in the previous section use partial local searches that consider only subsets of the immediate neighbourhood of the chosen parents. It would be possible to extend this to a complete local search that considers all possible offspring. Our experiments explore the effect of different intensities of the local search.

A second question concerns the effect of reducing the stochastic nature of the GP search by increasing the selection pressure towards high fitness offspring. In the standard GP, selection pressure is only applied in the selection of parents — the offspring produced are put into the next generation without selection. With a many-offspring breeding process performance or with constructive operators, selection pressure will be applied to the selection of offspring as well. In general, increasing selection pressure tends to confine the search process, speed up the loss of population diversity, and lead the search to premature convergence. It is important to explore the effect of this increasing selection pressure. Our experiments explore the appropriate balance between selection pressure in the parent selection process and selection pressure in the breeding process.

For both of these questions, we are considering a many-offspring local search merely as a means of simulating a constructive crossover operator, so that the cost of the performing the local search is ignored. Our goal is to determine whether it is worth trying to design constructive operators, and how they should be used in a GP system.

A third question concerns the effectiveness of a many-offspring local search as a technique in its own right as part of the GP process. There is no question that an intensive local search for good offspring of two parents is expensive and will take resources away from exploring more possible pairs of parents and more generations of programs. Our experiments explore whether the cost of this local search can be worthwhile in the context of a resource-limited GP process, and our goal is to determine the appropriate intensity of a local search in the breeding process to maximise the performance of a GP system.

2. SIMULATIONS OF CONSTRUCTIVE CROSSOVER OPERATORS

In the literature, the key point of differentiation between a *constructive* and a *destructive* operator is purely based on the performance-based fitness value. Nordin, et al. [12] define a destructive crossover operation as one in which the offspring have fitness values at least 2.5% worse than those of the parents, and a constructive crossover operation as one in which the offspring have fitness values better than those of the parents. Given these definitions, the conventional crossover operator has been shown to be more destructive than constructive [11, 12].

A many-offspring breeding process is not strictly a constructive crossover operator. At most it can be viewed as a simulated constructive crossover operator that produces the same results as a constructive crossover operator but will have to generate a large number of poor offspring in the search for good offspring in successor states.

Investigating the research questions given in Section 1.2 requires building at least two simulations of a constructive crossover operator. One simulation mimics an *ideal* constructive crossover operator (Ideal Xover). It considers all possible ways of recombining two chosen parents to produce all possible offspring, then evaluates them, and keeps two offspring with the best fitness values but throws the others away. The other simulation mimics a *partial* constructive crossover operator (Partial Xover). It chooses a crossover point randomly in one parent P_1 but considers all nodes in the other parent P_2 to produce offspring. Both simulations focus on optimising the offspring's fitness — problem solving quality, but Partial Xover contains some stochastic elements while Ideal Xover completely eliminates them. Partial Xover is very similar to the *context-aware crossover operator* [10] but has no depth constraint on choosing the possible crossover points in program P_2.

3. EXPERIMENT DESIGN

To investigate the effect of including selection pressure from constructive crossover operators in the GP process, our experiments consider the four combinations illustrated in Figure 1. The selection of parent programs can either have no selection pressure by using a random parent selection process, or can apply selection pressure, as in the standard GP algorithm. The selection of offspring in the breeding process can either have no selection pressure as in the standard breeding process, or can apply selection pressure using a many-offspring breeding process. We consider two levels of selection pressure in the many-offspring breeding process by using Ideal Xover to represent strong selection pressure, and using Partial Xover to represent weaker selection pressure.

The experiments explore the consequences of the six different combinations of selection pressure on three different domains: an Even-n-Parity problem (EvePar), a Symbolic Regression problem (SymReg), and a Binary Classification problem (BinCla).

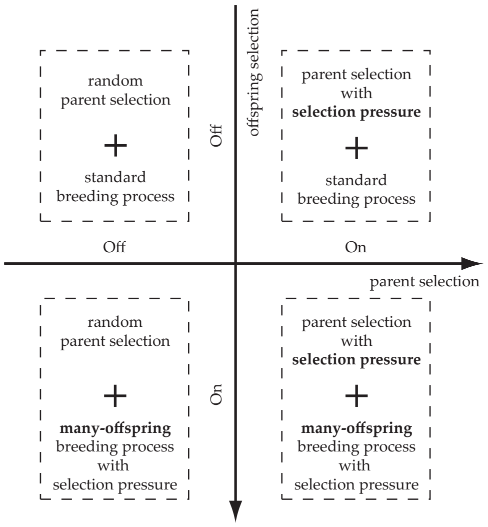
offspring selection
parent selection
Off
On
Off
On
random parent selection
+
standard breeding process
parent selection with **selection pressure**
+
standard breeding process
random parent selection
+
many-offspring breeding process with selection pressure
parent selection with **selection pressure**
+
many-offspring breeding process with selection pressure

Figure 1: Four groups of GP systems according to configurations of selection pressure on parent selection and offspring selection.

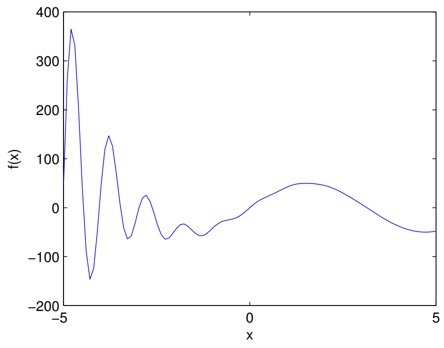

Figure 2: The symbolic regression problem.

3.1 Data Sets

EvePar takes an input of a string of n boolean values and outputs *true* if there are an even number of true's, and otherwise *false*. The most characteristic aspects of this problem are the low number of fitness cases, and the requirement to use all inputs in an optimal solution. In this study, the case of $n = 6$ is considered. Therefore, there are 2^6 combinations of unique 6-bit length strings as fitness cases.

SymReg is shown in Equation 1 and visualised in Figure 2. We generated 100 fitness cases by choosing 100 values for x from [-5,5] with equal steps.

$$f(x) = exp(1 - x) \times sin(2\pi x) + 50sin(x) \qquad (1)$$

BinCla involves determining whether examples represent a *malignant* or a *benign* breast cancer. The dataset is the Wisconsin Diagnostic Breast Cancer dataset chosen from the UCI Machine Learning repository [2]. The BinCla consists of 569 data examples, where 357 are benign and 212 are malignant.

3.2 Function Sets and Terminal Sets

The function set used for EvePar consists of the standard

Boolean operators { *and*, *or*, *not*} and *if* function. The *if* function takes three arguments and returns its second argument if the first argument is *true* (1), and otherwise returns its third argument. In order to increase the problem difficulty, we do not include the *xor* function in the function set.

The function set used for SymReg includes the standard arithmetic binary operators {+, -, *, /} and unary operators { *abs*, *exp*}. The / function returns zero if it is given invalid arguments. Note that for the purpose of simulating a real world problem, we intentionally do not include the *sin* function so that the GP algorithm needs to generate an approximation of it.

The function set used for BinCla includes the standard arithmetic binary operators {+, -, *, /}, unary operators { *abs*, *sqrt*, *sin*} and *if* function. The *sqrt* function automatically converts a negative argument to a positive one before operating on it. The *if* function takes three arguments and returns its second argument if the first argument is positive, and otherwise returns its third argument.

The terminal set for EvePar and SymReg consists of n boolean variables and a single variable x respectively. The terminal set for BinCla consists of 10 numeric features (see Table 1) computed from a digitised image of a fine needle aspirate of a breast mass and are designed to describe characteristics of the cell nuclei present in the image. The mean, standard error, and "worst" of these features are computed, resulting in 30 terminals [2].

Table 1: Ten features in the dataset of BinCla

a	radius	f	compactness
b	texture	g	concavity
c	perimeter	h	concave points
d	area	i	symmetry
e	smoothness	j	fractal dimension

Real valued constants in the range [-5.0, 5.0] are also included in the terminal sets for SymReg and BinCla. The probability mass assigned to the whole range of constants when constructing programs, is set to 5%.

3.3 Fitness Function

The fitness function in EvePar is the number of wrong outputs (misses) for the 64 combinations of 6-bit length strings. The fitness function in SymReg is the root-mean-square (RMS) error of the outputs of a program relative to the expected outputs. The fitness function for the classification problem is the classification error rate on the training data set (the fraction of fitness cases that are incorrectly classified by a program as a proportion of the total number of fitness cases in the training data set). A program classifies the fitness case as *benign* if the output of the program is positive, and *malignant* otherwise. All three problems have an ideal fitness of zero. Note that for EvePar, most random programs will have a fitness of about 32 misses due to the special characteristic of the problem [4].

3.4 Genetic Parameters

The genetic parameters are the same for all three problems. The ramped half and half method is used to create new programs and the minimum depth of creation is three and the maximum is five. The maximum size of a program is 50 nodes. The population size is 100. The crossover rate and the reproduction rate are 95% and 5%. For ease of analysis, the mutation operator is not used.

Selection pressure on the parent selection is switched on or off by setting the tournament size to 4 or 1 respectively. Tournament size of 4 is chosen based on empirical search. We expect the setting to provide a neutral environment when conducting performance comparisons. We also apply a multi-objective selection policy for selecting parents and offspring in Partial and Ideal Xover. The multi-objective selection policy consists of three measures. These are fitness value, number of nodes, and depth of tree. The three objectives are applied sequentially in order to select a program with a shallower tree depth and a smaller number of nodes if its fitness value is the same as its competitors.

3.5 Experiment Configuration

We performed two sets of experiments with different termination criteria. The first set of experiments (Exp1) treats the simulated constructive crossover operators as if they were real constructive operators, so that the cost of performing the local search for the best offspring is ignored. The runs are terminated when the number of generations reaches the pre-defined maximum of 51 (including the initial generation), or the problem has been solved.

The second set of experiments (Exp2), takes into account the cost of the local search, and terminates the runs when the total CPU time (seconds) exceeds a pre-defined limit, or the problem has been solved. The time limits were determined from analysis of the results in Exp1.

We ran experiments comparing GP systems with and without parent selection pressure using the standard crossover operator, the simulated partial crossover operator and the simulated ideal crossover operator respectively for each of the three problems. In each experiment, we repeated the whole evolutionary process 100 times independently.

The BinCla dataset represents a harder task than the other two, and was subject to overfitting. We therefore added additional strategies to deal with this problem. In Exp1, an additional termination criterion is applied to BinCla as an overfitting prevention strategy. We split the original BinCla data randomly and equally into a training data set, a validation data set and a test data set. We select the best program in a population based on its fitness on the training data set as the fittest program in the corresponding generation. We then monitor the fitness values of the best program on the training data set and the validation data set within a moving window of size 15. The window moves along evolution and its size of 15 is chosen based on empirical search. A run will terminate when the training fitness of the latest generation in the window has not been improved over the window. The run will also terminate when the validation fitness of the latest generation in the window is not better than that of the earliest generation in the window. For a run which stops according to the strategy, we examine the window and select a generation where the fitness on the validation set is the best in the window. The corresponding test fitness value is used as the performance measure of the run.

For Exp2, we used a different strategy, using 10-fold cross validation for BinCla with attempts to ensure class labels are evenly distributed. A log system records the detailed evolutionary information of each fold in a run, including the

Table 2: Performance of systems using 3 different crossover modes with/without selection pressure in Exp1.

GP	Parent	Xover	EvePar	SymReg	BinCla
Systems	Selection Pressure	Mode	Completion	RMS Error	Test Error Rate (%)
Sys1	On	standard	0%	62.5 ± 3.5	16.4 ± 7.3
Sys2		partial	12%	58.5 ± 3.3	10.9 ± 4.7
Sys3		ideal	23%	58.8 ± 5.7	9.7 ± 4.3
Sys4	Off	standard	0%	65.4 ± 1.0	16.2 ± 6.3
Sys5		partial	9%	55.4 ± 2.9	8.5 ± 2.6
Sys6		ideal	**100%**	37.2 ± 5.7	6.7 ± 2.4

training and test fitness values of the fittest program at each generation. We also base on the training fitness to select the fittest program in a generation. As overfitting may occur, in order to obtain accurate fitness measures for performance comparison purposes, a simple "pruning" algorithm is used to remove some overfitting noise from the detailed log. The simple pruning algorithm consists of two steps. For each fold in a run, in order to minimise the side effect of the performance fluctuation during the early stage of the search, the algorithm firstly filters out records where the fitness on the test data set is worse than a threshold (determined from analysis of Exp1). Then the algorithm selects the best fitness on the test data set as the performance measure of the fold. The performance measure of a run is the average of the best test fitness value over 10 folds.

4. RESULTS AND DISCUSSIONS: EXP1

4.1 Effectiveness

Table 2 shows the performance measures of the first set of experiments. The measure for EvePar is the completion rate, measuring the fraction of runs that successfully returned an optimal solution. The best value is 100%. The measures for SymReg and BinCla are the averages of the RMS error and the classification error rate on test data over 100 runs respectively, thus the smaller the value, the better the performance. Note that the standard deviation follows the ± sign.

When parent selection pressure is switched on, GP systems using Partial and Ideal Xover (Sys2 and Sys3) outperform the GP system using the standard crossover operator (Sys1). The performances in Sys2 and Sys3 are noticeably different in EvePar but very similar in SymReg and BinCla.

When parent selection pressure is switched off, there are significant differences between the performances of GP systems using different crossover modes. Sys6 provides the best performance.

From the results, it seems overall that reducing stochastic elements in the breeding process does not have any side effects. A constructive crossover operator is very effective, especially an ideal constructive crossover operator. This observation matches those made by proponents of a many-offspring breeding process.

When comparing the performances of GP systems with and without parent selection pressure, we realised that Sys1 and Sys4 using the standard crossover operator both perform badly. But we could not draw any sound conclusion about whether Sys4 is worse based on the small performance differences. We also realised that:

- For EvePar, Sys2 and Sys5 have the similar completion rates, but Sys2 is slightly higher than Sys5. Sys6 using Ideal Xover without parent selection pressure is much better than Sys3. All 100 runs successfully found the optimal solution in Sys6.
- For SymReg and BinCla, Sys5 and Sys6 are better than Sys2 and Sys3 respectively.
- Overall, the best performance is obtained by Sys6 using the simulated ideal constructive crossover operator without parent selection pressure.

These results suggest that premature convergence occurred in Sys3 more often than that in Sys6. To confirm this, we examined the index of the generation where the best-of-run appeared for the first time for Sys3 and Sys6. The results are illustrated in Table 3. We realised that, for instance in EvePar, the index in Sys3 ($\mu = 14$, $\sigma = 7$) is much earlier than that in Sys6 ($\mu = 21$, $\sigma = 6$), indicating that the GP system with parent selection pressure causes the search end up with premature convergence more often if stochastic elements are completely removed in the breeding process. Similar phenomena occurred in SymReg and BinCla as well.

Table 3: The average index of generation where the best-of-run appeared first time in Sys3 and Sys6 in Exp1.

GP Systems	EvePar	SymReg	BinCla
Sys3	14 ± 7	12 ± 6	6 ± 4
Sys6	21 ± 6	47 ± 4	16 ± 6

The maximum number of generations stopping criterion in Exp1 can be seen as setting a limited number of movements in a search process. In this situation, it is better to carefully make a wise movement for each step. The problem solving quality can be improved significantly generation by generation by always moving to the fittest status, indicating that the hill-climbing metaphor can be applied in the GP search algorithm to improve its performance. If we can develop an ideal constructive crossover operator (not a simulated one), we should certainly use it to replace the standard blind random crossover operator and we should certainly remove selection pressure from the parent selection.

4.2 Efficiency

Figure 3 shows the boxplots of CPU time consumed by 100 runs in each of the GP systems for the three problems. It is clear that systems using Ideal Xover required much more CPU time than others as they need to evaluate a huge number of offspring.

In EvePar, 77 runs in Sys3 using the Ideal Xover with parent selection pressure switched on have to keep searching until the 51st generation, while all runs in Sys6 with parent selection pressure switched off stop early as optimal solutions are found. Therefore it is not surprising that runs in Sys3 consumed much more time than those in Sys6.

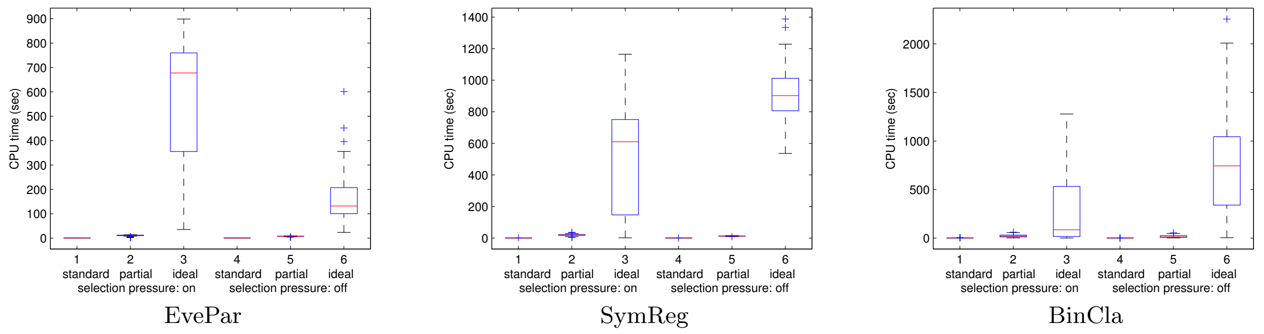

Figure 3: Boxplot of CPU time consumed in systems in Exp1.

In SymReg, as all runs stop at the 51st generation, we expected a similar amount of CPU time consumed for runs in corresponding systems. However, runs in Sys3 surprisingly consumed less time than those in Sys6. We then calculated the number of evaluations during evolution in Sys3 and Sys6. We realised that the number of evaluations in Sys3 ($\mu = 1.00 \times 10^6$, $\sigma = 0.58 \times 10^6$) is much less than that in Sys6 ($\mu = 2.14 \times 10^6$, $\sigma = 0.27 \times 10^6$). This indicates that parent selection pressure forces the search to focus on fit and smaller size programs. The positive effect is that the system controls the code bloat by effectively filtering out programs containing introns. Therefore, in Sys3, with smaller program size, the number of offspring evaluated is also smaller than that in Sys6, resulting in less CPU time used. However, a side effect is that the system also abandons currently unfit sub-trees that are possibly useful later and reduces the population diversity.

In BinCla, due to the use of the overfitting preventing strategy, runs often stop before the 51st generation. We then calculated the average total number of generations used over 100 runs in Sys3 and Sys6. We realised that the average total number of generations used in Sys3 ($\mu = 19$, $\sigma = 3$) is much smaller than that in Sys6 ($\mu = 29$, $\sigma = 6$). When considering the problem solving quality of Sys3, the smaller number of generations used is possibly due to the early occurrences of local optima. Therefore it is not surprising that runs in Sys3 consumed less CPU time than those in Sys6.

4.3 Determining Time Limits

The comparisons and suggestions in terms of the effectiveness made above are based on the assumption that we have a constructive crossover operator which can *directly* find a better or the best offspring without extra computational cost, and the actual computational resources required by the simulated constructive operators were irrelevant. The second set of experiments are intended to investigate the value of the many-offspring local search, and have to take this cost into account.

Sys6 took the greatest amount of CPU time in Exp1. To determine appropriate time limits for the runs in the second set of experiments, we identified the maximum CPU time taken by Sys6 for each of the problems in the first set of experiments, ignoring the outlier runs. On the basis, the CPU time limits in Exp2 are 400, 1200, and 2000 seconds for each run in EvePar, SymReg, and BinCla, respectively. For BinCla, as 10-fold cross validation is used, each fold is assigned an equal amount of CPU time (200 seconds). As a 16.40% classification error rate is the worst performance on average in Exp1, the value is used as the threshold in the pruning algorithm in Exp2.

5. RESULTS AND DISCUSSIONS: EXP2

Table 4 shows the performance measures of the second set of experiments. For EvePar, many runs may terminate before completely consuming the given upper bound CPU time as a result of finding optimal solutions. Therefore, an additional measure, *actual time*, is used for EvePar to measure the actual CPU time consumed in total. For SymReg and BinCla, there is no optimal solution found and all runs terminate when exceeding the allowed CPU time, so the actual time measure is omitted.

5.1 Parent Selection Pressure: On

When selection pressure is applied to the parent selection, surprisingly, the standard crossover operator is better than the simulated constructive crossover operators in EvePar and SymReg, and only has a slightly lower performance in BinCla. This observation suggests that, if sufficient search time is given, when selection pressure is applied to the parent selection for crossover, there is no advantage in using a many-offspring crossover to replace the standard blind random crossover operator. In other words, the standard crossover operator works *quite well* in comparison to a simulated constructive crossover operator using local search.

This observation is different from observation in Exp1 in the case of a real constructive crossover operator. A possible explanation is that the selection pressure on the parent selection removes some stochastic elements and then forces the search focusing on a smaller region. When stochastic elements, which are necessary in order to find global optima, are further removed in the many-offspring breeding process, the search will eventually end up at local optima. Contrarily, the standard breeding process keeps some stochastic elements which help the search escape local optima in a certain amount of searching time.

5.2 Parent Selection Pressure: Off

When parent selection pressure is switched off, Sys5 and Sys6 both produce significant performance improvements. In EvePar, Sys5 outperforms the other two, by not only the 100% completion rate, but also the much shorter CPU time consumed (about 21 seconds on average). Note that the completion rates in Sys6 are different between Exp1 and Exp2. This is possibly because some outliers in Exp1 re-

Table 4: Performance of systems using 3 different crossover modes with/without parent selection pressure in Exp2.

GP	Parent	Xover	EvePar		SymReg	BinCla
Systems	Selection Pressure	Mode	Completion	Actual Time (sec)	RMS Error	Test Error rate (%)
Sys1	On	standard	34%	299 ± 154	53.1 ± 7.6	8.5 ± 1.1
Sys2		partial	32%	288 ± 167	58.6 ± 3.6	7.1 ± 1.0
Sys3		ideal	22%	338 ± 122	58.0 ± 5.8	7.2 ± 1.0
Sys4	Off	standard	10%	370 ± 94	52.3 ± 5.2	6.9 ± 0.8
Sys5		partial	100%	**21 ± 15**	38.9 ± 5.0	4.1 ± 0.6
Sys6		ideal	96%	140 ± 86	37.3 ± 5.6	4.2 ± 0.6

quire more CPU time to complete their search but the CPU time set in Exp2 terminates those outliers before they find optimal solutions. In the other two problems, Partial Xover and Ideal Xover have similar performance.

The results suggest that when stochastic elements are fully preserved in selecting parents, it is necessary to conduct an intensive search in the successor states of chosen parents. The purpose is to remove some stochastic elements and make good movements so that the search will act differently from a completely random search.

The results for EvePar suggest using Partial Xover instead of the standard one or Ideal Xover. One possible explanation is that as the search can make as many movements as possible within the given time frame, it would be better just to look at a subset of all possible movements so that a larger number of less perfect movements can reach the goal faster than smaller number of perfect movements. But the subset has to be big enough to cover most important movements and has to be sufficiently smaller so that most of the worst movements are filtered out. The results suggest Partial Xover may lead the search into an optimal subset of successor states. From the exploration vs. exploitation [3] points of view, Ideal Xover just spends most time exploiting the known genetic material but less time exploring other potential useful search space. Therefore, if a tight time frame is given, it may often fail. To confirm this, we chose a new boundary of 55 seconds, which is about three standard deviations away from the mean in Sys5, and reexamined the completion rates in Sys5 and Sys6. Within the new time boundary, Ideal Xover had about 89% runs exceeding the time limit without finding optimal solutions, while Partial Xover only had about 4% runs that failed. The standard crossover is exactly opposite to Ideal Xover. It puts too much effort into exploring the search space but little effort on exploiting the known genetic material, thus fails also.

5.3 Overall

When comparing the performances in GP systems with and without the selection pressure being applied to the parent selection, we conclude that the performance in Sys4 is the worst in EvePar. But it is better than or comparable with those in all three systems with parent selection pressure on in SymReg and BinCla. This observation suggests that, if enough searching time is given, a random (beam) search can sometimes have similar problem solving quality to GP systems with selection pressure applied to the parent selection.

We also conclude that Sys5 and Sys6 are outstanding in all six systems. The results strongly suggest that if we intend to use a many-offspring crossover instead of the standard one, we should remove the selection pressure from the parent selection to avoid premature convergence in order to gain further performance improvement.

5.4 Further Discussion

One may argue that it is not necessary to turn off the selection pressure when using a many-offspring breeding process. The population diversity can be easily maintained by increasing the population size while keeping selection pressure on the parent selection.

To verify whether this argument is true, we conducted another set of experiments for Sys1, Sys2, and Sys3. In the third set of experiments, the population size is increased from 100 to 1000. Other parameters and stopping criteria are the same as those in Exp2. Table 5 lists the results.

By comparing the performances of corresponding GP systems in Tables 4 and 5, it is clear that the problem solving qualities are improved with a larger population size. However, when considering the performances of Sys5 and Sys6 in Table 4 together, it is also clear that, within the same CPU time limit, the improvements obtained by increasing the population size by a factor of 10 are not as significant as, or are similar to those obtained by just switching off parent selection pressure.

6. CONCLUSIONS AND FUTURE WORK

Stochastic elements exist in the parent selection process and the breeding process. Some stochastic elements need to be removed in order to distinguish the genetic search algorithm from a random search algorithm. On the other hand some stochastic elements must be retained in order to prevent the genetic search from being confined in local optima or converging prematurely.

Selection pressure on the parent selection removes some stochastic elements. After local search techniques are integrated into the breeding process, stochastic elements are further eliminated. The change was suggested as effective in [10, 14]. However, this paper observed different results by investigating four groups of GP systems involving two simulations of constructive crossover operators. The results show that stochastic elements cannot be removed in both processes based on the performance comparison under a fair situation in our experiments. The effectiveness of integrating local search techniques into the breeding process will be significantly reduced, and become worse than that of the standard breeding process if no corresponding change is made to selection pressure on the parent selection.

If stochastic elements are minimised or optimised in the parent selection process, for instance tuning the tournament size, it is better to keep some stochastic elements in the

Table 5: Performance of systems with population size of 1000 using 3 different crossover modes.

GP Systems	Parent Selection Pressure	Xover Mode	EvePar		SymReg	BinCla
			Completion	Actual Time (sec)	RMS Error	Test Error rate (%)
Sys1	On	standard	80%	125 ± 155	45.2 ± 7.0	5.3 ± 0.7
Sys2		partial	90%	70 ± 113	45.5 ± 4.7	4.0 ± 0.7
Sys3		ideal	54%	381 ± 45	40.8 ± 4.4	4.0 ± 0.5

breeding process, for instance using the standard blind random crossover operator. On the other hand, if stochastic elements are minimised or optimised in the breeding process, for instance performing enough intensive searching in successor states of chosen parents, then it is better to keep some stochastic elements in the parent selection process, for instance selecting parents randomly for crossover.

Integrating local search techniques into the breeding process is a good practice but effective only when selection pressure on the parent selection has been carefully adjusted. This paper shows that, in our experiments, taking off selection pressure on the parent selection correspondingly is a good practice for obtaining a significant performance improvement by effectively maintaining population diversity and avoiding local optima or premature convergence.

This paper also shows that when no selection pressure applied to parent selection, a good strategy of optimising stochastic elements in the breeding process is Partial Xover instead of the exhaustive Ideal Xover. However, further investigations are required in order to make a more general conclusion. More precisely, we will further investigate an optimal degree at which the intensive neighbourhood search can provide the most effective GP system in general.

This paper only investigated the primary genetic operator in the context of many-offspring breeding. We will further conduct similar studies on mutation in order to provide a complete picture.

In all evolutionary algorithms the selection mechanism considers the quality of the individuals as the basis to make choices. Therefore only the fitness values are used. The actual representations of solutions are irrelevant and information on the "inside" of an individual is not considered [3]. Our further research question is whether there are any other smart heuristics, such as the *semantics* in solutions, which can be used to guide a generate-and-test search in the breeding process in order to save the computational cost.

Acknowledgment

We would like to thank Neale Ranns for his wonderful GP package (Gouda), his endless technical support, and useful discussions.

7. REFERENCES

[1] T. Blickle and L. Thiele. A mathematical analysis of tournament selection. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 9–16. Morgan Kaufmann, 1995.

[2] C. B. D.J. Newman, S. Hettich and C. Merz. UCI repository of machine learning databases, 1998.

[3] A. E. Eiben and C. A. Schippers. On evolutionary exploration and exploitation. *Fundamenta Informaticae*, 35(1-4):35–50, 1998.

[4] S. M. Gustafson. *An Analysis of Diversity in Genetic Programming*. PhD thesis, University of Nottingham, 2004.

[5] K. Harries and P. Smith. Exploring alternative operators and search strategies in genetic programming. In J. R. Koza, et al, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 147–155, Stanford University, CA, USA, 1997. Morgan Kaufmann.

[6] J. R. Koza. *Genetic Programming — On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, 1992.

[7] J. R. Koza. A response to the ML-95 paper entitled "Hill climbing beats genetic search on a boolean circuit synthesis of Koza's". Distributed 11 July 1995 at the 1995 International Machine Learning Conference in Tahoe City, California, USA, 11 July 1995.

[8] K. J. Lang. Hill climbing beats genetic search on a boolean circuit synthesis of Koza's. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995. Morgan Kaufmann.

[9] S. W. Mahfoud. Crowding and preselection revisited. In R. Männer and B. Manderick, editors, *Parallel problem solving from nature 2*, pages 27–36, Amsterdam, 1992. North-Holland.

[10] H. Majeed and C. Ryan. A less destructive, context-aware crossover operator for GP. In P. Collet and et al, editors, *Proceedings of EuroGP 2006*, volume 3905 of *LNCS*, pages 36–48. Springer-Verlag, 2006.

[11] P. Nordin and W. Banzhaf. Complexity compression and evolution. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference*, pages 310–317, Pittsburgh, PA, USA, 15-19 1995. Morgan Kaufmann.

[12] P. Nordin, F. Francone, and W. Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In J. P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 6–22, 1995.

[13] S. J. Russell and P. Norvig. *Artificial Intelligence — A modern approach*. Pearson Education, New Jersey, US, 2nd edition, 2003.

[14] W. A. Tackett. *Recombination, selection, and the genetic construction of computer programs*. PhD thesis, University of Southern California, Los Angeles, CA, USA, 1994.

[15] M. D. Terrio and M. I. Heywood. On naive crossover biases with reproduction for simple solutions to classification problems. In K. Deb, et al, editors, *GECCO-2004, Part II*, volume 3103 of *LNCS*, pages 678–689, Seattle, WA, USA, 2004. Springer-Verlag.