# Search Operator Bias in Linearly Structured Genetic Programming

Garnett C. Wilson and Malcolm I. Heywood

Faculty of Computer Science
Dalhousie University, Halifax, NS B3H 1W5
{gwilson, mheywood}@cs.dal.ca

**Abstract.** GA solutions to the job-shop scheduling problem demonstrate that significant amounts of code context exist. Such observations have led to the introduction of biased search operators. In this work, we recognize that similar conditions exist in linearly structured GP (L-GP). An empirical study is made when biased search operators are applied to the San Mateo Trail (strategy), Two Box (regression), and Liver Disease (classification) benchmark problems. A preference is observed for biased mutation alone in the case of the regression problem, whereas the strategy and classification problems appear to prefer the combination of both biased mutation and crossover.

## 1 Introduction

Within the context of genetic algorithms (GA) applied to the Job-Shop Scheduling problem with an indirect encoding, biasing of the mutation (crossover) operator to occur towards the beginning (end) of the individual has resulted in significant performance gains [1]. Such a set of biases is based on the realization that genes towards the beginning of the individual effectively set the context for genes that follow. Thus, a code sequence towards the end of one individual may perform poorly under one context, but much better under a different context. Likewise, fitness may be significantly biased by the early location of a suboptimal initial code sequence that rapidly establishes itself across the population. In this work, we are interested in identifying whether similar observations hold for the case of linearly structured Genetic Programming (L-GP). To do so an empirical investigation is made over a set of benchmark problems taken from strategy learning (San Mateo Trail), classification (Liver) and regression (Two Boxes). We observe that a preference for biased mutation and crossover appears in the classification and strategy-learning problem, whereas the regression problem has a distinct preference for biased mutation alone.

The remainder of the paper is organized as follows. Section 2 introduces the form of L-GP employed here. The rationale for biasing of the operators in L-GP is then discussed. The performance measures we use to compare the performance of the implementations are introduced. Sections 3 to 5 detail results under each of the three problem categories (strategy, regression and classification). Finally, conclusions are drawn in Section 6.

## 2 Linearly structured Page-Based GP

### 2.1 Structure of an L-GP Individual

Linearly structured GP utilizes the same representation as a GA in which the gene alleles are defined over a range of integers [2]. Integers are now mapped into a set of permitted instructions, in this case a register addressing language (architecture independent assembly language). Operations are therefore defined in terms of a set of general-purpose registers (GPR) and operations (opcodes). Special purpose registers represent inputs (read only registers) and program counter. Naturally, the degree of problem decomposition is a function of the number of GPRs. Crossover merely exchanges (linear) sequences of integers (i.e. instructions) and mutation is applied at the instruction level. Results are assumed to always lie in GPR R0. In this work the page-based variant of L-GP is employed, where this fixes the locus of the crossover operator. Equal linear sequences of instructions are thus exchanged during crossover resulting in fixed length individuals. The number of instructions per page is allowed to vary in accordance with a cyclic annealing heuristic (dynamic paging) [3].

Upon initialization, individuals are described in terms of the number of pages (16 or 32) and the maximum number of instructions per page (4). The fixed length representation requires that individuals be initialized over the range {1,…, MaxPages}. Initialization of instructions is performed with uniform probability over the set of permitted instructions. Table 1 describes the parameterization of L-GP that is common to all three problems investigated. The 'swap' operator merely exchanges two instructions from the same individual with uniform probability.

**Table 1. Common L-GP Program Parameters**

| | |
|---|---|
| Tournament Style | Steady State |
| Tournament Size | 4 |
| Maximum Tournaments | 50,000 (≡ 50 generations of pop. of 4,000) |
| Probability of Mutation | 0.5 |
| Probability of Crossover | 0.9 |
| Probability of Swapping | 0.9 |
| Number of Registers | 4 or 8 |
| Maximum Number of Pages | 16 or 32 |
| Size of Pages | 1, 2, or 4 with dynamic paging |
| Size of Individuals at Initialization | 4 × {1, …, MaxPages} (MaxPages = 16 or 32) |
| Experiments | 50 independent runs |

### 2.2 Search Operator Biasing

The rationale for biasing the operators of crossover and mutation was first identified by Fang *et al*. [1] for the specific case of job-shop scheduling. In this case a sequence

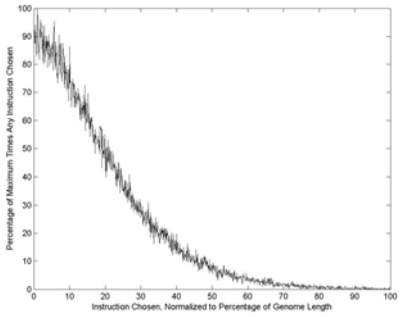
of integers was decoded to represent the order in which tasks of a job-shop manufacturing process are assigned to resource. It was recognized that the sequence of assignments at the beginning of the individual would set up context for the genes at the end of an individual. Moreover, premature convergence by the genes at the beginning of the individual was rarely corrected by crossover, whereas mutation was far more effective. Likewise, gene sequences at the end of an individual might perform better under one individual than another (depending on the initial gene sequences), making crossover more applicable.

In this work we are interested in establishing whether the linear execution of instructions in L-GP may also benefit from the application of biased search operators. To do so, rather than defining the application of a crossover (mutation) operator with uniform probability over the length of an individual, a bias towards the end (beginning) is introduced. Such a bias is defined in terms of the decaying region of the Gamma distribution, where the general form for the Gamma Function Γ(a) is,

$$\Gamma(a) = \int_0^\infty \exp(-t)t^{a-1}dt$$

where '*a*' is the shape parameter (takes the value of 2 in this work). Figure 1 provides an example of the resulting (discrete) distribution of random variants, as based on 90,324 samples.

**Fig. 1.** Distribution of times instructions in the genome are chosen based on Truncated Gamma probability density function.




Naturally, mutation uses the random variant as defined (i.e. highest probability at the beginning of the individual) whereas the case of crossover remaps the probability function about the mid point of the individual. In the following study the biased operators are investigated alone and together, and compared against unbiased selection of crossover/ mutation.

### 2.3 Performance Measures

In the San Mateo Trail and Two Boxes problem [4], performance measures include: the length of the best individual evolved, the number of problems solved out of 50 independent trails, and computational effort (also called "E-measure"). These measures

tell us solution succinctness, how good the algorithm is at solving the problem, and how hard the problem is for the algorithm, respectively.

The Liver Disease classification problem [5] does not lend itself to an analysis where the solution is found following a degree of computational effort (perfect solutions are not returned). Instead, the GP tournament typically runs for the full number of tournaments and then the best performing individual attempts to classify a test set of data. The success of the algorithm is measured, then, on the number of correct cases the best individual gets right in the test set. The succinctness of the solution is also provided by solution length.

## 3 San Mateo Trail Strategy Learning Problem

### 3.1 Problem Definition and Implementation

The Koza [4] implementation of the San Mateo Trail Problem is followed, where an ant must learn to follow a trail of food items on nine 13 × 13 grids (fitness cases) where the trail has five different types of discontinuity. Touching the edge of one of these grids terminates a fitness case, and moving onto a square containing food throws the program execution back to the beginning. The ant can spin left or right 90 degrees or move forward. There are 96 pieces of food altogether on the trail, and the raw fitness is the number of food pieces eaten. An ant moves between the separate grids of the trail (fitness cases) if the ant either touches the electrical fence (touches the edge), makes a total of 120 right or left turns, or moves 80 times. The ant always begins each grid in the middle of the top row, facing south. The food eaten in the nine parts of the grid is totaled and the ant's goal is to eat as much food as possible. Table 1 details the generic GP parameters, and Table 2 provides the problem specific parameters.

**Table 2.** Program implementation details for the San Mateo Trail.

| | |
|---|---|
| Objective | Eat 96 pieces of food on nine 13 × 13 grids |
| Population size | 125 |
| Terminal Set | {RIGHT, LEFT, MOVE} |
| Instruction Set | {fetch-from-register, load-into-register} |
| Fitness | Number of 96 food pieces eaten |
| Termination | 96 food pieces eaten (success) or maximum tournaments |

### 3.2 San Mateo Trail Results

Experiments are conducted across the two maximum page limits, Table 1, for all four variations of the search operators. The E-measure (computational effort) results are summarized in Figure 2, and Table 3 provides the percentage of 50 trials where the solution converged. It is apparent that the computational effort for individuals with

the lower page count is best when no bias is employed; whereas the higher page count favors the combination of biased crossover and mutation. Figure 3 indicates that solution length is best when only crossover is used for longer individuals, but at a cost of the highest computational effort (Figure 2). Table 3 shows that all combinations appear to reliably locate solutions.

**Fig. 2.** E-measure 1st, 2nd, 3rd quartile results for the 4 register San Mateo Trail problem.

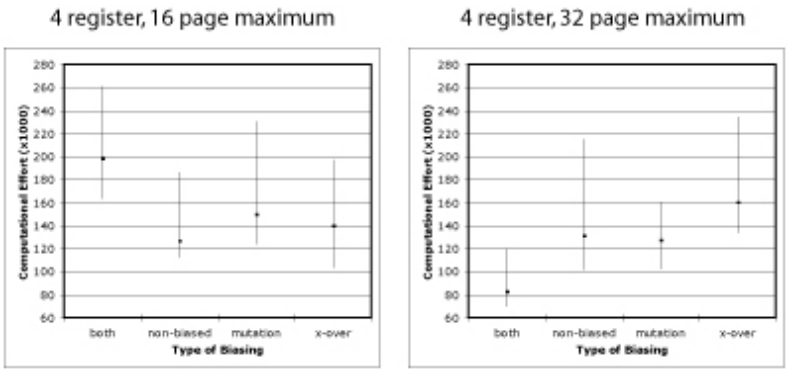



**Fig. 3.** Solution length 1st, 2nd, 3rd quartile results for the 4 register San Mateo Trail problem.

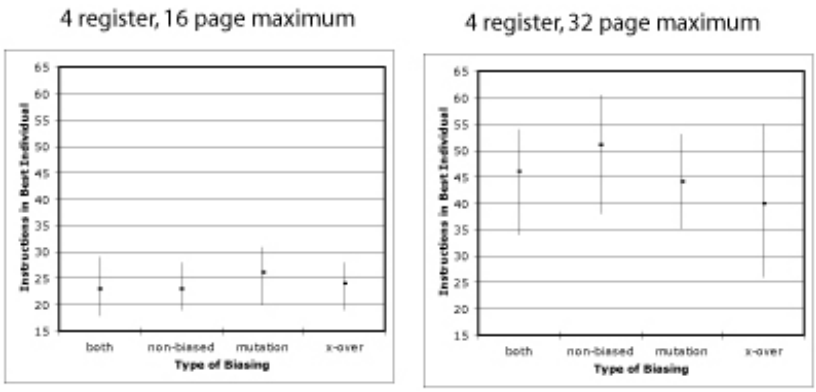



**Table 3.** Percentage of converging trials for the 4 register San Mateo Trail problem.

| | Mutation | | Xover | | Both | | Neither | |
|---|---|---|---|---|---|---|---|---|
| Maximum # of Pages | 16 | 32 | 16 | 32 | 16 | 32 | 16 | 32 |
| % Solutions (50 trials) | 94 | 98 | 98 | 98 | 90 | 98 | 94 | 100 |

## 4 Two Boxes Regression Problem

### 4.1 Problem Definition and Implementation

The Two Boxes problem relates six independent variables ($L_0$, $W_0$, $H_0$, $L_1$, $W_1$, and $H_1$) through a difference in the volumes of two boxes ($L_0W_0H_0 - L_1W_1H_1$). Ten fitness cases are created through uniform sampling of integers over the interval {1, ..., 10}. Table 4 details the problem specific parameterization. Fitness criteria are measured in terms of the number of hits (absolute error ≤ 0.01). Arithmetic operators are naturally protected against overflow and underflow.

**Table 4.** Program implementation details for Two Boxes.

| | |
|---|---|
| Objective | Fit equation to $x_1x_2x_3 - x_4x_5x_6$ |
| Population size | 500 |
| Terminal Set | $\{ x_1, x_2, x_3, x_4, x_5, x_6 \}$ |
| Functional Set | {+, -, *, %} |
| Instruction Set | {fetch-from-register, load-into-register} |
| Fitness Cases | 10 random values selected over {1, …, 10} |
| Fitness | Summed absolute error |
| Hits | Number of fitness cases with absolute error ≤ 0.01 |
| Termination | 10 hits (success) or maximum tournaments |

### 4.2 Two Box Results

The 4 register E-measure (computational effort) results are summarized below in Figure 4, where a distinct preference for mutation bias alone is apparent. Table 5 provides the percentage of 50 trials with a solution that converged, where it appears that the combination of biased crossover and mutation was significantly worse than the other three cases. No specific preference is apparent for the length of the solutions, Figure 5.

**Fig. 4.** E-measure 1st, 2nd, 3rd quartile results for the 4 register Two Boxes problem.

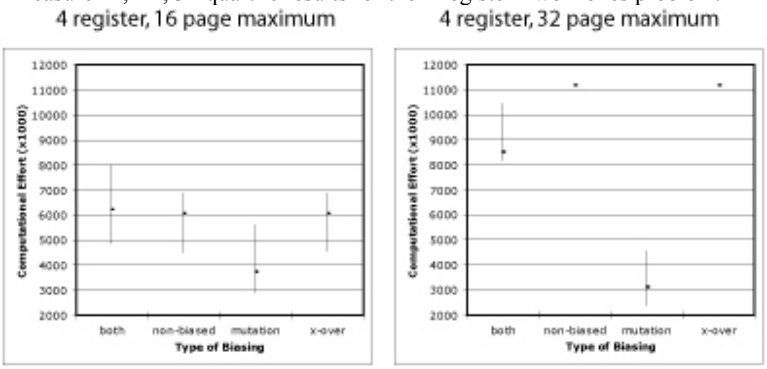


**Fig. 5.** Solution length 1st, 2nd, 3rd quartile results for the 4 register Two Boxes problem.

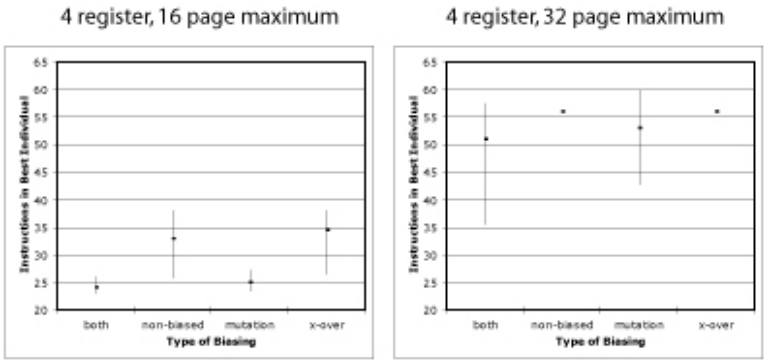



**Table 5.** Percentage of converging trials for the Two Boxes problem.

| | Mutation | | Xover | | Both | | Neither | |
|---|---|---|---|---|---|---|---|---|
| Maximum # of Pages | 16 | 32 | 16 | 32 | 16 | 32 | 16 | 32 |
| % Solutions (50 trials) | 12 | 8 | 12 | 2 | 8 | 6 | 12 | 2 |

# 5 Liver Disease Classification Problem

## 5.1 Problem Definition and Implementation

The Liver Disease Classification problem is divided into training (75%) and test (25%) sets [5]. The problem specific parameters are summarized below in Table 6.

**Table 6.** Program implementation details for the Liver Disease problem.

| | |
|---|---|
| Objective | Find a program to classify the Liver Disease Classification problem |
| Population size | 125 |
| Terminal Set | $\{ x_1, x_2, x_3, x_4, x_5, x_6 \}$ |
| Functional Set | {+, -, *, %, sin, cos, sqrt, $arg^2$-1} |
| Instruction Set | {fetch-from-register, load-into-register} |
| Wrapper | IF output > 0.0 AND label = 1 THEN correct class<br>ELSE IF output ≤ 0.0 AND label = 0 THEN correct class |
| Fitness Cases | 259 training, 86 test |
| Fitness | Number of correct classifications. |
| Hits | Same as fitness. |
| Termination | 259 hits or maximum tournaments |

### 5.2 Liver Disease Classification Results

Performance is measured by the percentage of test cases correctly classified after training is completed by the best case training individual. Figure 6 indicates that for training cases, regardless of maximum page number, non-biased and biased for cross-over performed equally and were the best performers. On classification of the test set, Figure 7 shows that biasing of both operators produced the best solutions for the problem for both maximum page restrictions. Figure 8 indicates that solution lengths varied little across parameterizations.

**Fig. 6.** Percentage of correctly classified training cases in the Liver Disease problem – 1st, 2nd, 3rd quartiles.

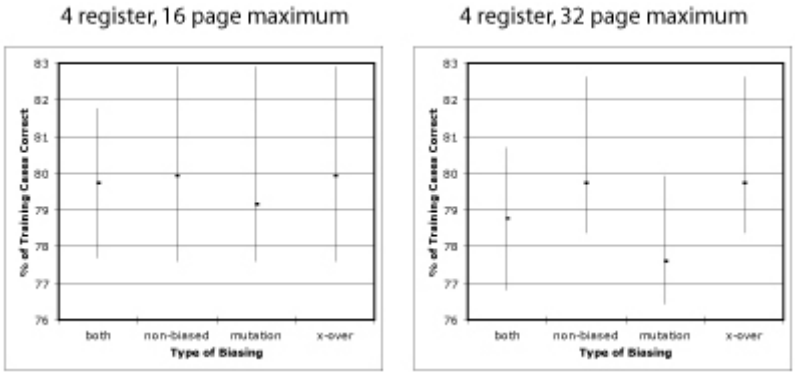



**Fig. 7.** Percentage of correctly classified test cases in the Liver Disease problem – 1st, 2nd, 3rd quartiles.

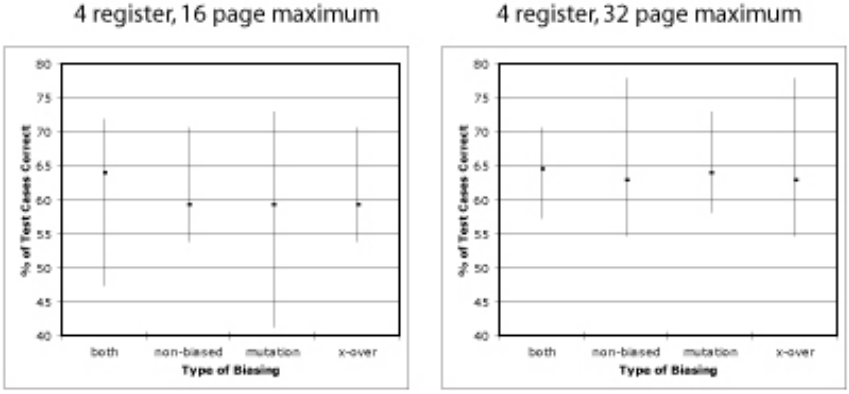
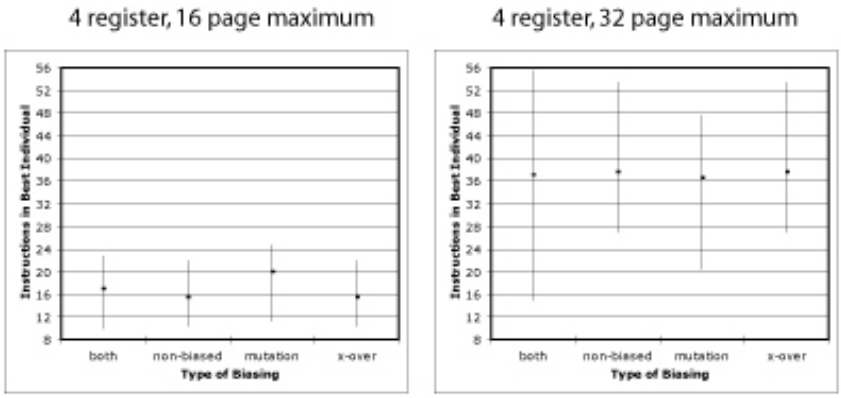


**Fig. 8.** Solution length in number of instructions for the 4 register Liver Disease problem.




## 6 Discussion and Conclusion

Linearly structured GP executes instructions in sequence from a program counter of 0 to the maximum value for the program counter (in this case the number of instructions in an individual). We conjectured that this linear execution of instructions resulted in instructions at the beginning of the program setting context for the code to follow. Although code evolved by GP does not support the notion of code context at the beginning of the program relating to instructions later in the program as directly as human authored code, our experiments with biased search operators do support the existence of code context in L-GP. Moreover, a specific preference for biased mutation (i.e. mutation more frequently applied to the beginning of the individual) was observed for the case of the Two Boxes regression problem. The strategy learning and classification problems both appeared to support the use of biased crossover (i.e. more frequent application to the end of the individual). This work provides an exhaustive investigation of four biasing combinations over three distinct benchmark problem types with respect to both computational effort and problem-solving success, and represents the first confirmation of code context through operator biasing in L-GP. Naturally, future work will address a wider study in order to establish the generality of the particular biasing preferences for the three problem-type sets (strategy-learning, regression, and classification).

## Acknowledgements

The authors gratefully acknowledge the support provided by an NSERC PGS-B and Honorary Izaak Walton Killam Scholarship (Garnett Wilson), and CFI New Opportunities and NSERC research grants (Dr. M. Heywood).

## References


1. Fang, H.-L., Ross, P., Corne, D.: A Promising Genetic Algorithm Approach to Job-Shop Scheduling, Rescheduling, and Open-Shop Scheduling Problems. Proc. of the Fifth International Conference on Genetic Algorithms (1993) 375-382
2. Cramer N.L.: A Representation for the Adaptive Generation of Simple Sequential Programs. Proceedings of the International Conference on Genetic Algorithms and Their Application. (1985) 183-187
3. Heywood M.I., Zincir-Heywood A.N.: Dynamic Page-Based Crossover in Linear Genetic Programming. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics. 32(3) (2002) 380-388
4. Koza, J. R.: Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, Cambridge (1994)
5. UCI machine learning repository: http://www.ics.uci.edu/~mlearn/MLRepository.html