On Random Numbers and the Performance of Genetic Algorithms

Erick Cantú-Paz
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94550
cantupaz@llnl.gov

Abstract

Pseudo random number generators (PRNGs) are the basic input to the stochastic selection, recombination, and mutation operations of genetic algorithms (GAs). Although it does not seem like a crucial decision, recent studies suggest that the choice of PRNG can affect the performance of GAs. The objective of this paper is to study the effect of PRNGs on a simple GA, and to identify the components that are most affected by the PRNG. The paper presents ablation experiments using two PRNGs and true random numbers from an atmospheric noise source. The experiments show that the PRNG used to initialize the population is critical, but the PRNG used as input to other operations does not affect the performance significantly. We confirmed these results with additional experiments that isolated single components of the GA. In a few cases, we obtained improved results with a poor PRNG, but we were unable to obtain improvements consistently across the test functions used or with different seeds. The results suggest that, in accordance with common practice in other fields, it is preferable to use the best PRNG available to avoid muddling the interpretation of the results.

1 INTRODUCTION

A basic component of genetic algorithms (GAs) is the pseudo-random number generator (PRNG) that provides input to the stochastic selection, recombination, and mutation operations. It is well known that the performance of GAs is greatly influenced by the solution encoding, population size, and choice of operators, and it may appear that the choice of PRNG is relatively unimportant. However, several studies show that the performance of evolutionary algorithms can be affected by the choice of PRNG. In genetic programming (GP), Daida et al. (1997, 1999) found surprising improvements (ranging from 36% to 800%) on different performance measures when a *poor* PRNG was used. Meysenburg and Foster (1999a) found similar but smaller differences in GP performance. In GAs, Meysenburg (1997) and Meysenburg and Foster (1997) found that, in very few cases, a poor PRNG resulted in modest performance improvements, but they found no evidence of better GA performance with good PRNGs. Later, Meysenburg and Foster (1999b) found additional evidence of poor PRNGs causing slightly better GA performance, and also found that good PRNGs caused worse performance in isolated cases.

Our own experiments show that small variations in the PRNG can cause large deviations in the GA's performance. Consider the example in figure 1. A simple GA is optimizing a fitness function formed by concatenating 13 copies of an 8-bit trap function (defined later). The first graph shows the average fitness value reached at the end of the experiments vs. the population size. The only difference among the four plots is the random number generator used. The two overlapping plots in the middle show the results using a good PRNG (a Mersenne Twister) and true random numbers (from an atmospheric noise source); the top and bottom plots were obtained with a *poor* PRNG seeded in two different ways. In some cases, the performance with the poor PRNG seeded with an arbitrarily chosen constant (10) is 35% better than with a good PRNG and 100% better than itself seeded with random numbers (bottom plot). We observed similar trends with 7- and 9-bit traps, but we found no significant differences using other seven test functions. The second graph shows the number of trap functions that were solved to optimality (a performance measure strongly correlated to fitness) vs. the population size.

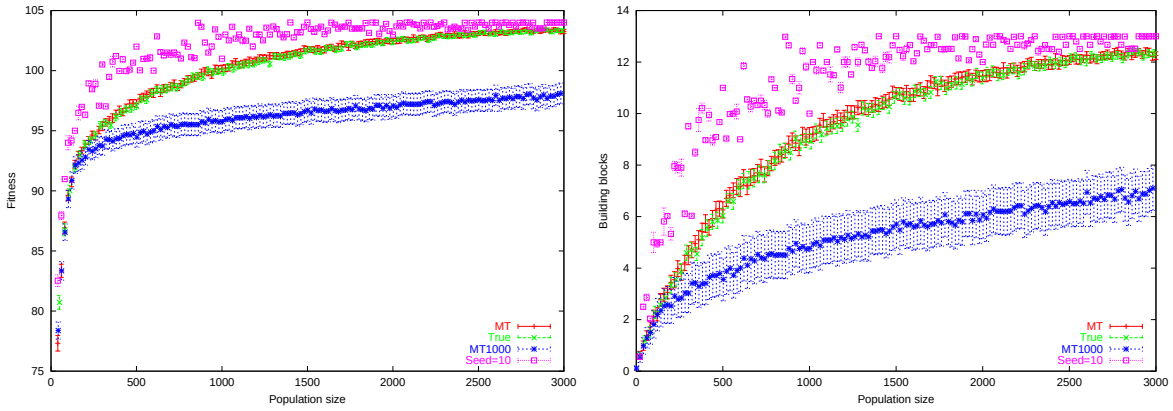


Figure 1: Example of different performance using different sources of (pseudo)random numbers. The error bars represent 95% confidence intervals. "MT" denotes experiments with a Mersenne Twister (MT), "True" denotes experiments with a source of true random numbers, "MT1000" refers to the MT with its period limited to 1000, and "Seed=10" is the limited MT initialized with the constant 10.

The objective of this paper is to continue the study of the effect of PRNGs on GAs. We used ablation experiments and simple fitness functions with known characteristics to isolate the stochastic components of the GA where the source of random numbers causes the greatest difference. In the ablation experiments, we used a "poor" PRNG as input to one of the stochastic components of the EA (say selection) while using a "good" PRNG for the rest of the algorithm. We performed additional experiments isolating single components of the GAs and compared the deviation of the algorithms to their expected behavior, which was calculated using existing models. We also performed experiments that included a source of true random numbers, but we found no difference in GA performance when compared to a "good" PRNG. We believe that this is the first time true random numbers have been used in GAs. The study was limited to simple genetic algorithms with fixed-length binary strings and popular operators.

The results of this study show that the PRNG used to initialize the population is critical to the performance of the GA, but the PRNG used as input to other GA operations does not affect the performance significantly. The experiments also show that, at least for the test functions used here, the choice of PRNG can cause large variations in performance (much larger than previously reported for GAs). Therefore, users and researchers of GAs should choose the PRNGs and their seeds carefully and report these choices appropriately, as has been advocated elsewhere (Daida et al., 1997; Daida et al., 1999).

The remainder is organized as follows: The next section describes the PRNGs and the source of true random numbers used in this study; section 3 describes the experiments and presents the results; and finally, section 4 summarizes the findings, issues recommendations, and suggests opportunities for future work.

2 (PSEUDO)RANDOM NUMBER GENERATORS

The consensus in many communities interested in stochastic simulations is to use the best PRNG available. Using the best PRNG helps to ensure that the results of a stochastic simulation are, in fact, a product of the algorithm and its inputs, and not an artifact of the PRNG.

The first PRNG used here is the Mersenne Twister (MT) (Matsumoto & Nishimura, 1998), which is considered to be one of the best PRNGs currently available (it has a period of $2^{19937}-1$ and is equidistributed in 623 dimensions). We used the implementation from the GNU Scientific Library version 0.4. In particular, this implementation uses the corrected seeding procedure recommended by the MT authors. The second PRNG used is also an MT, but its period has been artificially limited to 1000 numbers by re-seeding the generator every thousand calls with the original seed. We refer to the second PRNG as MT1000. Meysenburg and Foster (1999b) used similar generators in their experiments. In contrast to other studies that compared EA performance using numerous PRNGs, the experiments in this paper use only two generators that represent extremes in PRNG quality. This choice was motivated from the observations of Meysenburg

and Foster (1999b), where improvements in GA performance were observed only with a very poor PRNG, and no evidence of performance difference was found using other relatively good PRNGs.[1]

In addition to the two PRNGs, we use *true* random numbers obtained from an atmospheric noise source. These random numbers are available at www.random.org along with a description of the method used to create them. Briefly, a radio was tuned to a frequency where no one was transmitting, and the noise received was fed to a workstation where it was sampled as an 8-bit signal at 8KHz. The upper 7 bits of each sample were discarded, and the remaining bits were subject to a simple skew correction to ensure an even distribution of ones and zeroes.

To create our true random generator, we concatenated the four pregenerated 10Mb files available at www.random.org. These files are essentially streams of random bits that need some preprocessing before using them in a GA. The basic output from our PRNGs are uniform random numbers in $[0, 1]$. To obtain the same from the true random file, our C++ program read 4 bytes at a time into unsigned long (32 bit) integers and divided them by 2^{32}.

Unless specified otherwise, we initialized the PRNGs with 32-bit random integers obtained from the first 1Mb file from random.org. As our experiments below confirm, the initialization of the PRNGs—especially MT1000—was critical to the performance of the GA.

3 EXPERIMENTS

3.1 METHODS

The experiments used deceptive trap functions, which are used in numerous studies of genetic algorithms because they have known properties and their difficulty can be regulated easily (Deb & Goldberg, 1993). The values of the deceptive functions depend on the number, u, of bits set to one in their k-bit input substring. The fitness increases with more bits set to zero until it reaches a local optimum, but the global maximum is at the opposite extreme where all the bits in the input are set to one. The order-k traps are defined as

$$f_k(u) = \begin{cases} k - u - d & \text{if } u < k, \\ k & \text{if } u = k, \end{cases} \quad (1)$$

where d is the fitness difference of the two peaks, which in our case is always set to one. The trap functions become more difficult by increasing k and decreasing d. We varied k from 3 to 10. The fitness functions are formed by concatenating fully-deceptive trap functions and adding their individual contributions. We decided to set the length of the individuals to $l = \lceil 100/k \rceil * k$ bits (i.e., 100 bits or the smallest integer multiple of k larger than 100). For example, for the 6-bit trap problem, the individuals are $l = 102$ bits long and their fitness is calculated as $\sum_{i=0}^{16} f_6(u_{6i})$, where u_{6i} denotes the number of ones in the substring that starts at position $6i$.

The results reported here are from a simple GA with fixed-length binary encoding, pairwise tournament selection without replacement, one-point crossover with probability 1.0, and point-wise mutation with probability $1/l$. The population size for the 3,4,5,6-bit traps varied from 2 to 300 in steps of 2. For the 7-bit problem, the population size varied from 10 to 1000 in steps of 10, and for the 8,9,10-bit problems the population varied from 20 to 3000 in steps of 20. The experiments were terminated after 500 generations.

All the results were obtained repeating each experiment 100 times with different random seeds, and two-sided z tests with $\alpha = 0.05$ were used to verify if the observed means were different. The PRNGs were called each time that a random number was needed by the GA. For example, the PRNG was called once for each bit in the initial population (instead of, say, using the 32 bits returned by the PRNG to initialize 32 genes).

Our performance measure is the number of substrings that converged to the global optimal value (all ones) at the end of each run. We refer to these correct substrings as building blocks. This performance measure is adequate for the trap test functions, because the number of optimal subfunctions is a binomially distributed random variable that can be well approximated with a normal, which is what the z test assumes. This performance measure is strongly correlated with the fitness, as can be observed by comparing the two graphs in figure 1. This performance measure also allows us to calculate easily the expected behavior of the algorithm in some experiments below.

3.2 TRUE RANDOM NUMBERS

The first set of results compares the performance of the GA using the true random numbers with the MT and MT1000 PRNGs. For brevity we present only the results for 3-,4-,7-, and 10-bit problems in figure 2. The results for the 6-bit trap are similar to the 3-bit

[1] A short period is only one of the possible shortcomings of a PRNG: Correlations between consecutive samples and structural properties (such as the organization of the pseudorandom numbers in lattices) were not considered here.

problem: there are no noticeable differences, except in a few cases at relatively large population sizes, where the GA using the MT1000 performs worse. For the problems with 4,5,8,9, and 10 bit traps, the GAs using MT1000 perform noticeable worse than the GAs with true random or MT, except for small population sizes. The 7-bit problem has a similar behavior, but the transition to worse results appears at relatively large populations. These results contrast with some previous studies (Meysenburg & Foster, 1997; Meysenburg & Foster, 1999a) that showed that, in general, the performance of GAs was not adversely affected by poor PRNGs, and that sometimes poor PRNGs resulted in better results.

As mentioned in the introduction, the results in figure 1 use an 8-bit trap function. The overlapping middle plots correspond to the MT and the true random numbers. The bottom plot was generated with the MT1000 PRNG seeded with the random numbers as described in the previous section. The top plot was also generated with the MT1000 PRNG, but the seed was arbitrarily chosen to be the constant 10. Surprisingly, this poor PRNG with an arbitrary seed often outperforms all the other algorithms. We observed similar trends with the 7, and 9-bit problems, but with the other functions the results between the two MT1000 were statistically indistinguishable. While these results are intriguing, the performance with other arbitrarily chosen seeds was much worse than the *bottom* plot in figure 1. In essence, the best results were obtained by chance, and poor PRNGs do not seem to offer an advantage in general.

For all population sizes and in all problems tested, the GAs with the MT generator and the true random numbers performed equally well (there was not a single statistically significant difference). Therefore, in the following experiments we omit the results with the true random numbers.

3.3 ABLATION EXPERIMENTS

To further analyze the cause of the poor performance of GAs with MT1000, we performed ablation experiments. We started with a GA that uses MT for its four randomized components (initialization, selection, crossover, and mutation) and substituted MT1000 in each of these components at a time as specified in table 1. We also included the results where MT1000 is always used. Figure 3 has the results of this study for the 4-, 5-, 7-, and 10-bit functions. To minimize the clutter, the graphs omit the results for mutation (exp. 5), which did not differ from the results with MT.

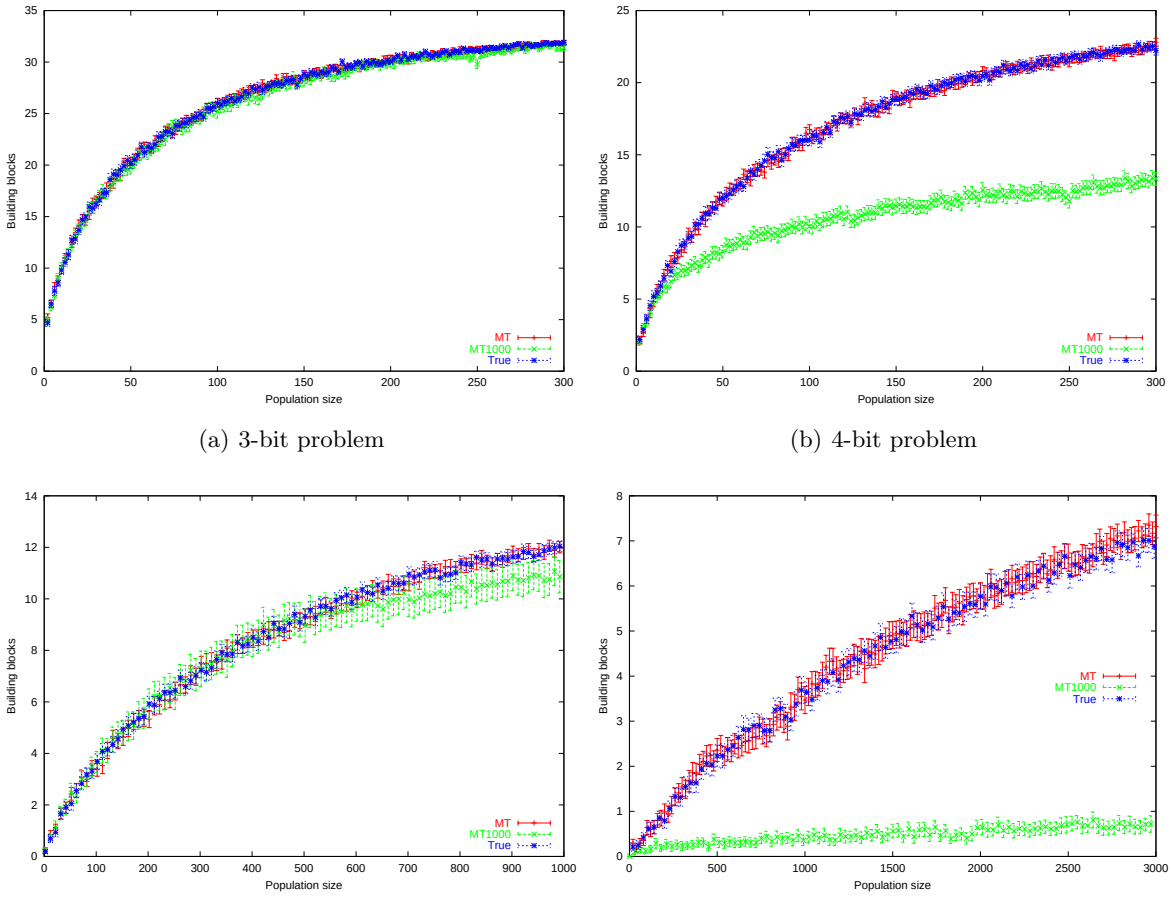
These results clearly show that the performance of the GA that used MT1000 to initialize the population (exp. 2) is strongly correlated with the performance of the GA that uses MT1000 for all its operations (exp. 6). This suggests that the initialization of the population is critical for the adequate performance of the GA, but the PRNG used in selection, crossover, or mutation seems unimportant. To provide additional support for this hypothesis, we performed an experiment where the population was initialized with MT, and the rest of the GA operations use MT1000 (experiment 7 in table 1). The results are also plotted in figure 3, and are not significantly different than those of the GA that uses MT exclusively.

Exp.	Init.	Sel.	X-over	Mut.
1	$\surd$	$\surd$	$\surd$	$\surd$
2	$\times$	$\surd$	$\surd$	$\surd$
3	$\surd$	$\times$	$\surd$	$\surd$
4	$\surd$	$\surd$	$\times$	$\surd$
5	$\surd$	$\surd$	$\surd$	$\times$
6	$\times$	$\times$	$\times$	$\times$
7	$\surd$	$\times$	$\times$	$\times$

Table 1: Ablation experiments setup. The $\surd$ and $\times$ represent the Mersenne Twister and the MT1000 PRNGs, respectively. Experiment 7 was used to verify the hypothesis that initialization was critical.

3.4 ADDITIONAL EXPERIMENTS

To try to understand why crossover and selection do not seem affected by the choice of PRNG, we performed additional experiments. To study crossover, we performed two different experiments. First, we fixed the population size to 100 and the number of generations until termination to 500. Using the 4-bit trap function and 100-bit long strings, we recorded the frequency that each possible crossover point was chosen. Ideally, we would expect that all points are chosen with the same frequency, since the probability of choosing each is uniform ($p = 1/(l-1)$). However, it is natural to expect some variability as the number of times a particular point is chosen is a random variable with a binomial distribution. Figure 4 presents the frequencies (sorted to aid in visualization) along with the expected frequency and 95% confidence intervals (calculated assuming that the binomial distribution can be approximated well with a normal). As we can see, the true random and the MT generator produce results that match our expectations, while using the MT1000 causes some crossover points to be chosen much more frequently than others. These results would suggest that the MT1000 is inadequate as input to crossover,



(a) 3-bit problem

(b) 4-bit problem

(c) 7-bit problem

(d) 10-bit problem

Figure 2: Performance of GAs using true random numbers and the MT and MT1000 PRNGs. The error bars represent 95% confidence intervals. The True and MT plots overlap, and are always at least as good as MT1000.

but in the ablation experiments we did not observe significant differences with good quality generators.

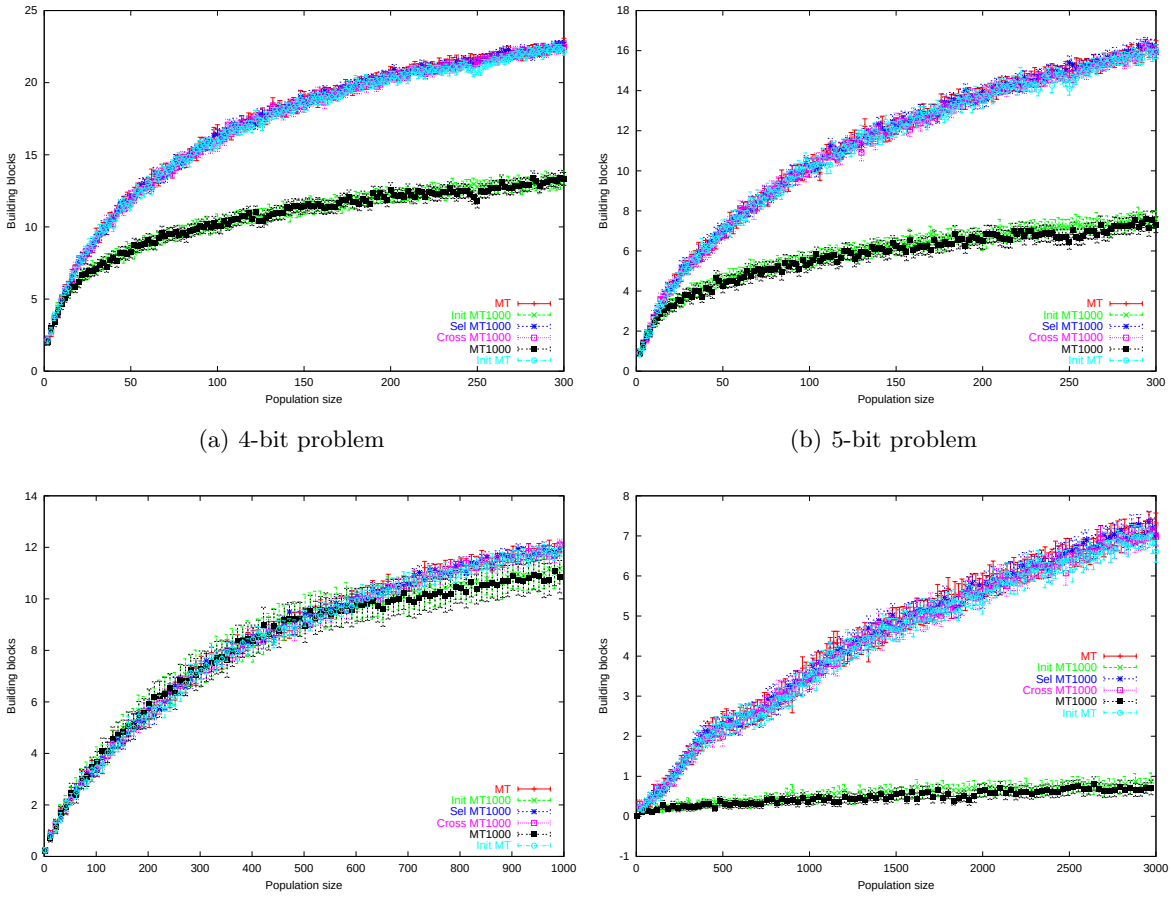
We did additional experiments recording the number of crossover operations that resulted in an offspring with at least one optimal subfunction more than each of the parents. Thierens and Goldberg (1993) call this occurrence a mixing event. As k increases, we expect fewer mixing events (all else being equal), and therefore the population was sized to 200 individuals for the functions with $k = 3, 4, 5, 6$ and to 1000 individuals for the remaining problems. The results in table 2 show that there are no significant differences in the number of mixing events using different sources of random numbers. This agrees with the ablation experiments, but it is puzzling that such a non-uniform distribution of crossover points has no apparent effect on our performance measure (or in fitness).

Separate experiments were done to investigate the effect of random inputs to selection. In particular, the following experiments verify if the expected fitness gain after selection matches the theoretical expectations. If the fitnesses are distributed normally, the mean fitness of the selected individuals can be calculated as (Mühlenbein & Schlierkamp-Voosen, 1993)

$$\mu_{\text{sel}} = \mu_{\text{orig}} + I\sigma_{\text{orig}}, \tag{2}$$

where μ_{orig} and μ_{sel} represent the mean fitness of the population before and after selection, I is the selection intensity, which in the case of pairwise tournaments is 0.5642 (Miller & Goldberg, 1995; Bäck, 1995), and σ_{orig} is the standard deviation of the fitness of the population before selection.

For these experiments, the GA used populations of 10000 individuals of length $100 * k$. We chose such a large population to ensure that MT1000 generator would cycle, and we used longer individuals to ap-



(a) 4-bit problem

(b) 5-bit problem

(c) 7-bit problem

(d) 10-bit problem

Figure 3: Performance of GAs in the ablation study. The error bars represent 95% confidence intervals. MT1000 and Init MT1000 are the two plots that overlap at the bottom; the rest of the plots overlap at the top.

proximate the assumed normal distribution. Since the fitness functions are summations of 100 random variables, it is reasonable to assume that the fitness of the initial population is distributed normally (the fitness of the selected population is certainly not normal).

We initialized the populations using the true random numbers to avoid any bias. Selection was driven by the true random numbers as well as the MT and MT1000 PRNGs. Pairwise tournament selection was applied once to the randomly initialized populations, and statistics of the selected individuals were recorded. Table 3 shows the expected and experimental results (averaged over 100 trials) of the mean fitness before and after selection. There are no statistically significant differences between the expected fitness value and the experimental results with the three sources of (pseudo)random numbers.

Finally, we performed experiments to determine the effect of PRNGs on the initialization of the population. We measured the number of optimal subfunctions in a randomly initialized population of size 1000. Table 4 has the average of 1000 repetitions using true random numbers and the different PRNGs initialized with random numbers. In addition, the table has results for MT1000 with a seed of 10, which produced the improvements in performance in figure 1. The length of the individuals was $l = \lceil 100/k \rceil * k$, and the expected number of optimal subfunctions is $l/(k * 2^k)$. Only MT1000 seeded with 10 shows significant deviations from the expected behavior. Note that MT1000 results have a higher variance (an order of magnitude) than the other generators.

k	True	MT	MT1000
3	945.48 (21.99)	947.92 (26.07)	933.42 (23.06)
4	755.87 (25.44)	770.13 (24.50)	742.82 (23.83)
5	553.56 (25.16)	554.77 (25.38)	558.03 (25.27)
6	376.56 (31.58)	372.46 (27.61)	356.67 (27.25)
7	2528.23 (107.95)	2621.69 (102.39)	2492.4 (109.20)
8	1882.13 (127.38)	1870.27 (137.06)	1806.73 (123.02)
9	957.94 (132.21)	928.89 (128.53)	847.57 (108.75)
10	421.65 (104.56)	436.84 (109.23)	423.89 (104.86)

Table 2: Number of mixing events. The numbers in parenthesis are the standard errors.

k	Original	Expected	True	MT	MT1000
3	99.9932	105.616	105.619	105.617	105.616
4	131.295	137.523	137.512	137.525	137.526
5	168.713	175.441	175.441	175.438	175.438
6	210.885	218.001	218.007	218.008	218.010
7	256.196	263.758	263.766	263.762	263.763
8	303.484	311.354	311.556	311.548	311.549
9	351.868	360.354	360.360	360.358	360.362
10	400.981	409.9156	409.921	409.911	409.924

Table 3: Population mean before and after selection driven by different PRNGs and true random numbers.

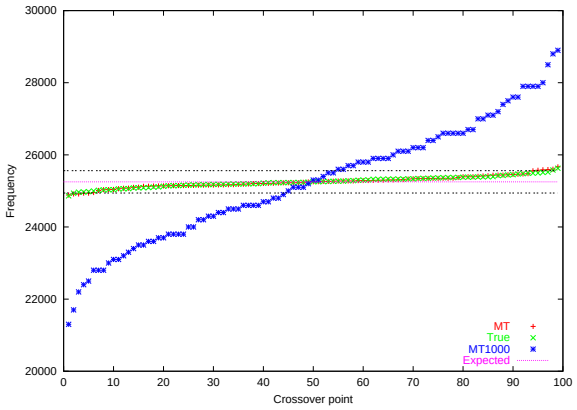


Figure 4: Frequency of choosing crossover points.

4 CONCLUSIONS

Previous studies have suggested that the choice of PRNG has a small effect on the performance of GAs. This paper presents additional experimental evidence of the effect of PRNGs on GAs, and the results suggest that the impact of the PRNG can be much more dramatic than reported previously. In agreement with other studies, we found that a poor PRNG can result in improved performance. However, this improvement is highly dependent of the seed, and we were unable to obtain good results consistently across the test functions used and different seeds. The ablation experiments suggest that the PRNG used to initialize the population is critical, while the PRNG used as input to other stochastic GA operations does not seem to affect the results. We performed additional experiments isolating individual components of the GA that seem to confirm these results. We did not observe any improvement in performance using the true random numbers over the MT generator.

The results of this study are limited to two PRNGs and to the trap functions used. Future work should apply the same experimental setup to additional functions and PRNGs. The effect of PRNGs on other evolutionary algorithms can also be studied with ablation experiments. The criticality of initialization on the performance of the GAs suggests that finding alternatives to the uniform random initialization may be beneficial.

While the choice of PRNGs seems to cause considerable fluctuations in performance, the design of reliable algorithms that consistently reach good solutions is not likely to be found in the experimentation with random seeds or different PRNGs. However, these large fluctuations require that experimenters choose their PRNGs and seeds carefully, and that these choices are reported appropriately. The results of this paper sug-

k	Expected	True	MT	MT1000 seed=random	MT1000 seed=10
3	4.25	4.2547 (0.0617)	4.2475 (0.0621)	4.2382 (0.5039)	4.653
4	1.5625	1.5668 (0.0368)	1.5632 (0.0392)	1.5635 (0.3795)	2.2
5	0.625	0.6255 (0.0236)	0.6248 (0.0259)	0.6217 (0.2462)	0.8
6	0.2656	0.2659 (0.0155)	0.2652 (0.0164)	0.2647 (0.1182)	0.34
7	0.1171	0.1175 (0.0107)	0.1172 (0.0108)	0.1149 (0.0711)	0.098
8	0.0507	0.0515 (0.0070)	0.0506 (0.0071)	0.0511 (0.0729)	0.104
9	0.0234	0.0235 (0.0045)	0.0236 (0.0049)	0.0233 (0.0295)	0.024
10	0.0097	0.0098 (0.0028)	0.0098 (0.0032)	0.0109 (0.0324)	0

Table 4: Number of optimal subfunctions in randomly initialized populations. The numbers in parenthesis are the standard errors.

gest that experimenters should use the best PRNG available to avoid "lucky" accidents that can muddle the interpretation of the results.

Acknowledgments

UCRL-JC-146850. This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

I thank the anonymous reviewers for their detailed and constructive comments that helped improve the paper.

References

Bäck, T. (1995). Generalized convergence models for tournament- and (μ, λ)-selection. In Eschelman, L. (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 2–8). San Francisco, CA: Morgan Kaufmann.

Daida, J., Ross, S., McClain, J., Ampy, D., & Holczer, M. (1997). Challenges with verification, repeatability, and meaningful comparisons in genetic programming. In Koza, J. R., Kalyanmoy, D., Dorigo, M., Fogel, D. B., Garzon, M., Iba, H., & Riolo, R. L. (Eds.), *Genetic Programming 97* (pp. 64–69). San Francisco, CA: Morgan Kaufmann Publishers.

Daida, J. M., Ampy, D. S., Ratanasavetavadhana, M., Li, H., & Chaudhri, O. A. (1999). Challenges with verification, repeatability, and meaningful comparison in genetic programming: Gibson's magic. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference 1999: Volume 2* (pp. 1851–1858). San Francisco, CA: Morgan Kaufmann Publishers.

Deb, K., & Goldberg, D. E. (1993). Analyzing deception in trap functions. In Whitley, L. D. (Ed.), *Foundations of Genetic Algorithms 2* (pp. 93–108). San Mateo, CA: Morgan Kaufmann.

Matsumoto, M., & Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, *8*(1), 3–30.

Meysenburg, M. M. (1997). *The effect of pseudo-random number generator quality on the performance of a simple genetic algorithm*. Master's thesis, University of Idaho.

Meysenburg, M. M., & Foster, J. A. (1997). The quality of pseudo-random number generators and simple genetic algorithm performance. In Bäck, T. (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms* (pp. 276–282). San Francisco: Morgan Kaufmann.

Meysenburg, M. M., & Foster, J. A. (1999a). Random generator quality and GP performance. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference 1999: Volume 2* (pp. 1121–1126). San Francisco, CA: Morgan Kaufmann Publishers.

Meysenburg, M. M., & Foster, J. A. (1999b). Randomness and GA performance, revisited. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference 1999: Volume 1* (pp. 425–432). San Francisco, CA: Morgan Kaufmann Publishers.

Miller, B. L., & Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, *9*(3), 193–212.

Mühlenbein, H., & Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization. *Evolutionary Computation*, *1*(1), 25–49.

Thierens, D., & Goldberg, D. E. (1993). Mixing in genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 38–45). San Mateo, CA: Morgan Kaufmann.