Category: Genetic Algorithms
Title: Randomness and GA Performance, Revisited

Mark M. Meysenburg
Department of Computer Science
Doane College
Crete, NE 68333
mmeysenburg@doane.edu
(402) 826-8267

James A. Foster
Laboratory for Applied Logic
Department of Computer Science
University of Idaho
Moscow, ID 83844
foster@cs.uidaho.edu
(208) 885-7062

Abstract

Previous studies by the authors have indicated that pseudo-random number generator (PRNG) quality has little effect on the performance of a simple genetic algorithm (GA). In this paper we examine this subject further, in the context of what we call the "granularity hypothesis. We detail a set of PRNG quality tests tailored specifically to the uses of randomness in a simple GA. We explain the application of detailed statistical analysis to the results of nearly ten-thousand individual GA runs, for large and small populations, over an eleven function GA test suite. We conclude that, although there is no evidence to support the notion that higher quality PRNGs cause better GA performance than lessor quality PRNGs, there is statistical evidence that certain PRNGs can provide improved GA performance.

1 INTRODUCTION

Previous studies conducted by the authors [Meysenburg, 1997, Meysenburg and Foster, 1997] studied the effect pseudo-random number generator (PRNG) quality had on the performance of a simple genetic algorithm (GA). We found statistical evidence suggesting that the quality of the PRNG chosen to drive the GA had little, if any, effect on the performance of the GA.

In this study, we attempted to find evidence to support what we call the "granularity hypothesis." In order to understand this hypothesis, consider the ways PRNGs are used in a simple GA utilizing binary representation:

Population initialization. Each bit in each individual is set to zero or one with probability $p_i = 0.5$ each. Call this use A.

Single-point crossover. Every generation, each individual in the population is selected for crossover with probability p_c. If an individual is selected for crossover, a "mate" is selected uniformly at random from the population, so the probability of an individual being selected as a mate is $\frac{1}{n}$, where n is the population size. Finally, a crossover point is selected uniformly at random, where the probability of any bit position being selected is $\frac{1}{L}$, where L is the length of a chromosome. Call these uses B, C, and D, respectively.

Point mutation. Every generation, each individual in the population is selected for mutation with probability p_m. If an individual is selected for mutation, a mutation point is selected uniformly at random, so the probability of any bit position being selected is again $\frac{1}{L}$. Call these uses E and F, respectively.

Two-tournament selection. For each individual in the next generation, two members of the current generation are selected uniformly at random. So, the probability of selecting an individual to participate in the tourney is again $\frac{1}{n}$. Call this use G.

Uses A through G above fall into two categories, which we call "two-buckets" and "n-buckets:"

Two-buckets. Uses A, B, and E are of this type. For example, a bit is set to one or zero, an individual is selected for crossover or not, and an individual is selected for mutation or not.

N-buckets Uses C, D, F, and G are of this type. One of the n individuals in the population is selected, or one of the L bits in an individual is selected.

These uses of randomness are what we call "coarse grained." The granularity hypothesis holds that, since a simple GA uses randomness in a very coarse grained manner, almost any PRNG could drive the GA to the same level of performance.

In a simple GA, the PRNG is used to choose between several options; this requires only that the PRNG produces a relatively uniform distribution of pseudo-random numbers. Even a PRNG that is poor in terms of overall quality can produce a uniform distribution. If the granularity hypothesis were true, even a poor quality PRNG should provide adequate GA performance, as long as it produces an adequate uniform distribution of numbers.

In order to test our granularity hypothesis, we constructed PRNG quality tests to determine which PRNGs are able to provide the kind of "coarse-grained" randomness required by the GA, and which are not. Then, we performed GA experiments, using PRNGs of differing quality, acting upon different GA test functions. Finally, we performed a detailed statistical analysis to look for evidence supporting our hypothesis.

Although we did not find evidence supporting the granularity hypothesis, we did find that particular PRNGs can effect GA performance. This paper summarizes our work on this subject.

2 TOOLS

We used a modular, object-oriented system consisting of the base GA, test functions for the GA, PRNGs, and test suites to evaluate the PRNGs.

2.1 The GA

The simple GA system used for this project employed binary representation, and utilizes tournament selection with tournament size two, single-point crossover, and point mutation. We coded the GA in Java.

2.2 The PRNGs

We repeated our GA experiments using several different PRNGs. The PRNGs represent several common algorithms for generating pseudo-random numbers, plus one PRNG with an artificially limited period. We coded the PRNGs in Java. The categories of PRNGs, and the PRNGs themselves, are listed in the following sections.

2.2.1 Linear Congruential Generators

rand. The `java.util.Random` PRNG, which uses the linear congruential algorithm described in Knuth's Volume 2, section 3.2.1 [Knuth, 1997].

rand1k. The **rand** PRNG, re-seeded with the initial seed value after every 1000 calls. This limits the period of **rand1k** to 1000.

pm. A Java version of the Park and Miller "minimal standard" linear congruential PRNG, based on the C implementation in Press, et. al. [Press et al., 1992].

2.2.2 Multiply With Carry Generators

mother. A multiply with carry generator termed "the mother of all PRNGs," described by Marsaglia [Marsaglia, 1994].

2.2.3 Additive and Subtractive Generators

add. The additive generator described in section 3.2.2 of Knuth's Volume 2 [Knuth, 1997].

sub. A Java version of the Knuth subtractive generator, based on the C implementation in Press, et. al. [Press et al., 1992].

2.2.4 Compound (a.k.a. "Shuffled") Generators

shsub. A version of **sub**, shuffled using **rand**, according to the shuffling algorithm described by Knuth [Knuth, 1997].

shpm. A shuffled version of **pm**, based on the C implementation from Press, et. al. [Press et al., 1992].

shlec. A Java version of the shuffled L'Ecuyer generator, based on the C implementation in Press, et. al. [Press et al., 1992].

2.2.5 Feedback Shift Register Generators

fsr. A feedback shift register generator of the type described by Schneier [Schneier, 1994].

tgfsr. A twisted generalized feedback shift register generator, described by Matsumoto and Kurita [Matsumoto and Kurita, 1992].

2.2.6 Tausworthe Generators

tauss. A Combined Tausworthe generator, described by Tezuka and L'Ecuyer [Tezuka and L'Ecuyer, 1991].

2.3 The PRNG Test Suites

To test our granularity hypothesis, we designed a new PRNG test suite, tailored to the way the GA uses randomness. We called the tests in our suite "bucket tests." The tests fell into two categories: two-bucket tests and n-bucket tests.

We ran each of the two-bucket and n-bucket tests described below for N = 1,000, 10,000, 100,000, 1,000,000, and 10,000,000 test points. We wanted to detect "local" non-randomness with smaller values of N, and "global" non-randomness with larger values of N. Local non-randomness might be characterized by poorly distributed short sequences offset by subsequent short sequences poorly distributed in the other direction. Global non-randomness might be characterized by non-uniformly distributed numbers over the long term.

2.3.1 Two-Bucket Tests

We used three two-bucket tests, one each for the uses of randomness A, B, and E referred to in Section 1; that is, one for initialization, one for crossover, and one for mutation. In each test, floating point numbers were obtained from the PRNG, and "dropped" into one of two buckets, with separate probabilities dictating how likely a number was to fall into each bucket. The only difference between the three tests was the *size* of the buckets (the probabilities for landing in each bucket).

In the initialization test, the buckets were the same size. If the floating point number from the PRNG was less than 0.5, it was dropped into the first bucket. Otherwise, it was dropped into the second bucket.

For crossover, we used $p_c = 0.6$. So, if the number from the PRNG was less than 0.6, it was dropped into the first bucket. Otherwise, it was dropped into the second bucket.

For mutation, we used $p_m = 0.01$. So, if the number from the PRNG was less than 0.01, it was dropped into the first bucket. Otherwise, it was dropped into the second bucket.

After one of these tests was completed, we used a Chi-Square (χ^2) test to measure how well the actual distribution between the two buckets compared with the distribution that would be expected from a truly random source.

2.3.2 n-Bucket Tests

We used our n-bucket tests for the uses of randomness C, D, F, and G referred to in Section 1. Unlike the two-bucket tests, where the buckets were of different sizes, the n-bucket tests used buckets all of the same size. The number of buckets, however, varied from test to test.

In our GA test suite functions, chromosomes took on the following lengths: 24, 30, 50, 120, 140, 200, 240, and 300. For our small population runs, we used populations of size 100. We had an n-bucket test for each of these values. For each test, random integers scaled to the correct range (say, from 0 to 23 for the 24-bucket test) were obtained from the PRNG, and then dropped into the appropriate bucket.

After a test was completed, we again used a χ^2 test to measure how well the PRNG distribution matched the expected result.

2.3.3 Diehard Tests

In addition to the bucket tests, we used Marsaglia's Diehard suite of tests [Marsaglia, 1993], available in executable form on the Internet [Marsaglia, 1998], to evaluate the overall quality of our PRNGs. The Diehard suite contains nineteen tests, which operate on large (approximately 10 MB) binary files created by the PRNG in question. The Diehard suite includes several tests similar to those presented by Knuth [Knuth, 1997], such as the "birthday spacings test" and the "runs test." In addition, the Diehard suite includes several tests that perform in-depth binary-level examinations of the sequences produced by the PRNG in question. So, the Diehard suite seemed to be a more stringent set of tests than the classic Knuth PRNG evaluation algorithms. Unlike the bucket tests, which were tailored to the use of randomness in the GA, the Diehard tests provided a more general or absolute measure of PRNG quality.

2.4 GA Test Suite

To evaluate the performance of the GA driven by different PRNGs, we used the eleven function test-suite described in our previous work [Meysenburg, 1997, Meysenburg and Foster, 1997]. The functions we chose are widely used for evaluating GA performance; they range from quite easy to solve using our GA to quite difficult.

Specifically, we used the following eleven functions: **F01**, DeJong's first function; **F02**, DeJong's second function; **F03**, DeJong's third function; **F04**, DeJong's fourth function; **F05**, Ackley's function; **F06**, Griewangk's function; **F07**, Rastrigin's function; **F08**, Schwefel's sine root function; **F09**, Shekel's foxhole function; **F10**, Michalewicz's function; and **F11**, Langerman's function.

3 METHODOLOGY

3.1 Seed Selection

Since the quality of some PRNGs can be significantly effected by the seed value used to start the sequence, we randomly created a set of thirty-two, sixty-four bit PRNG seed values. We created each seed value by rolling a four-sided die thirty-two times, interpreting each roll as a two-bit value. We then concatenated the bit patterns to form the seed value.

Most of our PRNGs take signed, sixty-four bit integers as seeds; hence our randomly created sixty-four bit binary patterns. However, we derived some of the PRNGs from C implementations; these PRNGs require thirty-two bit seeds. For these generators, we cast the sixty-four bit integers to thirty-two bits. In addition, one generator requires negative thirty-two bit integers; in this case, we cast to thirty-two bits and then flipped the sign bit of positive seeds.

3.2 PRNG Quality Evaluation

To summarize how a PRNG performed in our bucket tests, we used a scoring and ranking scheme similar to that in our previous work [Meysenburg and Foster, 1997]. For a particular test (with one set of runs for each of the thirty-two seed values) we used the Kolmogorov-Smirnov (KS) test to see how well the thirty-two χ^2 values approximated the actual χ^2 distribution. Then, we categorized the results of the KS tests (expressed as p-values between 0 and 1, with high p values indicative of a poor match) using Johnson's [Johnson, 1996] scheme for determining suspect and reject values.

We classified a p-value as "rejected" if $p \geq 0.998$. We classified a non-reject p-value as "suspect" if $0.95 \leq p < 0.998$. We classified all other p-values as "good." We assigned point values to the PRNGs: two points for every "reject" categorization, one point for every "suspect" categorization, and zero points for every "good" categorization. We then summed these points to produce a final Bucket score for each PRNG. Low Bucket scores indicated good PRNG quality, whereas high Bucket scores indicated poor PRNG quality.

We used a similar method to interpret the results of the Diehard test suite runs. In particular, we ran each PRNG through the test suite, with one run for each of the thirty-two seed values, and then assigned scores and ranks to the PRNGs based on the results of the Diehard tests. Each of the tests produced one or more p-values.

We categorized the Diehard p-values as "good," "suspect," or "rejected," as we did for the Bucket tests. Then, we assigned point values in the same way and summed these points to produce a final Diehard score for each PRNG. Low Diehard scores indicated good PRNG quality, whereas high Diehard scores indicated poor PRNG quality.

3.3 GA Runs

We performed two sets of GA experiments. For one set, we used populations of one-hundred individuals and runs of one-thousand generations. For the other set, we used populations of one-thousand individuals and runs of one-hundred generations. In all cases, we used $p_c = 0.6$ and $p_m = 0.01$.

For each set of experiments, we ran the GA for each test suite function, each PRNG, and each seed value. So, we performed a total of $2 \times 11 \times 13 \times 32 = 9152$ individual GA runs. For each GA run, we recorded the following statistics, on a generation-by-generation basis: the worst, best, and mean fitness of the current generation, and the worst and best fitness to date.

3.4 Statistical Methods

We used statistical methods to compare the performance of the GA driven by PRNG a versus the GA driven by PRNG b, for each (a, b) pair, with $a \neq b$, on a generation-by-generation basis. To do this, we established the following null and research hypotheses:

Null Hypothesis. For this generation, the GA driven by PRNG a does not perform better than the GA driven by PRNG b:

$$H_0 : \mu_a \leq \mu_b.$$

Research Hypothesis. The inverse of the null hypothesis; in other words, for this generation, the GA driven by PRNG a does perform better than the GA driven by PRNG b:

$$H_r : \mu_a > \mu_b.$$

For a given generation, we wished to use the fitness of individuals in the population as the measure of GA performance.

Many statistical methods for hypothesis testing require the experimenter to make assumptions about how the population is distributed. Typically the normal distribution is assumed, and statistical methods for hypothesis testing are then based on this assumption. In terms of GA performance, it is not clear that the fitness of individuals in the population is always distributed normally. Furthermore, the distribution will likely change from generation to generation, as the GA (hopefully!) converges to a solution. So, a statistical method that allows hypothesis testing for populations of unknown distribution was in order.

The Wilcoxson test described by Green and Margerison [Green and Margerison, 1978] is such a statistical measure; it is a method for hypothesis testing using the mean of a random variable with an unknown distribution. We performed the test in four steps.

1. **Compute u.** We let U be the random variable under scrutiny in our null and research hypotheses; u was a realization of U. Given PRNG a, PRNG b, and one of our eleven test suite functions (**F01** - **F11**), we computed u for a given generation t via the following algorithm. In the notation of the algorithm, S is the set of all thirty-two seed values, s_a is the particular seed used to initialize PRNG a, s_b is the seed used for PRNG b, μ_{s_a} and μ_{s_b} are the fitness means for generation t of the GA driven by PRNG a and PRNG b, respectively, and Opt is the optimal value for the test suite function under consideration. The algorithm is:

```
let $u = 0$
for each $s_a \in S$
    for each $s_b \in S$
        if $|\mu_{s_a} - Opt| = |\mu_{s_b} - Opt|$ then
            $u = u + \frac{1}{2}$
        else if $|\mu_{s_a} - Opt| \leq |\mu_{s_b} - Opt|$ then
            $u = u + 1$
    end for
end for
```

   The realization u should be normally distributed, for sample sizes as small as eight. Our sample size was thirty-two, one sample for each seed value.

2. **Compute T.** Based on the value of u, we calculated our test statistic T as follows, using $n_1 = n_2 = 32$:

$$T = \frac{u + \frac{1}{2} - \frac{n_1 n_2}{2}}{\sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}}}$$

3. **Find $\Phi(T)$.** We found the standard normal cumulative distribution of T, $\Phi(T)$.

4. **Reject or Accept H_0.** We rejected the null hypothesis in favor of the alternative hypothesis for a value of $\Phi(T)$ less than one percent.

After performing the Wilcoxson test computations on a generation-by-generation basis, we tallied the number of null hypothesis rejections for each (test suite function, PRNG a, PRNG b) triple. We normalized these tallies to range between zero and one, so that one represented the case where the null hypothesis was rejected at every generation. Zero represented the case where there were no null hypothesis rejections.

4 RESULTS

4.1 PRNG Quality

The Bucket and Diehard test suite scores for each PRNG are summarized in Table 1.

As Table 1 shows, the Bucket test ranking of PRNGs was not the same as the Diehard ranking. This was not especially surprising. Our Bucket tests can be thought of as extensions of the simple Equidistribution test from Knuth [Knuth, 1997]. That is, they measured whether numbers from a purported uniformly distributed random number source were indeed uniformly distributed. However, uniform distribution is only one aspect a PRNG must possess in order to be considered of high quality. The Diehard suite considered many more facets of PRNG quality than just uniform distribution. This was why the Diehard test rankings did not agree with the Bucket test rankings (with the exception of **rand1k**). Table 1 also shows that several of our PRNGs were indistinguishable, in terms of the Bucket tests.

If our granularity hypothesis was correct, one would expect that the GA driven by **rand1k** or perhaps by **tauss** would perform worse than the GA driven by any of the other PRNGs. However, we found no evidence to support this hypothesis in our study.

Although we did not find evidence to support our granularity hypothesis, we did find some interesting relationships between PRNG quality and GA performance.

4.2 GA Performance

Figures 1 through 5 summarize the number of null hypothesis rejections we found for several of our GA test suite functions, for both large and small populations. A complete set of figures, for all of our functions and population sizes, may be found at the author's web site [Meysenburg, 1998].

	Bucket		Diehard	
PRNG	Rank	Score	Rank	Score
pm	1	4	10	1619
shlec	1	4	7	751
shpm	3	6	8	799
shsub	4	7	1	548
fsr	4	7	2	573
rand	6	8	11	2129
add	6	8	3	577
sub	8	9	6	655
mother	8	9	5	602
tgfsr	8	9	4	584
tauss	11	19	9	935
rand1k	12	101	12	9337

Table 1: PRNG Test Suite Scores

In the figures, the null hypothesis PRNG is on the vertical axis, and the research hypothesis is on the horizontal axis. A large value in a particular cell means that there were a large number of null hypothesis rejections for that population size, test suite function, PRNG a, and PRNG b. In other words, a large value in a cell means that there was statistical evidence that the vertical axis PRNG caused the GA to perform better than the horizontal axis PRNG, for the particular population size and test suite function.

Further, a vertical column of high values indicates that the PRNG labeling the column caused the GA to perform worse than all other PRNGs in the study. A horizontal row of high values indicates that the PRNG labeling the row caused the GA to perform better than all other PRNGs in the study.

4.2.1 Large Population, Small Generations

For our large population set of experiments, results fell into two categories: where **rand1k** and **tgfsr** caused the GA to perform worse than the other PRNGs, and where the same PRNGs caused the GA to perform better. The first category is illustrated by Figure 1 (for function **F01**). Other functions for which **rand1k** and **tgfsr** caused worse GA performance are **F02**, **F04**, and **F06**. For **F03**, illustrated in Figure 2, **tgfsr** alone caused worse GA performance than the other PRNGs.

The second category is illustrated by Figure 3 (for function **F05**). Other functions for which **rand1k** and **tgfsr** caused better performance are **F07**, **F08**, **F09**, **F10**, and **F11**.

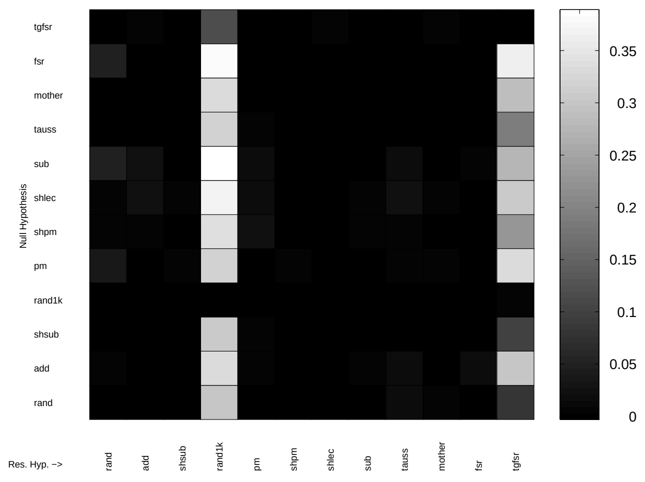

Figure 1: H_0 Rejections for large population, **F01**

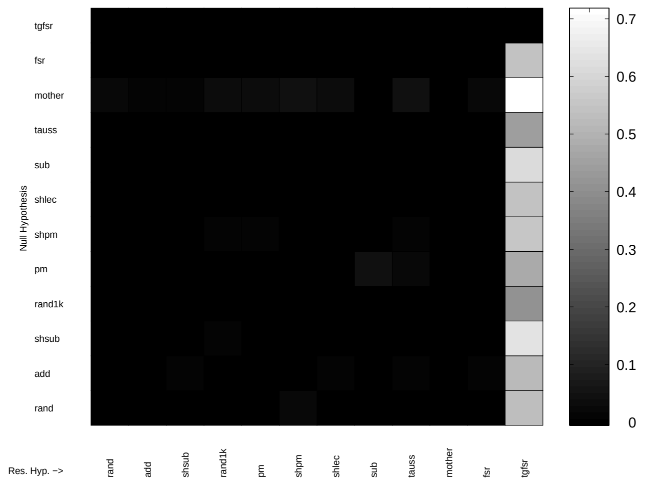

Figure 2: H_0 Rejections for large population, **F03**

4.2.2 Small Population, Large Generations

For our small population set of experiments, results fell into one main category, with three exceptions. All of the functions except **F01**, **F02**, and **F07** fit the pattern shown in Figure 5, which illustrates **F03**. In this pattern, **rand1k** caused the GA to perform better than the other PRNGs, and **mother**, **fsr**, and **tgfsr** caused the GA to perform worse. The results for function **F07**, fit this pattern, except that in addition to **mother**, **fsr**, and **tgfsr**, **shpm** also caused the GA to perform worse than did the other PRNGs.

For function **F01**, **rand1k** caused the GA to perform better than did the other PRNGs. For function **F02**, **tgfsr** caused the GA to perform slightly better than did the others.

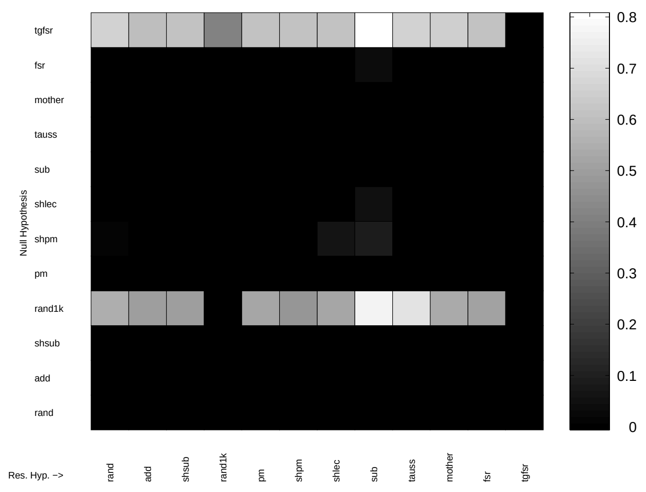

Figure 3: H_0 Rejections for large population, **F05**

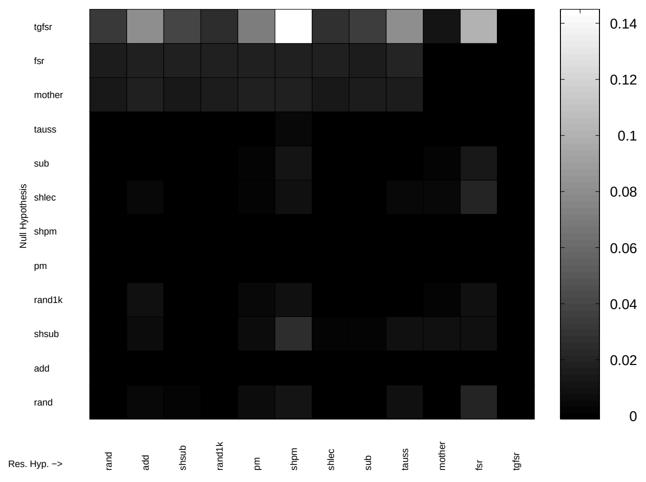

Figure 4: H_0 Rejections for small population, **F02**

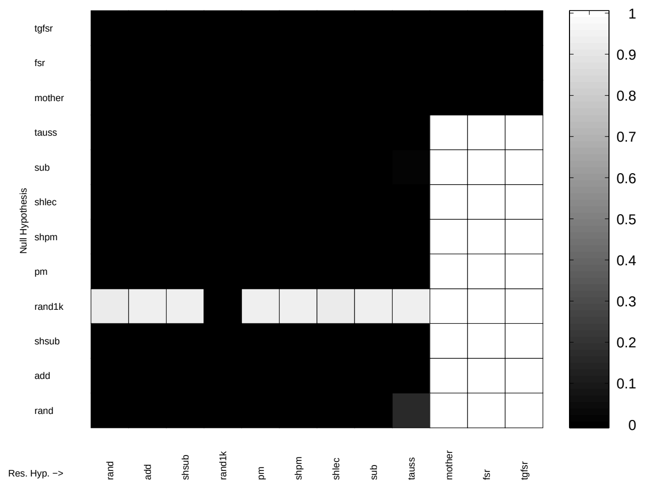

Figure 5: H_0 Rejections for small population, **F03**

5 CONCLUSIONS

We did not find evidence to support our granularity hypothesis. The PRNGs ranked best by our Bucket tests did not cause statistically better performance of the GA than did the PRNGs ranked worst. Nor did we find that high overall PRNG quality led to better GA performance.

However, we did find that different PRNGs caused different levels of GA performance, albeit in non-intuitive ways. We found that poor-quality PRNGs caused better GA performance (see Figures 3, and 5), that high-quality PRNGs caused worse GA performance (see Figures 1, 2, and 5), and that the same PRNG, of either high or low quality, caused better or worse performance, depending on the particular GA test suite function (see Figures 1 and 3).

In our previous work [Meysenburg, 1997], we applied statistical methods across all the test suite functions, using the online performance measure, expressed as a percentage of the optimal value. In that research, we found some very slight evidence that a poor quality PRNG could cause better GA performance than could a high quality PRNG. However, the overwhelming majority of cases showed no statistical effect on GA performance by PRNG quality, in either direction.

Our statistical methods in this study were applied at a much lower level. We examined the data from our GA runs on a functional, generation-by-generation basis, using mean fitness as our measure of GA performance. These methods allowed us to see differences in GA performance that were hidden by our methods in our previous work.

Our new evidence still does not show that higher quality PRNGs (in terms of either the Bucket or Diehard tests) cause better GA performance. The evidence does, however, suggest that a judiciously chosen PRNG (not necessarily the best quality PRNG), can cause improved GA performance.

6 FURTHER RESEARCH

In further research of this subject, we would like to determine *where* in a GA run performance gains are to be had by choosing one PRNG over another. That is, does the PRNG cause better performance early on in the GA run, late in the run, or throughout the run? We would like to understand more about how population size and length of run contribute to the differences in GA performance we observed. Why is it that **rand1k** may cause better or worse GA performance for a population of one-thousand, but not for a population of one-hundred? Also, we would like to add theoretical weight to our studies, which have been purely statistical to this point.

References

[Green and Margerison, 1978] Green, J. R. and Margerison, D. (1978). *Statistical Treatment of Experimental Data.* Elsevier Scientific Publishing Company, first edition.

[Johnson, 1996] Johnson, B. C. (1996). Radix-b extensions to some common empirical tests for pseudorandom number generators. *ACM Transactions on Modeling and Computer Simulation*, 6(4):261 – 273.

[Knuth, 1997] Knuth, D. E. (1997). *The Art of Computer Programming*, volume 2. Addison Wesley, third edition.

[Marsaglia, 1993] Marsaglia, G. (1993). Monkey tests for random number generators. *Computers & Mathematics with Applications*, 9:1–10.

[Marsaglia, 1994] Marsaglia, G. (1994). Yet another rng. Posted to sci.stat.math August 1, 1994.

[Marsaglia, 1998] Marsaglia, G. (1998). http://stat.fsu.edu/~geo/diehard.html.

[Matsumoto and Kurita, 1992] Matsumoto, M. and Kurita, Y. (1992). Twisted gfsr generators. *ACM Transactions on Modeling and Computer Simulation*, 2(3):179 – 194.

[Meysenburg, 1997] 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, 1998] Meysenburg, M. M. (1998). http://www.doane.edu/crete/academic /science/ist/mark.htm.

[Meysenburg and Foster, 1997] Meysenburg, M. M. and Foster, J. A. (1997). The quality of pseudo-random number generators and simple genetic algorithm performance. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 276 – 281. Morgan Kaufmann.

[Press et al., 1992] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C.* Cambridge University Press, second edition.

[Schneier, 1994] Schneier, B. (1994). *Applied Cryptography.* John Wiley And Sons.

[Tezuka and L'Ecuyer, 1991]
Tezuka, S. and L'Ecuyer, P. (1991). Efficient and portable combined tausworthe random number generators. *ACM Transactions on Modeling and Computer Simulation*, 1(2):99 – 112.