B. Williams, A. Carlson and R. Hiromoto. (Oct. 03-05, 2022) Novel Innovations that Failed to Improve Weak PRNGs, *2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, Kharagpur, India, 2022, pp. 1-6, doi: https://doi.org/10.1109/ICCCNT54827.2022.9984517]

Novel Innovations that Failed to Improve Weak PRNGs

Benjamin Williams
Computer Science
University of Idaho
Idaho Falls, U.S.
will9847@vandals.uidaho.edu

Albert Carlson
CSIT
Austin Community College
Austin, U.S.
albert.carlson@austincc.edu

Robert Hiromoto
Computer Science
University of Idaho
Idaho Falls, U.S.
hiromoto@uidaho.edu

Abstract—Weak Pseudo-random Number Generators (PRNGs) can be improved with additional operations and techniques. However, some techniques added to the RNGs in an attempt to improve the quality and randomness of the generator fail to produce the desired results. Literature surveys have produced very few ideas and information related to the constituent steps involved in the individual algorithms used in constructing the PRNGs. In preference to studying and understanding the individual elements of the algorithms, designers have preferred to create new PRNGs and have missed the opportunity to increase the basic understanding of the field. If the building blocks of PRNGs are well understood, along with their interactions, the design of high quality PRNGs is possible. Understanding what works also requires understanding what does not work. Those techniques can then be avoided. This paper is an exploration of some novel techniques for improving the quality of weak PRNGs that failed to live up to the promise of improving PRNGs.

Index Terms—Randomness, Random Number Generators, Pseudorandom Number Generators, LCGs, MCGs, PCGs, Geffe Generators, Polymorphic RNGs, Composite Generators

I. Introduction

Pseudo-random Number Generators (PRNGs) are ubiquitous and critically important in computing applications. They are used in everything from simulations to security. One major challenge of PRNGs is quality. Many applications require PRNGs that produce sufficiently high quality output as to be completely unpredictable. This used to be limited to cryptographic security applications, but the need for quality PRNGs has increased even in simulations [1], where weak LCGs were once favored for their speed. This presents another major challenge: Speed. Speed and statistical quality seem to be inversely proportional, thus there is a great deal of value in cheaply improving the quality of fast PRNGs, for use in applications where both speed and quality are important factors.

The process of discovering algorithms for improving the quality of weak PRNGs necessarily results in the discovery of algorithms that fail to improve the quality of weak PRNGs or even manage to make the quality worse. This paper presents a set of just such failures along with some discussion on why they failed to improve quality and potential uses for them, despite that failure.

A. Testing Limitations

Testing PRNGs can be difficult and time consuming. This is essentially a pattern recognition exercise, a problem with which computers tend to struggle. Common tests include goodness of fit, the gap test, the order test, the frequency test, the birthday test, and the bucket test. Dutang and Wuertz [2] also list some tests that can be executed on multiple output streams from the same generator, which are not part of the test suite used for this paper.

Testing was done using Dieharder [3], a strong suite of PRNG tests. Dieharder combines many of the original Diehard tests [4], some of the NIST tests [5], and a number of other tests. For the most part, poor quality tests from the original suites are not included[1]. Most of the tests included in Dieharder are variations on the previously mentioned test classes, using different parameters or dimensionality, shown to discover different types of patterns.

Dieharder is open source and is still actively maintained, adding new, high quality tests as they are discovered, making it one of the strongest PRNG testing suites currently available. Because Dieharder has such a robust suite of tests, however, test runs normally take two hours or more to run on good hardware. This limits the rate at which testing can be done.

Speed is not only a problem for Dieharder. It is also a problem for PRNGs. Some of the following strategies received significantly less testing, because they significantly reduced the PRNG speed, making testing too slow to go through as many runs. Slower PRNGs can take days to complete a single test run.

Dieharder's testing metrics calculate success or failure based on the probability of an observed series of values from the PRNG. This is fundamentally a comparison with true randomness. Because low probability series will sometimes be produced in real randomness, a good quality PRNG will sometimes produce low probability series as well. This can result in spurious failures or weak

[1]Dieharder still contains one low quality test, which was not known to be low quality when it was added. This test can produce false “WEAK” and “FAILED” results, but is not relevant here, because so many other tests also failed.

results. In Dieharder, “WEAK” indicates a result that has a 1% probability of occurring in true randomness, and “FAILED” indicates a 0.1% probability output. In practice, this means that even an extremely good PRNG should produce “WEAK” results in 1 test in 100 and “FAILED” results in 1 test in 1000.

The final limitation is the tests themselves. There is not any consistent method of determining the quality of a particular test. Some poor quality tests will manifest their poor quality easily, but it is likely some of the tests in Dieharder have undiscovered statistical weaknesses. This means that some portion of “WEAK” and “FAILED” results could be caused by weaknesses in the tests themselves. The only way to rigorously test PRNGs is to test them using Dieharder’s “test-to-destruction” mode, and then compare them with the results of the same testing against known high quality PRNGs. This mode takes literal months to complete against fast PRNGs and thus was not used here. In the cases discussed in this paper, the failures were so dramatic that limitations of the testing cannot reasonably be blamed.

II. Related Works

Much of the modern foundations of PRNGs is based on the work of L’Ecuyler [1], [6]–[11], Knuth [12], and Panneton [13]. Many of the standard techniques introduced by these authors are used today and emulated. Knowing how the researchers arrived at their conclusions has guided the field for decades. Another source of inspiration concerning the foundations of the mathematics of PRNGs comes from the field of encryption. Hartley and Shannon’s work on entropy helped to establish metrics for determining the mathematical quality of PRNGs [14], [15]. While this body of work does establish many metrics for determining PRNG quality, it does not identify techniques that fail to produce good results. Little other published work on topic of such constituent techniques used in PRNG algorithms exist. High quality PRNGs are a “holy grail” of the commercial technology world. Good PRNGs are considered trade secrets by their owners and vigorously defended in courts. This only happens because of the scarcity of available algorithms.

Mathematically proving a PRNG to be high quality is almost impossible, which is why statistical testing suites like Diehard were originally created. However, it is often quite possible to mathematically prove that techniques used in PRNGs impair quality. Two metrics commonly used for this purpose are λ and ρ [16]. The symbol λ represents the cycle length. A short cycle length indicates poor PRNG quality. In contrast, ρ represents the number of outputs required to determine the state of the PRNG. A small ρ is also indicative of poor quality. Large values for these quantities are necessary, but not sufficient, for high quality.

The values of both ρ and λ have related concepts found in Shannon Theory [15]. Consider the role of Abstract Algebra [17], which says that if the math for a problem is the same math that describes another problem, then the two are related. Further, it also indicates that the approach for one problem will work for the other problem. This same concept is one of the foundations of the work done in the field of Topology, as well [18]. Shannon Theory identifies unicity distance as the quantity of cipher text required to break a cipher. In the study of PRNGs, ρ indicates how much information is required to break the formula on which the PRNG is based. Therefore, the Shannon value n plays the same role as ρ in PRNG analysis. In Shannon Theory, the redundancy of the language (R_λ) indicates how often repeated characters are likely to be seen in an encrypted text. If the repeated text is considered to be the PRNG output stream then the redundancy is the cycle length with absolute precision. Therefore, the Shannon Theory quantity R_λ corresponds to λ in PRNGs. The correspondence between the two sets of numbers also indicates that entropy [14] heavily underlies and influences both fields. Since both are so interrelated, the tools of Information Theory (IT) [19] are applicable to both problems, and the techniques of IT can also be applied similarly. PRNGs and their algorithms should be examined in light of this relationship and in the context of encryption and IT.

Users and practitioners typically do not talk about failure because the techniques failed. However, there is significant knowledge that can be derived from the failures experienced by other researchers. Too much time is wasted in repeated research into techniques that previously were shown to be ineffective. If those results were reported then researchers could have avoided the lost time and concentrated on more effective lines of research. It is in this spirit that the results of research that was not fruitful are offered.

Crypt1 and PCG provide a contrast in methodology that highlights the value of testing and cataloging individual techniques used in PRNGs to avoid ineffective techniques that waste resources. PCG included the creation of a number of innovative output permutations and applied them largely as a group [20]. Work on Crypt1 started with the permutations introduced in PCG and began by testing each permutation independently [21]. This testing found that many of the permutations used in PCG provide no significant benefits and that one actually reduces statistical quality. Making the knowledge of which permutations failed to produce good results publicly known is no less important than identifying the permutations that are good, so that future researchers will not waste their resources rediscovering what others have already found. To this end, this paper seeks to disclose techniques that were attempted but failed to produce any improvement, during the development of a simple PRNG designed for temporary use in another research project.

III. Prime Numbers

Prime numbers are especially valuable in random number generation and other security applications. One major reason for this is that they can be used to create very long cycles, reducing the frequency of repetition in PRNGs. Due to memory constraints, digital computing is fundamentally cyclical. Given any statically typed integral variable, continually adding 1 to it will eventually cause it to overflow its maximum value and start over at its minimum value, producing cyclic behavior. Adding larger values will cause a similar effect, ultimately always cycling, but prime numbers are guaranteed to always visit every possible value the variable can hold, so long as the prime number is not a factor of the variable range. In general, repeatedly adding a value that is coprime to the range of the data type will always visit every possible value of the data type exactly once in each cycle, maximizing cycle length. Prime numbers maximize this potential, because they are less likely to be coprime than composite numbers. If, however, the value used to increment shares a factor with the variable type's range, the cycle length will be substantially reduced, and some of the values will never be visited. In the context of base-2 computing systems, where variable ranges are typically powers of 2, any odd value is sufficient. However, the best results come from prime numbers and odd numbers with only one of each prime factor. An advantage of prime numbers is that when multiplied by other prime numbers, the result is a number with only unique prime factors. Repeated factors create the potential for smaller cycles.

A. Prime Stride

PRNGs generate sequences of values through an iterative process. Normally, the state of the PRNG is used to produce output after each iteration. Knowing that the entire state range of a PRNG will be visited, even if some iterations are skipped, so long as the skip length is coprime to the cycle length, it seems intuitive that skipping some prime value of iterations for each output will reduce the correlation between adjacent outputs. It seems like this should be an effective strategy for improving the quality of weak but fast PRNGs, especially since this sort of skipping can often be trivially integrated into the PRNG without any performance cost[2].

Unfortunately, this strategy does not actually work. Testing using Dieharder, with strides of 3, 5, 7, 11, 13, 17, 19, 23, and 101 produced a total of 4853 passed tests, 464 weak tests, and 4700 failed tests. This is a failure rate of 46.9%, which is marginally higher than the failure rate of 46.3% of the core 32 bit LCG without adding the prime strides. No individual prime stride produced significantly better results than the others.

[2]For example, in an LCG, one could add a stride of 7, by multiplying both the multiplicative constant and additive constant by 7, which can be trivially optimized into new constants by the compiler.

It is not clear why this strategy failed, but there are some speculative conclusions. One notable factor is that for this particular PRNG, increasing the stride to be larger than 1 is equivalent to using different constants. LCGs are very sensitive to the constants used, and the LCG used in this case uses constants taken from a list of known good quality constants for LCGs. This suggests that this particular modification was doing nothing more than changing the initial LCG into a different one, which happens to have similar statistical weakness. It is notable, however, that the new LCGs (all nine of them) had similar statistical properties to the original. Most potential constants for LCGs produce far worse results than this. This suggests that given a known good constant for an LCGs, new good constants can be discovered merely by multiplying the constants by a prime number[3]. Despite the failure of this strategy, it does seem to have revealed some valuable information about the properties of LCGs.

B. Irregular Prime Stride

A more advanced strategy than using a single stride is using multiple prime strides, that are rotated through, such that each iteration of the PRNG skips a different number of states. Unlike a static prime stride, this is not equivalent to merely using a different pair of constants. This is more like iterating through 53 different LCGs, each one starting with the previous state and iterating once using constants equivalent to the core constants multiplied by this iteration's prime number. This makes it very similar to a Geffe generator [22], using 53 different LCGs. The prime numbers used in testing included all prime numbers between 3 and 255 inclusive, of which there are 53. Such a PRNG cycles through the list in an irregular order (using a stride of 31, modulus the length of the list). One particular advantage of this strategy is that it increases the cycle length of the PRNG. The number of primes in the list is 53, which is itself a prime number, increasing the cycle length of the PRNG by 53 times. This strategy again met with failure. Testing using Dieharder, produced 341 passed tests, 35 weak tests, and 422 failed tests, across 6 test runs. This is a failure rate of 52.9%, again compared with the core LCG's failure rate of 46.3%. The figure is beyond the expected rate of deviation, suggesting that this strategy actually made the statistical quality of the PRNG worse.

That the failure of this strategy is worse than the simple prime stride is counterintuitive. It seems to be a more advanced strategy. On the other hand, LCG is a fairly fragile algorithm to begin with, requiring carefully selected constants to avoid falling into very short cycles. Perhaps more experimentation using this strategy with other PRNGs and larger prime numbers would reveal

[3]This cannot be 2, as the will reduce the cycle length by at least half. Since the modulus is a power of 2, having both constants divisible by 2 will constrain the state to odd or even value depending on the initial state.

more information. This does show that the strategy of using prime multiples of known good constants does have limits.

IV. Output Permutations

In general, the use of output permutations has proven very effective. O'Neil's PCG [20] and Williams' Crypt1 [21] proved the effectiveness of simple rotate and xorshift permutations, while Crypt1 proved the effectiveness of both homogeneously and heterogeneously partitioned rotate and xorshift permutations. Presented here are several variations of permutation.

A. Byte Boundary Crossing Output Permutations

An innovation on heterogeneous partitions is partition sizes that are not powers of two. In theory this should achieve more shuffling in less predictable ways, because it can achieve more cross-byte mixing in a single permutation than even simple heterogeneously partitioned permutations. This strategy appeared to work, however, it required significantly more instructions to accomplish. The result was permutations that executed significantly more slowly than byte aligned partitions but produced approximately the same improvement in statistical quality when both strategies were used in series of two or more permutations. Given that a single byte boundary crossing permutation takes at least twice as long as a single simple heterogeneously partitioned permutation there is never a situation where byte boundary crossing permutations can add statistical value.

It is possible that with hardware designed for arbitrary alignment and length bit manipulation, some statistical benefit could be obtained from this strategy. With the limited statistical benefits observed, however, it is not clear the improvement would be more than minor. Perhaps the greatest potential benefit would be in having far more permutations to choose from when generating unique PRNGs from components.

B. Poor Quality Permutations

Not all partitioned permutations produce good quality results. The quality of the various partitioning schemes vary substantially. The vast majority produce very good results, but some seem to be mediocre or worse.

The homogeneously partitioned rotate using 32 bits partitions on a 64 bit value consistently failed to produce good results. Further, it tended to substantially reduce the statistical quality of any permutation series it was used in. Out of 53 rotate permutations, including the basic 64 bit rotate; 32, 16, and 8 bit homogeneously partitioned rotates; and 49 heterogeneously partitioned rotates, comprising all possible 64 bit rotate permutations that stay within native type alignments, this was the only rotate permutation that produced such poor results.

It is not clear why this particular permutation caused such dramatic quality issues, but it is clear that not all permutations are equal. This suggests there is significant potential for curated sets of permutations that are proven to be of superior quality to the average. Unfortunately, comprehensive testing of all rotate and xorshift permutations (of which there are 53 of each). Even in small series this methodology is completely infeasible. The ideal series length seems to be in the range of 4 to 8 permutations. With 106 total permutations, just the series of length 4 result in 126.2 million possibilities in number. On high end hardware (Intel i7-10750H, which has 6 cores), a single Dieharder test on default settings takes well over an hour, even using all 6 cores in parallel, it would require more than 2,401 years to run a complete test. It might be possible to reduce this to a handful of years using one or more supercomputers, but the cost would be prohibitive. Statistical analysis of a much smaller number of series might be sufficient to exclude permutations most likely to be of lower quality or to select a subset of permutations that are more likely to be higher quality though.

V. Generators

A number of generators were tested with permutations. In most cases the permutations significantly improve the statistical quality of the generator. Tested PRNGs included LCG, Xoroshiro128+, Xoroshiro128**, Xoshiro128+, Xoshiro128**, and LCG128mix, an LCG based PRNG, with a specialized xorshift output permutation[4]. These are all 128 bit PRNGs, thus the permutations were applied to only half of the bits, which where used to produce the output after application of the permutations.

Xoroshiro128+ consistently failed to get good results, with permutations applied. Without permutations, it tested similarly to the other generators, but when permutations were applied, the quality either failed to improve or sometimes even declined, where the other generators very consistently improved quite dramatically.

Again, it is unclear precisely why this particular PRNG was affected negatively by the permutations when the others did so well. This hints at some hidden statistical weakness of Xoroshiro128+, which Dieharder was unable to discover in the output of the raw generator but which the permutations made discoverable. While these results suggest Xoroshiro128+ has some statistical issues, they seem to hint at another valuable use of permutations, in testing PRNGs in general. It is unknown exactly how the permutations make the statistical weaknesses discoverable, but discovering this would also be useful in designing additional statistical tests which could be added to Dieharder.

VI. Conclusions

Exploring failure can be very useful in learning, and the failures discussed here all have significant room for future exploration. Better understanding why these failures occur

[4]This is not based on O'Neill's xorshift permutation family.

could have enormous implications in PRNG design, which could have a huge impact in the field of cryptography [23]. Many applications of PRNGs are low stakes, and even simple LCGs would be sufficient. There are a handful of applications, however, where statistical quality is a critical factor.

Perhaps the most obvious application where statistical quality matters is cryptography, where poor statistical quality can seriously compromise security. LCGs are commonly used in Monte Carlo simulations, for their speed, but such simulations can suffer significantly in quality, when a poor quality PRNG is used. Additionally, in applications where multiple PRNG instances are used and correlation between them is problematic, quality can be an enormous factor in performance. For example, in collision detection and avoidance in wireless communications, where PRNGs are used to desynchronize devices that are colliding, raw LCGs are terrible, because even with different seeds, there is correlation in their outputs that can make desynchronization difficult to achieve quickly.

The main limitation in this paper is the small number of techniques tested. The testing was done as part of another project, and it was limited by the needs of that project. There are many techniques used in PRNGs that have not been tested. There are also some limitations in the statistical testing. Mathematical analysis may be able to prove some existing techniques ineffective. The application of Information Theory could be extremely valuable in this endeavor. The only limitation that cannot currently be overcome is the weaknesses of statistical testing, which means this is likely to be an ongoing effort, as new statistical tests are developed.

Developing a better understanding of PRNGs may be absolutely critical to the future of cryptography, especially as faster and faster computing technologies are developed. Quantum computing may not be the end of secure encryption many initially believed it to be, but it will almost certainly significantly increase the necessary statistical quality of cryptographic algorithms, including PRNGs, to maintain security in the long term. In this endeavor, a better understanding of what does not work is no less important than understanding what does.

VII. Future Work

In all of these failures, there is substantial room for additional exploration. The failure of prime strides for the LCG used to test it suggests weaknesses in the LCG, but it may demonstrate weaknesses in the use of prime numbers themselves. Further, if the weakness is in the LCG, perhaps there is some potential for using prime strides in statistical testing of other PRNGs. A good starting place for future work on this is testing prime strides with other PRNGs.

The failure of irregular prime strides also suggests weaknesses in LCG. Because of the irregularity, it is unclear what the implications might be with respect to the use of prime numbers in this context. Again, there might be some use of irregular prime strides in statistical testing, and a good starting place would be testing with other generators.

Failure of byte boundary crossing permutations was not entirely surprising. Manipulation of arbitrary length bit strings is known to be quite expensive on modern hardware. It is possible, however, to design custom hardware that can accommodate this. Hardware encryption devices or circuits could easily provide fast and cheap byte boundary crossing permutations. Before spending significant resources on this, however, it would be necessary to prove the value of such permutations. More comprehensive testing of these permutations would be necessary to this end. A good starting place for this would be to produce significantly more byte boundary crossing permutations, and the next step might be testing a suite of these permutations, applied to the output of many different PRNGs.

The failure of poor quality permutations has already been discussed in this context. The next step is larger scale testing of the 106 existing permutations in series of 4 to 8, using statistical methods to estimate the quality of the individual permutations.

Failure of the Xoroshiro128+ PRNG with permutations, where apparently similar quality PRNGs did not fail with permutations, seems to indicate that Dieharder's test suite still has room for additional tests. There is significant importance in this, in the context of cryptography. It is not clear what the best starting place for future work here is. Perhaps a good place would be deeper mathematical analysis of the algorithm. Given this, it seems likely the field of pseudo-random number generation would benefit significantly from more collaboration with the field of mathematics. Indeed, it is possible more discoveries in mathematics could be made through this sort of collaboration.

Often failures go unexplored. However, the cryptographic value of learning more about how PRNGs work and do not work is so great that the exploration of these failures and similar ones could be critical to the continuing security and privacy of our governments, institutions, businesses, and individuals.

References

[1] P. L'Ecuyer, "Random numbers for simulation," Communications of the ACM, pp. 85 – 98, 1990.

[2] C. Dutang and D. Wuertz, "A note on random number generation," tech. rep., CRAN-R Project, 2009.

[3] R. G. Brown, D. Eddelbuettel, and D. Bauer, "Robert g. brown's general tools page." https://webhome.phy.duke.edu/~rgb/General/dieharder.php, 2022.

[4] M. M. Alani, "Testing randomness in ciphertext of block-ciphers using diehard tests," IJCSNS International Journal of Computer Science and Network Security, vol. 10, no. 4, 2010.

[5] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," Tech.

Rep. 800-22, National Institute of Standards and Technology, Gaithersburg, MD 20899-8930, 4 2010.

[6] P. L'Ecuyer, "Efficient and portable combined random number generators," Communications of the ACM, vol. 31, no. 6, pp. 742 – 750, 1988.

[7] P. L'Ecuyer, "Maximally equidistributed combined trustworthy generators," Mathematics of Computation, vol. 65, no. 213, pp. 203 – 213, 1996.

[8] P. L'Ecuyer, "Good parameters and implementations for combined multiple recursive random number generators," Operations Research, vol. 47, no. 1, pp. 159 – 164, 1999.

[9] J. Banks, ed., Handbook of Simulation: Principles, Methodology, Advances, Application and Practice, ch. Random Number Generation, pp. 93 – 137. New York: Wiley, 1998.

[10] P. L Ecuyer, F. Blouin, and R. Couture, "A search for good multiple recursive random number generators," ACM Transactions on Modeling and Computer Simulation, vol. 3, no. 2, pp. 87 – 98, 1993.

[11] P. L'Ecuyer, Random Number Generation, pp. 35 – 71. Springer.

[12] D. E. Knuth, The Art of Computer Programming, vol. 2. Boston, United States: Addison-Wesley, 1968.

[13] F. Panneton, P. L'Ecuyer, and M. Matsumoto, "Improved long-period generators based on linear recurrences modulo 2'," ACM Transactions on Mathematical Software, vol. 31, no. 1, pp. 1 – 16, 2006.

[14] P. Garrett, The Mathematics of Coding Theory. Upper Saddle River: Pearson/Prentice Hall, 2004.

[15] C. Shannon, "Communication theory of secrecy systems," Bell System Technical Journal, vol. 28, pp. 656 – 715, 1949.

[16] A. Carlson, S. R. Mikkilineni, M. W. Totaro, and C. Briscoe, "A venona style attack to determine block size, language, and attacking ciphers," The Seventeenth International Conference on Systems and Networks Communications, 8 2022.

[17] J. B. Fraleigh, A First Course in Abstract Algebra. Addison-Wesley, 7^{th} ed., 2003.

[18] J. Kelley, General Topology. Princeton: D. Van Nostrand Company, 1955.

[19] T. Cover and J. Thomas, Elements of Information Theory. New York: John Wiley & Sons, Inc, 2nd ed., 2005.

[20] M. E. O'Neill, "Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation," Tech. Rep. HMC-CS-2014-0905, Harvey Mudd College, Claremont, CA, 9 2014.

[21] B. Williams, R. Hiromoto, and A. Carlson, "A design for a cryptographically secure pseudo random number generator," pp. 864–869, 09 2019.

[22] P. Geffe, "How to protect data with ciphers that are really hard to break," Electronics, pp. 46, 99–100, 1973.

[23] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Cryptanalytic attacks on pseudorandom number generators," Fast Software Encryption, Fifth International Workshop Proceedings, pp. 168 – 188, 1998.