Victorian Electoral Commission

BALLOT DRAW REVIEW

BMM Australia Pty Ltd

18 July 2022

EXECUTIVE SUMMARY

The Victorian Electoral Commission (VEC) has requested BMM to audit the randomness of the new Ballot Draw service. This service has been updated since the previous version was evaluated by BMM in August 2020 in report VEC.1007.01, to make use of a more suitable random number generator and its available functions provided by the .NET software framework. The new random number generator used by the service produces unpredictable sequences of random values. The draw algorithm itself produces random draws from a uniform distribution. As a result of the statistical testing and source code review, BMM believes that the new Ballot Draw service provides uniformly distributed random draws fit for the intended purpose of randomly ordering candidates.

TABLE OF CONTENTS

1 PURPOSE OF EVALUATION

The Victorian Electoral Commission (VEC) has requested BMM to audit the randomness of the new Ballot Draw service. This service has been updated since the previous version was evaluated by BMM in August 2020 in report VEC.1007.01, to make use of a more suitable random number generator and its available functions provided by the .NET software framework. The source code used for the review was provided to BMM on 27 June 2022 and is included in Appendix A.

2 DESCRIPTION OF RNG

The Ballot Draw service runs in a Microsoft DotNet ASPNet Core 3.1 package running in a Docker container on a Linux server. The service draws random numbers using functions provided by the RandomNumberGenerator class in the System.Security.Cryptography namespace, which in turn draws from the system's cryptographically secure random number generator.

2.1 Seeding

The underlying system RNG is seeded by the system with entropy drawn from numerous system sources. This makes for an entirely unpredictable state after seeding, with no user managed seeding involved.

2.2 Scaling

The Ballot Draw service uses functions provided by the RandomNumberGenerator class to produce random numbers in usable ranges from a uniform distribution without introducing bias.

2.3 Security

The underlying system RNG uses a cryptographically secure algorithm, combined with periodic injection of entropy collected from system sources, to produce a sequence of random numbers that cannot be predicted.

3 STATISTICAL TESTING

Statistical tests were performed on the output from the RNG. Raw output from the RNG was subjected to a range of tests in the Empirical, Diehard and NIST test suites. Appendix B describes the tests run in each test suite.

3.1 Test Results

Each test tests the hypothesis that the RNG is a random source of numbers. A "p-value" is produced for each test run, which is the probability that a truly random process would produce the same or a more extreme result. P-values are expected to be uniformly distributed between 0 and 1. Each test is performed at least 100 times, and the p-values for each test are evaluated using an Anderson-Darling test. This produces a single p-value, which is the probability that the individual p-values have been produced from a uniform distribution.

Finally, the p-values from each test in the same test suite are combined using the Holm-Bonferroni method to provide an overall p-value. This process adjusts each p-value to ensure that the overall probability of accepting the RNG as random matches the confidence interval used. The overall p-value, equal to the minimum of the adjusted p-values, is compared to a specific alpha value to determine if the RNG is accepted or rejected as being random for a specific confidence interval. For a 95% confidence interval, the alpha value used is 0.05.

The following tables summarise the test results. See Appendix B for a description of the statistical tests used.

3.1.1 Empirical Tests

Test	P-values	99% Confidence
Frequency Test	1.000000	PASS
Serial Correlation Test	0.978384	PASS
Runs Test	0.978384	PASS
Gap Test	0.658824	PASS
Coupon Collector Test	0.978384	PASS
Subsequences Test	1.000000	PASS
Poker Test	0.978384	PASS
Overall	**0.658824**	**PASS**

Conclusion: The RNG is **ACCEPTED** as random at the 99% confidence interval.

3.1.2 Diehard Tests

Test	P-values	99% Confidence
Binary Rank 32x32 Test	1.000000	PASS
Binary Rank 6x8 Test	1.000000	PASS
Birthday Spacings Test	1.000000	PASS
Bitstream Test	1.000000	PASS
Count The 1's Stream Test	1.000000	PASS
Count The 1's Specific Test	1.000000	PASS
Runs Test	1.000000	PASS
Squeeze Test	1.000000	PASS
Overall	**1.000000**	**PASS**

Conclusion: The RNG is **ACCEPTED** as random at the 99% confidence interval.

3.1.3 NIST Tests

Test	P-values	99% Confidence
Approximate Entropy Test	1.000000	PASS
Block Frequency Test	1.000000	PASS
Cumulative Sums Test	1.000000	PASS
Discrete Fourier Transform Test	1.000000	PASS
Frequency Test	1.000000	PASS
Linear Complexity Test	1.000000	PASS
Longest Run of Ones Test	1.000000	PASS
Non-Overlapping Template Matchings Test	1.000000	PASS
Overlapping Template Matchings Test	1.000000	PASS
Random Excursions Test	1.000000	PASS
Random Excursions Variant Test	1.000000	PASS
Rank Test	1.000000	PASS
Runs Test	1.000000	PASS
Serial Test	1.000000	PASS
Universal Test	1.000000	PASS
Overall	**1.000000**	**PASS**

Conclusion: The RNG is **ACCEPTED** as random at the 99% confidence interval.

4 RANDOM DRAW ALGORITHM

The draw algorithm works by creating a list of indexes that correspond to each candidate. A new list is created by repeatedly selecting and removing an index at random from the first list and adding it to the new list, until the first list is empty, and the new list contains some permutation of all the indexes. This process correctly shuffles the positions uniformly without introducing any bias.

5 CONCLUSION

The new random number generator used by the Ballot Draw service produces unpredictable sequences of random values. The draw algorithm itself produces random draws from a uniform distribution. As a result of the statistical testing and source code review, BMM believes that the new Ballot Draw service provides uniformly distributed random draws fit for the intended purpose of randomly ordering candidates.

[illegible]

Christopher van Prooije, Senior Systems Consultant, Mathematics

APPENDIX A SOURCE CODE

The following is the source code in BallotDrawHelper.cs used by the Ballot Draw application. It was provided to BMM on 27 June 2022.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography;

namespace BallotDrawRandNumGen.WebApi
{
    public class BallotDrawHelper
    {
        private static readonly object syncLock = new object();

        public virtual int[] GenerateRandomPositions(int count)
        {
            if (count <= 0)
                throw new ArgumentOutOfRangeException("count");

            lock (syncLock)
            {
                //get the set of numbers from 1 to {count}
                var positions = Enumerable.Range(1, count).ToList();
                // create a list to hold the new numbers
                var perm = new List<int>();
                for (int i = 0; i < count; i++)
                {
                    // find the random index for list
                    int n = RandomNumberGenerator.GetInt32(positions.Count);
                    // push the value corresponds to the random index to the result list
                    perm.Add(positions[n]);
                    // remove the selected number from the lot
                    positions.RemoveAt(n);
                }
                return perm.ToArray();
            }
        }
    }
}
```

APPENDIX B STATISTICAL TESTS

The following tests were used to test the statistical properties of the RNG.

B1 EMPIRICAL TESTS

The Empirical Tests are based on the tests described by Donald Knuth in The Art of Computer Programming Volume 2: Seminumerical Algorithms (1968, revised in 1997). They test sequences of numbers scaled to specific ranges.

Frequency Test	Counts of each number occurring across the sample set.
Serial Correlation Test	Counts of non-overlapping groups of numbers occurring together. Group sizes of two, three, and four are tested separately.
Runs Test	Counts of ascending and descending sequences of numbers. Note that this is a different test to the Runs Test in the Diehard and NIST Tests.
Gap Test	Counts of the size of gaps between successive occurrences of a given number. Each number in the range is tested separately.
Coupon Collector Test	Counts of sequence lengths required to complete a full set of each number in the range.
Subsequences Test	Similar to the Serial Correlation Test for pairs of numbers, except looking at numbers separated by a specific gap. Step sizes of 5, 10, 15, and 20 are tested separately.
Poker Test	The sequence is split into groups of five. The number of unique values in each group is counted.

B2 DIEHARD TESTS

The Diehard Tests are based on the test suite published by George Marsaglia in 1995. They test sequences of raw binary output from the RNG.

Binary Rank 32x32 Test	Matrices are created using 32 32-bit words. The ranks of the resulting matrices are counted.
Binary Rank 6x8 Test	Same as the Binary Rank 32x32 Test, except each matrix is formed using 6 values, each taking 8 bits from successive 32-bit words with a specific offset. All possible offsets are tested separately.
Birthday Spacings Test	32-bit words are taken as values, sorted, and the spacings between them calculated. The number of spacings of the same size are counted.
Bitstream Test	Blocks of 2^18 values are treated as a stream of overlapping 20-bit values. The number of possible 20-bit values that are not found in each block is counted.
Count The 1's Stream Test	8-bit values are taken and assigned a “letter” based on the number of one’s appearing in the binary representation of each value. Overlapping groups of 5 “letters” are counted.
Count The 1's Specific Test	Similar to the Count The 1’s Stream Test, except 8-bit values are taken from successive 32-bit words with a specific offset. All possible offsets are tested separately.
Runs Test	Counts sequences of increasing and decreasing 32-bit words. Note that this is a different test to the Runs Test in the Empirical and NIST Tests.
Squeeze Test	A value of 2^31 is repeatedly multiplied by 32-bit words, dividing by 2^32 and taking the ceiling of the result each time. The number of successive words that are required to reduce the value down to 1 is counted. The value is reset to 2^31 and the process is repeated.

B3 NIST TESTS

The NIST Tests are based on the suite of tests released by the National Institute of Standards and Technology in Special Publication 800-22, Revision 1a (revised April 2010). They test sequences of raw binary output from the RNG.

Approximate Entropy Test	Similar to the Serial Test, count each possible m-bit value, except it does so for two adjacent m bit lengths and compares the two.
Block Frequency Test	Similar to the Frequency Test, except the data is split into equally sized blocks. The number of ones and zeroes in each block is counted.
Cumulative Sums Test	Random walks are created by converting the data to +1 / -1 for 1 / 0 respectively and summing consecutive values.
Discrete Fourier Transform Test	The data is transformed using a Discrete Fourier Transform. The number of peaks within the 95% threshold are counted.
Frequency Test	The number of ones and zeroes in the binary output is counted.
Linear Complexity Test	The length of the linear complexity of the random sequence is determined.
Longest Run of Ones Test	The data is split into equally sized blocks. The longest run of ones in each block is determined and counted.
Non-Overlapping Template Matchings Test	The data is split into equally sized blocks. Each block is searched for a specific pattern of bits and counted. A separate test is run for various bit patterns. Each bit pattern searched does not overlap with itself. That is, when the pattern is matched, the end of the pattern cannot be the start of another match.
Overlapping Template Matchings Test	Similar to the Non-Overlapping Template Matchings Test, except only one pattern is searched, which may overlap with itself.
Random Excursions Test	As with the Cumulative Sums Test, random walks are created by converting the data to +1 / -1 for 1 / 0 respectively and summing consecutive values. The number of times a given state is visited between returns to zero are counted. Separate tests are run for various states from -4 to +4, not including 0.
Random Excursions Variant Test	Similar to the Random Excursions Test, except the number of times the given state is visited is counted for the entire sequence. Separate tests are run for various states from -9 to +9, not including 0.
Rank Test	Matrices are created using 32 32-bit words. The ranks of the resulting matrices are counted. Note that this is fundamentally the same test as the Binary Rank 32x32 Test in the Diehard Tests, although the implementation may differ.
Runs Test	Runs of consecutive bits of the same value of various lengths are counted.
Serial Test	Counts of each possible m-bit values. Separate tests are run for various m bit lengths.
Universal Test	Distances between repeated patterns of bits are counted.

--- END OF REPORT ---