MAG My Array Generator (a new strategy for random number generation)

Abstract: MAG is an algorithm (cellular automata) that creates an apparently random stream. It is very simple and very fast. The performance is obtained by using an arbitrary wide word (multicolored cell) instead of traditional operating on a bit scale. Therefore eight basic operations are needed for creating a random word, which can be 16, 32, 64… bits wide. The MAG rule uses the selection programming structure as an alternative to Boolean / Algebraic functions for imposing complexity within the system. In addition, the MAG algorithm can be easily transformed to use a randomized approach to cipher design.

Keywords: MAG, random number generator, cellular automata, fast algorithms, stream cipher design, non-classical cryptography.

1 Introduction

This paper is an evaluation of a computational peculiarity (essentially, cellular automata), which can be applied to the field of random number generation. This idea comes from an apparent disaccord between how computational operations are theoretically defined and their practical application in mathematics, physics and computing.

Computational theory can be summarized as follows:

[TCOI] Computation operations can be characterized from Babbage's idea of an "Analytical Engine":

- "(1) The arithmetic functions $+,-,\times$ are operations (where $x-y=0$ if $y \geq x$)."
- "(2) Any sequence of operations is an operation."
- "(3) *Iteration.* The *n*-fold iteration of an operation *P* (where *n* is the number in a specified register, whose content is not affected by *P*) is an operation."
- "(4) *Conditional iteration.* If *P* is an operation and *T* is a test on the numbers in certain registers, then the result of iterating *P* until *T* succeeds is an operation.
- "(5) *Conditional transfer.* If *P* and *Q* are operations then the result of doing *P* if a test *T* succeeds, *Q* if it fails, is an operation.

[C#PI] The simpler formulation from Bohm and Jacopini's work demonstrates that all programs could be written in terms of only three control structures:

- (a) The sequence structure.
- (b) The selection structure.
- (c) The repetition structure.

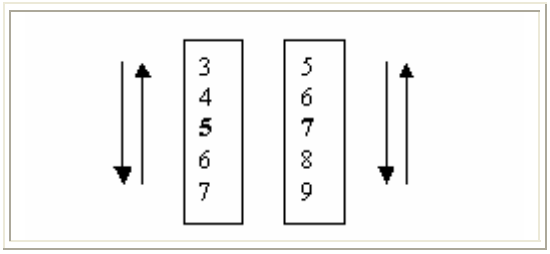
The above formulations can be translated to the programming lexicon as:

- (1) and (2) operations from Babbage's Analytical engine and (a) structure are incorporated in the common computer languages which means that every statement is executed one after the other in the order in which they appear in the program.
- (3) operation and (c) structure are ***for loop*** programming paradigm**.**
- (4) operation and (c) structure are ***while loop*** paradigm.

- (5) operation and (b) structure is ***if/then/else*** programming tool.

From the practical application perspective, almost all mathematical expressions can be managed by (1) (2) (3) and (4) computational operations (Babbage) or by (a) and (c) programming structures (Bohm and Jacopini). In short: Babbage's (5) operation or Bohm and Jacopini's (b) structure are not normally used for the explicit inner working of an algorithm as a mathematical function. Below is an example of how (5) and (b) may be explicitly used in the algorithm.

Example: Two variables *x* and *y* are used in the following function with the values 5 and 7 respectively. The first step of the function is to compare *x* and *y* and to transform the first variable (*x*). If $x < y$ add 4 to the *x* and keep only the rightmost value as a result, otherwise subtract 3 from *x* using the following rule: if the result is negative, add 10 to the result. A wheel with numbers from 0 to 9 can be imagined and the position can be changed going up for 4 places or going down 3 places. In this example, the pair (5, 7) is changed to the pair (9, 7).

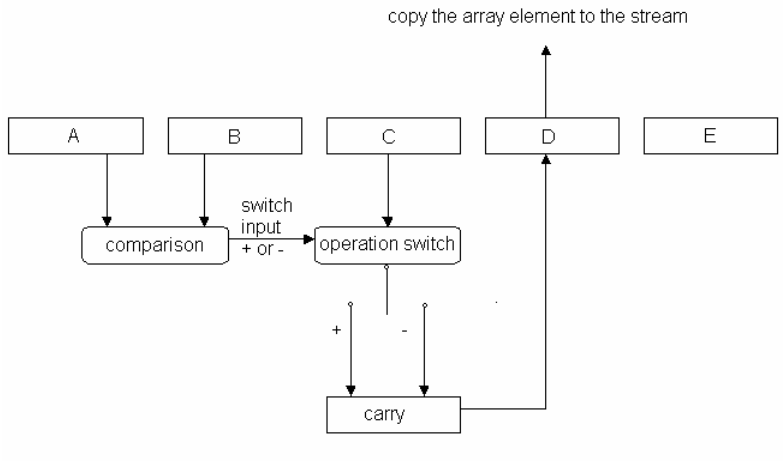


The second step is to transform the second variable *y*. Again, two variables are compared. If $y < x$ add 4 otherwise subtract 3 in same fashion as above. In this case, the pair (9, 7) is changed to the pair (9, 1). From there, the first and second steps are repeated with the following results:
(6, 5); (3, 2); (0, 9); (4, 6); (8, 0); (5, 4); (2, 1); (9, 5); (6, 9) and so on.

One characteristic of above algorithm is that the transforming operation is arbitrary and is decided (on the fly) on the particular relations between elements of the pair in question. My research examines this attribute in order to develop a new approach to random number generation.

2.0 Introduction to MAG (My Array Generator) Algorithm Design

An unsigned 32-bit integer array [A; B; C; D; E ….] and 32-bit element "carry" are used as the data structure for manipulating output sequences:



1. Compare A and B.
2. Position the operation switch to + or – according to the comparison outcome.
3. The carry element is xored with element C (+) or xored with one complement of element C (-) to form the new value for the carry.
4. The element D is xored with the carry to form a new value for element D.
5. The value of element D is forwarded to the output.
6. The whole operation is shifted to the right and all previous steps are repeated (B and C are compared, D changes the carry, and the carry changes E).

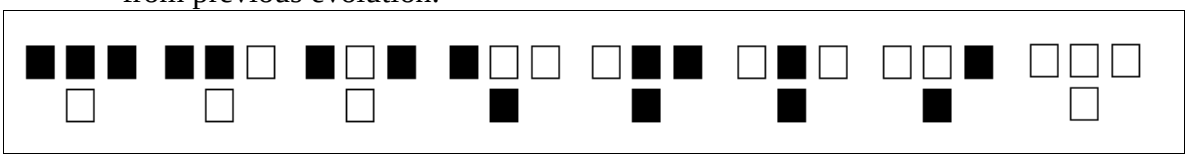
It should be noted that only the carry element and element D change value through one algorithm iteration, as shown in the previous table, which explains MAG's workings.

2.1 Wolfram's rule 30 cellular automata (CA) and MAG

Stephen Wolfram's rule 30 belongs to class 3 behavior cellular automata (CA). This class has a complex structure and can be regarded as a chaotic/random class.

Rule 30 [p869 NKS] can be expressed as:

- algebraic expression is $Mod[p+q+r+qr,2]$
- logic expression is $Xor[p,Or[q,r]]$
- Visually rule looks like figure below, where newly formed cell depends on three cells from previous evolution.

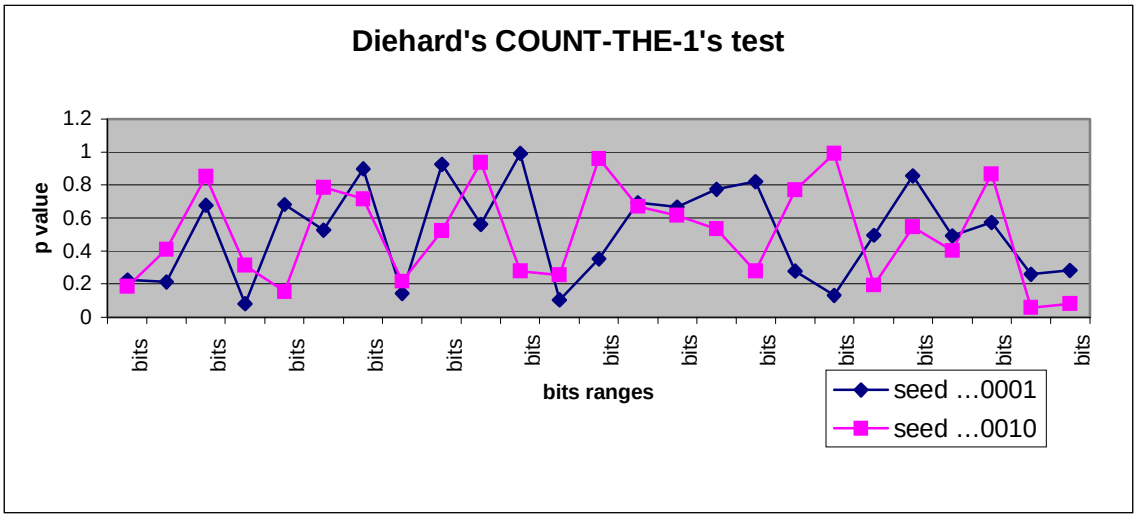


- English formulation is [p27 NKS]: “First, look at each cell and its right-hand neighbor. If both of these were white on the previous step, then take the new color of the cell to be whatever the previous color of its left-hand neighbor was. Otherwise, take the new color to be the opposite of that.”

There is a major difference between MAG and Stephen Wolfram’s rule 30 CA:

The rules for a one-dimensional two-color cell CA are based on a switching system where all the possibilities of three two-colored cells are mapped. That mapping can be generalized (as algebraic, logical, even lexical expression), as is the case with rule 30. There is no straightforward way to apply this particular rule (or class 3 behavior with the same properties) to multicolor cells. Someone has to search for the rule empirically, which becomes exponentially more difficult when larger words are used.

While traditional CA systems use two-color cells, MAG uses 32-bit (or any multi-bit word) color cells. The disproportion of cell sizes translates directly to the performance of the CA system. Using multicolor cells does not influence the 3rd class (chaotic) behavior of bits. Below is the distribution of the 1’s (for bits ranges 1-8, 2-9, 3-10, 4-11 …) for steams initialized by 32-bit unsigned integers 1, 2 (0…0001, 0…0010), which can illustrate the point:



MAG algorithm shows that algorithm selection criteria (Babbage’s conditional transfer) are actually underlying structure for class 3 behavior CA (which is only apparent in the lexical formulation of Wolfram’s rule 30). Consequently the MAG algorithm can be used for CA with arbitrary wide cells and still producing a random/chaotic evolution behaviors.

3. 0 MAG Analysis

There are only a few “algebraic” relations, which can be derived from the algorithm explained previously:

The dependence between the first generation child and parent, and second generation child and grandparent evolution cycles is shown in the next table. It is apparent that formulas cannot be written without the ± sign, which indicates that there is considerable difficulty in expressing MAG in algebraic terms.

Parent	Child-Parent dependence	Grandchild-Grandparent dependence
A	A+(X±D)	(A+(X±D))+((X±D±A±B±C) ±(D+(X±D±A±B±C))
B	B+(X±D±A)	(A+(X±D))+((X±D±A±B±C) ±(D+(X±D±A±B±C)) ± (B+(X±D±A)))
C	C+(X±D±A±B)	…
D	D+(X±D±A±B±C)	…
X (carry)	X±D±A±B±C	…

A, B, C, D, … are array elements;
X is the carry element and has the same word size as A, B, C, D, …;
+ sign in tables is XOR;
± sign means either XOR or one complement of XOR is the transforming operation;

It will be easy to reduce the tables’ dependences formulas if XOR was the only transforming operation. But because there is one complement XOR operation as well, instances of formulas tend to be more and more complex (and longer) evolution wise.

Another approach to analyze MAG is to observe evolution transformations on the bit level. But there is difficulty from the fact that binary branching (if else) is decided on the word level. It is difficult to pinpoint which bit in the word is going to decide branching path (which word has greater binary value).

It can be observed that there is no mathematical equations which can describe MAG as it is the case with Wolfram’s rule 30 (and for that matter rule 45) . That phenomenon implies that there is mechanical process, which cannot be defined by mathematical tools. In other words series of words produced by MAG is defined by initial state and cannot be reproduced or defined by any other means.

On the other hand MAG complexity can be expressed by software testing methods, particularly by cyclomatic complexity. “Cyclomatic complexity [ACM] measures the amount of decision logic in a single software module. It is used for two related purposes in the structured testing methodology. First, it gives the number of recommended tests for software. Second, it is used during all phases of the software lifecycle, beginning with design, to keep software reliable,

testable, and manageable. Cyclomatic complexity is based entirely on the structure of software's control flow graph." Below is simplified method for applying degree of complexity in respect to the binary branching of the program control.

"If all decisions are binary and there are p binary decision predicates, v(G) = p + 1. A binary decision predicate appears on the control flow graph as a node with exactly two edges flowing out of it. Starting with one and adding the number of such nodes yields the complexity. This formula is a simple consequence of the complexity definition. A straight-line control flow graph, which has exactly one edge flowing out of each node except the module exit node, has complexity one. Each node with two edges out of it adds one to complexity."[p 33 ST]

MAG evolution uses binary branching through iteration. Therefore complexity of path which one cell evolves increases exponentially. From software testing point of view MAG algorithm is unmanageable (it is not possible to cover all paths for testing or to predict algorithm behavior).

3.1 MAG's repetition period

It is apparent that the output stream cannot be unique forever. There are two possibilities, which will repeat the pattern eventually:

1. The stream reaches the initial state and starts to repeat itself:
$\underline{A}$=> B=> C=> D=> E=> $\underline{A}$=> B …

2. The stream reaches some state where repeating of the stream occurs:
A=> B=> C=> $\underline{D}$=> C=> $\underline{D}$=> C=> $\underline{D}$ …

If one to one correspondence between elements in the stream exists then the stream will behave as shown in the case (1). The case (2) will happen if some state in the stream can be reached from two points (B=> C and D=> C).

It is clear that the only transformation that occurs in MAG is xor and one complement of xor. In other words: from an array A (by MAG rule) an array A' is formed and *A is xored with A' to produce B*, which guarantees a unique output for any input. The evolving configuration of A can only produce B: A=>B. Also, MAG is reversible, (A<=B) which indicates one to one correspondence between evolution cycles ($A \Leftrightarrow B$).

Because one to one correspondence was not established from the beginning of development, a period repetition test was designed which covers both cases (1 and 2) mentioned above.

The period repetition test procedure is:

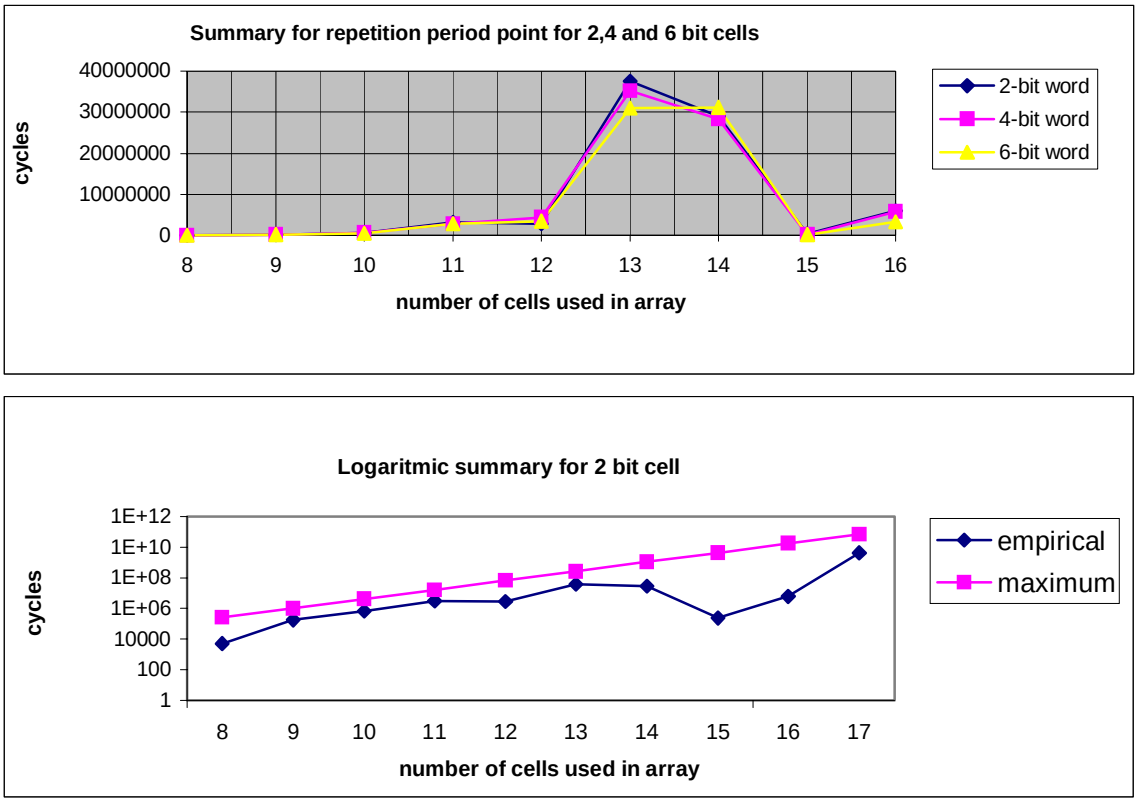
- The algorithm is run for a predefined number of iterations.
- If a starting point is not matched through evolution then the last iteration is recorded.
- The algorithm is run again and checking is performed to see if the last iteration is unique.

The reasoning behind this test is: if the last sequence part is unique in respect to the whole stream, all the previous iterations are unique as well.

The second issue about the period is: If the stream acts as in case one (1), does that stream walk through all possible states? And if not, what is the period dependence in respect to the number of cells and word length of the cells. To find the ratio of actual repetition periods to the maximum possible periods a set of empirical tests were performed.

3.1.1 The Period Testing Environment

There are two variables that can influence the period length. The variables are: word size (l) of the cell, and the number (n) of cells in array. The word size is varied employing 2-bit, 4-bit and 6-bit lengths and the array size is varied from 8 to 17 elements. Low figures are chosen in an attempt to actually reach a point where a period occurs. Every combination of word size and array size (3 word size x 10 array size) is performed 64 times and then the mean is extracted. The mean plots then would be the basis for evaluation. During testing, figures for the 17-word array size could not be obtained in reasonable time (the repetition period test was conducted to 2^{32} levels of iteration).



Below are some deductions from the charts shown previously:

- The summary plots for 8 to 16 word array sizes shows that the number of iterations does not significantly vary in respect to word size. That means the sequence depends on word size linearly (sequences are twice or tree times larger for 4-bit and 6-bit word cell configurations).
- Different numbers for reaching a repetition point within one combination of word / array size is constantly observed. That collaborates with the proposition that the period can be different for different initializations.

- Constantly reaching the period (for 8 to 16 word array sizes) shows that the proposed algorithm's every state of iteration produces a unique new state (one to one correspondence).
- It can be said that period trend of MAG rises with the amount of cells and it lies anywhere between Wolfram's rules 30 and 45 (p260 NKS). During the testing of the MAG design when a 127 cells wide array was used, more than several hundreds of gigabytes of data was produced and tested for period repetition. The repetition point was not reached.

3.2 MAG's performance

Two methods were used to determine the relative performance of the proposed algorithm.
One method was to directly time the proposed algorithm and compare results with the Mersenne Twister, an algorithm which is widely regarded as the state-of-the-art in random number generators. The second approach was to analyse the number of computations per output for the proposed algorithm and compare that value with similar methodologies and figures.

3.2.1 Mersenne Twister (MT) vs. MAG

The source code (19937int.c) for the Mersenne Twister was found at http://random.mat.sbg.ac.at/news/ and that link is mentioned and recommended in the MT white paper. It is essentially the same as the source code in the white paper except that the output is an unsigned long integer, which is consistent with the proposed algorithm.

Relevant hardware and software:
Pentium MMX 200Mhz Intel processor 196 Mb RAM
Operating system: Red Hat Linux version 6 GNU compiler and C language (shipped with Red Hat 6)

Output size for both algorithms was 10,000,000 long unsigned 32-bit integers. Three case scenarios were applied:
- The MT and MAG algorithm output was via the "fwrite" c function on to a binary file.
- MT and MAG algorithm was iterated for production of 10,000,000 long unsigned 32-bit integers but no output was produced.
- The same as for case 2 (immediately above) except MAG was optimized (one loop was unrolled).

The results indicate that MAG performs better. The following table summarizes the performance testing for MT and MAG.

	MT	**MAG**
LINUX's time command	User time in sec	User time in sec
Output via "fwrite"	15.110	13.630
No output	5.710	3.690
No output; MAG optimised	5.710	0.860

3.2.2 Fast Software Ciphers vs. MAG

A comparison is made between the numbers of simple operations contained in each algorithm. This method is used to compare the "(alleged) RC4" algorithm with the proposed (MAG) algorithm. The rationale for this approach lies in the fact that the proposed algorithm is far from an optimal design therefore any empirical test in these circumstances cannot reveal an objective result. Comparing C source code can at least give an indication of the proposed algorithm's performance. Below are the ARC4 ((alleged) RC4) source code and MAG source code:

```
     i=(i+1) & 0xFF;          /*      update      i      */
     ARC4
     tmpI=S[i];
     j=(j+tmpI) & 0xFF;          /* update j */
     tmpJ=S[j];
     S[j]=tmpI;               /* swap S[i], S[j] */
     S[i]=tmpJ;
     t=(tmpI+tmpJ) & 0xFF;           /*  compute  ``random''
     index */
        *p ^= S[t];                  /*  XOR  keystream  into
data */
```

There are four Boolean operations (three *&"and"* and one ^*"xor"*).
There are also four assignments.

```
if           (ptr[compA]           >           ptr[compB])
MAG
     ptr[carry] ^= ptr[input];
else
     ptr[carry] ^= (~ptr[input]);
ptr[output] ^= ptr[carry];
outFile.write(reinterpret_cast<const
char*>(&ptr[output]),WORDSIZE_32);
```

One comparison.
Either two or three Boolean operations (two ^*"xor"* or two ^*"xor"* and one ~*"one complement"*) depending on which branch is executed.
Four assignments.

If the above summary assumptions hold true, it can be asserted that the proposed algorithm is around four times faster than the ARC4 algorithm if a MAG cell is 32-bits wide (1 Byte vs. 4 Bytes output for approximately the same operations). Also MAG is not limited to 32-bit output (may be 64-bit, 128-bit, etc.) as ARC4 is limited to one byte (B. Schnier actually proposed a 16-bit version in his Applied Cryptography; while a 32-bit version requires a 2^{32} element wide working array).

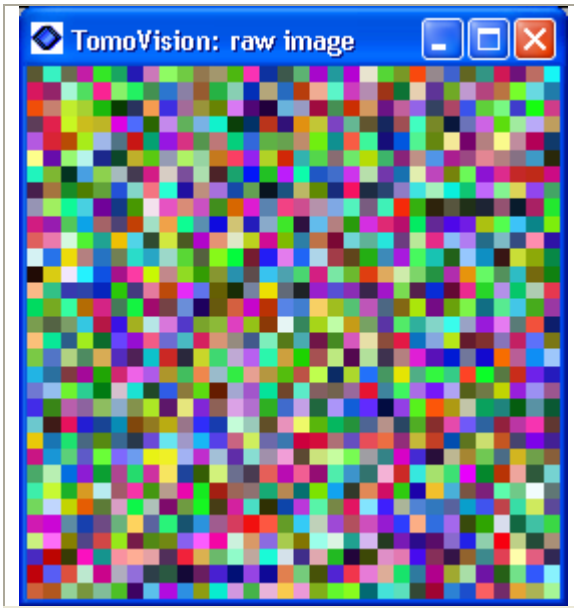
3.3 MAG empirical randomness testing

MAG output streams were tested for patterns in every stage of development (several gigabytes of data) and no patterns were found. Systematic (recorded) pattern testing was performed on 500 megabytes of output data and the details of these results are discussed below.

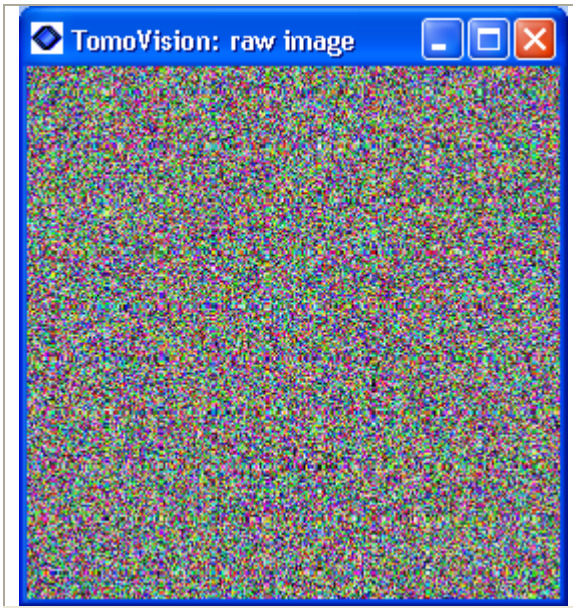
ENT, Diehard and CryptX-98 are test suites used for empirical testing of MAG and should represent the wide spectra of possible statistical analysis. The first two test suites (ENT and Diehard) are highly recommended in the sci.crypt newsgroup, an online community of cryptographers and interested parties. CryptX-98 is a QUT (Queensland University of Technology; Brisbane; Australia) developed program for testing stream ciphers, block ciphers and key generators using “black box” testing methods.

First of all the seed is produced by the COMBO RNG, which is part of the DIEHARD testing suite [TD]. Secondly, the resulting seed is used to produce 50 files over 10 megabytes each. Following these procedures, three batteries of tests are applied to each file.

It can be concluded that the 50 files produced by MAG pass the three batteries of tests (ENT, DIEHARD, CRYPTX-98), which suggests that considerable confidence can be accredited to MAG. The source code for repeating results can be found at http://www.geocities/radence_v or requested by email radence_v@yahoo.com. The next illustrations show 32-bit (RGBA) representations of MAG’s output in the .raw format using TomoVision freeware (Virtual Magic Inc. http://www.TomoVision.com).



32 * 32 resolution; 32 bit color



256 * 256 resolution; 32 bit color

3.4 About MAG non-linearity

It appears that MAG algorithm is essentially a linear type of RNG. But that is not the case. If table from section 2.2 is analyzed closer, the two relations can be observed:

- a child cell is dependant on a corresponding parent and all cells between which can be transformed in exponentially different ways. The transforming function can be in 2^N states where *N* is number of elements in an array (pair D; D+(X±D±A±B±C)).
- Neighboring cells are different by at least one value with potential that one value in equation is inverse (pair A+(X±D); B+(X±D±A)). That practically means a child is produced by xoring corresponding parent and either neighboring cell or neighboring cell inverse.

These correlations can not be linear in nature if XOR and one complement XOR is used as "**±"** operators. The empirical randomness testing performed on MAG suggests that usual patterns observed in linear type of RNG are not present in MAG. For example:

- Shift Register Family fails several Diehard tests [ACV] (MAG passes all).

LATTICE	PARKLOT	MTUPLE	OPSO	BDAY	OPERM	RUNS	RANK
Pass	FAIL	FAIL	FAIL	FAIL	Pass	FAIL	FAIL

- The similar table shows LCG (Linear Congruential Generators) [ACV] (MAG passes all).

LATTICE	MTUPLE	OPSO	BDAY	OPERM	RUNS	RANK
FAIL	Pass	FAIL	FAIL	Pass	Pass	Pass

- MAG passes Serial Correlation Coefficient [ACP p64-65] test (ENT testing suite).

Also is important to mention that MAG consistently passes CryptX-98 Linear complexity test based on the Berlecamp Massey algorithm.

4.0 About cryptography

CA (cellular automata) is known to the cryptography community. The Wolfram's rule 30 is around for quite some time (proposed 1985) [CCA] and Meier and Staffelbach [CP] presented crypto-analysis which showed that original proposal (127 bits wide CA) is not secure and suggesting security around 1000 bits wide CA. The latest Wolfram's rule 30 proposal was convincingly presented in Wolfram's NKS [page 605]. There is some of Wolfram's comment s about rule 30 security [page 606 NKS] "… with standard methods of crypto-analysis, as well as a few others, there appears to be no easy way to deduce the key for rule 30 from any suitably chosen encrypting sequence…As a practical matter one can say that not only have direct attempts to find easy ways to deduce the key in rule 30 failed, but also-despite some considerable effort-little progress has been made in solving any of various problems that turn out to be equivalent to this one".

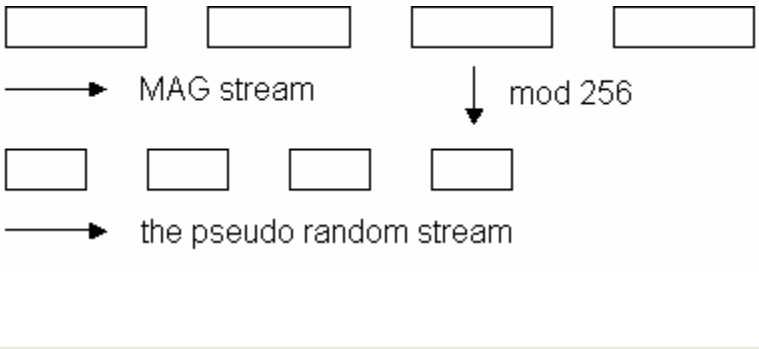
However it turns that security of the class 3 CA can be argued on computational irreducibility principle alone. The paradigm of computational irreducibility of the 3rd class CA, is explained and extensively argued in [NKS p737-750] and is applicable to MAG. In short, a class 3 CA evolution cannot be simulated by any other means but repeating the same CA with the same initial conditions. It is also important that steps in an evolution cannot be skipped or calculated in advance. To inspect the state in *n*th step of evolution all *n* steps between must be performed first. In that respect computational irreducibility stays in odds with current chaos theories which state that complexity resides in initial condition of the system and if an initial condition of the complex system is known there are laws / math which can predict future states of the system.

From the practical point of view class 3 CA output cannot be patterned in any way. That phenomenon confirms the principle of computational irreducibility and it contradicts common view that every algorithm / machine (random number generator) will fail eventually empirical random testing.

From that reasoning the reduction of the 3rd class CA evolution knowledge makes ideal situation for producing one way function. In particular the 3rd class CA evolution is easy to produce, and if partial information of that 3rd class CA evolution is given there remain only exhaustive search to recover original evolution state because only exact state and exact rule and exact step can give knowledge of the system (the computational irreducibility principle). Stephen Wolfram shows visually how reduction in evolution output exponentially decreases decrypting prospects [NKS p605].

4.1 MAG and Cryptographical applicability

It is straight forward procedure to reduce knowledge of the MAG stream. Simple hash function is applied to the every cell in the evolution. See following figure.



This approach takes one byte from each of the 32-bit cells in the MAG output stream by applying the mod 256 operation to create a pseudo random stream (secure stream).

This proposal is based on the assertion that is not possible to derive any knowledge of the next evolution cycle without a full knowledge of the previous cycle and the corresponding algorithm rule. The table from section 2.2 is useful to illustrate the point.

Parent	Child-Parent dependence	Grandchild-Grandparent dependence
A	**A+(X±D)**	(A+(X±D))+((X±D±A±B±C) ±(D+(X±D±A±B±C))
B	**B+(X±D±A)**	(A+(X±D))+((X±D±A±B±C) ±(D+(X±D±A±B±C)) ± (B+(X±D±A)))
C	C+(X±D±A±B)	…
D	**D+(X±D±A±B±C)**	…
X (carry)	X±D±A±B±C	…

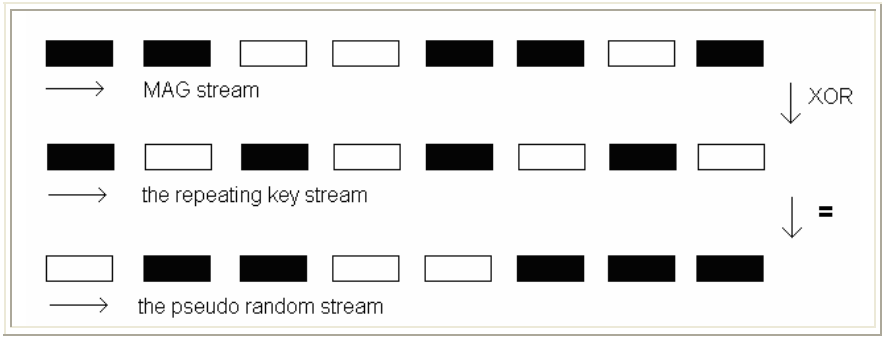
For example if the "mod 256" approach is used, one byte from the observed MAG stream clearly reduces 32-bit wide cell to 24-bit wide uncertainty for the corresponding cell in the MAG stream (the 32-bit amount of possible states that the cell can take is reduced to 24-bit). From the dependencies mentioned in section 2.2 and section 6 it may be possible to reduce one cell uncertainty further. Particularly relation of the pair A+(X±D) and B+(X±D±A) (neighbors; the relation column wise) appears to be more promising one. It transpires however, that further reduction is irrelevant (unless the reduction is total) for the time cost evaluation because to validate the guessed value of the cell, the whole evolution has to be used to predict the next cycle.

Because the knowledge of one state of the evolution cycle is needed to accomplish complete knowledge of the system, the traveling with the relations column wise (guessing game) has to go through the whole column, which amounts to the relation step row wise (pair D; D+(X±D±A±B±C)). Furthermore to relate cells more than one step either column or row wise increases the uncertainty exponentially.

If a cell's possible values (amount of elements that satisfy relation from the observed byte to an unknown 32-bit word) is enumerated as g and the number of cells as n then the computational time cost can be expressed as $O(g^n)$. Therefore if g is 1 then the cost is in polynomial time, otherwise (g>1) cost is in non-polynomial time and continues on exponentially, depending of the amount cells in an array. Essentially, to solve the MAG is an all or nothing proposition (the case g=1 or the case g>1), meaning that the reversing procedure (one-way function reversal) shall predict the whole evolution array state at once to qualify for a P class of problems.

Other approaches to produce a pseudo random stream from MAG are:

- The proposal by Stephen Wolfram [p605 NKS], which essentially sample fewer cells to the crypto stream than are actually produced by the CA. That approach is affecting the performance of a cipher. For example if every fourth element of the MAG stream is put to the output the performance of MAG is about the same as ARC4 (which is the case with "mod256" version as well).
- The approach which combines a MAG 32-bit stream with a stream that is made by repeating a secret key to produce a pseudo random stream (secure stream). See next figure. This approach influences performance by adding one operation.



4.2 MAG security / performance

One aspect of MAG is that the algorithm is in *O(1)*complexity in respect to the level of security (8-9 basic operations is needed per byte regardless on desired security level?!). Security/performance examples for 32 and 64 bit word hardware are listed below, where first example is actually implemented one.

For the 127 cell / 32-bit word / mod 256 hash variant $g = 2^{24}$ (where *g* is 32-bit / mod 256 hash) and $n = 127$ (where *n* is number of cells) giving security magnitude from 2^{127} to $(2^{24})^{127}$ and performance slightly slower than ARC4 (9 operations per byte output).

For the 511 cell/ 64-bit word/ mod 2^{32} hash variant $g = 2^{32}$ and $n = 511$ giving security magnitude from 2^{511} to $(2^{32})^{511}$ and performance around 4 times better than ARC4 (9 operations per 4 bytes output).

The MAG implementation is written in C++ for 32-bit hardware and for research purpose only. Input is a seed (key) which has to be at least one 32-bit word (unsigned long int). Output is a pseudo random or a cryptographically secure pseudo random stream of desired length. The period check functionality is implemented as well (for more details see http://www.geocities/radence_v/).

5.0 Discussion

MAG computational irreducibility opens way for randomized approach to stream cipher design [SC]. The rationale behind this approach is to create an unfeasibly large problem for the crypto-analysts to solve. The idea is to maximize the area of crypto analysts' investigation while the secret key remains relatively small in size. For example, if there is large public random string, the secret key can specify which part of that public string is gong to be used for encryption. From crypto-analytical perspective, the only option is to search through the whole public random string. The security of this encryption scheme is a ratio of the size of the encrypted message to the size of the public random string.

The MAG set of all possible states for particular setting (127 cells 32-bit wide setting gives 32^{127} states) can replace public random string from randomized approach scheme. In the same way it is impossible to determine where the MAG hashed output string in question starts because the string position in the set is statistically / empirically indistinguishable. That MAG string is defined by evolution state only. If that state is hashed, that creates many to one relationship. There is only two ways to recover state of evolution. One is to search through 32^{127} bits long

string (number of all MAG possible states). Other is to search through one to many relationship which has ratio of $\frac{1}{24^{127}}$ (Note that only complete knowledge of state evolution is needed for stepping through evolution).

The advantage of the randomized approach is: A set of criteria such as linear complexity, nonlinearity, statistics, confusion and diffusion does not have to be addressed directly as is the case with more complicated system-theoretic approaches. The whole security issue is shifted to the computational irreducibility principle alone.

The concept of security by obscurity can be applied to the MAG evolution as well. It is difficult to have any insight in a observed stream if the stream algorithm is unknown and there are no statistical biases. MAG empirical studies show lack of statistical biases MAG cell evolve by executing one of the two operations. If whole array iteration is examined, it can be seen that the two operations occurrence position cannot be predicted. For 127 cell wide MAG array there is 2^{127} different function configurations which can evolve one step of an evolution. Because of high complexity of MAG from the algorithm testing perspective it is difficult to manage path coverage of MAG branching, especially if every branch path is equally possible. Therefore for every step through evolution, there is a different function (which is hidden by hash /one to many relationship) from the set of 2^{127} functions.

6. Summary

The MAG algorithm strategy shows some desirable attributes for creating an PRNG:

- There is no evidence of patterns or regularities to be found for the MAG algorithm sequence outputs.
- The test and comparison shows that MAG outperforms the best RNG / cipher (MT and ARC4).
- The empirical studies show that MAG's period is relatively high (around $2^{0.5N}$ where N is the number of cells). Although there are no explicit (formal) guarantees for the period length, the period is easily tested against users' requirements.
- It is easy to transform the MAG output as a primitive for the randomized stream design approach. All security depends on computational irreducibility principle alone.

"The world can be seen as a process of flow and change, with the same material constantly going around and around in endless combinations. Geneticist Richard Lewontin called these scientists "Heraclitians," after the Ionian philosopher who passionately and poetically argued that the world is in a constant state of flux. "When I read what Lewontin said," says Arthur (Brian Arthur), "it was a moment of revelation. That's when it finally became clear to me what was going on" I thought to myself, `Yes! We're finally beginning to recover from Newton. `" [C p335]

7 References

[ACM]
"A Complexity Measure," McCabe, T. IEEE Transactions on Software Engineering, December 1976.

[ACP] "The Art of Computer programming" Volume 2 D.E. Knuth Addison-Wesley, 1969 ISBN 0-201-89684-2.

[ACV] "A Current View of Random Number Generators" G. Marsaglia 16th Symposium on the Interface, Atlanta, 1984. It appeared in the Proceedings of the Conference, published by Elsevier Press.

[C] Complexity "The Emerging Science at the Edge of Order and Chaos" M. Mitchell Waldrop Touchstone 1992 Q175.W258 1992 ISBN: 0 – 671 – 76789 – 5

[CCA] "Cryptography with cellular automata", Advances in Cryptology–CRYPTO '85 S. WOLFRAM, (LNCS 218), 429–432, 1986.

[CP] "Correlation properties of combiners with memory in stream ciphers", Advances in Cryptology–EUROCRYPT '90 W. MEIER AND O. STAFFELBACH, (LNCS 473), 204–213, 1991.

[C#PI] p74 "C#: Programmer's Introduction" Deitel Developer Series ISBN 0-13-046132-6 referencing -> Bohm, C., and G. Jacopini, "Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules," Communications of the ACM, Vol. 9, No. 5, May 1996, pp. 336-371.

[NKS] "A New Kind of Science" Stephen Wolfram; ISBN I-57955-008-8; QA267.5.C45 W67 2001

[SC] "Stream Ciphers," R.A. Rueppel, Contemporary Cryptology: The Science of Information Integrity, G.J.Simmons, ed, IEEE Press 1992, pp. 65-134.

[ST] "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric"
Arthur H. Watson Thomas J. McCabe Prepared under NIST Contract 43NANB517266
Dolores R. Wallace, Editor Computer Systems Laboratory National Institute of Standards and Technology Gaithersburg, MD 20899-0001 September 1996.

[TCOI] p-57 "The Confluence of Ideas in 1936" Robin Gandy paper published in The Universal Turing Machine A Half-Century Survey by Springer-Verlag ISBN 3-211-82628-9 (see notes (D) on page 57 for further references)

[TD] The Diehard battery download at http://stat.fsu.edu/~geo/diehard.html