Masarykova univerzita
Fakulta informatiky

Usage of evolvable circuit for statistical testing of randomness

Bachelor thesis

Martin Ukrop

Brno, spring 2013

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Martin Ukrop

Advisor: RNDr. Petr Švenda, Ph.D.

Acknowledgement

Thanks will be here.

Abstract

Abstract will be here.

Keywords

Keywords will be here.

Contents

1 Introduction

Text ...

2 Statistical randomness testing

- idea: statistic (maths) -> test
- fast
- universal
- usage: assess quality of (pseudo)random data, ???

2.1 Statistical Test Suite by NIST

- nist standard
- short description (?)

2.2 Diehard battery of tests

- author
- one of the first and most-well known in those years
- old, but still considered "gold standard" along with sts-nist
- short description of tests (?)

2.3 Dieharder: A Random Number Test Suite

- framework idea
- progress: diehard -> sts-nist -> new

2.4 Limits and disadvantages of statistical testing

- idea -> test (idea is always the predecessor)
- check only one specific property
- can only rarely be adapted to specific situation
- results interpretation (what is wrong?)

3 Evolution based randomness testing

- introduction, need for other method for randomness testing
- idea: instead of math analysis we will let program guess the correct analysis for this case
- advantages will be: no prior math knowledge needed, can potentially discover new metrics fir statistical randomness
- we will use evolutionary algorithms, namely genetic programming

3.1 Basic principles of genetic grogramming

- inspired by biology, method of supervised learning
- we create population of random individuals (solvers)
- we evaluate fitness of each individual (pre-generated data along with solutions)
- so called "fitness function" - crucial part!
- we make crossover among the best (survival of the fittest)
- we randomly apply "mutation" (to avoid getting stuck in local optimum)
- we iterate untill desired success rate is achieved
- points to keep in mind when using GA
  - only tasks with available partial solutions can be used (see next)
  - small change in solver should result in small chenge in fitness
  - designing the fitness function is crucial
  - fine-tuning GA settings (mutation/crossover probability, population size) is important
- generating data with solutions for learning can be computationally expensive
  - possibility to use them multiple times
  - risk of overlearning, individuals might learn to solve this particular instance of problem

3.2 Using software-emulated circuits

- represent individual in GP as a hardware-like circuit solving the problem
- circuit has gates with elementary functions and interconnecting wires
- circuits are emulated by software, internally represented as array of unsigned integers
- short description of functions
- it would be sufficient to allow nand only, however more functions improve understandability of evolved circuit for humans, also enables us to limit the circuit to small number of layers and nodes
- small similarity to neural networks, deep neural netowrks in particular

3.3 General working principles of EACirc

- EACirc is framwork for general problem solving using sw-emulated circuits and EA

- based on SensorSim developed at LaBAK, FI MUNI, further improved by Matej Pristak and Ondrej Dubovec
- main modules (+picture):
  - GAlib (MIT), used for evolution
  - project module, used for generating data for supervied learning (test vectors)
  - circuitg emulator, used for running circuits
  - evaluator, used for interpreting circuit outputs and computing fitness of individual circuits
  - set of random generators, needed since computaion is randomized
  - testing files, CATCH
  - help libraries (tinyXML)
- external static checker used for consistency-checking

3.4 Current capabilities of EACirc

- old core, basic capabilities of EA provided by GAlib and software emulation taken from work by Matej Pristak, originally from SensorSim
- main object model has been revised to uitlize the principle of modules, thus enabling integration of multiple projects and evaluators according to actual needs.
- guaranteed bit-reproducibility now - use of random generators revised, hierarchical system of generators created, any run can be reproduced by providing original input files and one central seed => crucial to experiment verification
- bit-reproducibility enabled us to implement computation recommencing - EACirc can save and load its complete internal state => useful for computation-expensive experiments (when the machine is rebooted, we can continue from last saved state instead of starting allover again)
- evolved circuits are exported to 4 different formats (from SensorSim or original work on EACirc?), binary (can be reloaded into EACirc if needed), text and graphical (using Graphviz) (used to ease humal analysis of evolved results) and C source code implementing the circuit (useful for independent verification of circuit work)
- new static checker created, circumvents most of the EACirc framework (especially circuit emulator), used for static checking on pre-generated test vectors
- project for distinguishing between eStream cipher output and random stream of data, taken from work by Matej Pristak and slightly revised to operate withing the new object model and allow more detailed configuration
- project for distinguishing between SHA-3 function output and random stream of data, ideas and hash functions implementations taken from Ondrej Dubovec, gest vector geenration reimplemented from scratch
- small project for distinguishing among external binary files

Most of the code that was taken over was revised and slightly refactor to ease the understanding of its function and to standardise naming and principles used in EACirc. For further details, use and developmetn documentation see EACirc wiki at GitHub.

Note, that EACirc is wider project beyond the scope of this thesis. Some parts were added and/or redesigned in pre process, so not all experiments had all now-supported features. Coinfiguration files may not be compatible across versions, etc.

4 Experiment settings and output data

- introduction - this chapter will describe the settings used in the experiments...
- our reasons for using these settings
- description of output files
- interpretation of result numbers

4.1 EACirc settings

- most of the settings taken from Matej Pristak's thesis (with no optimality verification)
- GA settings (population size 20, prob mutation 0.05, prob crosover 0.20)
- 30000 generation with test vector sets changed every 100th generation (thus, 300 unique test sets altogether)
- circuit - 5 layers, 8 functions in layer, input 16 bytes, output 2 bytes, maximum of 4 connectors (?), all functions allowed except for READX
- distinguisher - correct output for stream 1 id 0x00, correct for stream 2 is 0xff
- evaluator - "agent based", each byte is separate output, less or more than 128
- fitness function = #(correctly predicted vectors in set)/#(all vectors in set)
- 1000 test vectors in each test set
- only distiguisher experiments, always used 500:500 (according to Matej Pristak, imballance causes test vectors to only guess what type is more frequent in current run)
- experiment-specific settings described in appropriate section with results

4.2 EACirc output data

- main goal: finding strong distinguisher (over 99% for 50 consecutive generations)
- displayed average stable generation across 30 independent runs
  (stable = fitness over 99% for at least next 50 test sets)
- if none stable generation was found, average average maximum fitness after test vector change is displayed in parentheses.

4.3 Random data sources

- EACirc is randomized algorithm -> need good random data source
- in most experiments a distinguisher from random data stream is being evolved => crucial to have extremely realiable random data source
- using quantum random data, generated by measuring quantum effects
- 2 different sources, HU Berlin, institute in Croatia
- these 2 sources thoroughly compared, see section 5.2

4.4 Settings and output data for statistical test batteries

- to compare our results with existing statistical batteries, statistical randomness of all generated streams was checked using STS NIST and Dieharder
- 250 MB of data generated for statistical testing, same streams used for both STS NIST and Dieharder
- STS NIST
  - 100 runs, 100000 bits per run (=> 11.92 MB used for testing)
  - all tests were run, significance level of ???
  - some runs had problems with tests RandomExcursions and RandomExcursionsVariant, to ensure statistical accuracy of results, these test are ommited in results
  - for each test, following stats are computed:
    - p-value for each run - if this p-value is out of bounds of the interval determined by the significance level, the run is considered failed
    - the number of passed runs is infered
    - the cumullative p-value of all 100 runs
    - if either the cumulative p-value or the number of passed runs for this particular test is out bounds of the interval determined by the significance level, this test is considered as failed
  - results expressed as cumullative score for the entire stream, 0 for each failed test, 1 for each passed test. expressed as part of total 162
- Dieharder
  - test corresponding to original Diehard (except for Diehard sums test, due to probable error in test implementation)
  - 1 run of each test, stream length determined by the test itself (comes from design of Dieharder, we want to prevent from revinding the file)
  - total data used for testing: ??? MB (smallest test: ??, biggest test: ??)
  - each test interpreted as PASSED, WEAK or FAILED (significance levels of ???)
  - results for stream again in the form of cumullative score (0 for failed, 0.5 for weak, 1 for passed), out of total 20

5 Control distinguishers

- introduction – the need of reference numbers before analysis
- we need to define what does it mean "indistinguishable" in our setting
- we use quantum random data from Humboldt Universitat and Quantum random bit generator service as a standard for randomness

5.1 Looking for non-randomness in quantum random data

- trying to distinguish quantum random data from quantum random data
  => we presume to fail
- using random data from Quantum random bit generator service
- statistical batteries: data are random (Dieharder: 20/20, STS NIST: 188/188)
- evolution: no stable distinguisher found, AAM of 0.52 (differences in various runs in 3rd or 4th decimal place)
- presumption: dependence on test set size and population size
- presumption confirmed (Table 5.1), AAM decreases with smaller population and bigger test set size

		number of test vector in a set					
		200	500	1000	2000	5000	10 000
individuals in population	5	–	–	(0.509)	-	-	-
	10	–	–	(0.514)	-	-	-
	20	(0.544)	(0.527)	(0.520)	(0.514)	(0.509)	(0.506)
	50	-	-	(0.526)	-	-	-
	100	-	-	(0.530)	-	-	-

Table 5.1: Dependence of AAM on population size and test vector set size.

5.2 Distinguishing quantum random data from different sources

- distinguishing streams of quantum random data from Humboldt University and streams of quantum random data from Ruđer Bošković Institute
  - Quantum Random Bit Generator Service, Centre for Informatics and Computing, Ruđer Bošković Institute, Zagreb, Croatia
  - Quantum Random Number Generator Service, Department of Physics, Humboldt University, Berlin, Germany
- 6 files of 5 MB from each source
- fixed initial reading offsets as (0,0)
  => same data from given file in each run
- looking for distinguisher for each pair
- interpretation of results (Table 5.2):

  - data from both sources are equally random for our purposes
  - there is no single statistically different stream in these
    =>they can be used interchangeably

		QRBG service (Ruđer Bošković Institute, Croatia)					
		stream 1	stream 2	stream 3	stream 4	stream 5	stream 6
QRNG service (HU, Germany)	stream 1	(0.521)	(0.520)	(0.520)	(0.519)	(0.519)	(0.519)
	stream 2	(0.158)	(0.519)	(0.520)	(0.520)	(0.520)	(0.519)
	stream 3	(0.519)	(0.522)	(0.519)	(0.520)	(0.519)	(0.519)
	stream 4	(0.520)	(0.520)	(0.519)	(0.518)	(0.519)	(0.519)
	stream 5	(0.519)	(0.520)	(0.519)	(0.518)	(0.520)	(0.520)
	stream 6	(0.520)	(0.519)	(0.520)	(0.520)	(0.519)	(0.519)

Table 5.2: Distinguishing binary quantum random streams from independent sources.

5.3 Uncompressed audio streams

- 12 files
  - 3 quantum random files
  - 3 noise files (white, pink, brown)
  - 3 noise files with oscillating volume (2 cycles in given 30 seconds)
  - 3 samples of transcendental khaoblack metal music
- each file is 30sec of uncompressed WAV audio (5.3MB)
- evolving distinguisher for each pair
- interpretation of results (Table 5.3):
  - quantum random stream are undistinguishable (we already know)
  - random stream 2 is more "noise-like"
    => reminds us to use random data cautiously, and use statistics to evaluate results
  - uncompressed WAV audio is quite easily distinguishable from random stream of data (generally over 80%)
  - different types of noise can be quite successfully distinguished from one another (generally over 70%)
  - it is difficult to distinguish noise from its oscillating version (around 58%)
  - when using oscillating noise for distinguishing, fitness is not oscillating
    => volume is not statistically important in these sample noise files
  - given samples of metal music can be quite successfully distinguished from noise
  - given samples of metal music nearly cannot be distinguished from one another
- most of the runs have slow rising tendency in fitness
  => if more generations, the average maximum value might be slightly higher
-
- noise after 128kbs mp3 compression had 480 kB

		random streams			noise (true)			noise (via mp3)			metal music		
		random stream 1	random stream 2	random stream 3	white noise	pink noise	brown noise	white noise (via mp3)	pink noise (via mp3)	brown noise (via mp3)	metal music (sample 1)	metal music (sample 2)	metal music (sample 3)
random	random stream 1	n/a	(0.52)	(0.52)	(0.52)	(0.80)	(0.84)	(0.59)	(0.93)	(0.89)	(0.84)	(0.87)	(0.83)
	random stream 2	(0.52)	n/a	(0.52)	(0.52)	(0.83)	(0.83)	(0.57)	(0.82)	(0.84)	(0.90)	(0.85)	(0.82)
	random stream 3	(0.52)	(0.52)	n/a	(0.52)	(0.94)	(0.91)	(0.58)	(0.83)	(0.83)	(0.89)	(0.83)	(0.85)
noise (true)	white noise (true)	(0.52)	(0.52)	(0.52)	n/a	(0.83)	(0.81)	(0.59)	(0.87)	(0.89)	(0.86)	(0.93)	(0.81)
	pink noise (true)	(0.80)	(0.83)	(0.94)	(0.83)	n/a	(0.76)	(0.86)	(0.52)	(0.76)	(0.65)	(0.65)	(0.66)
	brown noise (true)	(0.84)	(0.83)	(0.91)	(0.81)	(0.76)	n/a	(0.86)	(0.76)	(0.56)	(0.71)	(0.69)	(0.68)
noise (mp3)	white noise (via mp3)	(0.59)	(0.57)	(0.58)	(0.59)	(0.86)	(0.86)	n/a	(0.91)	(0.83)	(0.84)	(0.80)	(0.78)
	pink noise (via mp3)	(0.93)	(0.82)	(0.83)	(0.87)	(0.52)	(0.76)	(0.91)	n/a	(0.78)	(0.63)	(0.68)	(0.70)
	brown noise (via mp3)	(0.89)	(0.84)	(0.83)	(0.89)	(0.76)	(0.56)	(0.83)	(0.78)	n/a	(0.71)	(0.69)	(0.67)
metal music	metal music (sample 1)	(0.84)	(0.90)	(0.89)	(0.86)	(0.65)	(0.71)	(0.84)	(0.63)	(0.71)	n/a	(0.54)	(0.56)
	metal music (sample 2)	(0.87)	(0.85)	(0.83)	(0.93)	(0.65)	(0.69)	(0.80)	(0.68)	(0.69)	(0.54)	n/a	(0.53)
	metal music (sample 3)	(0.83)	(0.82)	(0.85)	(0.81)	(0.66)	(0.68)	(0.78)	(0.70)	(0.67)	(0.56)	(0.53)	n/a

Table 5.3: Distinguishing random streams and uncompressed audio (noise, oscillating noise, metal music).

6 Distinguishing cipher outputs from random stream

- introduction, idea, running EACirc along with statistical batteries
- stream ciphers from eStream competition

6.1 Stream ciphers used

- ciphers except for ?? (why??)
- from last phase
- those that could be limited in rounds are tested in weaker variant as well
- differences from Metej Pristak thesis

6.2 Generating binary stream from stream ciphers

- cipher modes (iv+key initialization frequency)
- case of LEX (not weakening the cipher, only making shorter output)
- case of TSC (producing binary stream of 0 for 1-8 rounds) => problems in 3 Dieharder tests

6.3 Results interpretation

- ???
- more or less as statistical batteries
- dieharder better in some case than STS-NIST (is newer and some tests are redesigned)
- statistical tests has much more input data compared to EACirc
- using evolved distinguisher is quick

# of rounds	IV and key reinitialization								
	once for run			for each test set			for each test vector		
	Dieharder (x/20)	STS NIST (x/162)	EACirc (AAM)	Dieharder (x/20)	STS NIST (x/162)	EACirc (AAM)	Dieharder (x/20)	STS NIST (x/162)	EACirc (AAM)
1	0.0	0	$n = 2681$	0.0	0	(0.85)	0.0	5	$n = 1431$
2	0.5	0	(0.54)	1.0	0	(0.54)	15.5	146	(0.52)
3	1.0	0	(0.53)	1.0	0	(0.53)	15.0	160	(0.52)
4	3.5	79	(0.52)	3.0	78	(0.52)	20.0	160	(0.52)
5	4.5	79	(0.52)	3.5	91	(0.52)	17.5	161	(0.52)
6	19.0	158	(0.52)	19.0	159	(0.52)	18.0	162	(0.52)
7	18.5	162	(0.52)	19.0	161	(0.52)	20.0	161	(0.52)
8	20.0	162	(0.52)	20.0	159	(0.52)	19.0	161	(0.52)

Table 6.1: Random distinguishers for Decim ciphertext.

# of rounds	IV and key reinitialization								
	once for run			for each test set			for each test vector		
	Dieharder (x/20)	Sts Nist (x/162)	EACirc (AAM)	Dieharder (x/20)	Sts Nist (x/162)	EACirc (AAM)	Dieharder (x/20)	Sts Nist (x/162)	EACirc (AAM)
1	20.0	162	(0.52)	20.0	161	(0.52)	18.0	162	(0.52)
4	20.0	162	(0.52)	20.0	162	(0.52)	20.0	162	(0.52)

Table 6.2: Random distinguishers for FUBUKI ciphertext.

# of rounds	IV and key reinitialization								
	once for run			for each test set			for each test vector		
	Dieharder (x/20)	Sts Nist (x/162)	EACirc (AAM)	Dieharder (x/20)	Sts Nist (x/162)	EACirc (AAM)	Dieharder (x/20)	Sts Nist (x/162)	EACirc (AAM)
1	0.0	0	$n = 221$	0.0	0	(0.67)	18.5	162	(0.52)
2	0.0	0	$n = 471$	0.5	0	(0.66)	20.0	162	(0.52)
3	19.5	160	(0.52)	20.0	162	(0.52)	20.0	162	(0.52)
13	20.0	162	(0.52)	20.0	161	(0.52)	19.5	162	(0.52)

Table 6.3: Random distinguishers for Grain ciphertext.

# of rounds	IV and key reinitialization								
	once for run			for each test set			for each test vector		
	Dieharder (x/20)	Sts Nist (x/162)	EACirc (AAM)	Dieharder (x/20)	Sts Nist (x/162)	EACirc (AAM)	Dieharder (x/20)	Sts Nist (x/162)	EACirc (AAM)
1	20.0	162	(0.52)	20.0	162	(0.52)	20.0	162	(0.52)
10	20.0	160	(0.52)	20.0	162	(0.52)	20.0	162	(0.52)

Table 6.4: Random distinguishers for Hermes ciphertext.

# of rounds	IV and key reinitialization								
	once for run			for each test set			for each test vector		
	Dieharder (x/20)	Sts Nist (x/162)	EACirc (aam)	Dieharder (x/20)	Sts Nist (x/162)	EACirc (aam)	Dieharder (x/20)	Sts Nist (x/162)	EACirc (aam)
1	0.0	0	$n = 148$	0.0	0	$n = 7274$	3.0	1	$n = 154$
2	4.0	1	$n = 221$	4.0	1	$n = 304$	3.5	1	$n = 254$
3	0.5	1	$n = 378$	3.5	1	$n = 491$	4.0	1	$n = 361$
4	20.0	162	(0.52)	19.5	162	(0.52)	20.0	161	(0.52)
10	19.5	162	(0.52)	19.5	160	(0.52)	20.0	160	(0.52)

Table 6.5: Random distinguishers for LEX ciphertext.

# of rounds	IV and key reinitialization								
	once for run			for each test set			for each test vector		
	Dieharder (x/20)	Sts Nist (x/162)	EACirc (aam)	Dieharder (x/20)	Sts Nist (x/162)	EACirc (aam)	Dieharder (x/20)	Sts Nist (x/162)	EACirc (aam)
1	5.5	1	(0.87)	8.5	1	(0.67)	17.5	161	(0.52)
2	5.5	1	(0.87)	7.0	1	(0.67)	19.5	162	(0.52)
3	20.0	162	(0.52)	20.0	162	(0.52)	19.5	161	(0.52)
12	20.0	162	(0.52)	19.5	161	(0.52)	19.0	161	(0.52)

Table 6.6: Random distinguishers for Salsa20 ciphertext.

# of rounds	IV and key reinitialization								
	once for run			for each test set			for each test vector		
	Dieharder (x/20)	Sts Nist (x/162)	EACirc (aam)	Dieharder (x/20)	Sts Nist (x/162)	EACirc (aam)	Dieharder (x/20)	Sts Nist (x/162)	EACirc (aam)
1–8	0.0*	0	$n = 104$	0.0*	0	$n = 101$	0.0*	0	$n = 104$
9	1.0	1	$n = 234$	1.5	1	$n = 491$	2.0	1	$n = 121$
10	2.0	13	$n = 188$	3.0	13	$n = 218$	3.0	12	$n = 158$
11	10.0	157	(0.52)	11.5	157	(0.52)	14.0	159	(0.52)
12	16.0	162	(0.52)	17.0	161	(0.52)	17.5	162	(0.52)
13	20.0	162	(0.52)	20.0	162	(0.52)	19.0	162	(0.52)
32	20.0	161	(0.52)	20.0	162	(0.52)	20.0	161	(0.52)

Table 6.7: Random distinguishers for TSC-4 ciphertext.

7 Analysis of Salsa20 output stream

- learns current vectors quicker than other ciphers
- the case of six

8 Distinguishing hash outputs from random stream

- introduction, idea
- hash function candidates from SHA-3

8.1 Hash functions used

- except for 2 (?? source code size, compilation)
- from last phase
- those that could be limited in rounds are tested in weaker variant as well
- differences from Ondrej Dubovec Bc thesis

8.2 Generating binary stream from hash functions

- length set to 256b
- hashing 4 byte counters starting from random value (in fact, cutting each hash in half)

8.3 Determining optimal set change frequency

- previously,we used change every 100 generations
- 100 was taken from Matej Pristak's thesis
- Ondrej proposes 10 as best, however, data is not provided
- interpretation of results (Table 8.1):
  - ???

	change frequency for test vector set							
	5	10	20	50	100	200	500	1000
30 000 g.	(0.614)	(0.614)	(0.607)	(0.602)	(0.599)	(0.598)	(0.591)	(0.582)
run-time	70 m.	52 m.	42 m.	37 m.	32 m.	28 m.	23 m.	20 m.
300 sets	(0.567)	(0.583)	(0.585)	(0.589)	(0.599)	(0.608)	(0.617)	(0.618)
run-time	4 m.	6 m.	9 m.	19 m.	32 m.	57 m.	115 m. 220 m.	

Table 8.1: Determining optimal change frequency for test vector set.

8.4 Results interpretation

- ???

	number of rounds							
	0	1	2	3	4	5	6	full
ARIRANG	$n =$ 694	$n =$ 707	$n =$ 467	$n =$ 1071	(full)	–	–	(0.52)
Aurora	$n =$ 5614	(0.75)	(0.78) !!!	(0.52)	–	–	–	(0.52)
Blake	$n =$ 474	(0.52)	–	–	–	–	–	(0.52)
Blue Midnight Wish	(0.52)	–	–	–	–	–	–	(0.52)
Cheetah	$n =$ 181	$n =$ 574	$n =$ 708	(0.90) !!!				(0.52)
CHI	(0.52)	–	–	–	–	–	–	(0.52)
CRUNCH	$n =$ 104	$n =$ 534	$n =$ 954		34-(0.52)			(0.52)
CubeHash	$n =$ 104	(0.52)	–	–	–	–	–	(0.52)
DCH	$n =$ 104	(0.73) !!!	(0.52)	–	–	–	–	(0.52)
Dynamic SHA	$n =$ 484	$n =$ 2337	$n =$ 1773 !!!	(0.95) !!!				(0.52)
Dynamic SHA2	–	(0.94) !!!	(0.74)	(0.75)	(0.57)			(0.52)

Table 8.2: Random distinguishers for SHA-3 candidate functions.

9 Conclusions and future work

9.1 Conclusions based on experimental data

- summary of what we did
- control distinguishers (random-random, hr-de, audio)
- estream (round limited ciphers)
- analysis of Salsa20
- sha3 (round limited hash functions)
- different approach than statistical batteries -> possibly new things
- dynamically adapting distinguisher - both advantage and disadvantage
- comparable to statistical tests, however smaller inputs
- speed: slow learning (more computational power needed), fast distinguishing
- problem with interpreting results

9.2 Proposed future work

- deep analyses instead of wide
- possibilities of longer input
  - READX
  - memory circuit
- tools for interpreting results
  - histogram of outputs in nodes
- fixing functions in layers