Extended Essay Subject:

Computer Science

Topic:

Effects of parameters of additive scrambling (a digital encoding technique) on its wall time

Research Question:

What is the impact of connection vector and initial state of additive scrambling have on its wall time and what is the best combination of initial bits?

Word Count: 3553

Candidate Code: hqw378

Table of contents

Introduction

Scrambling of data or signals masks the data and makes it seem random[1]. This is a very important tool used by the telecommunications industry, military and many others. Data scrambling is important because it helps in protection of data and prevents unauthorized access. It does this by changing the data by using an algorithm and a set of inputs which are secret. The scrambled data can then be only descrambled if the algorithm and its inputs are known. Another such method is cryptography. Unfortunately while cryptography is the gold standard for data security and privacy[2], it is computationally expensive[3]. Thus it can not be used for telecommunications where the data needs to be transmitted and received with minimum latency [4] often using very simple and low power electronics rather than a powerful and power hungry computer. While thanks to improvements in embedded systems we now have system-on-chips capable of real time encryption they are significantly more complex and expensive. Thus scramblers are still widely used. The algorithm and its input for scrambling is rather simple and can be replicated using an interchanged collection and logic gates. The data can then be only descrambled by the complementary descrambler with a specific configuration. There are various ways of scrambling data and I will be assessing additive scrambling as it is the most common type of scrambling used.

---

[1] "What is Data Scrambling? - Soaring Eagle Consulting." 16 Jun. 2019, https://soaringeagle.biz/what-is-data-scrambling/. Accessed 5 Feb. 2020.
[2] "Why Cryptography Is the New Gold for Banking and Blockchain." 30 Sep. 2016, https://securityintelligence.com/cryptography-new-gold-banking-blockchain/. Accessed 5 Feb. 2020.
[3] "Why Cryptography Is the New Gold for Banking and Blockchain." 30 Sep. 2016, https://securityintelligence.com/cryptography-new-gold-banking-blockchain/. Accessed 5 Feb. 2020.
[4] "In real time over the network: low latency is the key | Deutsche ...." https://www.telekom.com/en/company/details/in-real-time-over-the-network-low-latency-is-the-key-436142. Accessed 5 Feb. 2020.

For my experiment I want to explore what parameters affect the performance of the additive scrambling algorithm. This topic interested me when we were learning about encoding in school and I had to dive deeper into this topic. While researching scrambling types, I reached additive scrambling and then to dive deeper, I did my research on this. I am using time as my assessment variable. Speed is of utmost importance in real time communication. Specifically, I will be measuring the actual time taken to execute the process (wall time)[5] of the scrambling algorithm and will not include the time used to configure and process the data that is eventually scrambled. I will be varying the input size, initial state- the initialization set of bits that is used for addition and connection vector- the bits that are used for addition.

The results of this experiment will aid telecommunication developers determine what choices or alterations should be made to additive scrambling in order to increase the wall time. This will also help them find the right balance between the time and the processing power that should be used for their setup. The developers would want to integrate a system that is efficient in terms of saving resources and decreasing the scrambling time and make an overall better scrambler by understanding the correct combination of initial state and connection vector.

[5] "What is wall time (real-world time or wall-clock time ...." https://whatis.techtarget.com/definition/wall-time-real-world-time-or-wall-clock-time. Accessed 12 Jan. 2020.

The algorithm

Data scrambling is the process of jumbling the inputted data to an extent where the data is not only different, but completely illegible. Data scrambling is done to protect or hide data from unwanted viewers. This is done by passing the data in the scrambling algorithm, which applies a mathematical/logical function on it bit by bit. It involves a set of registers which store the state.

The scrambling process starts with the transformation of the input data to a data stream(stream of bits). In my algorithm, the data is a collection of bits either all empty, alternately 0 and 1 or random. I want to observe if the type of input has any affect on the performance of the algorithm and also on the randomness of the final output.

Before the algorithm runs the state must be initialized. Thus the initial state is an important input to the algorithm. The size of the state would affect the individual steps of the additive scrambling algorithm. This I will compare and assess several sizes and values of the input state to understand how it affects the performance of the additive scrambling algorithm.

To get the output bit the algorithm compares the state with the connection vector, sums up the number of active connections by Modulo-2 addition and compares this with the input bit using exclusive or (XOR). This is why it is known as additive scrambling. I have implemented the modulo-2 addition as decimal addition followed by taking its modulo-2 by the remainder operator i.e. (will get the remainder of sum when divided by 2).

The state shifts as the data stream progresses. This is called bit shift. The final bit of the state is lost and the entire array is shifted by one index. The Modulo-2 sum is then put at the start. With this one iteration of the scrambling is done. This process is repeated for all input bits. In practice

this process is carried out using a collection of logic gates i.e. a logic circuit called the Linear Feedback Shift Register (LFSR). The data to be scrambled is generated as data frames as both audio and video are encoded on the basis of time. The scrambled frames have a sync word between them. This helps in proper descrambling and avoids the mixing of frames. This is not a concern for us as I will only use one frame for our analysis.

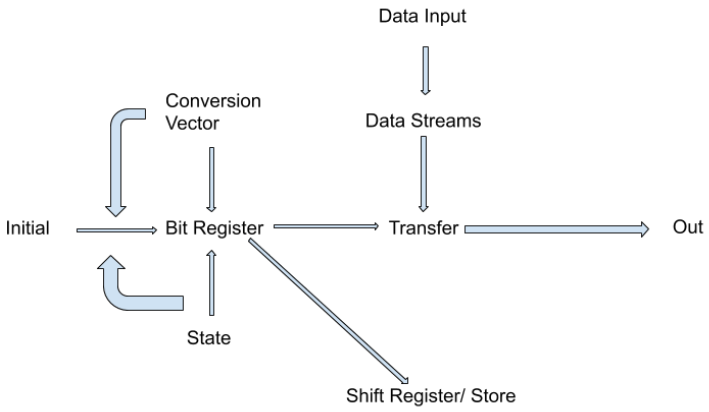


Figure 1. Additive scrambling functioning

Parameters and how it affects the wall time:

Size of the data input

The size of the inputted data will be increased. An increase in the size of the data should increase the wall time because there is more data to be changed. This will hence take more time. The expected dependence is linear. The chosen input sizes(number of bits) are based on logarithmic sampling i.e 10, 10^2 ,10^3, 10^4, 10^5 and 10^6

Number of layers

The number of layers/ registers that can be connected corresponds to the size of the state. Again, the expected dependence is linear as the size of the state will affect the time required to carry out the modulo 2 addition.

Connection vector

The connection vector specifies the specific bits of the state that are to be considered for addition. It is expected that a more complex set of connections will lead to more time required to calculate the sum. The expected dependence is linear with the number of connections.

Initial state

The state vector is essential to calculate the sum. The calculation of this sum should ideally only depend on the number of layers and the connection vector. Thus the value stored in the state should not affect the time required for the calculation. Therefore no dependence of performance on initial state is expected. However initial state may affect the quality of scrambling as it will be the only source of variation in the case of an empty input. Thus the outputs of the algorithm will be compared.

Hypothesis:

My hypothesis is that the time complexity of the algorithm is linear with size as the individual steps of scrambling are not dependent on the size of input and thus are constant with respect to size of input. It is expected that changes in connection vector, initial state and number of layers will have a more significant impact as they affect the individual steps of the scrambling algorithm.

Methodology

Independent variables

Variable Name	Values	Purpose
Size	10, 10^2, 10^3, 10^4, 10^5 and 10^6	Algorithmic efficiency is evaluated on the basis of time required as a function of the input size.
Initial State	Variation in index Variation in sum Variation in pattern(i.e. Alternating vs consecutive)	The scrambling process theoretically should be very sensitive to the initial state.
Connection Vector	Variation in index	The sum of the connection vector should affect the time required to calculate the modulo 2 sum.
Number of layers	2,3,4,5,10,20,50	Number of layers theoretically should affect the

		time required to do each iteration in the process

Dependent variables

Variable	Measurement	Purpose
Wall time	Using time function to get time before and after the code that scrambles the bits and reporting the difference in a csv file.	To estimate the time efficiency of the algorithm.
Output bits	Qualitative measurement of randomness	To find preferred or unfavourable configurations

Controlled variables

Variable	Value	Purpose
Hardware	2018 Mac Mini with 3 Ghz 8th generation intel core i5 processor & 8GB DDR4 RAM & soldered nvme ssd	The hardware affects the absolute time taken for a set of calculations.
OS	Mac OS X- Catalina	The OS provisions the usage of hardware and using a different OS and other software may influence the usage of hardware unpredictably
Programming language	Python 3.7 - base install no 3rd party packages	The differences in implementation of the internals may have an unpredictable effect on computational efficiency.
IDE	VScode	VScode is a very light code editor with great python support

Algorithm	LFSR type additive scrambling custom algorithm (see appendix)	This is the subject of analysis
Data output	CSV file using inbuilt CSV package on python	The code is written in a way to not consider the time required to output the data as a file. Thus this variable has no effect of the experiment but was still controlled for consistency.
Data type	Binary data stored as a python bytes string object	Algorithms are extremely sensitive to data type and data structure thus its was kept constant
Ambient Temperature	Maintained at 25 degrees Celsius by a 1.5 ton air conditioner in a small room. The computer was kept at the same position and orientation near the middle of the room.	The working of semiconductors is affected by heat. Thus the temperature of the room was maintained at room temperature.

Procedure

Before starting the experiment, the configuration files along with the code which will be used for parsing the configuration files and saving the output in a CSV file will be prepared. To start of this experiment, the configuration file in the Integrated Development Environment (IDE) will be parsed. The configuration files will include five connection vectors and five initial states. Collectively, the configurations will have all 25 combinations of these input bits. Each of these configurations will take run one by one for five trials. The CSV files will be exported in Microsoft Excel for further analysis like making graphs, tables and finding averages of the trial. The average of these trials will help reduce random errors.

Results & Observations

Effect of Connection Vector and Initial State

To understand the functioning of the algorithm and determine the configurations used for further analysis, a few tests were tried. In these tests there was only one 1 in the connection vector and its index was varied. For the first set the index of the 1 in the initial state was increased. The second set 1s in the initial state were cumulative. The final set used alternating values for the initial vector. These experiments will also help understand the dependence of quality of scrambling on the initial state. The number of layers and size of input was limited.

Input State Index

For this set of configurations the input state vector sum was 1 while the position of that 1 varied i.e. input state index 2 means input state was [0,0,1,0,0]. In a similar way the connection vector was varied. As can be seen in the table the interval between connections(represented as 1 in the output as 1) increases as the connection vector index increases. On the other hand, when the initial state index is increased the initial connection vector indices (lower than the initial state index) become redundant and do not scramble the data.

	Connection index 0	Connection index 1	Connection index 2	Connection index 3	Connection index 4
Input State Index 0	1111111111111111111111111111111111111111111111111111111111111111	0101010101010101010101010101010101010101010101010101010101010101	0010010010010010010010010010010010010010010010010010010010010010	0001000100010001000100010001000100010001000100010001000100010001	0000100001000010000100001000010000100001000010000100001000010000
Input State Index 1	0000000000000000000000000000000000000000000000000000000000000000	1010101010101010101010101010101010101010101010101010101010101010	0100100100100100100100100100100100100100100100100100100100100100	0010001000100010001000100010001000100010001000100010001000100010	0001000010000100001000010000100001000010000100001000010000100001
Input State Index 2	0000000000000000000000000000000000000000000000000000000000000000	0000000000000000000000000000000000000000000000000000000000000000	1001001001001001001001001001001001001001001001001001001001001001	0100010001000100010001000100010001000100010001000100010001000100	0010000100001000010000100001000010000100001000010000100001000010
Input State Index 0	0000000000000000000000000000000000000000000000000000000000000000	0000000000000000000000000000000000000000000000000000000000000000	0000000000000000000000000000000000000000000000000000000000000000	1000100010001000100010001000100010001000100010001000100010001000	0100001000010000100001000010000100001000010000100001000010000100

	0	0	0	0	0

Table 1a. Output bits

Input state sum

For this set of configurations the input state vector sum was incremented, progressively filling it with 1 i.e. input state sum 2 means input state was [1,1,0,0,0] while input state sum 3 would mean input state was [1,1,1,0,0]. In a similar way to the case with variation in initial state index the connection vector was varied. As can be seen in the table the interval between connections(represented as 1 in the output as 1) increases as the connection vector index increases post the last connection. Like observed in the case of varying initial state index, when the initial state sum is increased the initial connection vector indices (lower than the initial state sum) become redundant but unlike the case with variation in initial state index the output is flipped i.e. all the 0s have been flipped to 1s.

	Connection index 0	Connection index 1	Connection index 2	Connection index 3	Connection index 4
Input State Sum 1	11111111111 11111111111 11111111111 11111111111 11111111111 11111111111 11111111111 11111111111 1111	01010101010 10101010101 01010101010 10101010101 01010101010 10101010101 01010101010 10101010101 01010101010 1	00100100100 10010010010 01001001001 00100100100 10010010010 01001001001 00100100100 10010010010 01001001001 0	00010001000 10001000100 01000100010 00100010001 00010001000 10001000100 01000100010 00100010001 00010001000 1	00001000010 00010000100 00100001000 01000010000 10000100001 00001000010 00010000100 00100001000 01000010000 1
Input State Sum 2	11111111111 11111111111	11111111111 11111111111	01101101101 10110110110	00110011001 10011001100	00011000110 00110001100

	1111111111111111111111111111111111111111111111111111111111111111	1111111111111111111111111111111111111111111111111111111111111111	1101101101101101101101101101101101101101101101101101101101101101	1100110011001100110011001100110011001100110011001100110011001100	0110001100011000110001100011000110001100011000110001100011000110
Input State Sum 3	1111111111111111111111111111111111111111111111111111111111111111	1111111111111111111111111111111111111111111111111111111111111111	1111111111111111111111111111111111111111111111111111111111111111	0111011101110111011101110111011101110111011101110111011101110111	0011100111001110011100111001110011100111001110011100111001110011
Input State Sum 4	1111111111111111111111111111111111111111111111111111111111111111	1111111111111111111111111111111111111111111111111111111111111111	1111111111111111111111111111111111111111111111111111111111111111	1111111111111111111111111111111111111111111111111111111111111111	0111101111011110111101111011110111101111011110111101111011110111
Input State Sum 5	1111111111111111111111111111111111111111111111111111111111111111	1111111111111111111111111111111111111111111111111111111111111111	1111111111111111111111111111111111111111111111111111111111111111	1111111111111111111111111111111111111111111111111111111111111111	1111111111111111111111111111111111111111111111111111111111111111

Table 1b. Output bits

Input State Alternate

For this set of configurations the input state vector filled alternately at various indices for a sum of 2 and 3, i.e. [1,0,1,0,0], [0, 1, 0, 1, 0], [0, 0, 1, 0, 1] & [1, 0, 1, 0, 1]. In a similar way to the case with variation in initial state index the connection vector was varied. As can be seen in the table the interval between connections(represented as 1 in the output as 1) increases as the connection vector index increases post the last connection. Like observed in the case of varying initial state index, when the initial state sum is increased the initial connection vector indices (lower than the initial state sum) become redundant but unlike the previous cases the redundancy is much lower. Also the fully alternating one i.e. [1, 0, 1, 0, 1] shows alternating redundancy. Due to the presence of both comparative redundancy which is neither unscrambled nor flipped and alternating outputs with the change in connection vector. This alternating case was chosen as the base configuration for further analysis.

	Connection index 0	Connection index 1	Connection index 2	Connection index 3	Connection index 4
[1, 0, 1, 0, 0]	1111111111111111111111111111111111111111111111111111111111111111	0101010101010101010101010101010101010101010101010101010101010101	1011011011011011011011011011011011011011011011011011011011011011	0101010101010101010101010101010101010101010101010101010101010101	0010100101001010010100101001010010100101001010010100101001010010
[0, 1, 0, 1, 0]	0000000000000000000000000000000000000000	1010101010101010101010101010101010101010	0100100100100100100100100100100100100100	1010101010101010101010101010101010101010	0101001010010100101001010010100101001010

	00000000000 00000000000 00000000000 00000000000 00000000000 00000000000 00000000000 0	10101010101 01010101010 10101010101 01010101010 10101010101 01010101010 10101010101 0	10010010010 01001001001 00100100100 10010010010 01001001001 00100100100 10010010010 0	10101010101 01010101010 10101010101 01010101010 10101010101 01010101010 10101010101 0	01001010010 10010100101 00101001010 01010010100 10100101001 01001010010 10010100101 0
[0, 0, 1, 0, 1]	00000000000 00000000000 00000000000 00000000000 00000000000 00000000000 00000000000 00000000000 00000000000 0	00000000000 00000000000 00000000000 00000000000 00000000000 00000000000 00000000000 00000000000 00000000000 0	10010010010 01001001001 00100100100 10010010010 01001001001 00100100100 10010010010 01001001001 00100100100 1	01000100010 00100010001 00010001000 10001000100 01000100010 00100010001 00010001000 10001000100 01000100010 0	10100101001 01001010010 10010100101 00101001010 01010010100 10100101001 01001010010 10010100101 00101001010 0
[1, 0, 1, 0, 1]	11111111111 11111111111 11111111111 11111111111 11111111111 11111111111 11111111111 11111111111 1111	01010101010 10101010101 01010101010 10101010101 01010101010 10101010101 01010101010 10101010101 01010101010 1	10110110110 11011011011 01101101101 10110110110 11011011011 01101101101 10110110110 11011011011 01101101101 1	01010101010 10101010101 01010101010 10101010101 01010101010 10101010101 01010101010 10101010101 01010101010 1	10101101011 01011010110 10110101101 01101011010 11010110101 10101101011 01011010110 10110101101 01101011010 1

Table 1c. Output bits

Effect of size

Algorithms are compared using the time complexity. The time complexity is given as the type and order of the function that represents the time taken as a function of input size. To get good sampling, sizes were taken as the powers of 10, i.e. 10 bits, 100 bits, 1000 bits & so on till 1 million bits. As the individual step of the algorithm is a sum which is dependent on the number of layers and is thus constant, the expected outcome is that the time should vary linearly with input size because only the number of iterations increase but the function does not become more complex. The number of layers was 5, connection vector was [0,0,0,1,0] initial state was [1,0,1,0,1]. The input data was empty i.e. all bits were 0. The following table(table 1) & graph(figure 2) shows the data obtained from this empty scramble.

input size (bits)	Trial 1	Trial 2	Trail 3	Trail 4	Trial 5	average time	expected time (linear)	error(stdev)
10	2.50E-05	2.10E-05	2.10E-05	2.00E-05	2.00E-05	2.14E-05	2.14E-05	1.8615E-06
100	0.0001781	0.00017691	0.00017715	0.00017691	0.0001862	1.79E-04	2.14E-04	3.6032E-06
1000	0.00190091	0.0018611	0.00185966	0.00199294	0.00208807	1.94E-03	2.14E-03	8.8242E-05
10000	0.02502465	0.02262402	0.02386594	0.02311206	0.02149701	2.32E-02	2.14E-02	0.00118401
100000	0.37523365	0.37492299	0.37172318	0.36891079	0.37852573	3.74E-01	2.14E-01	0.00328166

1000000	21.006866	20.9511609	20.9991491	20.979805	20.6297538	2.09E+01	2.14E+00	0.14309398

Table 2a. Wall time with ranging input bit size

Table 1 Size vs Time for Empty scramble with init state as [1,0,1,0,1] and connection vector as [0,0,0,1,0] and thus no. of Layers was 5.

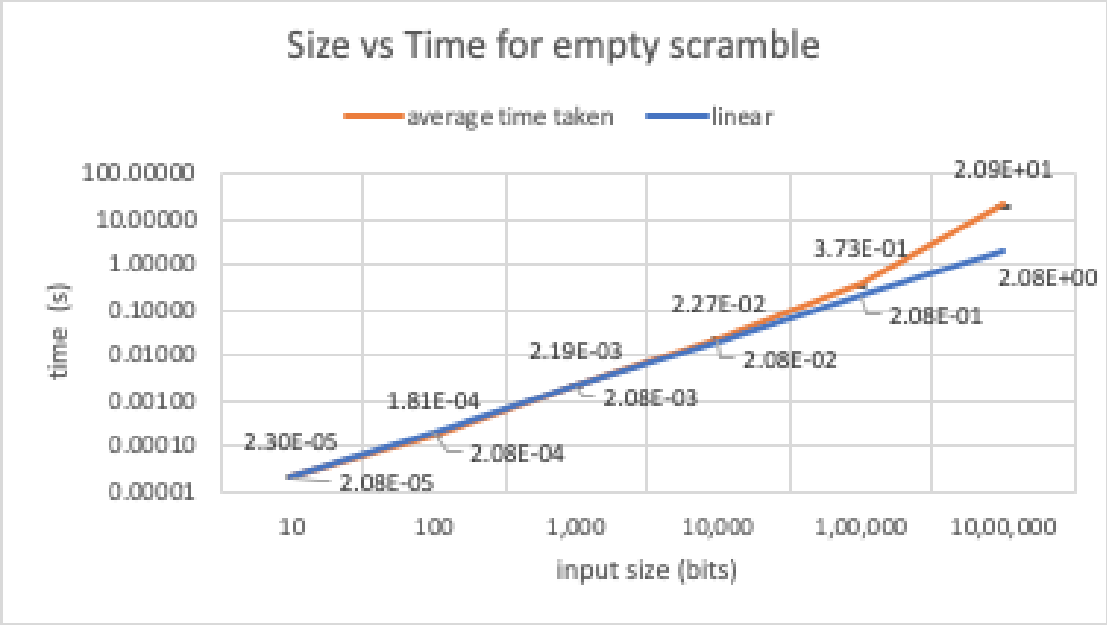


Figure 2

Figure 2 Size vs Time for Empty scramble with init state as [1,0,1,0,1] and connection vector as [0,0,0,1,0] and thus no. of Layers was 5.

As expected the dependence of time with size was linear but the data indicates that towards the higher sizes the linear dependence is broken. This unexpected based on the theory thus it could be an effect of low level implementation details of the language and operating system used. To verify this, it must be tested on several OSs and programming languages.

Effect of type of input

It is expected that all the characteristics of input data do not affect the time required, the only characteristic that should have any effect should be size. To test this the same configuration was run for empty data(00000...) alternating data(01010101...) and random data. The number of layers was 5, connection vector was [0,0,0,1,0] initial state was [1,0,1,0,1]. The time data is expected to be the same for all three cases. The simplified tables & graphs are given below

Input Size(bits)	Empty Scramble		Alternate Scramble		Random Scramble	
	average time	expected	average time	expected	average time	expected
10	2.14E-05	2.14E-05	2.28E-05	2.28E-05	2.31E-05	2.31E-05
100	1.79E-04	2.14E-04	1.93E-04	2.28E-04	2.07E-04	2.31E-04
1000	1.94E-03	2.14E-03	0.001887846	2.28E-03	0.002390003	2.31E-03
10000	2.32E-02	2.14E-02	0.019360018	2.28E-02	0.021279478	2.31E-02
100000	3.74E-01	2.14E-01	0.363060284	2.28E-01	0.370293427	2.31E-01
1000000	2.09E+01	2.14E+00	19.76738939	2.28E+00	19.85815492	2.31E+00

Table 2b. Wall time with ranging input bit size

Table 2 Size vs Time for Empty, Alternate and Random scramble with init state as [1,0,1,0,1] and connection vector as [0,0,0,1,0] and thus no. of Layers was 5.

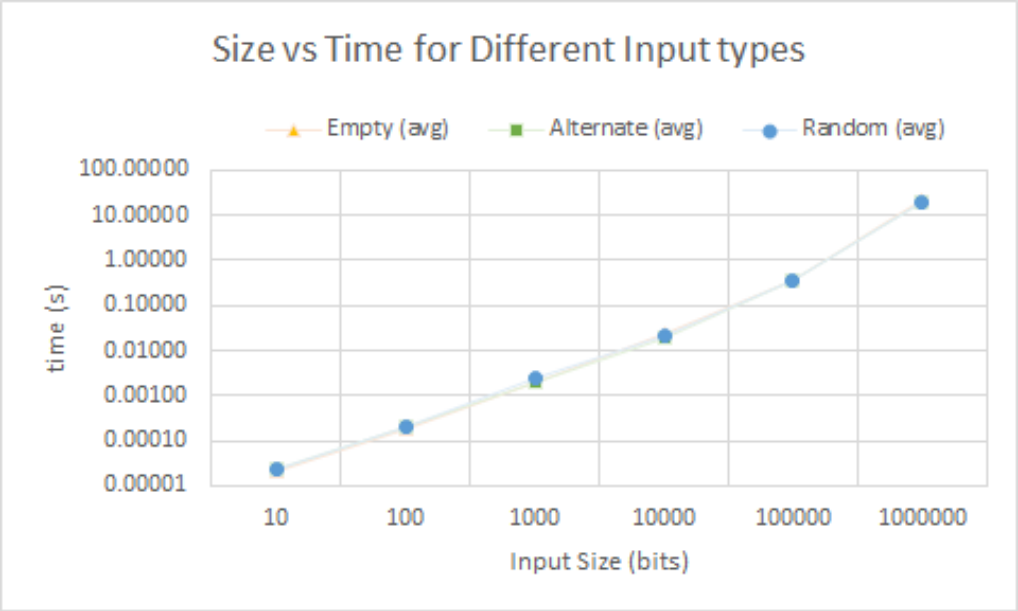


Figure 3

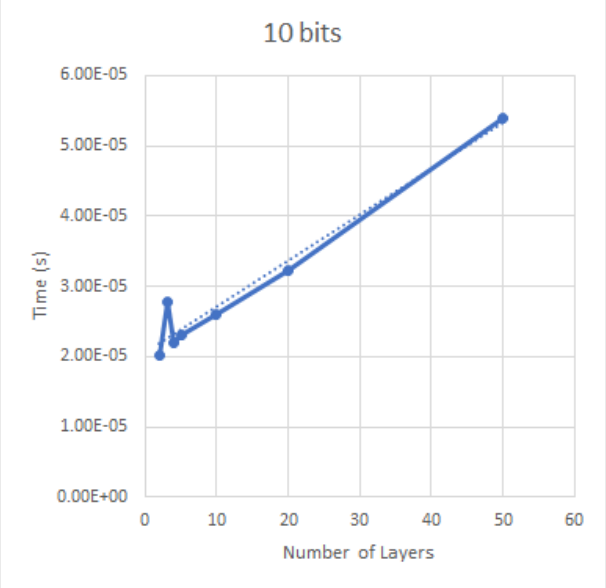
Figure 3 Size vs Time for Empty, Alternate and Random scramble with init state as [1,0,1,0,1] and connection vector as [0,0,0,1,0] and thus no. of Layers was 5.

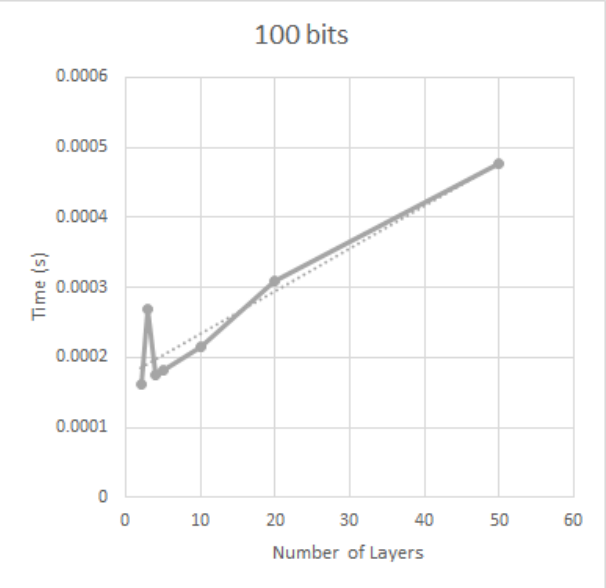
The observation clearly agrees with our expectations based on theory i.e. the input type should have no effect on the time taken by the algorithm. There are some deviations which peak at 1000 this may be relevant on a linear scale but on a logarithmic scale it is hardly observable. These deviations may be due to fluctuations in temperature or background tasks running on the OS.

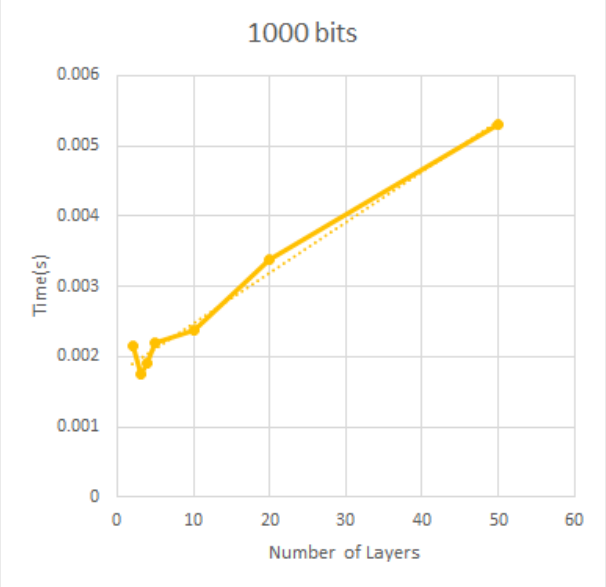
Effect of number of layers

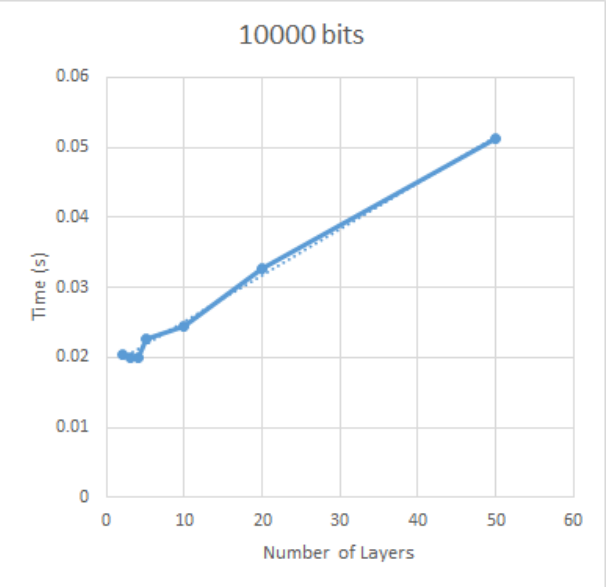
The number of layers decides the time taken by the addition step thus a linear dependence of the time with number of layers is expected. To assess this the size , input, initial state and connection vector were kept constant.

What is the impact of connection vector and initial state of additive scrambling have on its wall time and what is the best combination of initial bits?









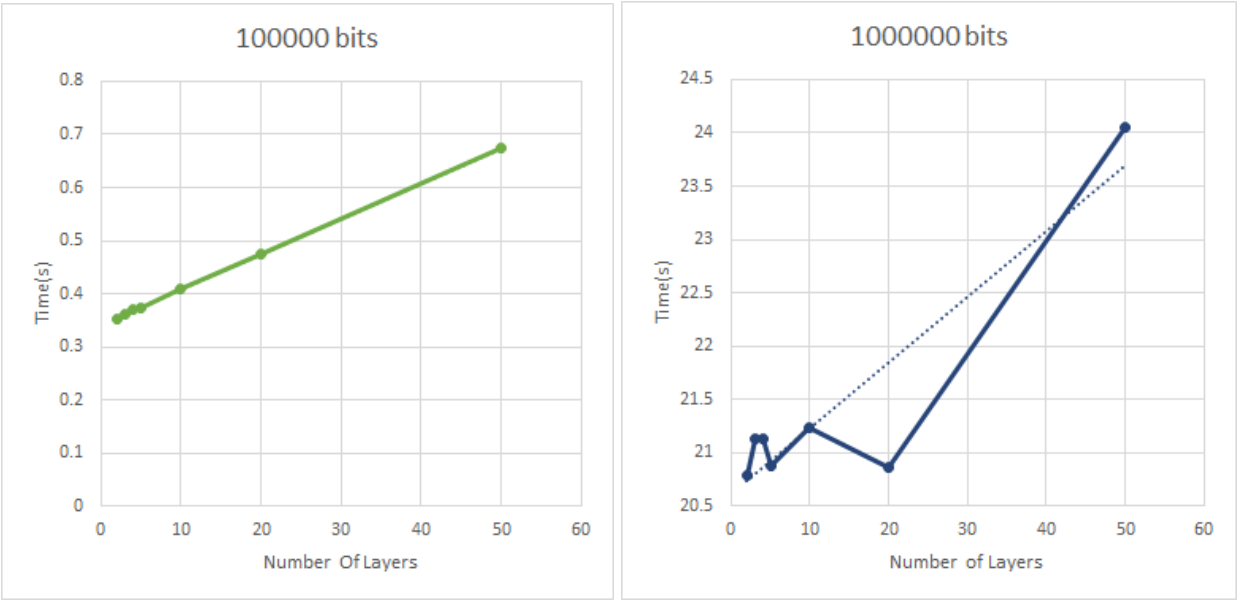


Figure 4. Time vs Number of Layers for different sizes ranging from 10 bit to 1 million bits.

As expected the time required increases with the number of layers. This is clearly seen in the case of 100,000 bits where the trent line has merged with the data. For others there is slight deviations especially in the lower range on the number of layers. This effect is magnified in the case of 1 million bits.

Conclusion

Based on the theoretical considerations the dependence of time on both the size and number of layers is linear. The assumption seems to become unstable at 1 million bits of input data. The dependence of time on size seems to become polynomial at this size. Unfortunately due to hardware limitations, I could not try higher input sizes and verify this further. This deviation and other deviations may be caused by outliers and will need further trials. Unlike other methods the

linear time complexity of the algorithm with respect to the input and the lack of effect of input type, makes it favourable to use in real time transmission because of the sense of urgency is established. If the scrambling would take more time, it would no longer be in real time because of the delay in sending the signal. A delay in the sending of the signal makes communication harder because of the wait. In other uses, like live streaming, the event is no longer what it was meant to be because of the lag.

The time dependence on the initial state is constant. Thus, any initial state and connection vector can be used without affecting the performance. All combinations of input state and connection vector are however not compatible because the interval is large and the 1s do not align to scramble. This is shown and explained under the heading *'Input State Sum'*. Thus, alternating initial state seems to be better overall as there is less redundancy because of the reason mentioned. In conclusion, to decrease the wall time and better the scrambling, they must use an appropriate balance of an alternating-resembling initial state with an adequate number of layers by finding the right complexity of the connection vector.

Bibliography:

"Data Scrambling". *Docs.Oracle.Com*, 2020, https://docs.oracle.com/cd/E11857_01/em.111/e18709/T508706T512067.htm. Accessed 25 Nov 2019.

"Digital Encoding Technique Of Scrambling Computer Science Essay." UKEssays.com. 11 2018. All Answers Ltd. 11 2019 <https://www.ukessays.com/essays/computer-science/digital-encoding-technique-of-scrambling-computer-science-essay.php?vref=1> Accessed 25 Nov 2019.

Preissler Jr, Sigmundo. "Seasonality In Python: Additive Or Multiplicative Model?". *Medium*, 2018, https://medium.com/@sigmundojr/seasonality-in-python-additive-or-multiplicative-model-d4b9cf1f48a7. Accessed 26 Nov 2019.

Mioc, Mirella, and Mircea Stratulat. *Naun.Org*, 2014, http://www.naun.org/main/NAUN/computers/2014/a042007-097.pdf. Accessed 26 Nov 2019.

Valmori, Flipp. "SCRAMBLING". *Filippovalmori.Wixsite.Com*, 2018, https://filippovalmori.wixsite.com/eletlcdsp/scrambling.Accessed 26 Nov 2019.

distribution, non-uniform. "US20030099359A1 - Method And Apparatus For Data Scrambling/Descrambling - Google Patents". *Patents.Google.Com*, 2020, https://patents.google.com/patent/US20030099359A1/en. Accessed 28 Nov 2019.

Cake Labs, Inc. "Bit Shifting (Left Shift, Right Shift) | Interview Cake". *Interview Cake: Programming Interview Questions And Tips*, 2020, https://www.interviewcake.com/concept/java/bit-shift. Accessed 28 Nov 2019.

"Generate A Random Binary Number - Online Random Tools". *Onlinerandomtools.Com*, 2020, https://onlinerandomtools.com/generate-random-binary-numbers. Accessed 15 Dec 2019.

"What is wall time (real-world time or wall-clock time ...." https://whatis.techtarget.com/definition/wall-time-real-world-time-or-wall-clock-time. Accessed 12 Jan. 2020.

Appendix

Additive Scrambling algorithm:

```
import csv
import time
import sys
import random

def AddScramb(
  InBits, # input as bits
  ConVect, # connection vector that is used to calculate the registry bit from the state
  InitState ) :
  RegState = InitState[:] # it sets up the state vector
  OutBits = '' # it sets up the output string
  for j in range(len(InBits)) :
     RegBit = (sum([     # 3. gets the number of indices where both state and connection vector have 1
        v1*v2 # 2. does logical and
        for v1,v2 in
        zip(RegState,ConVect)]) # 1. zips the data at the same indices in the state & con vect
        %2) # if sum is odd then reg bit is 1 else 0
       OutBits=''.join([OutBits,bin(int(str(InBits[j]),base=2)^RegBit)[2:]]) # XOR of the inbit and regbit is appended to the
outbits
     RegState[1:] = RegState[:-1] # move the state ahead removing the last bit
     RegState[0] = RegBit    # load the regbit at the start of the state
  return OutBits # returns scrambled output
 b
def emptyScramble(n,init_state,con_vect): # this function does additive scrambling for an empty input, i.e. all the bits in
the inputs are 0s
  print('size: ',n)
  InfoLen = n
  InfoBytes = b'\x00'*InfoLen
  start = time.time()
  result = AddScramb( InfoBytes, con_vect, init_state )
  end = time.time()
  return [result,end-start]

def run_scrambler(config,csvwriter,scrambler): # a function that takes a config and runs the scrambler using it, and put
sit in a csv file.
  size = config['log_size']
  csvwriter.writerow([
```

```
        'output','time taken'
        ])
  for run in range(config['runs']):
    print('run :',run)
    csvwriter.writerow(
      scrambler(
        10**size,
        config['init_state'],
        config['con_vect'])
      )

def run_config(config): # initialises and manages the csv file and calls run scrambler
  with open(config['filename'], 'w') as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow([config])
    run_scrambler(config,csvwriter,config['scrambler'])
```

Example of Configuration:

```
from additive_scrambling import emptyScramble,run_config
from datetime import datetime
if __name__=="__main__":
  n_layers=5
  for i in range(0,n_layers):
    InitStA = [1,0,1,0,0]  # initial state
    ConVectA = [0,0,0,0,0]  # connection vector
    ConVectA[i] = 1
    timeStamp= datetime.now()
    filename='initaltsum2-index0'+'-'+'convectindex-'+str(i)+'-'+str(timeStamp)+'.csv' # tabulation format
    config={
      'filename' : filename,
      'runs' : 1,
      'log_max_size' : 7,
      'log_size' : 2,
      'init_state': InitStA,
      'con_vect': ConVectA,
      'scrambler': emptyScramble
```

```
}
run_config(config)
```

Data results:

{'filename': 'initaltsum3-convectindex-1-nlayers-2-2020-01-24 08:32:22.911971.csv', 'runs': 5, 'log_max_size': 6, 'log_size': 0, 'init_state': [1, 0], 'con_vect': [0, 1], 'scrambler': <function emptyScramble at 0x10959e170>}									
size	time taken								
10	3.08E-05								
10	1.79E-05		size	10	100	1000	10000	100000	1000000
10	1.72E-05		trial 1	3.08E-05	0.000154	0.001737	0.022016	0.353373	20.74568
10	1.79E-05		trial 2	1.79E-05	0.000198	0.001774	0.021865	0.362233	20.45438
10	1.72E-05		trial 3	1.72E-05	0.000153	0.002223	0.019059	0.353916	20.70397
100	0.000154018		trial 4	1.79E-05	0.000153	0.002917	0.02009	0.3508	20.68306
100	0.000198126		trial 5	1.72E-05	0.000153	0.002145	0.018866	0.348763	21.36075
100	0.000152826		average Time	2.02E-05	1.62E-04	2.16E-03	2.04E-02	3.54E-01	2.08E+01
100	0.000153065								
100	0.000153065								
1000	0.001737118								

1000	0.001774311								
1000	0.002223015								
1000	0.002917051								
1000	0.002145052								
10000	0.022016048								
10000	0.021865129								
10000	0.019058943								
10000	0.020089865								
10000	0.018866062								
100000	0.353373051								
100000	0.362232924								
100000	0.353916168								
100000	0.350799799								
100000	0.348762989								
1000000	20.74568439								
100000	20.45437								

0	717								
1000000	20.70397496								
1000000	20.68306088								
1000000	21.3607471								

{'filename': 'initaltsum3-convectindex-1-nlayers-3-2020-01-24 08:33:15.941482.csv', 'runs': 5, 'log_max_size': 6, 'log_size': 0, 'init_state': [1, 0, 1], 'con_vect': [0, 1, 0], 'scrambler': <function emptyScramble at 0x100fa9170>}									
output	time taken								
10	6.82E-05								
10	1.88E-05		size	10	100	1000	10000	100000	1000000
10	1.69E-05		trial 1	6.82E-05	0.00028	0.001801	0.021862	0.353229	20.84566
10	1.79E-05		trial 2	1.88E-05	0.00033	0.00175	0.01999	0.358887	21.17384
10	1.69E-05		trial 3	1.69E-05	0.000264	0.001713	0.018956	0.359456	21.1391
100	0.00028		trial 4	1.79E-05	0.000252	0.001751	0.019201	0.368249	21.26629
100	0.00033		trial 5	1.69E-05	0.00022	0.001778	0.019556	0.361367	21.26373
100	0.000264		average Time	2.78E-05	2.69E-04	1.76E-03	1.99E-02	3.60E-01	2.11E+01

100	0.000252								
100	0.00022								
1000	0.001801								
1000	0.00175								
1000	0.001713								
1000	0.001751								
1000	0.001778								
10000	0.021862								
10000	0.01999								
10000	0.018956								
10000	0.019201								
10000	0.019556								
100000	0.353229								
100000	0.358887								
100000	0.359456								
100000	0.368249								
100000	0.36136								

	7								
1000000	20.84566								
1000000	21.17384								
1000000	21.1391								
1000000	21.26629								
1000000	21.26373								

{'filename': 'initaltsum3-convectindex-1-nlayers-10-2020-01-24 08:35:18.104669.csv', 'runs': 5, 'log_max_size': 6, 'log_size': 0, 'init_state': [1, 0, 1, 0, 1, 0, 1, 0, 1, 0], 'con_vect': [0, 1, 0, 0, 0, 0, 0, 0, 0, 0], 'scrambler': <function emptyScramble at 0x10172bd40>}									
output	**time taken**								
10	**3.50E-05**								
10	**2.41E-05**		**size**	**10**	**100**	**1000**	**10000**	**100000**	**1000000**
10	**2.41E-05**		**trial 1**	**3.50E-05**	**0.000215**	**0.002245**	**0.024421**	**0.407664**	**21.38446**
10	**2.41E-05**		**trial 2**	**2.41E-05**	**0.000215**	**0.002273**	**0.024481**	**0.414137**	**20.93048**
10	**2.29E-05**		**trial 3**	**2.41E-05**	**0.000215**	**0.002301**	**0.024912**	**0.407379**	**21.04719**
100	**0.000215**		**trial 4**	**2.41E-05**	**0.000213**	**0.002709**	**0.024293**	**0.406317**	**21.48405**

100	**0.000215**		**trial 5**	**2.29E-05**	**0.000214**	**0.002381**	**0.024183**	**0.407324**	**21.31226**
100	**0.000215**		**average Time**	**2.60E-05**	**2.14E-04**	**2.38E-03**	**2.45E-02**	**4.09E-01**	**2.12E+01**
100	**0.000213**								
100	**0.000214**								
1000	**0.002245**								
1000	**0.002273**								
1000	**0.002301**								
1000	**0.002709**								
1000	**0.002381**								
10000	**0.024421**								
10000	**0.024481**								
10000	**0.024912**								
10000	**0.024293**								
10000	**0.024183**								
100000	**0.407664**								
100000	**0.41413**								

	7								
100000	**0.407379**								
100000	**0.406317**								
100000	**0.407324**								
1000000	**21.38446**								
1000000	**20.93048**								
1000000	**21.04719**								
1000000	**21.48405**								
1000000	**21.31226**								

{'filename': 'initaltsum3-convectindex-1-nlayers-20-2020-01-24 13:44:00.438147.csv', 'runs': 5, 'log_max_size': 6, 'log_size': 0, 'init_state': [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0], 'con_vect': [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'scrambler': <function emptyScramble at 0x10eccd170>}									
output	**time taken**								
10	**4.29E-05**								
10	**2.98E-05**		**size**	**10**	**100**	**1000**	**10000**	**100000**	**1000000**
10	**3.00E-05**		**trial 1**	**4.29E-05**	**0.000275**	**0.002962**	**0.034314**	**0.47038**	**21.00625**

10	**3.00E-05**		**trial 2**	**2.98E-05**	**0.000273**	**0.004487**	**0.031885**	**0.470868**	**20.91278**
10	**2.88E-05**		**trial 3**	**3.00E-05**	**0.000274**	**0.003588**	**0.032695**	**0.475508**	**20.80493**
100	**0.000275**		**trial 4**	**3.00E-05**	**0.000354**	**0.002927**	**0.032136**	**0.47856**	**20.81158**
100	**0.000273**		**trial 5**	**2.88E-05**	**0.000371**	**0.002953**	**0.03256**	**0.47563**	**20.73885**
100	**0.000274**		**average Time**	**3.23E-05**	**3.09E-04**	**3.38E-03**	**3.27E-02**	**4.74E-01**	**2.09E+01**
100	**0.000354**								
100	**0.000371**								
1000	**0.002962**								
1000	**0.004487**								
1000	**0.003588**								
1000	**0.002927**								
1000	**0.002953**								
10000	**0.034314**								
10000	**0.031885**								
10000	**0.032695**								
10000	**0.03213**								

	6								
10000	**0.03256**								
100000	**0.47038**								
100000	**0.470868**								
100000	**0.475508**								
100000	**0.47856**								
100000	**0.47563**								
1000000	**21.00625**								
1000000	**20.91278**								
1000000	**20.80493**								
1000000	**20.81158**								
1000000	**20.73885**								

{'filename': 'initaltsum3-convectindex-1-nlayers-50-2020-01-24 13:55:46.239321.csv', 'runs': 5, 'log_max_size': 6, 'log_size': 0, 'init_state': [1, 0, 1, 0], 'con_vect': [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'scrambler': <function emptyScramble at 0x107c03170>}									
output	**time taken**								
10	**7.13E-05**								

10	**4.98E-05**		**size**	**10**	**100**	**1000**	**10000**	**100000**	**1000000**
10	**4.89E-05**		**trial 1**	**7.13E-05**	**0.000471**	**0.005718**	**0.051579**	**0.668512**	**24.32458**
10	**5.08E-05**		**trial 2**	**4.98E-05**	**0.000473**	**0.004787**	**0.051119**	**0.684119**	**23.58587**
10	**4.89E-05**		**trial 3**	**4.89E-05**	**0.000485**	**0.004784**	**0.051396**	**0.678873**	**24.40921**
100	**0.000471**		**trial 4**	**5.08E-05**	**0.000483**	**0.005369**	**0.050833**	**0.67173**	**23.83153**
100	**0.000473**		**trial 5**	**4.89E-05**	**0.000471**	**0.005896**	**0.051168**	**0.667786**	**24.13703**
100	**0.000485**		**average Time**	**5.39E-05**	**4.77E-04**	**5.31E-03**	**5.12E-02**	**6.74E-01**	**2.41E+01**
100	**0.000483**								
100	**0.000471**								
1000	**0.005718**								
1000	**0.004787**								
1000	**0.004784**								
1000	**0.005369**								
1000	**0.005896**								
10000	**0.051579**								
10000	**0.05111**								

	9								
10000	**0.051396**								
10000	**0.050833**								
10000	**0.051168**								
100000	**0.668512**								
100000	**0.684119**								
100000	**0.678873**								
100000	**0.67173**								
100000	**0.667786**								
1000000	**24.32458**								
1000000	**23.58587**								
1000000	**24.40921**								
1000000	**23.83153**								
1000000	**24.13703**								