MARCELO SALHAB BROGLIATO

ESSAYS IN COMPUTATIONAL MANAGEMENT SCIENCE

ESSAYS IN COMPUTATIONAL MANAGEMENT SCIENCE

MARCELO SALHAB BROGLIATO

Escola Brasileira de Administração Pública e de Empresas
Fundação Getulio Vargas

January 2018

Marcelo Salhab Brogliato: *Essays in Computational Management Science,* © January 2018

SUPERVISORS:
Alexandre Linhares LOCATION:

Rio de Janeiro

ABSTRACT

Short summary of the contents in English...

RESUMO

Aqui entre o resumo...

Showing gratitude is one of the simplest yet most powerful things humans can do for each other.

— Randy Pausch, The Last Lecture

ACKNOWLEDGMENTS

Finally, thanks to everyone who helped me in any way, for each word, for caring, for your support in difficult moments and for all jokes and talks that made my days worthwhile. Surely this work was done thanks to all of you. After all, nobody does it alone!

CONTENTS

i INTRODUCTION 1

1 INTRODUCTION 3
- 1.1 Distributed financial ledgers 5
- 1.2 Artificial intelligence 6
- 1.3 Diffusion of innovation 8
- 1.4 The fine print... 8

ii AN ALTERNATIVE STRUCTURE TO A BLOCKCHAIN: CONFIRMATIONS WITHOUT BLOCKS 11

2 INTRODUCTION 13

3 BITCOIN & BLOCKCHAIN 17

4 ANALYSIS OF BITCOIN 21
- 4.1 Hash function 21
- 4.2 Mining one block 21
- 4.3 Mining several blocks 23
- 4.4 Mining for a miner 24
- 4.5 Orphan blocks 26
- 4.6 Analysis of network's hash rate change 26
  - 4.6.1 Hash rate smoothly changing 27
  - 4.6.2 Piecewise linear model of hash rate change 29
- 4.7 Attack in the Bitcoin network 30
- 4.8 Confirmation time and network capacity 34
- 4.9 Fork analysis 35

5 DAG MODEL 37
- 5.1 Nuclear submarine attack 39
- 5.2 Proposal: Questions to be explored 40

6 METHODOLOGY 43

7 ANALYSIS OF DAG MODEL 45

8 CONCLUSION 47

iii SPARSE DISTRIBUTED MEMORY: A CROSS-PLATFORM, MASSIVELY PARALLEL, OPEN SOURCE REFERENCE IMPLEMENTATION 49

9 INTRODUCTION 51

10 NOTATION 55

11 SPARSE DISTRIBUTED MEMORY 57
- 11.1 Neurons as pointers 63
- 11.2 Concepts 63
- 11.3 Read operation 64
  - 11.3.1 Generalized read operation 65

11.4 Critical Distance 66
12 FRAMEWORK ARCHITECTURE 69
12.1 Bitstring 70
12.1.1 The distance between two bitstrings 70
12.2 Address space 71
12.2.1 Scanning for activated hard locations 71
12.2.2 OpenCL kernels 72
12.3 Counters 73
12.4 Read and write operations 82
13 RESULTS (I): FRAMEWORK VALIDATION 83
13.1 Distance between random bitstrings 83
13.2 Number of activated hard locations 84
13.3 Intersection of two circles 85
13.4 Storage and retrieval of sequences 86
13.4.1 k-fold memory using only one SDM 87
14 RESULTS (II): CRITICAL DISTANCE 89
14.1 A deviation from the equator distance? 93
14.2 Counter bias 96
14.3 Read bias 99
14.4 Critical distance of 209 101
15 RESULTS (III): LOSS OF NEURONS 103
16 RESULTS (IV): GENERALIZED READ OPERATION 109
17 RESULTS (V): PERFORMANCE 113
17.1 Kernels comparison 114
17.2 Scanners comparison 115
17.3 Read and write operations 115
17.3.1 Summary of results 115
18 RESULTS (VI): SUPERVISED CLASSIFICATION APPLICATION 137
19 RESULTS (VII): IMAGE NOISE FILTERING APPLICATION 147
20 RESULTS (VIII): THE POSSIBILITY OF UNSUPERVISED REINFORCEMENT LEARNING 153
20.1 Training 156
20.2 Results 156
21 RESULTS (IX): INFORMATION-THEORETICAL WRITE OPERATION 161
22 CONCLUSION 165
22.1 “i” versus “l” 165
22.2 Magic numbers 166
22.3 Symmetrical, rapidly accessible, hard locations 166
22.4 Illuminating, paving, clearing 167
23 APPENDIX 171

IV DIFFUSION AND DISMISSAL OF INNOVATION: FORECASTING THE NUMBER OF FACEBOOK'S ACTIVE USERS 175

24 INTRODUCTION 177

25 THE BASS MODEL 181

26 THE EXTENDED MODEL 183

27 MODELS FOR R(t) 185
27.1 Model 1 185
27.2 Model 2 186
27.3 Model 3 187
27.4 Model 4 188

28 ESTIMATION METHOD 191

29 PRELIMINARY RESULTS 193

30 CONCLUSION 197

V CONCLUSION 199

31 CONCLUSION 201

VI APPENDIX 205

A RECENT RESULTS IN THEORY OF COMPUTING - I 207
A.1 The Halting Problem is Solvable 207

BIBLIOGRAPHY 209

Figure 1 Slides from an old Linhares' class; a viewpoint that has influenced the choice of topics found in this thesis. In the history of computing business, the most interesting level of analysis seems to be that of the *platform*. It is where things can get built on top of, and communities emerge, and standards fight against each other, and fortunes are built or lost. It is, in a sense, a major part of the Big Drama of our moment in history. Each time a new computing platform appears, it seems as the opportunities are ripe for the taking; as if a multitude of doors have opened simultaneously. Note that the slides were made in 2007, and expected an AI-based 'semantic revolution' by 2015 [64]. In between, both the iPhone (and competitors) and Bitcoin (and competitors) have created giant platforms on top of which immense wealth has been created (Uber, Instagram, etc, on the case of smartphones; and exchanges, miners, investors, payment processors, etc., in the case of blockchains). The key point is that we should (i) expect new, unforeseen, technological platforms, (ii) rapidly identify them, and (iii) throw our energy at them, as they offer leverage to make an asymmetric impact. 4

Figure 2 Probability density function of Y_6, i.e., probability of finding 6 blocks after time t. The shaded areas shows the lower 5% and upper 5% of the pdf. 24

Figure 3 $\mathbf{E}[T]$ when H increases linearly with $u(t) = 1 + at$. 30

Figure 4 Both the attacker and the network are mining. Each step up is a new block found by the network with probability p. Each step right is a new block found by the attacker with probability $1 - p$. It ends when the network finds k blocks — in this example, $k = 6$. The red path has probability $p^6(1-p)^3$, while the blue path has probability $p^6(1-p)^7$. Notice that the blue path is a successfull attack, because the attacker has found more blocks than the network. In the red path, the attacker still have to catch up 3 blocks to have a successful attack, which happens with probability ρ^3, if $p < 0.5$. 32

Figure 5 Probability of a successful attack according to the network's hash rate of the attacker (β). 34

Figure 6 White nodes represent transactions that have been confirmed at least once. Green circles represent unconfirmed transactions (tips). Gray and dashed nodes are the transactions currently solving the proof-of-work in order to be propagated. 38

Figure 7 Suddenly the number of transactions per second increases and the width of the swarm grows. After a while, the number of transactions per second decreases and the width of the swarm shrinks. 39

Figure 8 The red nodes are transactions which had some conflict with previous transaction and were invalidated by the network. Notice that none of them have been confirmed. 39

Figure 9 Histogram of how long has been a transaction waiting until its first confirmation. It was a simulation of 15 minutes with new transactions rate changing between 1 and 15 tx/s. 41

Figure 10 Histogram of how long has been a transaction waiting until its first confirmation. It was a simulation of 5 minutes with new transactions rate of 50 tx/s, i.e., very high load. 42

Figure 11 Here we have Q_n, for $n \in \{3, 7, 10\}$. Each node corresponds to a bitstring in $\{0,1\}^n$, and two nodes are linked iff the bitstrings differ by a single dimension. A number of observations can be made here. First, the number of nodes grows as $O(2^n)$; which makes the space rapidly intractable. Another interesting observation, better seen in the figures below, is that most of the space lies 'at the center', at a distance of around $n/2$ from any given vantage point. 58

Figure 12 Activated addresses inside access radius r around the center address. 59

Figure 13 Shared addresses between the target datum η and the cue η_x. 60

Figure 14 Hard-locations randomly sampled from binary space. 61

Figure 15 In this example, four iterative readings were required to converge from η_x to η. 62

Figure 16 Hard-locations pointing, approximately, to the target bitstring. 63

Figure 17 How far, in Hamming distance, is a read item from the original stored item? Kanerva demonstrated that, after a small number of iterative readings (6 here), a critical distance behavior emerges. Items read at close distance converge rapidly; whereas farther items do not converge. Most striking is the point in which the system displays the tip-of-tongue behavior. Described by psychological moments when some features of the item are prominent in one's thoughts, yet the item still cannot be recalled (but an additional cue makes convergence 'immediate'). Mathematically, this is the precise distance in which, despite having a relatively high number of cues (correct bits) about the desired item, the time to convergence is infinite. Heatmap colors display the Hamming distance the associative memory is able to cleanly converge to—or not. In the x-axis, the distance from the desired item is displayed. In the y-axis, we display the read operation's behavior as the number of items registered in the memory grows. These graphs are computing intensive, yet they can be easily tested by readers in our provided Jupyter notebooks. Note the different scales. 67

Figure 18 Address space's bitstrings are stored in a contiguous array. In a 64-bit computer, each bitstring is stored in a sub-array of 64-bit integers, with length $8 \cdot \lceil n/64 \rceil$. 71

Figure 19 Histogram of 10,000 distances between two random bitstrings with 1,000 bits. The curve in red is the theoretical normal distribution with $\mu = 500$ and $\sigma = \sqrt{500}/2$. 83

Figure 20 Histogram of the number of activated hard locations in 10,000 scans from a random bitstring. The curve in red is the theoretical normal distribution with $\mu = Hp$ and $\sigma = p(p-1)H$. 84

Figure 21 Histogram of the distances of activated hard locations to the center of the circles. The curve in red is the theoretical distribution of Equation 3 85

Figure 22 Number of hard locations in the intersection of circles around two bitstrings x bits away. 86
Figure 23 Kanerva's original Figure 7.3 (p. 70) predicting a ~500-bit distance after a point. 89
Figure 24 Results generated by the framework diverging from Kanerva's original Figure 7.3. Here we had a 1,000 bit, 1,000,000 hard location SDM with 10,000 random bitstrings written into it, which was also Kanerva's configuration. 90
Figure 25 Results generated by the framework similar to Kanerva's original Figure 7.3. Here we have a 1,000 bit, 1,000,000 hard location SDM with (a) just 100 random bitstrings written into it and (b) steps of 1,000 random bitstrings written into it. 91
Figure 26 This graph shows the interaction effects more clearly. As we change the single read to a 6-iterative read, the effect has vanished, and all bitstrings above $x = 500$ have converged to 500-bit distance bitstrings. Here we have precisely the same configuration of Figure 24, except for the iterative read. 92
Figure 27 Kanerva's original Figure 7.3 generated using the equations from Brogliato et al. [20]. 93
Figure 28 Given an address x and a dimension i, how many hard locations with bitmatches in i are activated by reading at x? The histogram was obtained through numerical simulation. The red curve is the theoretical normal distribution found in Theorem 14. 96
Figure 29 The value of the counters after $s = 10,000$ writes shows the autocorrelation in the counters in autoassociative memories ("x at x"). The histogram was obtained through simulation. The red curve is the theoretical normal distribution found in equations (13) and (14). 98
Figure 30 Autocorrelation in the counters in autoassociative memories ("x written at x"). The histogram was obtained through simulation. The red curve is the theoretical distribution. 98

Figure 31 The histogram was obtained through simulation. The red curve is the theoretical normal distribution. 100
Figure 32 The histogram was obtained through simulation. The red curve is the theoretical normal distribution. 100
Figure 33 The histogram was obtained through simulation. The red curve is the theoretical normal distribution. 101
Figure 34 New distance after a single read operation in a bitstring η_d, which is d bits away from η. The new distance was calculated between η_d and read(η_d). Notice that when $d \geqslant 520$, the intersection between η and η_d is zero, which means there is only random bitstrings written into the activated hard locations. The distance 220 equals $1000 \cdot 0.220$ which is the probability find in Figure 33. 101
Figure 35 Zoom-in around $d = 209$ of Figure 27. 102
Figure 36 Zoom-in around $d = 209$ of Figure 24. 102
Figure 37 This graph shows the SDM's robustness against loss of neurons in a SDM with $n = 1,000$ and $H = 1,000,000$. It shows that a loss of 200,000 neurons, 20% of the total, does not seem to affect SDM whatsoever. 104
Figure 38 This graph shows the SDM's robustness against loss of neurons in a SDM with $n = 1,000$ and $H = 1,000,000$. The more neurons are lost, the smaller the critical distance, i.e., the worse the SDM recall. 104
Figure 39 This graph shows the SDM's robustness against loss of neurons in a SDM with $n = 1,000$ and $H = 1,000,000$. Even when 50% of neurons are dead, SDM recall is barely affected, which is an impressive result and matches with some clinical results of children submitted to hemispherectomy. 106
Figure 40 This graph shows the SDM's robustness against loss of neurons in a SDM with $n = 1,000$ and $H = 1,000,000$. 106
Figure 41 This graph shows the SDM's robustness against loss of neurons in a SDM with $n = 256$ and $H = 1,000,000$. 107

Figure 42 (a) and (b) show the behavior of a single read. As stated previously, we can see a deterioration of convergence, with lower critical distance as $z > 1$. Another observation can be made here, concerning the discrepancy of Kanerva's Fig 7.3 and our data. It seems that Kanerva may not have considered that a single read would only 'clean' a small number of dimensions *after the critical distance*. What we observe clearly is that with a single read, as the distance grows, the system only 'cleans' towards the orthogonal distance 500 after a number of iterative readings. 110

Figure 43 (a) and (b) show the behavior of Figure 42, now executed with 6-iterative reads. What we observe clearly is that with a single read, as the distance grows, the system only 'cleans' towards the orthogonal distance 500 after a number of iterative readings. 111

Figure 44 Kernel comparisons for MacBook Pro Retina 13-inch Late 2013 with a 2.6GHz Intel core i5 processor, 6GB DDR3 RAM, and Intel Iris GPU. 117

Figure 45 Scanner comparisons for MacBook Pro Retina 13-inch Late 2013 with a 2.6GHz Intel core i5 processor, 6GB DDR3 RAM, and Intel Iris GPU. 118

Figure 46 Read operation comparisons for MacBook Pro Retina 13-inch Late 2013 with a 2.6GHz Intel core i5 processor, 6GB DDR3 RAM, and Intel Iris GPU. 119

Figure 47 Write operation comparisons for MacBook Pro Retina 13-inch Late 2013 with a 2.6GHz Intel core i5 processor, 6GB DDR3 RAM, and Intel Iris GPU. 120

Figure 48 Kernel comparisons for iMac Retina 5K 27-inch 2017 with a 3.8GHz Intel core i5 processor, 8GB DDR4 RAM, and a Radeon Pro 580 8G GPU. 122

Figure 49 Scanner comparisons for iMac Retina 5K 27-inch 2017 with a 3.8GHz Intel core i5 processor, 8GB DDR4 RAM, and a Radeon Pro 580 8G GPU. 123

Figure 50 Read operation comparisons for iMac Retina 5K 27-inch 2017 with a 3.8GHz Intel core i5 processor, 8GB DDR4 RAM, and a Radeon Pro 580 8G GPU. 124
Figure 51 Write operation comparisons for iMac Retina 5K 27-inch 2017 with a 3.8GHz Intel core i5 processor, 8GB DDR4 RAM, and a Radeon Pro 580 8G GPU. 125
Figure 52 Kernel comparisons for Amazon EC2 p2.xlarge with Intel Xeon E5-2686v4 processor, 61GB DDR3 RAM, and NVIDIA K80 GPU. 127
Figure 53 Scanner comparisons for Amazon EC2 p2.xlarge with Intel Xeon E5-2686v4 processor, 61GB DDR3 RAM, and NVIDIA K80 GPU. 128
Figure 54 Read operation comparisons for Amazon EC2 p2.xlarge with Intel Xeon E5-2686v4 processor, 61GB DDR3 RAM, and NVIDIA K80 GPU. 129
Figure 55 Write operation comparisons for Amazon EC2 p2.xlarge with Intel Xeon E5-2686v4 processor, 61GB DDR3 RAM, and NVIDIA K80 GPU. 130
Figure 56 Kernel comparisons for Amazon EC2 p3.2xlarge with Intel Xeon E5-2686v4 processor, 488GB DDR3 RAM, and NVIDIA Tesla V100 GPU. 132
Figure 57 Scanner comparisons for Amazon EC2 p3.2xlarge with Intel Xeon E5-2686v4 processor, 488GB DDR3 RAM, and NVIDIA Tesla V100 GPU. 133
Figure 58 Read operation comparisons for Amazon EC2 p3.2xlarge with Intel Xeon E5-2686v4 processor, 488GB DDR3 RAM, and NVIDIA Tesla V100 GPU. 134
Figure 59 Write operation comparisons for Amazon EC2 p3.2xlarge with Intel Xeon E5-2686v4 processor, 488GB DDR3 RAM, and NVIDIA Tesla V100 GPU. 135
Figure 60 Examples of noisy images with uppercase letters, lowercase letters, and numbers. 137
Figure 61 One noisy image for each of the 62 classification groups. 138
Figure 62 100 noisy images generated to train label A. 139

Figure 63 Images generated using a 20% noise for the high noise scenario. 139
Figure 64 Images generated for the no noise scenario. 140
Figure 65 Images of different characters which may be confusing depending on the noise level. 141
Figure 66 Characters in the low noise scenario in which the classifier has made at least one mistake. In all the other cases, it correctly classified the images. We may notice that the groups of "i" and "l" have been completely merged by the classifier, because it cannot distinguish them, not even with no noise. 142
Figure 65 Characters in the high noise scenario in which the classifier has made at least one mistake. In all the other cases, it correctly classified the images. 145
Figure 66 Progressive noise into letter "A", from 0% to 45% in steps of 5%. 147
Figure 67 Training images written into the SDM. They were written in their own addresses — write(address=bs_image, datum=bs_image). 148
Figure 68 In order to test the SDM as a noise filter, we read from noisy images expecting to get a clean image. It is interesting to highlight that SDM has never seen a clean version letters "T" and "I". 148
Figure 69 Probability of getting the right pixel when reading from an image with noise p. It assumes that SDM was trained with 200 images with 15% noise. 149
Figure 70 Training images in which the intersection between images is too high. They were written in their own addresses — write(address=bs_image, datum=bs_image). 150
Figure 71 When the intersection between images becomes too high, there appears some interference in the resulting image. All cases have 10% noise. We can notice that the empty space on the right side of the "C" letter generates some white pixels on the right side of both "B" and "D" letters. 150

Figure 72 Using labels solves the interference problem when the intersection between images becomes too high. All cases have 20% noise. 151
Figure 73 Each action is a cell in the TicTacToe board and is mapped to slice of the bitstring. 154
Figure 74 Example of a game with 7 movements in which X wins. 154
Figure 75 Positive reward bitstrings used in our reinforcement learning algorithm. 155
Figure 76 Results playing against the random player. Each cycle was made of 100 games for training, and then 100 games for measuring statistics. 157
Figure 77 Results playing against the smart player. Each cycle was made of 100 games for training, and then 100 games for measuring statistics. 158
Figure 78 Results playing against another SDM player. Each cycle was made of 100 games for training, and then 100 games for measuring statistics. 159
Figure 79 Results playing against a randomly chosen player between random player, smart player, and another SDM player. Each cycle was made of 100 games for training, and then 100 games for measuring statistics. 160
Figure 80 Shannon write operation: Computing the amount of information of a signal to each hard location in its access radius. (a) entirety of the space; (b) region of interest; (c) Fast integer computation is possible through a stepwise function. 163
Figure 81 Behavior of the critical distance under the information-theoretic weighted write operation when $n = 1,000$, $H = 1,000,000$ and $r = 451$. 164
Figure 82 Fit of Model 2 with Facebook's active users dataset. $mF(t)$ is the total users, $mR(t)$ is the inactive users, and $mA(t)$ is the active users. The unit of these functions are thousands of people. The parameters are $m = 1,497.50$, $p = 0.000331$, $q = 0.100088$, $w = 0.140595$, and $\nu = 0.187188$. The goodness of fit are $R^2 = 99.84\%$ and BIC=10,566.52. 194

Figure 83 Fit of Model 3 with Facebook's active users dataset. $mF(t)$ is the total users, $mR(t)$ is the inactive users, and $mA(t)$ is the active users. The unit of these functions are thousands of people. The parameters are $m = 1,967.64$, $p = 0.000184$, $q = 0.097867$, $w = 0.330511$, and $\nu = 0.006912$. The goodness of fit are $R^2 = 99.83\%$ and BIC=11,485.68 194

Figure 84 Fit of Model 4 with Facebook's active users dataset. $mF(t)$ is the total users, $mR(t)$ is the inactive users, and $mA(t)$ is the active users. The unit of these functions are thousands of people. The parameters are $m = 1,854.85$, $p = 0.000183$, $q = 0.099738$, $w = 0.334454$, and $\nu = 0.007007$. The goodness of fit are $R^2 = 99.84\%$ and BIC=10,724.55 195

LIST OF TABLES

Table 1 Write operation example in a 7-dimensional memory of data η being written to ξ, one of the activated addresses. 62

Table 2 Comparison of Kanerva's read and Chada's read. Each ξ_i is an activated hard location and the values come from their counters. Gray cells' value is obtained randomly with probability 50%. 66

Table 3 MacBook Pro Retina 13-inch Late 2013 with a 2.6GHz Intel core i5 processor, 6GB DDR3 RAM, and Intel Iris GPU. The SDM settings were: (i) $n = 256$, $r = 103$, and $H = 1,000,000$; (ii) $n = 1,000$, $r = 451$, and $H = 1,000,000$; and (iii) $n = 10,000$, $r = 4850$, and $H = 1,000,000$. There is no benchmark for $n = 10,000$ because memory is not enough on either RAM or GPU—it would consume 37.25 GB of RAM and 1.2GB of memory in the GPU. For the histogram of durations, see Figures 44, 45, 46, and 47. 116

Table 4 iMac Retina 5K 27-inch 2017 with a 3.8GHz Intel core i5 processor, 8GB DDR4 RAM, and a Radeon Pro 580 8G GPU. The SDM settings were: (i) $n = 256$, $r = 103$, and $H = 1,000,000$; (ii) $n = 1,000$, $r = 451$, and $H = 1,000,000$; and (iii) $n = 10,000$, $r = 4850$, and $H = 1,000,000$. There is no benchmark for read and write operations with $n = 10,000$ because RAM is not enough to allocate the counters—it would consume 37.25 GB of RAM. For the histogram of durations, see Figures 48, 49, 50, and 51. 121

Table 5 Amazon EC2 p2.xlarge with Intel Xeon E5-2686v4 processor, 61GB DDR3 RAM, and NVIDIA K80 GPU. Running an SDM with $n = 256$ bits, $H = 1,000,000$, and $r = 103$. The SDM settings were: (i) $n = 256$, $r = 103$, and $H = 1,000,000$; (ii) $n = 1,000$, $r = 451$, and $H = 1,000,000$; and (iii) $n = 10,000$, $r = 4850$, and $H = 1,000,000$. There is no benchmark for kernel single_scan5_unroll because it returns the wrong result in this GPU. The problem is related to the premises of the optimization used by this kernel, which are not true for this GPU. For the histogram of durations, see Figures 52, 49, 50, and 51. 126

Table 6 Amazon EC2 p3.2xlarge with Intel Xeon E5-2686v4 processor, 488GB DDR3 RAM, and NVIDIA Tesla V100 GPU. The SDM settings were: (i) $n = 256$, $r = 103$, and $H = 1,000,000$; (ii) $n = 1,000$, $r = 451$, and $H = 1,000,000$; and (iii) $n = 10,000$, $r = 4850$, and $H = 1,000,000$. There is no benchmark for kernel single_scan5_unroll because it returns the wrong result in this GPU. The problem is related with the premises of the optimization used by this kernel, which are not true for this GPU. For the histogram of durations, see Figures 56, 57, 58, and 59. 131

Part I

INTRODUCTION

1 INTRODUCTION

It is still an unending source of surprise for me to see how a few scribbles on a blackboard or on a sheet of paper could change the course of human affairs.

— Stanislaw Ulam

If anything good can ever be said about the second world war, it might be this: the war effort sparked a massive number of scientific fields.

Though most fields existed prior to the war, after the war they were funded by the public as strategic pieces of the major nations arsenal against future conflagrations. One of the fields in question was that of Management Science (also called Operations Research in military circles, as researchers filled the ranks of planners of war operations). Management Science had started as an industrial field, in movements stemming from Taylor and the origin of the production line by Henry Ford. That was the first moment in industry in which operations were systematically subject to some of the tools of science: measurement, experimentation, hypothesis-testing, statistics, mathematical optimization, etc.

This humble beginnings date from almost 100 years ago. Today the field has advanced to a great number of nations, and the amount of applications has grown explosively. Of particular interest to us is the advent of the computer, and of engineering efforts that brought exponential growth in computational power to the hands of individuals. Whilst, during the war, computations were mostly done by hand, the electronic computer took over afterward; up to an extent that it is not outlandish to say that this original field can be referred to, today, as *computational management science*.

Applied mathematics and computer science serve simultaneously as a theoretical foundation and the major tool available to the field. Though this is a doctoral thesis concerning business, in this document one should expect to find the language and nomenclature of mathematical modeling and computer science as our primary and most natural language.

This thesis will explore three different topics related to *computing business platforms* (Figure 1). Though the range of the topics is large, as it usually is in management science, it is my hope to convince readers of the value of this doctoral thesis brought by three specific, self-contained, scientific papers. The first of which studies the possibility of distributed financial ledgers.

You only get one life

What to do with it?

(a)

1975: I lost the microcomputer revolution

(b)

1985 (-1): I lost the GUI revolution

(c)

1995: I lost the web revolution

(d)

2005: I lost the Web 2.0 revolution

(e)

~2015: the semantic revolution

(f)

Figure 1: Slides from an old Linhares' class; a viewpoint that has influenced the choice of topics found in this thesis. In the history of computing business, the most interesting level of analysis seems to be that of the *platform*. It is where things can get built on top of, and communities emerge, and standards fight against each other, and fortunes are built or lost. It is, in a sense, a major part of the Big Drama of our moment in history. Each time a new computing platform appears, it seems as the opportunities are ripe for the taking; as if a multitude of doors have opened simultaneously. Note that the slides were made in 2007, and expected an AI-based 'semantic revolution' by 2015 [64]. In between, both the iPhone (and competitors) and Bitcoin (and competitors) have created giant platforms on top of which immense wealth has been created (Uber, Instagram, etc, on the case of smartphones; and exchanges, miners, investors, payment processors, etc., in the case of blockchains). The key point is that we should (i) expect new, unforeseen, technological platforms, (ii) rapidly identify them, and (iii) throw our energy at them, as they offer leverage to make an asymmetric impact.

1.1 DISTRIBUTED FINANCIAL LEDGERS

The World Bank estimates that there are two billion people without access to financial services. As banks are unable to sustain operations in numerous poverty-stricken areas, services such as money transfers, access to credit, digital/distant payments, inflation protection, etc., remain beyond reach for 'the unbanked'. This seems to be one of the factors that perpetuate poverty. The Bill and Melinda Gates Foundation chose as focus of its "Level-One project": to provide basic financial services through cell phones. Another initiative, the United Nations World Food Programme has begun, in 2017, an experiment in Jordan, in which the organization provides funds for thousands of people towards its goal of food relief. An interesting aspect of this program has been the format of the funds distributed: they have been all on the ethereum blockchain [102, 85, 86].

The possibility of having a completely digital financial system without the overheads of traditional banking systems has appeared with the release of Bitcoin and similar blockchain technologies. This field questions numerous traditional assumptions in computer science, record-keeping, banking & finance, and economic inclusion. The seminal work of Nakamoto [73] described the architecture of Bitcoin, a peer-to-peer electronic cash system, also known as cryptocurrency. Bitcoin's currency ledger is public and stored in a blockchain across thousands of computers. Even so, no one is able to spend either somebody else's funds nor to double spend their own funds. In order to be confirmed, each transaction must be both digitally signed by the owner of the money and the funds verified in the blockchain by Bitcoin's miners. The question of whether Bitcoin (or related works) can scale to billions of people is, however, far from settled.

One of the interesting parts of Bitcoin are the incentives. On one hand, users have incentive to use Bitcoin because the fees are small, the money transfer is quick and global, and the currency issuance rate is well known. On the other hand, the miners have incentive to be part of Bitcoin's network because, every ten minutes, new coins are found and transactions' fees are collected. These incentives keep the community together and have maintained Bitcoin alive.

The impact of Bitcoin in society — and hence in the companies and the government — has been growing every day. People are increasingly using Bitcoin to exchange money and transfer money overseas. Companies are looking into Bitcoin as an alternative to reduce banking fees. The poor may be included in the finance system through Bitcoin. People may hedge their assets against their governments' money issuance and inflation — as in the case of Venezuela.

Bitcoin is the first and most famous cryptocurrency, used worldwide, with a highly volatile market cap, as of this writing, of $ 192 bi. Even so, it faces serious scalability challenges; such as serious quality of service and network congestion when the number of transactions per second is high, and an increase in the transaction fees and uncertain delays in transactions' confirmations.

Note that these problems have been a deliberate decision from the current developers of the "bitcoin-core", which believe that is it risky to increase the blocksize (in which all transactions are stored). It is not known whether a blocksize, say, of 1GB, would be feasible to sustain the decentralization of the network.

Iota is a second cryptocurrency that, instead of using a blockchain, proposes the use of a "tangle' architecture': a different way to register the currency ledger across thousands of computers. Although it has not been confirmed in practice yet, its architecture seems to be significantly more scalable than Bitcoin's blockchain. As we will see, the problem here is exactly the opposite of Bitcoin's. Iota needs a minimum of transactions per seconds in order to work properly.

Our analysis suggests an architecture for a distributed currency which is inspired in both Bitcoin's blockchain and Iota's tangle in order to solve the scalability problems. While Bitcoin's network saturates when it hits a certain number of transactions per second, Iota's does not work properly with less than a certain number of transactions per second. Our proposed architecture seems to work in both scenarios: low and high number of transactions per second.

In this first study we will investigate some issues regarding this possibility, namely: (i) cryptographic security and game-theoretical attacks; (ii) scalability; (iii) self-governance of the system; (iv) appropriate incentive system to all participants.

A second topic that may have an outsized influence on business and that we will be taking a closer look is a model of artificial intelligence.

1.2 ARTIFICIAL INTELLIGENCE

Technology has been one of the underlying engines behind economic growth. It has been changing the whole society – people, companies, and governments. Cities and houses had to be rethought when cars became popular. Trains allowed distant places to exchange high volume of goods. Airplanes and boats opened countries to overseas business. And, finally, the internet has had a profound impact in nearly everyone's life, as it changed everything – from the way we communicate, behave, do business, do shopping, share ideas, and so forth.

One area of technology that has been redefining business is computer science. Together with the internet, computer science has been one of the most important tools to scale a business model – and create many others which were impossible before. More and more expensive human labor has been replaced by algorithms. Managers are able to make better decisions because they receive real-time information. The supply chain has incredibly evolved thanks to advances in logistics supported by routing algorithms, storage algorithms, and many others optimization algorithms.

Artificial intelligence has been disrupting many businesses. Uber is able to handle hundreds of thousands of requests. Amazon optimizes the location of each product based on demand. Netflix increases the quality of their services offering movies specific to the taste of each customer. Spotify learns which kind of music users like the most and suggests playlists. Banks prevent fraud classifying which patterns seem to be erratic towards their customers' previous behavior.

It is gradually becoming impossible to imagine a world without artificial intelligence.

Behind artificial intelligence systems, there is pattern recognition: The capacity to match information from new data with information which has already been seen and is stored in memory. It may be used in classification, face recognition, character recognition, and so forth. Even if AI cannot yet solve numerous hard problems — such as the Turing Test, Bongard Problems or simply understanding when 'Lawyers are sharks' [43, 44, 65, 45] —, it is clear that the technology must be taken seriously.

The second paper lies at the intersection of cognitive psychology, computer science, neuroscience, and artificial intelligence. Sparse Distributed Memory, or SDM for short, is a theoretical mathematical construct that seems to reflect a number of neuroscientific and psychologically plausible characteristics of a human memory. SDM has already been used to different pattern recognition problems, like noise reduction, handwriting recognition, robot automation, and so forth.

We implement a RB-Complete[1] SDM framework that not only shows small discrepancies from previous theoretical expectations, but also may be of use to other researchers interested in testing their own hypotheses and theories of SDM. The computer code has been used in a previous Ph.D. Thesis; the code has shown some small discrepancies from theoretical expectations; the code has been run on a number of different architectures and information-processing devices (e.g., CPUs, GPUs). The framework enables us to have a

1 'Ridiculously Buzzword Complete: the model is (i) Open-Source, (ii) Cross-Platform; (iii) highly parallel; (iv) able to execute on CPUs and/or GPUs; (v) it can be run on the 'cloud'; etc.

visual exploration previous experiments and new possibilities for SDM.

1.3 DIFFUSION OF INNOVATION

In 2014, a group of Princeton's researchers predicted that Facebook's users would abandon the platform by 2017 [23]. The forecast was done applying a disease spreading model which has correctly predicted the abandonment of "MySpace". Facebook replied after applying Princeton's methodology:

> "Using the same robust methodology featured in [Princeton's] paper, we attempted to find out more about this 'Princeton University' — and you won't believe what we found!". Then, they conclude: "This trend suggests that Princeton will have only half its current enrollment by 2018, and by 2021 it will have no students at all, agreeing with the previous graph of scholarly scholarliness. Based on our robust scientific analysis, future generations will only be able to imagine this now-rubble institution that once walked this earth".

Whilst this brouhaha reminds one of the dangers of extrapolation, our third paper will revisit the prospects of our esteemed colleagues in Facebook. Lying at the intersection of Marketing, Diffusion of Technological Innovation, and modeling, the Bass model of diffusion of innovation will be extended, in order to account for users who, after adopting the innovation for a while, decide to reject it later on (possibly bringing down the number of active users—something impossible in Bass' original model). Four alternative mathematical models are presented and discussed with the Facebook's users dataset.

1.4 THE FINE PRINT...

Before embarking on the technical topics, small qualifications must be asked from my readers. First, as stated above, though these problems have immense and urgent importance to the fields of study in business, the language in which we will approach them and discuss them most naturally will be that of mathematics and computer science. There will not be surveys, interviews, questionnaires, or such methods typically used in the social sciences: This is basically a work of modeling.

A second and final qualification: It is my hope that readers of this thesis will accept the format of self-contained studies, as just as valid

as a monograph on a particular topic[2]. With these qualifications, we are ready to venture into the world of computational management science.

2 e.g., a reference to the "fundamental research theorem", found in the Appendix, seems inescapable.

Part II

AN ALTERNATIVE STRUCTURE TO A BLOCKCHAIN: CONFIRMATIONS WITHOUT BLOCKS

2 INTRODUCTION

We are watching History being made — or History being repeated.

— David Collum, Cornell University, 2013

The main problem when one is trying to create digital money is how to prevent double spending. As the money is digital, and copies can be made *ad nauseam*, what can prevent counterfeiting? That is, what would prevent users from sending copies of the same money to two (or more) people? That is precisely the problem solved by Bitcoin and its underlying blockchain technology. The current solution behind fiat money is having a single issuer, a central bank, and trusting the financial institutions.

Bitcoin (BTC) is the first digital currency, also known as digital money, internet money, and cryptocurrency. It is the first currency based on cryptography techniques, distributed and decentralized, and with no central bank. Bitcoin is distributed since its ledger is public and is stored in thousands of computers. It is decentralized because there is no authority (or government) who decides its future — any decision must be accepted by its community. The security of Bitcoin relies on digital signature technology and network agreement. While digital signature ensures ownership, i.e., the funds may only be spent by their owners, and nobody else; the network agreement both prevents double spending and ensures that all processed transactions have sufficient funds. In short, every transaction must spend only unspent funds, must have enough funds available, and must be signed by its owners, authorizing its execution. When all these requirements are met, the funds are transferred.

According to Barber et al. [8], despite the 30-year literature on e-cash, most of the proposed schemes requires a central authority which controls the currency issuance and prevents double spending. The no central point of trust and predictable money supply together with a clever solution to the double-spending problem is what separates Bitcoin from the previous e-cash philosophies.

Bitcoin provides interesting incentives to all players, namely the users and the miners. On the one hand, users may have incentives to use Bitcoin because of the following: (i) the fees are small and does not depend on the amount being transfered — but only in the size (in bytes) of the transaction — (ii) the transfers will be confirmed in

a well-known period; (iii) it is not possible to revert an already confirmed transfer, not even with a judicial order; and (iv) and the currency issuance rate is well-known and preset in Bitcoin's rules, which makes the Bitcoin supply predictable and trustworthy, different from fiat currencies which depends on decisions of their central banks — i.e., it would be virtually impossible to face a hyper inflation in Bitcoin due to currency issuance. On the other hand, miners have incentive to mine Bitcoin because new Bitcoins are found every ten minutes, and they may also receive the fees of unconfirmed transactions. These incentives have kept the Bitcoin network up and running since 2009 with barely any interruptions (99.99% uptime).

Since 2009, Bitcoin has been growing and becoming more and more used all around the world. It started as an experiment based on a seminal work by Nakamoto [73] and expanded to the most important and successful cryptocurrency with a highly volatile $192 billion market capitalization, as of this writing [28]. There are hundreds of companies investing in different uses of the technology, from exchanges to debit cards, and billions of dollars being invested in the new markets based on Bitcoin's technology.

Despite Bitcoin's huge success, there are still many challenges to be overcome. We will focus on a specific subset of those challenges, namely the scaling, decentralization, and spam challenges. One important challenge that we will skip is to reduce the size of the ledger (or blockchain), which today is around 125GB and is growing at a rate of 4.5GB per month [14].

The network must scale to support hundreds of transactions per second, while its capacity is around only eight transactions per second. Thus, the more Bitcoin becomes popular, the more saturated the network is. Network saturation has many side effects and may affect the players' incentive to keep the network running. The transaction fees have to be increased to compete for faster confirmation. The pool of unconfirmed transactions grows indefinitely, which may cause some transactions to be discarded due to low memory space available, as the previously predictable confirmation time of transactions becomes unpredictable.

Bitcoin seems to have the most decentralized network between the cryptocurrencies, even so, there are few miners and mining pools which together control over 50% of the network's computing (hash)power. Thus, they have an oversized influence when it comes to changes in the Bitcoin protocol's behavior, and it is also seen as a problem to be solved. The more decentralized, the more trustworthy Bitcoin is.

Generating new transactions in Bitcoin has a tiny cost because one only has to generate the transaction itself, digitally sign it, and propagate in the Bitcoin network. On the one hand, it means that

any device is capable of generating new transactions, but, on the other hand, it makes Bitcoin susceptible to spam attacks. One may generate hundreds of thousands of new valid transactions, overloading the unconfirmed transactions pool and saturating the network.

The number of ideas and publications focusing on improving Bitcoin's design and overcoming those challenges is increasing every day. Many of these proposals are organized into BIPs (Bitcoin Improvement Proposals) which are discussed and implemented by the community; while others come in the form of whitepapers and alternative software forks (which would include the need of a protocol upgrade). Other proposals are published in blogs and forums, describing new cryptocurrencies. Bitcoin's community hardly ever publishes their ideas in academic journals, preferring instead, of BIPs, white papers, and web discussions.

After the launch of Bitcoin, more than 1,000 other cryptocurrencies have been created (REF). In general, they are Bitcoin-like, which means they use similar technologies, including the blockchain. Some cryptocurrencies differs a lot from Bitcoin, like the ones which use the DAG model [38, 84, 62, 96, 63, 100]. We are especially interested in one of them: Iota.

Iota uses a DAG model, called tangle, which has a different design than Bitcoin's blockchain. It has neither mining nor confirmation blocks and transaction fees. Each transaction has its own proof-of-work and is used to confirm other transactions, forming a directed acyclic graph. In Iota, as transactions confirm transactions, the network benefits from a high volume of new transactions. Hence, theoretically, it scales to any large number of transactions per second. The scaling problem of tangle is exactly the opposite of Bitcoin's. It must have at least a given number of transaction per seconds; otherwise, the transactions are not confirmed, and the cryptocurrency does not work.

The present work intends to analyze a new architecture which lies between Bitcoin and Iota and may be a viable solution to both Bitcoin's scaling, centralization, and spam problems. We also present a mathematical analysis of Bitcoin's mining, forking, and safety.

3

BITCOIN & BLOCKCHAIN

In Nakamoto's (2009) seminal paper, there is no distinction between bitcoin and blockchain. They are just one thing which solves an important theoretical problem: how to create a distributed and decentralized digital form of hard money on the internet, in which all users can agree as to whom is entitled to which funds.

But, in practice, it is interesting to separate these concepts. Bitcoin uses the blockchain technology to create a distributed ledger, while the blockchain is a technology which allows information to be stored in an immutable and distributed way.

The blockchain technology works through the creation of new blocks. Each new block confirms that all the previous blocks are valid and have not been tampered with. The mechanism that assures the immutability is proof-of-work, which makes it computationally infeasible to tamper with previous transaction records without having to recalculate all the previous proof-of-works faster than all of the remaining machines of the network. The network agrees that work should be done in the block at the longest chain in the blockchain.

The proof-of-work is a mathematical problem with the following characteristics: (i) it is hard to find a solution; (ii) this hardness level may be adjusted; and (iii) it is fast to check whether the proposed solution is correct.

Bitcoin's blockchain uses the mathematical problem of finding a random number which, after being applied to the hash function SHA-256 twice, results in a number smaller than a given threshold A. As SHA-256 is a pseudo-random function, its output is uniformly distributed between 0 and $2^{256} - 1$ [47]. Thus, if the given number is $A = 2^{255}$, one has probability 50% of finding a solution (just the most significant bit of the hash needs to be zero). But if the given number is $A = 2^{240}$, one has probability 0.0015% of finding a solution (as the 16 most significant bits of the hash must equal zero). Hence, finding a solution is a hard problem which difficulty depends on the given number A. The lesser the given threshold A, the higher the difficulty. On the other side, checking whether a solution is correct is fast, one just has to apply the SHA-256 twice and compare.

By design, Bitcoin's blockchain proof-of-work difficulty is dynamically adjusted every 2016 blocks to keep an average pace of 10 minutes between block creation. Thus, the goal is to adjust the difficulty every 14 days. If it takes less than 14 days to find 2016 blocks, that means the network's hash power has increased; thus the

difficulty is increased. If it takes more than 14 days to find 2016 blocks, it means the network's hash power has decreased; thus the difficulty is decreased.

When miners are finding a solution to a new block, they are mining or working in the new block. A block is found when a solution to the proof-of-work is found. When a new block is found, it indirectly confirms all the previous blocks in the chain and their transactions. It may happen that two miners find two different blocks in a small interval of time. In this case, both miners will propagate their blocks, and the network will randomly choose one of them as the next block. This phenomenon is called a fork. Thus, when the next block is found, one of those blocks will be confirmed, and the other will be ignored and referred to as an orphan block. As blocks confirm transactions, all the transactions in the orphan block which have not been confirmed by another block will return to the unconfirmed transaction pool. Hence, it is not safe to accept a transaction when it is confirmed by only one block. It is the idea behind the rule of thumb of waiting for at least six confirmations before accepting a transaction.

Newly propagated blocks are validated by the Bitcoin's network. If the solution of the proof-of-work is incorrect or if any transaction included in the block has any issue, then the block is discarded. In order to work properly, the whole network must agree in what is allowed and what is not. Should one think that something should be allowed and accept it in their blocks, the remaining of the network will discard their newly propagated blocks. That is why Bitcoin's network is distributed and decentralized. Everything depends on the agreement of the network, or, precisely, the agreement of the owners of at least 50% of the hash power. Even if the remaining 49% disagrees, the 50% or more who agree will generate, on average, more blocks than the remaining of the network and their rules will prevail on the longest chain. If a disagreement between miners' rules happens, that is referred as either a hard-fork or a soft-fork (in general, a hard-fork relaxes the constraints, while the latter hardens them).

A practical example of a network disagreement is the increase of the block size. No group with more than 50% of the network's hash power agreed into increasing the maximum block size to increase the number of transactions confirmed by a block and thus increasing the network's capacity. Hence, the capacity remains the same, and the community has been discussing the issue in search of a consensus.

Bitcoin uses the blockchain technology to create a distributed ledger. It allows every new block to generate new bitcoins and also to collect the fees from the confirmed transactions within the block. Bitcoin's transactions have two main parts: (i) inputs, and (ii) outputs. Each transaction sends bitcoins from one or more input

addresses to one or more output addresses. In order to prove that one is the owner of the input bitcoins, one must digitally sign the transaction proving such ownership.

The digital signature scheme used by Bitcoin is based on a pair of private and public keys. The private key is used to sign the transaction, while the public key is used to check whether the signature is valid. Thus, the owner of some Bitcoin funds is, in fact, the owner of a pair of private and public keys. The private key must never be publicly published, as whoever has access to the private key is able to spend its funds. In other words, in order to protect their funds, the owners must protect the private key. If one loses their private key, unfortunately, access to their funds will be lost forever. The public key may be used to a proof-of-ownership, i.e., one may publish the public key with some message digitally signed by the private key, proving that he/she is the owner of the funds.

Bitcoins (BTCs) owned by someone are, in fact, unspent outputs in one or more transactions. For instance, one may have 6 BTCs spread between three transactions' unspent output: the first with 1 BTC, the second with 2 BTCs, and the third with 3 BTCs.

In a transaction, the inputs are pointers to other transactions' outputs (which they are spending). A transaction output may only be spent once and thus may not be partially spent. For instance, when one has 3 BTCs in one transaction output and would like to send 1 BTC to a friend, they have to create a transaction with one input spending the 3 BTCs and two outputs, one for the friend with 1 BTC and one's change with 2 BTCs.

Each transaction's output has a script that is executed by the miners to check whether one has or has not permission to spend that output. In other words, whether one has the ownership of that output. In order to execute these scripts, the miners also need some data. This data is given by the transaction which is spending the output.

The output's scripts usually checks whether the public key is valid and whether the digital signature was signed by the private key associated to that public key. Although there are only 3 commonly used scripts, one may create a custom script using the Bitcoin's script language [1].

The input contains the data which prove that the sender is the owner of the referred outputs, i.e., the input which must be accepted by the output scripts being spent. Usually, each input has the public key of the sender and a digital signature.

Those users accustomed to block explorers may have been misled by the transaction information that these websites provide. For example, suppose a miner receives a transaction with "one input

1 Your courageous author once tried to make a transaction with custom script to try to double spend a deposit to an exchange, only to learn through this intrepid adventure that Bitcoin allows only 3 script patterns and the others are treated as invalid.

from address A1". This "one input" actually consists of a pointer to a previous unspent output (i.e., there is no "input" address, as is displayed, but only a pointer). This pointer reference allows lookup to be executed in $O(1)$ time.

After lookup, the miner knows how many BTC tokens are available at that unspent output. But, in order to certify ownership of that output, the miner receives instructions in the form of a script with the rules that lead to the desired unspent output address (and this one is displayed as the input address by those websites). Because this process requires a digital signature, only the holder of the corresponding private key is able to sign such transaction.

Next, we will do a mathematical analysis of Bitcoin in order to better understand its minings properties, how a fork would affect the network and its security against attackers.

4

ANALYSIS OF BITCOIN

The primary objective of this chapter is to increase our understanding of the Bitcoin through mathematical tools.

4.1 HASH FUNCTION

Hash functions has been widely studied in computer science. In short, a hash function $h : \{0,1\}^\infty \to \{0,1\}^n$ has the following properties:

1. $x = y \Rightarrow h(x) = h(y)$
2. $h(x) \sim \mathcal{U}(0, 2^n - 1)$, where $\mathcal{U}$ is the uniform distribution, i.e., $\forall a \in [0, 2^n - 1], \mathbf{P}(h(x) = a) = \frac{1}{2^n}$

In other words, when two inputs are the same, they have the same output. But, when the inputs are different, their outputs are uniformly distributed. Clearly, the hash functions are surjective but not injective. They are not injective because the image of h has only 2^n elements and the domain has infinite elements. When $x \neq y$ and $h(x) = h(y)$, we say that x and y are a collision. A hash function is considered to be safe when it is unknown how to quickly find a collision of a given hash, i.e., one has to check all possible values until the correct one is found (known as the brute-force attack).

Bitcoin uses two hash functions: HASH-160 and HASH-256. The first has $n = 160$ and consists of the composition of *SHA-256* and *RIPEMD-160*. The latter has $n = 256$ and applies *SHA-256* twice. The first is used in transactions' scripts and the latter in the mining algorithm. For both hash functions, it is infeasible to run a brute-force attack because it would demand, on average, either 2^{160} or 2^{256} trials, and those would take a tremendous amount of time even for the fastest known processors.

For further information about hash functions, see Gilbert and Handschuh [47], Dobbertin et al. [39].

4.2 MINING ONE BLOCK

Let $\mathbb{B}$ be the set of Bitcoin blocks and $h : \mathbb{B} \to \{0,1\}^{256}$ be the Bitcoin *HASH-256* function. The mining process consists of finding $x \in \mathbb{B}$ such as $h(x) < A$, where A is a given threshold. The smaller the A, the harder to find a new block. In fact, $\mathbf{P}(h(x) < A) = \frac{A}{2^{256}}$.

Hence, in order to find a new block, one must try different inputs $(x_1, x_2, \ldots, x_k)$ until they find a solution, i.e., all attempts will fail

($h(x_i) \geqslant A$ for $i < k$) but the last ($h(x_k) < A$). The probability of finding a solution exactly in the k^{th} attempt follows a geometric distribution. Let X be the number of attempts until a success, then $\mathbf{P}(X = k) = (1-p)^{k-1}p$, where $p = \frac{A}{2^{256}}$. Also, we have $\mathbf{P}(X \leqslant k) = 1-(1-p)^k$. The average number of attempts is $\mathbf{E}(X) = 1/p$ and the variance is $\mathbf{V}(X) = \frac{1-p}{p^2}$.

In the Bitcoin's protocol, the given number A is adjusted so that the network would find a new a block every 10 minutes, on average. Suppose that the Bitcoin's network is able to calculate H hashes per second — H is the total hash rate of the network. The time required to find a solution would be $T = X/H$, and $\mathbf{E}(T) = \mathbf{E}(X)/H$ would be the average number of seconds to find a new block. So, the rule of finding a new block every 10 minutes ($\eta = 600$ seconds) — on average — leads to the following equation: $\mathbf{E}(T) = \eta = 600$. So, $\mathbf{E}(T) = \mathbf{E}(X)/H = \frac{1}{pH} = \eta = 600 \Rightarrow p = \frac{1}{\eta H}$. Finally, $\mathbf{E}(X) = \eta H$, $\mathbf{E}(T) = \eta$, $\mathbf{V}(X) = (\eta H)^2 - \eta H$, and $\mathbf{V}(T) = \eta^2 - \eta/H$.

The cumulative distribution function (CDF) of T is $\mathbf{P}(T \leqslant t) = \mathbf{P}(X/H \leqslant t) = \mathbf{P}(X \leqslant tH) = 1-(1-p)^{tH} = 1-\left(1-\frac{1}{\eta H}\right)^{tH}$. But, as the Bitcoin's network hash rate is really large, we may approximate the CDF of T by $\lim_{H\to\infty} \mathbf{P}(T \leqslant t) = 1-e^{-\frac{t}{\eta}}$, which is equal to the CDF of the exponential distribution with parameter $\lambda = \frac{1}{\eta}$.

Theorem 1. *When $H \to +\infty$, the time between blocks follows an exponential distribution with parameter $\lambda = \frac{1}{\eta}$, i.e., $\lim_{H\to+\infty} \mathbf{P}(T \leqslant t) = 1-e^{-\frac{t}{\eta}}$.*

Proof.

$$\begin{aligned}\mathbf{P}(T \leqslant t) &= 1-(1-p)^{tH} \\ &= 1-\left(1-\frac{1}{\eta H}\right)^{tH}\end{aligned}$$

Replacing $u = \eta H$,

$$\begin{aligned}\lim_{H\to+\infty} \mathbf{P}(T \leqslant t) &= \lim_{u\to+\infty} 1-\left(1-\frac{1}{u}\right)^{\frac{tu}{\eta}} \\ &= \lim_{u\to+\infty} 1-\left[\left(1-\frac{1}{u}\right)^{u}\right]^{\frac{t}{\eta}} \\ &= 1-(1/e)^{\frac{t}{\eta}} \\ &= 1-e^{-\frac{t}{\eta}}\end{aligned}$$

□

Now, we would like to understand from which value of H it is reasonable to assume that T follows an exponential distribution.

Theorem 2. $x > M \Rightarrow |(1+1/x)^x - e| < e/M$.

Proof. Let's use the classical inequality $\frac{x}{1+x} < \log(1+x) < x$ for $x > -1$. So, $\frac{1/x}{1+1/x} < \log(1+x) < 1/x$. Simplifying, $\frac{1/x}{1+1/x} = 1/(1+x)$. Thus, $1/(1+x) < \log(1+1/x) < 1/x \Rightarrow x/(1+x) < x\log(1+1/x) < 1$.

As $\log(1+\frac{1}{M}) > 0$ and $1 < 1+\log(1+\frac{1}{M})$.

$x > M \Rightarrow 1/x < 1/M \Rightarrow 1+1/x < 1+1/M \Rightarrow 1/(1+1/x) > 1/(1+1/M) \Rightarrow x/(1+x) > M/(1+M)$.

Again, $\log(1+x) < x \Rightarrow \log(1-1/M) < -1/M \Rightarrow 1+\log(1-1/M) < (M-1)/M < M/(1+M)$, since $(x-1)/x < x/(x+1)$.

Hence, $1+\log(1-1/M) < M/(1+M) < x/(1+x) < x\log(1+1/x)$, and $x\log(1+1/x) < 1 < 1+\log(1+\frac{1}{M})$.

Finally,

$$
\begin{aligned}
1+\log(1-1/M) < x\log(1+1/x) &< 1+\log(1+\frac{1}{M}) \\
e^{1+\log(1-1/M)} < e^{x\log(1+1/x)} &< e^{1+\log(1+\frac{1}{M})} \\
e\cdot e^{\log(1-1/M)} < e^{\log((1+1/x)^x)} &< e\cdot e^{\log(1+\frac{1}{M})} \\
e(1-1/M) < (1+1/x)^x &< e(1+\frac{1}{M}) \\
e-e/M < (1+1/x)^x &< e+e/M \\
-e/M < (1+1/x)^x - e &< e/M
\end{aligned}
$$

Therefore, $|(1+1/x)^x - e| < e/M$. □

We may consider H big enough to say that T follows an exponential distribution when $e/H < \epsilon$, where ϵ is the maximum approximation error. When $\epsilon = 10^{-6} \Rightarrow H > e\cdot 10^6$. So, when $H > 2.6$Mh/s, our approximation is good enough.

The symmetrical confidence interval with level α would be $[t_0, t_1]$, where $\mathbf{P}(t_0 < T < t_1) = 1-\alpha$, $\mathbf{P}(T < t_0) = \alpha/2$, and $\mathbf{P}(T > t_1) = \alpha/2$. These conditions give the following equations: $1-e^{-t_0/\eta} = \alpha/2$, and $e^{-t_1/\eta} = \alpha/2$. Solving these equations, we have $t_0 = -\eta\ln(1-\alpha/2)$, and $t_1 = -\eta\ln(\alpha/2)$.

For instance, if $\alpha = 10\%$, then $t_0 = 30.77$ and $t_1 = 1797.44$ (or $[0.51, 30.76]$ in minutes). Thus, 90% of the time the intervals between blocks are between 30 seconds and 30 minutes, with average of 10 minutes.

The fact that the time between blocks follows an exponential distribution with $\lambda = 1/\eta = p$H may be used to estimate the total network's hash rate (or a miner's hash rate). For further information, see [80].

4.3 MINING SEVERAL BLOCKS

Let $T_1, T_2, T_3, \ldots, T_n$ be the time to find the first block (T_1), then the time to find the second block (T_2), and so on. Let's analyze the dis-

tribution of $Y_n = \sum_{i=1}^{n} T_i$ which is the total time to find the next n blocks. As Y_n is the sum of random variables which follow an exponential distribution with same $\lambda = \frac{1}{\eta}$, then $Y_n \sim \text{Erlang}(n, \frac{1}{\eta})$. Thus, the CDF of Y would be $\mathbf{P}(Y_n < t) = 1 - \sum_{k=0}^{n-1} \frac{1}{k!} e^{-\lambda t} (\lambda t)^k$.

Many exchanges require at least six confirmations in order to accept a deposit in Bitcoin. So, for $n = 6$, $\mathbf{P}(Y_6 < 1 \text{ hour}) = \mathbf{P}(Y_6 < 3600) = 0.5543$, i.e., only 55% of the deposits will be accepted in one hour. The symmetrical confidence interval with $\alpha = 10\%$ is $[27, 105]$ in minutes. Thus, 90% of the times, it will take between 27 minutes and 1 hour and 45 minutes to have your deposit accepted — assuming that your transaction will be confirmed in the very next block. The pdf of Y_6 is shown in Figure 2, in which the 10% symmetrical confidence interval is shown in the white area. The average total time of six confirmations is $\mathbf{E}(Y_6) = 6 \cdot 600 = 3600 = 60$ minutes.

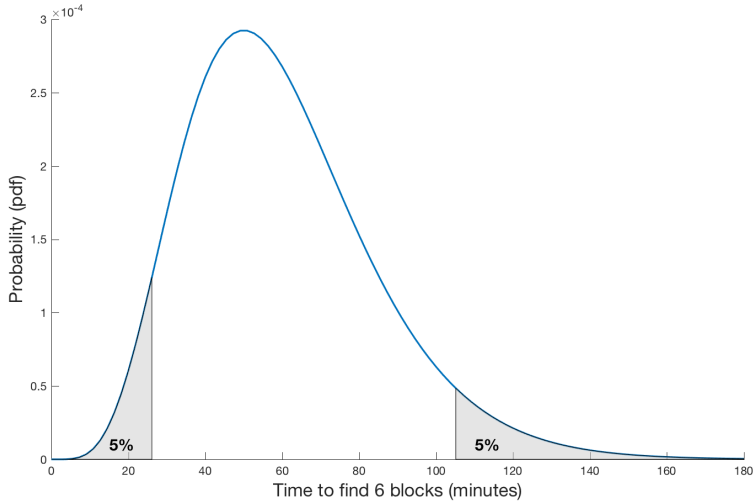


Figure 2: Probability density function of Y_6, i.e., probability of finding 6 blocks after time t. The shaded areas shows the lower 5% and upper 5% of the pdf.

4.4 MINING FOR A MINER

Let's analyze the probability of finding a new block for a miner who has α percent of the network's total hash rate. Let $T_\alpha = \frac{X}{\alpha H}$ be the time required for the miner to find a new block. As $T_\alpha = \left(\frac{1}{\alpha}\right) T$, when $H \to +\infty$, T_α also follows an exponential with parameter $\lambda_\alpha = \frac{\alpha}{\eta}$. Hence, we confirm the intuition that the miner with α percent of the network's total hash power will find α percent of the blocks.

Theorem 3. *When the miner with α percent of the network's total hash rate is part of the mining network, $\mathbf{P}(\text{next block is from } T_\alpha) = \alpha$.*

Proof.

$$\begin{aligned}\mathbf{P}(\text{next block is from } T_\alpha) &= P\left(T_\alpha = \min\{T_\alpha, T_{1-\alpha}\}\right)\\ &= \frac{\lambda_\alpha}{\lambda_\alpha + \lambda_{1-\alpha}}\\ &= \frac{\alpha/\eta}{\alpha/\eta + (1-\alpha)/\eta}\\ &= \frac{\alpha}{\alpha + 1 - \alpha}\\ &= \alpha.\end{aligned}$$

□

Theorem 4. *When one miner with α percent of the network's total hash rate multiplies their hash rate by m, the probability of this miner find the next block is multiplied by $\frac{m}{m\alpha+1-\alpha}$.*

Proof. When miners increase their hash rate, they also increase the network's total hash rate. Let H be the network's hash rate before the increase. Thus, the network's total hash rate after the increase is $H + (m-1)\alpha H = (1 - \alpha + m\alpha)H$. So,

$$\begin{aligned}\mathbf{P}(\text{next block is from } T_{m\alpha}) &= P\left(T_{m\alpha} = \min\{T_{m\alpha}, T_{1-\alpha}\}\right)\\ &= \frac{\lambda_{m\alpha}}{\lambda_{m\alpha} + \lambda_{1-\alpha}}\\ &= \frac{m\alpha/\eta}{m\alpha/\eta + (1-\alpha)/\eta}\\ &= \frac{m\alpha}{m\alpha + 1 - \alpha}\\ &= \alpha\left(\frac{m}{m\alpha + 1 - \alpha}\right).\end{aligned}$$

□

Corollary. *If one miner has a really tiny percent of the network's total hash rate, then multiplying their hash rate by m approximately multiplies their probability of finding the next block by m.*

Proof.

$$\lim_{\alpha\to 0} \mathbf{P}(\text{next block is from } T_{m\alpha}) = \lim_{\alpha\to 0} \frac{m}{m\alpha + 1 - \alpha} = m.$$

□

That way, it is not exactly correct to say that when one doubles their hash rate, their probability will double as well. It is only true for small miners.

4.5 ORPHAN BLOCKS

An orphan block would be created if a new block is found during the propagation time of a new block. Let α be the percentage of the total hash rate of the node which is outdated, and Δt the propagation time in seconds. Thus, $\mathbf{P}(\text{new orphan}) = \mathbf{P}(T < \Delta t) = 1 - e^{-\frac{\alpha \Delta t}{\eta}}$.

Bitcoin peer-to-peer network is a gossip network, where miners are semi-randomly connected to each other, and each miner sends all information it receives to all its peers. According to Decker and Wattenhofer [33], the average time for a new block propagate over the network is 12.6 seconds, while the 95% percentile is 40 seconds, which indicates a long-tail distribution. BitcoinStats [12] has measured the propagation time between 2013 and 2017. During 2017, the worst daily 90% percentile was 21 seconds. Notice that both results may not be contradictory because Bitcoin network is continuously evolving.

For instance, if a node has 10% of the total hash rate and it takes 30 seconds to receive the update, then $\mathbf{P}(\text{new orphan}) = 1 - e^{-\frac{0.1 \cdot 30}{600}} = 0.004987$, which is almost 0.5%. I would say that a node with 10% of the total hash rate would be well connected and it would take less time to receive the update, so, the probability would be even smaller than 0.5%.

Another important factor is that, as Bitcoin is open-source, miners are free to change the gossip algorithm, which leads to the network incentives. See Babaioff et al. [5] for an analysis of the incentives to miners forward new blocks and transactions in the network.

For further information about gossip algorithms, see Shah et al. [92].

4.6 ANALYSIS OF NETWORK'S HASH RATE CHANGE

The difficulty, given by the number A, is adjusted every 2016 blocks. As, $\mathbf{P}(13 \text{ days} < Y_{2016} < 15 \text{ days}) = \mathbf{P}(13 \cdot 24 \cdot 3600 < Y_{2016} < 14 \cdot 24 \cdot 3600) = 0.9986$, it is expected that the total time to find 2016 blocks will be between 13 and 15 days, assuming that the network's hash rate remains constant. If it takes less than the expected time, it means that the network's total hash rate has increased. While if it takes more than the expected time, it means that the network's total hash rate has decreased. So, let's analyze what happens when the network's hash rate changes significantly.

Let $H \cdot u(t)$ be the network's total hash rate over time. So, the number of hashes calculated in t seconds is $H \int_0^t u(t)dt$. Hence, $\mathbf{P}(T \leqslant t) = \mathbf{P}(X \leqslant H \int_0^t u(t)dt)$. When $H \to +\infty$, $\mathbf{P}(T \leqslant t) = 1 - e^{-\frac{1}{\eta} \int_0^t u(t)dt}$, and the pdf of T is $\frac{u(t)}{\eta} \cdot e^{-\frac{1}{\eta} \int_0^t u(t)dt}$.

Let's say that the network's total hash rate has suddenly multiplied by α. So, $u(t) = \alpha$, $\int_0^t u(t)dt = \alpha t$, and T also follows an exponential

distribution, but with $\lambda = \frac{\alpha}{\eta}$. Thus, $Y_n^\alpha = \sum_{i=1}^{n} T_i^\alpha \sim \text{Erlang}(n, \frac{\alpha}{\eta})$. Thus, $\mathbf{E}[Y_n^\alpha] = \frac{\mathbf{E}[Y_n]}{\alpha}$, i.e., the average total time required to find n blocks will be divided by α, while $\mathbf{V}[Y_n^\alpha] = \frac{\mathbf{V}[Y_n]}{\alpha^2}$ and the variance will be divided by α^2. Hence, on one hand, when the network's hash rate increases ($\alpha > 1$), the 2016 blocks will be found earlier. On the other hand, when the network's hash rate decreases ($\alpha < 1$), the 2016 blocks will be found later.

For example, if the network's total hash rate suddenly doubles ($\alpha = 2$), then $\mathbf{P}(6.5 \text{ days} < Y_{2016} < 7.5 \text{ days}) = 0.9986$, and the time required to find 2016 blocks halved. On the other side, if the network's total hash rate suddenly halves ($\alpha = 0.5$), then $\mathbf{P}(27 \text{ days} < Y_{2016} < 29 \text{ days}) = 0.9469$, and the time required to find 2016 blocks doubled. It is an important conclusion, since it shows that even if half of the network stops mining, it will only double the time to the next difficulty adjustment, i.e., the time between blocks will be 20 minutes for, at most, the next 29 days, at which point the adjustment will occur and everything will be back to the normal 10 minutes between blocks.

4.6.1 *Hash rate smoothly changing*

Let $u(t) = \frac{1+abx}{1+bx}$. It is an useful function because $u(0) = 1$ and $\lim_{t\to\infty} u(t) = a$. The bigger the b, the faster $u(t) \to a$. For example, if $a = 2$, it means H would be smoothly doubling. If $a = 0.5$, it means H would be smoothly halving.

It is easy to integrate $u(t)$ because $\frac{1+abx}{1+bx} = \frac{1-a}{1+bx} + a \Rightarrow \int_0^t u(x)dx = at + \frac{1-a}{b}\log(1+bt)$. So,

$$F_T(t) = 1 - (1+bt)^{\frac{\lambda(a-1)}{b}} e^{-\lambda a t}.$$

$$f_T(t) = \lambda\left(\frac{1+abt}{1+bt}\right)(1+bt)^{\frac{\lambda(a-1)}{b}} e^{-\lambda a t}.$$

Assuming that $n = \frac{\lambda(a-1)}{b}$ is integer, we have:

$$F_T(t) = 1 - (1+bt)^n e^{-\lambda a t}$$

Thus,

$$\begin{aligned}
\mathcal{L}\{F_T(t)\} &= \mathcal{L}\{1 - (1+bt)^n e^{-\lambda a t}\} \\
&= \mathcal{L}\{1\} - \mathcal{L}\{(1+bt)^n e^{-\lambda a t}\} \qquad (\mathcal{L} \text{ is a linear operator}) \\
&= \frac{1}{s} - \mathcal{L}\{(1+bt)^n e^{-\lambda a t}\} \\
&= \frac{1}{s} - \sum_{k=0}^{n} \binom{n}{k} b^k \mathcal{L}\{t^k e^{-\lambda a t}\} \\
&= \frac{1}{s} - \sum_{k=0}^{n} \binom{n}{k} b^k \frac{k!}{(s+\lambda a)^{k+1}}
\end{aligned}$$

Hence, as $\mathcal{L}\{f_T(t)\} = s\mathcal{L}\{F_T(t)\}$,

$$\mathcal{L}\{f_T(t)\} = 1 - \sum_{k=0}^{n} \binom{n}{k} \frac{s b^k k!}{(s+\lambda a)^{k+1}}$$

Then,

$$\begin{aligned}
\frac{d}{ds}\mathcal{L}\{f_T(t)\} &= -\sum_{k=0}^{n} \binom{n}{k} b^k k! \frac{d}{ds} \frac{s}{(s+\lambda a)^{k+1}} \\
&= -\sum_{k=0}^{n} \binom{n}{k} b^k k! \left[\frac{1}{(s+a\lambda)^{k+1}} - \frac{s(k+1)}{(s+a\lambda)^{k+1}} \right] \\
\frac{d}{ds}\mathcal{L}\{f_T(t)\}|_{s=0} &= -\sum_{k=0}^{n} \binom{n}{k} b^k k! \frac{1}{(\lambda a)^{k+1}} \\
&= -\frac{1}{a\lambda} \sum_{k=0}^{n} \binom{n}{k} k! \left(\frac{b}{\lambda a}\right)^k \\
&= -\frac{1}{a\lambda} \sum_{k=0}^{n} \frac{n!}{(n-k)!} \left(\frac{b}{\lambda a}\right)^k \\
&= -\frac{1}{a\lambda} \left[n! \sum_{k=0}^{n} \frac{1}{(n-k)!} \left(\frac{b}{\lambda a}\right)^k \right] \\
&= -\frac{1}{a\lambda} \left[n! \sum_{k=0}^{n} \frac{1}{k!} \left(\frac{b}{\lambda a}\right)^{n-k} \right] \qquad (k \to n-k) \\
&= -\frac{1}{a\lambda} \left[n! \left(\frac{b}{\lambda a}\right)^n \sum_{k=0}^{n} \frac{1}{k!} \left(\frac{b}{\lambda a}\right)^{-k} \right] \\
&= -\frac{1}{a\lambda} \left[n! \left(\frac{b}{\lambda a}\right)^n \sum_{k=0}^{n} \frac{1}{k!} \left(\frac{\lambda a}{b}\right)^k \right]
\end{aligned}$$

Finally, as $\mathbf{E}[T] = -\mathcal{L}\{f_T(t)\}|_{s=0}$,

$$\mathbf{E}[T] = \frac{1}{\lambda a} \left[n! \left(\frac{b}{\lambda a}\right)^n \sum_{k=0}^{n} \frac{1}{k!} \left(\frac{\lambda a}{b}\right)^k \right], \text{ where } n = \frac{\lambda(a-1)}{b}$$

Let's check this equation for already known scenarios. When $a = 1$, then $n = 0$ and $\mathbf{E}[T] = 1/\lambda$. When $b \to +\infty$, it reduces to the case in which the hash rate is multiplied by a, which we have already studied. In fact, $n \to 0$, $u(t) \to a$, and $\mathbf{E}[T] = \frac{1}{\lambda a}$.

Theorem 5.

$$a > 1 \textit{ and } x > M \Rightarrow \left| \frac{1+abx}{1+bx} - a \right| < \frac{a-1}{1+bM}$$

Proof. $x > M \Rightarrow \frac{1}{1+bx} < \frac{1}{1+bM}$. As $1-a < 0$, $\frac{1-a}{1+bx} > \frac{1-a}{1+bM}$. Thus, $\frac{1-a}{1+bM} < \frac{1-a}{1+bx} + a - a = \frac{1+abx}{1+bx} - a < 0 < \frac{a-1}{1+bM}$. Hence, $-\frac{a-1}{1+bM} < \frac{1+abx}{1+bx} - a < \frac{a-1}{1+bM}$. □

For instance, if we would like to know the impact of smoothly double the hash rate in the next week, then the parameters would be $\lambda = 1/600$, $a = 2$, $M = 1\,\text{week} = 3600 \cdot 24 \cdot 7 = 604,800$, b can be calculated using $\epsilon = \frac{a-1}{1+bM} < 0.01 \Leftarrow b > 0.000163690 \Leftarrow n < 10.1818$. So, for $n = 10$, then $b = 0.000166666$ and $\epsilon = 0.009823 < 0.01$, as expected. Finally, $\mathbf{E}[T] = 557.65$. In other words, during the next week, the average time between blocks will be 9 minutes and 17 seconds, instead of the normal 10 minutes. If the hash rate had suddenly doubled, the average time between blocks would be 5 minutes.

4.6.2 *Piecewise linear model of hash rate change*

Let's analyze what would happen if the network's hash rate is growing linearly with angular coefficient a^2, i.e., $u(a, b, t) = a^2 t + b$. Thus, $\mathbf{P}(T \leqslant t) = 1 - e^{-\frac{bt + a^2t^2/2}{\eta}}$.

It is well known that $\mathbf{E}(T) = \int_0^\infty 1 - \mathbf{P}(T \leqslant t)dt$. Thus, replacing $y = \frac{a^2t+b}{a\sqrt{2\eta}}$, and using the fact that $\int_0^\infty e^{-x^2}dx = \frac{\sqrt{\pi}}{2}\operatorname{erf}(x)$, we have:

$$\begin{aligned}
\mathbf{E}(T)|_{t_1}^{t_2} &= \int_{t_1}^{t_2} \exp\left(-\frac{bt + a^2t^2/2}{\eta}\right) dt \\
&= \frac{\sqrt{2\eta}}{a} \exp\left(\frac{b^2}{2a^2\eta}\right) \int_{y_1}^{y_2} \exp(-y^2)dy \\
&= \frac{\sqrt{2\eta}}{a} \exp\left(\frac{b^2}{2a^2\eta}\right) \frac{\sqrt{\pi}}{2}[\operatorname{erf}(y_1) - \operatorname{erf}(y_2)] \\
&= \frac{\sqrt{2\pi\eta}}{2a} \exp\left(\frac{b^2}{2a^2\eta}\right) [\operatorname{erf}(y_2) - \operatorname{erf}(y_1)] \qquad (1)
\end{aligned}$$

Where $y_1 = \frac{a^2t_1+b}{a\sqrt{2\eta}}$ and $y_2 = \frac{a^2t_2+b}{a\sqrt{2\eta}}$.

Thus, $\mathbf{E}(T) = \mathbf{E}(T)|_0^\infty$. When $t_1 = 0 \Rightarrow y_1 = \frac{b^2}{2\sqrt{2\eta}}$ and $t_2 \to \infty \Rightarrow y_2 \to \infty \Rightarrow \operatorname{erf}(y_2) = 1$, then:

$$\mathbf{E}(T) = \frac{\sqrt{2\pi\eta}}{2a} \exp\left(\frac{b^2}{2a^2\eta}\right) \left[1 - \operatorname{erf}\left(\frac{1}{a\sqrt{2\eta}}\right)\right]$$

The function $\mathbf{E}(T)|_{t_1}^{t_2}$ may be used to a piecewise linear analysis of any hash rate change. Let's analyze the hash doubling in one week. Then, $u(1\text{week}) = u(604800) = 2$, thus $a^2 = \frac{1}{604800} \Rightarrow a = \frac{1}{120\sqrt{42}}$. Let's sample the interval $[0, 1\text{week}]$ every hour, i.e., (t_0, t_1, t_2, ..., t_168), where $t_i = i \cdot 1\text{hour} = i \cdot 3600$.

For each point t_i, let $g_i(t) = [H(t - t_i) - H(t - 604800)]u(t) + 2H(t - 604800)$, then $\mathbf{E}(T) = \mathbf{E}(T)|_{t_i}^{t_{i+1}} + 1 - e^{\frac{t}{\eta}}$. The result is presented in Figure .

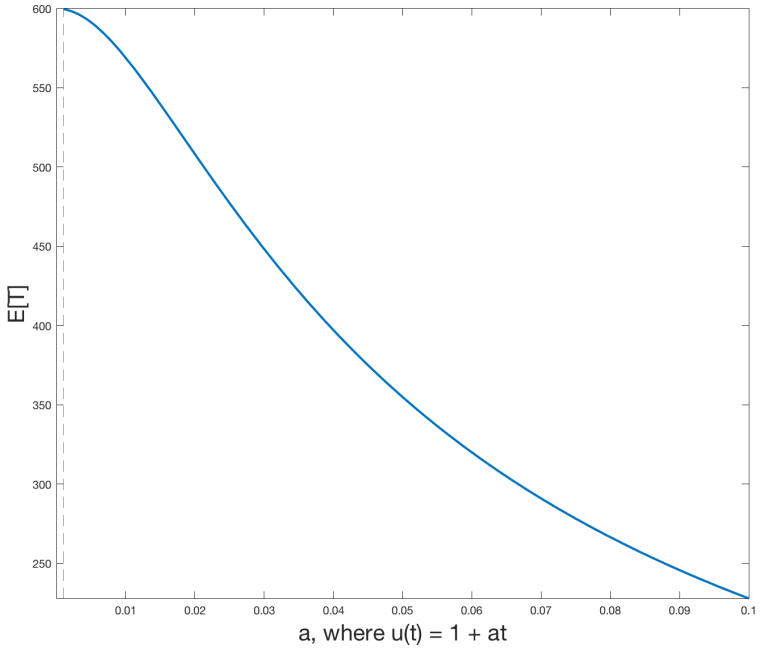


Figure 3: $\mathbf{E}[T]$ when H increases linearly with $u(t) = 1 + at$.

4.7 ATTACK IN THE BITCOIN NETWORK

Karame et al. [56]

There are many possible ways to attack the Bitcoin (REF). In this section, we are interested in a particular attack: the double spending attack.

In the double spending attack, the attacker's send some funds to the victim, let's say a merchant. They wait for k confirmations of the transaction, and the victim delivers the good or the service to the attacker. Then, the attacker mine enough blocks with a conflicting transaction, double spending the funds which was sent to the victim. If the attacker is successful, the original transaction will be *erased* and the victim will be left with no funds at all. In order to be successful, the attacker must propagate more blocks than the network in the same period, propagating a chain longer than the main chain. Hence, we would like to understand what the odds are that the attacker will be successful. This attack was originally discussed by Nakamoto [73].

In order to maximize their odds, the attacker must start to mine the new blocks as soon as they send the funds to the victim. In this moment, it starts to mine in the head of the blockchain, just like the rest of the network. So, in the beginning, the attacker and the network are in exactly the same point.

Let βH be the hash rate of the attackers, and γH be the network's hash rate without the attackers. Thus, when $H \to +\infty$, we already

know that $T_{attackers}$ and $T_{network}$ follow exponential distributions with parameters $\lambda_{attacker} = \frac{\beta}{\eta}$ and $\lambda_{network} = \frac{\gamma}{\eta}$, respectively.

As [73] has done, we will also model the attack using the Gambler's Ruin. In this game, a gambler wins \$1 at each round, with probability p, and loses \$1, with probability $1-p$. The rounds are independent. The gambler starts with \$$k$ plays continuously until he either accumulates a target amount of \$$m$, or loses all his money. Let $\rho = \frac{1-p}{p}$, then the probability of losing his fortune is:

$$\mathbf{P}(\text{losing his fortune}) = \begin{cases} \frac{\rho^k - \rho^m}{1-\rho^m}, \text{ if } \rho \neq 1, \\ \frac{m-k}{m}, \text{ if } \rho = 1. \end{cases}$$

When $m \to +\infty$,

$$\mathbf{P}(\text{losing his fortune}) = \begin{cases} \rho^k, \text{ if } \rho < 1, \\ 1, \text{ if } \rho \geqslant 1. \end{cases}$$

The gambler winning \$1 is the same as the network finding a new block, the gambler losing \$1 is the same as the attacker finding a new block. The initial \$$k$ is the same as the number of blocks the attacker is behind the network. Thus, the gambler loses his fortune is the same as the attacker successfully finds k or more blocks than the network, i.e., losing his fortune means that the attack was successful.

In our case, $p = \frac{\lambda_{network}}{\lambda_{network}+\lambda_{attacker}} = \frac{\gamma}{\beta+\gamma}$, thus $\rho = \frac{\beta}{\gamma}$. Hence, $\rho < 1 \Leftrightarrow \beta < \gamma$.

Suppose that the attacker is mining with the network. Suddently, he stops mining with the network and starts attacking, i.e., starts to mine in another chain. In this scenario, since the attacker's hash rate is not mining with the network anymore, $\gamma = 1 - \beta$. Thus, $\beta < \gamma \Rightarrow \beta < 0.5 \Leftrightarrow \rho < 1$. Here comes the conclusion that, if the attacker has 50% or more of the network's hash rate, then his attack will be certainly successful. We got exactly the same equations and conclusions as [73].

But this scenario seems not to be the optimal attack, because the attacker has waited k confirmations before starting the attack. A better approach would be to start attacking just after propagating the transaction. In this case, our previous model is not good, because even if the attacker have found more blocks than the network, he cannot propagate those blocks before the network has found k confirmations. So, we have to model the probabilities before the network has found the k block. Then, if the attacker has more blocks than the network, he has successfully attacked. Otherwise, we return to the previous model, in which the attacker must still find more blocks.

Theorem 6. *Assuming that the attacker starts the attack just after publishing the transaction, the probability of the attacker has already found*

exactly s *blocks while it waits the network to find* k *blocks is* $\mathbf{P}(S = s) = \binom{k+s-1}{s}(1-p)^s p^k$.

Proof. The attacker must find exactly s blocks while the network must find exactly k blocks. It is as they would be walking the grid from the point $(0,0)$ to (s,k), where it is only allowed to go up or right, like in Figure 4. When the attacker finds a block, it would be a movement to the right. When the network finds a block, it would be an upward movement. No matter the order which the blocks are found, all the paths occur with probability $(1-p)^s p^k$.

The walking ends when $(\cdot, k)$ is reached, i.e., when the network finds k blocks, regardless of how many blocks the attacker has found – i.e., it is not allowed to walk above the line $(\cdot, k)$. Thus, the number of paths between $(0,0)$ and (s,k) moving only upward or to the right, without going into the line $(\cdot, k)$ is exactly the number of paths between $(0,0)$ and $(s,k-1)$, which is equal to the number of permutations of the sequence $(u,u,\ldots,u,r,r,\ldots,r)$ in which there are s movements to the right (r) and $k-1$ upward movements (u). This number of permutations is $\frac{(k-1+s)!}{s!(k-1)!} = \binom{(k-1)+(s)}{s}$ because there are s repetitions of the element r and $k-1$ repetitions of the element u.

Finally, the probability is $\binom{k+s-1}{s}(1-p)^s p^k$.

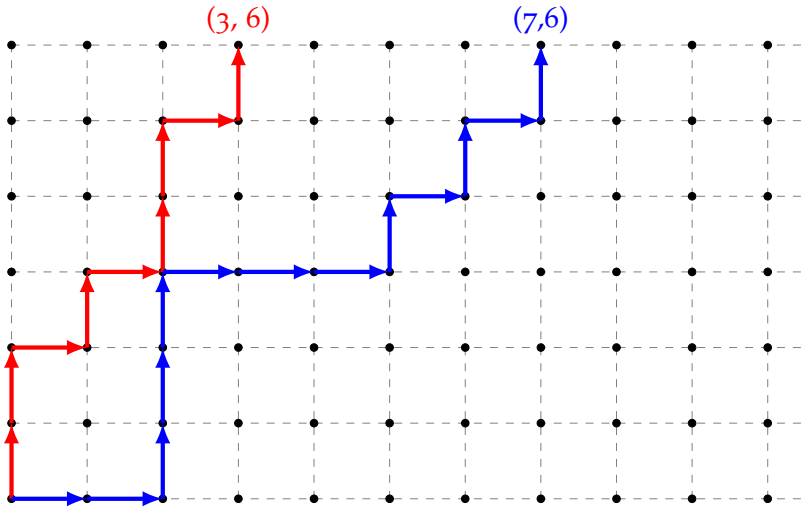


Figure 4: Both the attacker and the network are mining. Each step up is a new block found by the network with probability p. Each step right is a new block found by the attacker with probability $1-p$. It ends when the network finds k blocks — in this example, $k = 6$. The red path has probability $p^6(1-p)^3$, while the blue path has probability $p^6(1-p)^7$. Notice that the blue path is a successfull attack, because the attacker has found more blocks than the network. In the red path, the attacker still have to catch up 3 blocks to have a successful attack, which happens with probability ρ^3, if $p < 0.5$.

□

Assuming that the attacker starts mining just after publishing the victim's transaction, the probability of the attacker will have found more than k blocks while it waits the network to find k blocks is $\mathbf{P}(S \geqslant k) = \sum_{s=k}^{\infty} \binom{k+s-1}{s}(1-p)^s p^k$.

Theorem 7.

$$\mathbf{P}(S \geqslant k) = 1 - \sum_{s=0}^{k-1} \binom{k+s-1}{s}(1-p)^s p^k.$$

Proof. Let's use the following identity:

$$\frac{1}{(1-z)^{a+1}} = \sum_{i=0}^{\infty} \binom{i+a}{i} z^i, \text{ for } |z| < 1$$

Thus, replacing $z = 1-p$, $i = s$, and $a = k-1$, we have:

$$\frac{1}{p^k} = \sum_{s=0}^{\infty} \binom{s+k-1}{s}(1-p)^s$$

$$1 = \sum_{s=0}^{\infty} \binom{s+k-1}{s}(1-p)^s p^k.$$

Now, just split $\sum_{s=0}^{\infty} = \sum_{s=0}^{k-1} + \sum_{s=k}^{\infty}$ and it is done. □

Using this last theorem, we moved from an infinity sum to a finity sum.

Theorem 8. *Let* $p = \frac{\gamma}{\beta+\gamma}$.

$$\mathbf{P}(\textit{successful attack}) = \begin{cases} 1 - \sum_{s=0}^{k-1} \binom{k+s-1}{s} \left((1-p)^s p^k - (1-p)^k p^s\right), p \geqslant 0.5 \\ 1, p < 0.5. \end{cases}$$

Proof.

$$\mathbf{P}(\text{successful attack}) = \mathbf{P}(S \geqslant k) + \sum_{i=0}^{k-1} \mathbf{P}(s = i)\rho^{k-i}$$

□

For $k = 6$, $p = 0.9$, $\mathbf{P}(\text{successful attack}) = 0.0005914121600000266$.
For $k = 6$, $p = 0.7$, $\mathbf{P}(\text{successful attack}) = 0.15644958192000014$.

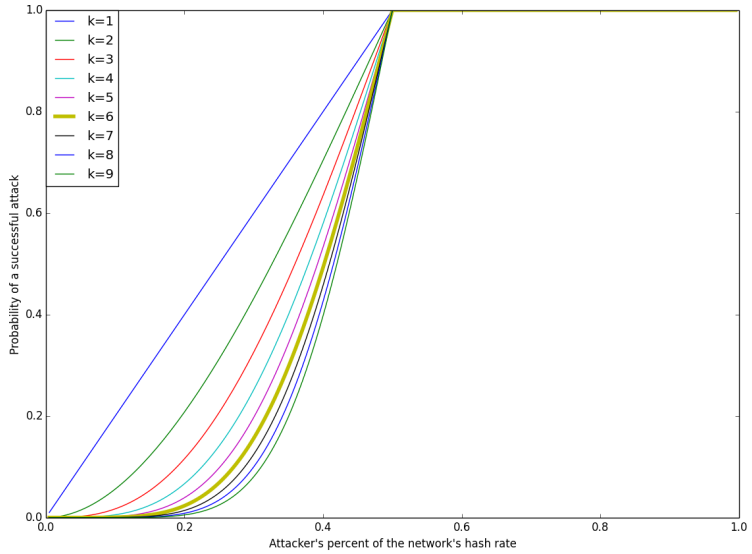


Figure 5: Probability of a successful attack according to the network's hash rate of the attacker (β).

4.8 CONFIRMATION TIME AND NETWORK CAPACITY

Let's say that when a new transaction is propagated it is enqueued in the unconfirmed transaction queue. Then, when a new block is found, some of these transactions in the queue are confirmed. We are interested in some measures of the queue, like the expected time to confirm a transaction and the queue's length.

Let's assume that all transactions have exactly the same size S and pay exactly the same fee. If the Bitcoin block's maximum size is M, there would be room for $s = \lfloor M/S \rfloor$ transactions in each block. Using the results from Bailey [7], we have found that $\pi_n = \frac{z_s - 1}{z_s^{n+1}}$ is the probability of having n unconfirmed transactions in the pool subjected to $s > m$, where $m = \frac{\lambda_{TX}}{\lambda_{blocks}}$ and z_s is the single root of the polynomial $z^s(1 + m(1 - z)) - 1$ with $|z_s| > 1$. The average size of the unconfirmed transaction pool is $\mathbf{E}(\pi) = \frac{1}{z_s - 1}$. Thus, $s > m \Rightarrow \lambda_{TX} < s\lambda_{blocks}$. The probabilities form a simple geometric series, and so they are exponentially decreasing. We may interpret it as a stable system, i.e., the unconfirmed transactions pool size is finite. In the Bitcoin's network, $\lambda_{blocks} = 1/\eta$ and the average number of transactions per block is $s = 2,250$, so, this solution is valid when $\lambda_{TX} < 2,250/600 = 3.75$. Hence, 3.75 is the maximum number of new transactions per second that the Bitcoin's network may handle. When $\lambda_{TX} > 3.75$, the unconfirmed transaction pool starts to grow indefinitely.

The average waiting time of a transaction to be confirmed, when $m < s$, is $\mathbf{E}(w) = \frac{1}{\lambda_{TX}(z_s - 1)}$.

When $m \ll s$, $z_s \to 1 + 1/m$. So, the average number of unconfirmed transactions is $\mathbf{E}(\pi) \to m$ and the average waiting time $\mathbf{E}(w) \to \frac{1}{\lambda_{\text{blocks}}} = \eta = 600$ seconds. In the Bitcoin's network, $s \gg m \Rightarrow \lambda_{TX} \ll 3.75$.

When $\lambda_{TX} \to s$, $z_s \to 1$. So, $\mathbf{E}(\pi) \to +\infty$.

Finally, we conclude that the Bitcoin's network capacity is $\lambda_{\text{blocks}} s = s/\eta = s/600$ transactions per second, where s is the average number of transactions per block.

4.9 FORK ANALYSIS

When a disagreement between miners' rules happens, that is referred as either a hard-fork or a soft-fork. It is said to be a soft-fork when the rules are backward compatible, and a hard-fork when the rules are not backward compatible. In general, a hard-fork relaxes the constraints, while the latter hardens them.

Suppose that the miners' are split in two groups, G_1 and G_2, with different rules. Let's say H_1 and H_2 are their hash rate, respectively. We have two different scenarios to analyze: (i) when neither of them accepts other's blocks; and (ii) when G_2 accepts G_1's blocks, but not the other way around.

Scenario (i) is easy to analyze, because the network would just split and, after a while, both difficulties will be adjusted. Then, they will be just like two different Bitcoin networks.

Scenario (ii) is more trick. As G_2 accepts G_1 blocks, their hashrate will matter. If $H_1 > H_2$, then G_2 will frequently skip their blockchain, because G_1's blockchain will be longer most of the time. If $H_1 < H_2$, then G_2 may have its own blockchain after a while — a true fork. But what would happen if H_1 keeps increasing and eventually gets larger than H_2? It if happens for sufficient time, the whole G_2's blockchain may be discarded in order to move to G_1's blockchain when G_1's gets longer.

Theorem 9. *When* $H_1 > H_2$, *the*

5

DAG MODEL

DAG model proposes a whole different approach to confirmations. It proposes that there is no need for a block to confirm transactions, as transactions can confirm themselves. Here, each transaction has its own proof-of-work, named weight, and they must confirm two other, previous, transactions. Hence, each transaction has an accumulated weight, which is the accumulated proof-of-work that has confirmed it so far. In this sense, instead of a chain of blocks, the transactions and their confirmations form a directed and acyclic graph, as in Fig. 6.

Like the Blockchain, the DAG model is another technology to store immutable data and may be the underlying technology to different applications, such as cryptocurrencies, digital contracts (Ethereum-like), digital notaries, and so forth. The main question which this work proposes to answer is: Is it reliable to use DAG model? In which conditions?

The transactions may be either confirmed or unconfirmed. The confirmed transactions have been already confirmed by at least one more transaction. It does not mean they are already irreversible and protected against a double spend attack — it just means at least one transaction has done some work to confirm it. The unconfirmed transactions are called tips and they are eager to be confirmed. Usually a new transaction selects two tips to confirm, but this rule may not be followed.

Transactions may have any format, including Bitcoin-like transactions—with inputs, outputs, and scripts. But it also may be completely different, just like in the cryptocurrency Iota [84]. This work will not discuss details of the transactions' format, because we understand that it does not affect exactly the network scalability and security.

The accumulated weight of a transaction A is the sum of all weights of the transactions which confirm A, including A itself, i.e., $w_A + \sum_{A \rightsquigarrow P} w_P$. For example, in Fig. 6, the accumulated weight of transaction 3 is the sum of the weights of the transactions 5, 6, 7, and 8. The accumulated weight may be interpreted as how hard it is to rollback a transaction. It is analogous to the number of confirmations of a block in Bitcoin.

The score of a transaction A is the sum of all weights of the transactions which are being confirmed by A, including A itself, i.e., $w_A + \sum_{P \rightsquigarrow A} w_P$. For example, in Fig. 6, the score of transaction 3 is the sum of the weights of the transactions 1 and 2. It is a measure of

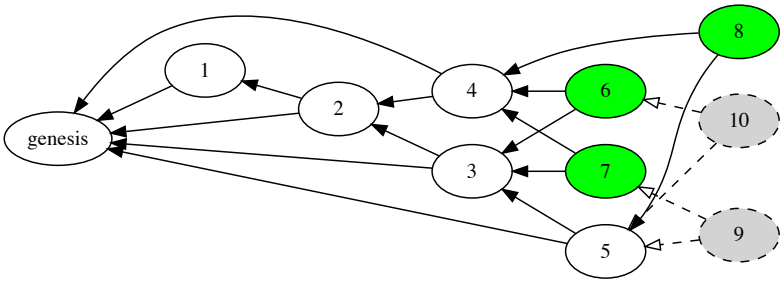


Figure 6: White nodes represent transactions that have been confirmed at least once. Green circles represent unconfirmed transactions (tips). Gray and dashed nodes are the transactions currently solving the proof-of-work in order to be propagated.

how much proof-of-work has been done before confirming this transaction. It may indicate whether the confirmed transactions have received enough attention (and hashpower) or whether it may have received tangential attention.

The height of a transaction A is the length of the longest path from transaction A to the genesis transaction. For example, in Fig. 6, the height of transaction 5 is four ($5 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow$ genesis). It may be interpreted as the "'age" of the transaction. The lower the height, the older the transaction.

The depth of a transaction A is the length of the longest path in the inverted graph from transaction A to any unconfirmed transaction (tip). For example, in Fig. 6, the depth of transaction 2 is three ($2 \rightarrow 3 \rightarrow 5 \rightarrow 8$). It is the opposite of the height. It may be interpreted as the youth of the transaction. The lower the depth, the younger the transaction. When a new transaction is confirming two transactions with high depth, it is referred as lazy transaction.

An important factor of DAG model is how the new transactions choose which transactions they will confirm. There are several possible approaches, such as randomly selecting two of the unconfirmed transactions (tips). In Fig 6, the reader may have noticed that transaction 8 will confirm transaction 4, which has already been confirmed by transaction 7. It may be on purpose, or maybe transaction 4 was unconfirmed when it was chosen, but it got confirmed during the calculation of the proof-of-work or the network propagation of the transaction. The selection algorithm seems to be important to protect the network against double spend attacks.

The higher the volume of new transactions, the more unconfirmed transactions will appear. In Fig. 7, the reader can notice that the number of new transactions was increased for a while, and then

decreased back to the original value. The DAG has behaved well when exposed to a high load scenario, since it reduced the number of tips to only three. It is like a moving swarm which gets wider when the number of new transactions increases and gets thinner when the number of new transactions decreases.

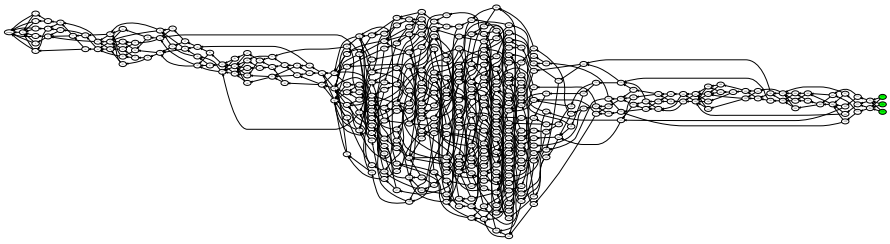


Figure 7: Suddenly the number of transactions per second increases and the width of the swarm grows. After a while, the number of transactions per second decreases and the width of the swarm shrinks.

Conflicting transactions may happen when two or more transactions try to spend the "same money" — or, in the Bitcoin's transaction format, try to spend the same output. In this case, the network must choose which of the transactions will be accepted and the other one will be invalidated, even when both have already been confirmed. In fact, when one transaction is invalidated, the whole sub-DAG which confirms it is also invalidated. In this case, it may happen to reverse some transactions.

Intuitively, when there is a conflict, the network should accept the transaction which has greater accumulated weight, invalidating the others (see Fig. 8). But it may be not enough to prevent a nuclear submarine attack.

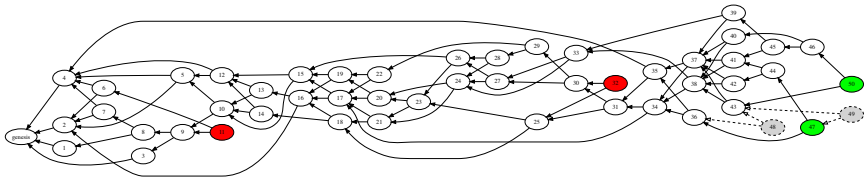


Figure 8: The red nodes are transactions which had some conflict with previous transaction and were invalidated by the network. Notice that none of them have been confirmed.

5.1 NUCLEAR SUBMARINE ATTACK

The nuclear submarine attack, also known as the parasite chain attack, is when the attacker generates a separate DAG (or a side DAG), with many transactions and a lot of proof-of-work. This side DAG is off the network, i.e., its transactions have not been propagated. Then, at a convenient moment, the attacker suddenly

propagates these transactions. The whole network needs to decide how to handle these transactions.

If the transactions have no conflict with any transaction of the main DAG, i.e., there is no transaction spending the "same money", then it seems easy to handle the transactions. But, as it is an attack, probably there will be some conflicts, and it is not easy to choose which transaction should be invalidated. As the attacker has been generating a separate DAG, the conflicting transaction may have an accumulated weight similar or greater than the transaction in the main DAG. Thus, using only the accumulated weight may not be enough to prevent this attack.

For example, the attacker may generate a transaction which transfers all their funds to another address. Then, they start to generate many new transactions which confirms themselves and even confirms some of the transactions in the main DAG. But none of these transactions are propagated to the network. Then, the attacker buys something in the real world, pays with cryptocurrency, and wait until the payment gets the accumulated weight demanded by the merchant. Finally, the attacker suddenly propagates all the transactions to the network in a small window of time. This may cause the network to accept the attacker's original transaction instead of the one used to pay the merchant. Hence, the merchant transaction is invalidated, and the double spend attack has succeeded.

5.2 PROPOSAL: QUESTIONS TO BE EXPLORED

Given these preliminaries, these are some questions with which we will be concerned.

In this paper, we will analyze the network scaling and security. How a DAG network scales, simulating different loads and measuring the bandwidth, computational effort, and storage space necessary to handle all the transactions in time. What is the minimum transaction's aggregated weight which it may be considered unlikely to be reversed?

We are interested in how the rate of new transactions affects how long it takes to a new transaction to be confirmed for the first time. We had already run some simulations under normal load (Fig. 9) and high load (Fig. 10).

Besides the time to the first confirmation, it is also important to measure how long it takes to a new transaction reach a specific accumulated weight. It is useful for exchanges and merchants to set their minimum requirements. Bitcoin's exchanges and merchants usually requires a minimum of 6 confirmations blocks.

As the DAG network is distributed, its users may use different parameters — they cannot violate the network's rules, but they may

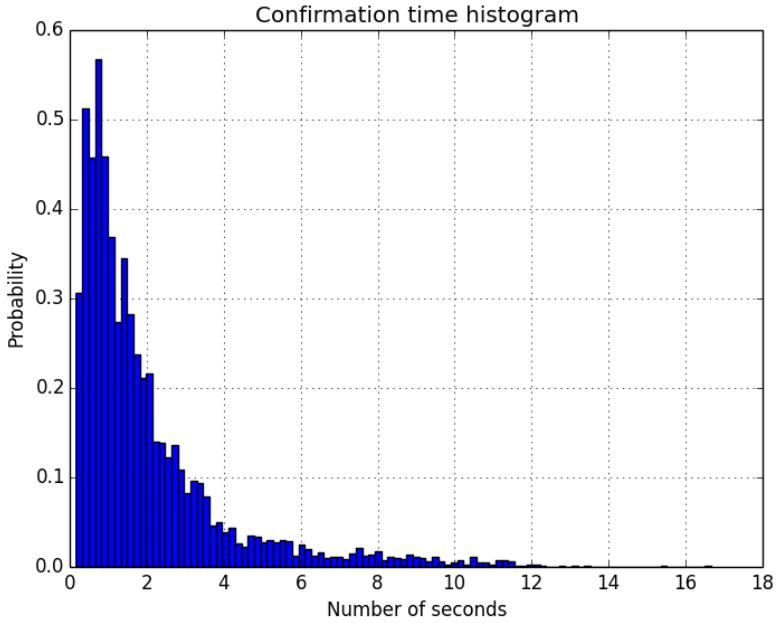


Figure 9: Histogram of how long has been a transaction waiting until its first confirmation. It was a simulation of 15 minutes with new transactions rate changing between 1 and 15 tx/s.

use different tip selection algorithms, for instance. We would like to explore how different users' parameters may affect the network, how much is necessary to cause either a hard-fork or a soft-fork.

[84] suggests that constraining transactions' weight in a range would both prevent spam, because it would be necessary a minimum work to propagate a new transaction, and attacks, because an attacker would not be allowed to propagate a transaction with very high weight. We agree with the spam argument, and would like how a minimum weight would affect mobile devices's new transactions. We partially agree with the upper bound, because an attacker would be able to generate many transaction with lower weight. We will analyze if constraining transactions' weight would be effective against nuclear submarine attacks.

We have another suggestion to prevent nuclear submarine attacks: set a maximum depth for the transactions confirmed by a new transaction. So, new transactions would have to confirming newer transactions instead of old transactions, and a nuclear submarine attack would be controlled because the attacker would be not be able to create a large separate DAG. But the maximum depth rule would possibly also affect transaction created by low (hash)power devices, because it would take a longer time to solve the proof-of-work and, if the confirmations are going fast, the transaction would possibly be invalidated. This raises another important question: what would be an optimal maximum depth allowed?

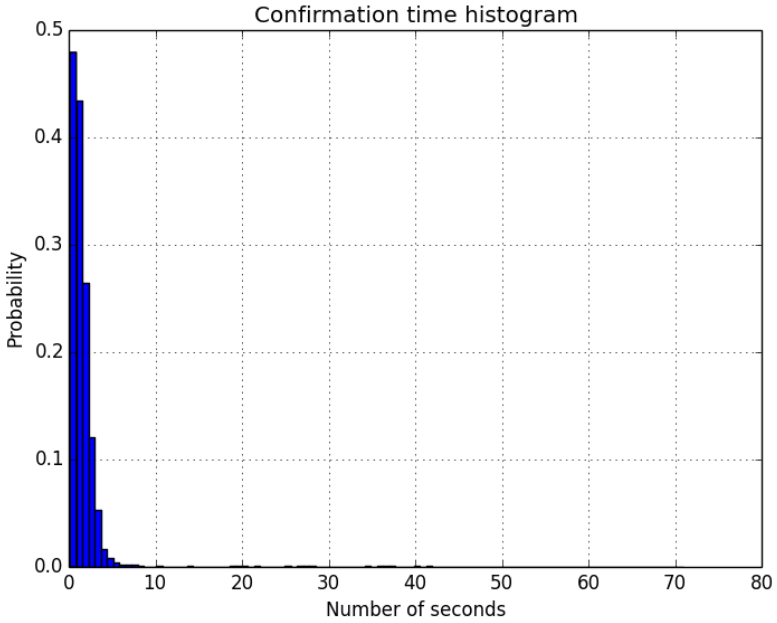


Figure 10: Histogram of how long has been a transaction waiting until its first confirmation. It was a simulation of 5 minutes with new transactions rate of 50 tx/s, i.e., very high load.

If we would like to include fee transactions, how would it be distributed between confirmations? As transactions may have multiple direct confirmations, it is not obvious who would receive the transaction fee. An intuitive suggestion would be to give the fee to the first confirmation with a minimum weight, but it might have some propagation problems, because it takes a while to propagate a transaction in the network and there would be conflict.

In order to solve the fee distribution problem, we will simulate how the network would behave if we include blocks. These blocks would be like Bitcoin's, with an adjustable proof-of-work according to the network's hashpower. They would receive the fees from all confirmed transactions which had not been confirmed by any block before, and they would be able to generate new coins. These blocks would also help solving other problems.

6 METHODOLOGY

The methodology we have been using is computer simulation. Through the simulation of many scenarios of DAG model with different parameters, we will understand how the network behaves in complex scenarios, mainly when the load is suddenly increased, and when the network is under attack. We will first test our hypothesis through simulation and than prove them mathematically.

The simulator has been developed using an event-based design which is capable of running hours of simulation in just a few minutes. It creates agents who decide to make a transaction, then they select which transactions will be confirmed, then they spend some time working in the proof-of-work, and, finally, they propagate the transaction to the network. The other agents receive the transaction and may accept or deny it. The agents may use different parameters among themselves.

When a new transaction is added to the DAG, it uses a depth-first search [29] (i) to update the aggregated weight of the directly and indirectly confirmed transactions, (ii) to calculate its own score, and (iii) to generate a topological sort [29]. The topological sort is used to calculate the longest path from all other transactions to the new transaction, and this longest path is used to update the depth of the whole DAG. If the new transaction confirms only transactions which has already been confirmed, the depth update is skipped thanks to Theorem 10.

The necessary time to generate the proof-of-work is estimated by the function $k \cdot 2^{weight}$, where k is a constant related to the computing power of the agent and weight is randomly chosen.

The time between two new transactions is sampled from the exponential distribution with parameter λ. The value of λ is changed over time, causing changes in the load of the network. We consider 1 tx/s to be a low load, and 15 tx/s to be a high load. Even though we have already tested up to 100 tx/s.

The simulator will output different reports: (i) snapshots of the DAG at interesting moments, such as Fig. 6, 7, and 8; (ii) histogram of confirmation time, such as Fig. 9 and 10; and many others.

7

ANALYSIS OF DAG MODEL

In order to a further understanding of the DAG Model, we will analyze the mathematical model behind it.

The DAG Model's mining starts just like Bitcoin's, i.e., $\mathbf{P}(h(x) < A) = \frac{A}{2^{256}}$. Then, it follows that solving a transaction proof-of-work follows a geometric distribution. Let X be the number of attempts until a success, then $\mathbf{P}(X = k) = (1-p)^{k-1}p$, where $p = \frac{A}{2^{256}}$. Also, $\mathbf{P}(X \leqslant k) = 1-(1-p)^k$. The average number of attempts is $\mathbf{E}(X) = 1/p$.

Let w be the transaction's weight, i.e., $\mathbf{E}(X) = w \Rightarrow w = 1/p$. Let $T = X/H$ be the time required to solve the transaction's proof-of-work, where H is the hash rate of the transaction's owner. Then, $\mathbf{P}(T < t) = \mathbf{P}(X < tH) = 1-(1-p)^{tH} = 1-\left(1-\frac{1}{w}\right)^{tH} = 1-e^{tH\log\left(1-\frac{1}{w}\right)}$, i.e., T follows an exponential distribution with $\lambda = -H\log\left(1-\frac{1}{w}\right)$.

The number of transactions solving the proof-of-work may be modeled by a Birth and Death process, since if we have k transactions solving the proof-of-work only two things may happen: either a new transaction will be created or one the transactions will finish solving the proof-of-work. Let the time between new transactions follows an exponential distribution with μ parameter, then the probability of a new transaction emerges before any transaction finishes its proof-of-work is $q_k = \mathbf{P}(T_{\text{new tx}} = \min\{T_{\text{new tx}}, T_1, T_2, \ldots, T_k\}) = \frac{\mu}{\mu+\sum_{i=1}^{k}\lambda_i}$. Hence, the probability of any transaction finishes its proof-of-work before a new transaction emerges is $p_k = 1-q_k = \frac{\sum_{i=1}^{k}\lambda_i}{\mu+\sum_{i=1}^{k}\lambda_i}$.

Every time a new transaction emerges, it chooses two tips to confirm before solving the proof-of-work. When it finishes solving the proof-of-work, it is propagated and becomes a tip. So, two new transactions may choose the same tips to confirm. If there are t tips, a new transaction will randomly choose 2 out of these t tips, even if they have already been chosen by other new transactions — in fact, they do not know which have been chosen because these new transactions have not being propagated yet.

The following theorems will be mathematically proved. They have already been used in the implementation of the simulator.

Theorem 10. *When a new transaction confirms an already confirmed transaction, the depth of all transactions remains the same, i.e., the depth of the transactions are only changed when a new transaction confirm an unconfirmed transaction.*

Theorem 11. *The height of a transaction is equal to the maximum height of its parents plus one.*

Theorem 12. *If new transactions choose randomly the unconfirmed transactions to be confirmed, then no unconfirmed transaction will be left behind, i.e., all transactions will be confirmed in due time.*

8 CONCLUSION

Bitcoin's underlying technology blockchain has been called by many as a major invention comparable to the invention of the internet. But it is unlikely that Bitcoin and blockchain have achieved the final or most optimal design for a secure and scalable electronic transaction system. In this work, we will investigate whether DAG model is suitable to be a real alternative to blockchain.

Preliminary results obtained from our simulator have shown that, using a random tip selection strategy, the network seems to support up to 50 transactions per second, with no tip abandoned (Fig. 7) or taking too long to have at least one confirmation (Fig. 10). Today, Bitcoin network can barely handle 8 transactions per second without increasing the unconfirmed transaction list to hundreds of thousands — several transactions take days to be confirmed.

We have not yet tested the security of the network against double spend attacks. It will be one of the next steps and we expect to block a nuclear submarine attack including a rule for invalidate new conflicting transactions which confirm transactions with depth greater than a given threshold. We are unsure whether this rule would invalidate legitimate transactions on a high load scenario.

We also plan to improve the analysis of the confirmation time. So far, we have the histograms of how long it takes to a transaction to be confirmed for the first time (Fig. 9 and 10). We will also generate the histograms of how long it takes to a transaction to have an accumulated weight of at least a given number. It is an important histogram because it helps merchants and users to decide how long they should wait to consider the transaction almost irreversible.

We will also work on the mathematical modeling of DAG networks, and the equations will help us to understand the limits of these networks.

Part III

SPARSE DISTRIBUTED MEMORY: A CROSS-PLATFORM, MASSIVELY PARALLEL, OPEN SOURCE REFERENCE IMPLEMENTATION

9 INTRODUCTION

How is memory gradually built up during one's conscious, or even unconscious, life and thought? My guess is that everything we experience is classified and registered on very many parallel channels in different locations

— Stanislaw Ulam

Pentti Kanerva's memory model was a revelation for me: it was the very first piece of research I had ever come across that made me feel I could glimpse the distant goal of understanding how the brain works as a whole. It gave me a concrete sense for how familiar mental phenomena could be thought of as distributed patterns of micro-events, thanks to beautiful mathematics.

— Douglas Hofstadter

Sparse Distributed Memory (SDM) [54] is a mathematical model of long-term memory that has a number of neuroscientific and psychologically plausible dynamics. This model may be applied in all sort of applications because of its incredible ability to closely reflect the human capacity to remember past experiences from clues of the present. For instance, when one is walking on a dark alley and is afraid of something, one cannot explain where one's fear comes from. One just feels it. We may interpret this situation as clues of the present — a dark alley; a giant metropolitan area; people going on about their lives mostly indifferent from each other; constrained routes ahead and behind you; a general lack of activity; that very feeling that something isn't right... without knowing precisely what isn't right, or even what 'right' means... etc. — recalling past experiences from memory and thus generating the feeling. Our memory is able to make a parallel between previous experiences and the clues. Although one has never been in the exactly same situation, one's brain involuntarily makes an analogy and recognizes the possibility of danger. This flexibility into mapping one situation in another is an essential human feature which is hard to replicate in computers.

SDM has been applied in many different fields, like pattern recognition [77, 88], noise reduction [70], handwriting recognition [41], robot automation [87, 69], and so forth. Linhares et al. [66] has

shown that SDM respects the limits of short-term memory discussed by Miller [71] and Cowan [31]. Despite all those applications, there is not a reference implementation which would allow one to replicate the results published in a paper, to check the source code for details, and to improve it. Thus, even though intriguing results have been achieved using SDM, it requires counter-productive, duplicate effort from researchers to build on top of previous work.

It is our belief that a tool such as a framework could bring orders of magnitude more researchers and attention if they were able to use the model, at zero cost, with an easy to use high-level language such as Python, in an intuitive platform such as Juypyter notebooks. Neuroscientists interested in long-term memory storage should not have to worry about high-bandwidth vector parallel computation. This new tool would provide a ready to use system in which experiments could be executed almost as soon as designed — and it might accelerate research [93].

Our motivation was our own effort to run our models. As there is no reference implementation, we had to develop our own and run several simulations to ensure that our implementation was correct and bug-free. Thus, we had to deviate from our primary goal — which was to test our hypothesis and explore the 'ideas space' — and to focus on the implementation itself. Furthermore, new members of our research group had to go through different source codes developed by former members.

Extensions of SDM have been used in many applications. For example, Snaider and Franklin [95] extended SDM to store sequences of vectors and trees efficiently. Rao and Fuentes [87] used a modified SDM in an autonomous robot. Meng et al. [70] modified SDM to clean patterns from noisy inputs. Fan and Wang [41] extended SDM with genetic algorithms. Chada [24] extended SDM creating the Rotational Sparse Distributed Memory (RSDM), which is used to modeling network motifs, dynamic flexibility, and hierarchical organization, all results from neuroscience literature.

The main contribution of this work is a reference implementation which yields (i) orders of magnitude gains in performance, (ii) has several backends and operations, (iii) has been validated against the mathematical model, (iv) is cross-platform, and (v) is easily extended to test new research ideas. Our reference implementation may, hopefully, accelerate research into the model's dynamics and make it easier for readers to replicate any previous results and easily understand the source-code of the model. Moreover, it is compatible with Jupyter notebook and researchers may share their notebooks possibly accelerating the advances in their fields [93].

Other contributions have also been introduced, which include (i) a noise filtering approach, (ii) a supervised classification algorithm, (iii) and a reinforcement learning algorithm, all of them using only

the original SDM proposed by Kanerva, i.e., with no additional mechanisms, algorithms, data structures, etc. Although some of these applications have already been explored in previous work [70, 41, 88], all of them have adapted SDM to fit their problems, and none of them have used just the ideas introduced by Kanerva. We have presented different approaches with no adaptations whatsoever.

Finally, I have striven to provide a visual tour of the theory and application of SDM: whenever possible, detailed figures should tell the story — or at least do the heavy lifting. In this study, we will see an anomaly in one of Kanerva's predictions, which I believe is related to SDM capacity. We will see tests of a generalized reading operation proposed by Physics Professor Paulo Murilo (personal communication). We will see what happens when neurons — and all their information — is simply and suddenly lost. We will see whether information-theory can improve some of Kanerva's ideas. From (basic) noise filtering to learning to play tic-tac-toe, we will review the entirety of Dr. Pentti Kanerva's proposal.

This time, however, it will be running on a computer.

10 NOTATION

n	Number of dimensions, i.e., $n = 1,000$.
N	Size of the binary space, $\lvert\{0,1\}^n\rvert = 2^n$.
N'	Number of hard locations samples from $\{0,1\}^n$. Its typical value is 1,000,000, as suggested by Kanerva [54].
H	Same as N'.
r	Access radius, i.e., when $n = 1,000$ and $N' = 1,000,000$, its typical value is 451. This value is calculated to activate, on average, one-thousandth of N'.
η	A bitstring, usually a datum.
η_x	A clue x bits away from η, i.e., $\text{dist}(\eta, \eta_x) = x$.
ξ	A bitstring, usually an address.
$\text{dist}(x, y)$	Hamming distance between x and y.
$d(x, y)$	Same as $\text{dist}(x, y)$.

11

SPARSE DISTRIBUTED MEMORY

Sparse Distributed Memory (SDM) is a mathematical model developed and suggested as a theory of human memory by Finish Scientist Pentti Kanerva [54]. It introduces many interesting mathematical properties of n-dimensional binary space that, in a memory model, seem to be remarkably psychologically plausible. Most notable among these are the tip-of-the-tongue phenomenon, conformity to Miller's magic number [66], and robustness against loss of neurons.

The data — and address space on which it is stored — are represented by large sequences of bits, called *bitstrings*. The *Hamming distance* provides comparisons between bitstrings and is used as a metric for the system. The Hamming distance is defined for two bitstrings of equal length as the number of positions in which bits differ. For example, 00110_b and 01100_b are bitstrings of length 5 and their Hamming distance is 2.

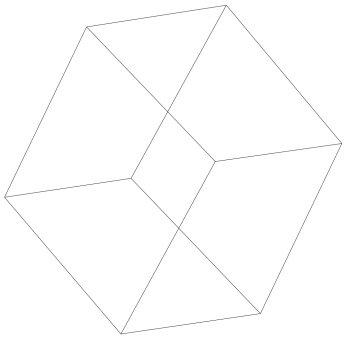
The space studied by Kanerva is also called the *hypercube graph*, or Q_n, as in Figure 11. For a fixed $n \in \mathbb{Z}$, the graph $G = (V, E)$, in which $v \in V$ iff there is a bijective function $b : V \to \{0,1\}^n$ and $(v_i, v_j) \in E$ iff $H(b(v_i), b(v_j)) = 1$, where H is the Hamming distance. That is, n-sized bitstrings correspond to nodes, and edges exist between nodes $\mathtt{iff}$ they flip a single bit. Though Kanerva has derived many combinatorial properties of the space, additional results have been found by the graph-theoretical community. A good survey is provided by Harary et al. [49].

One has to be careful when thinking intuitively about distance in SDM because the Hamming distance does not have the same properties of, say, our 3-dimensional space.

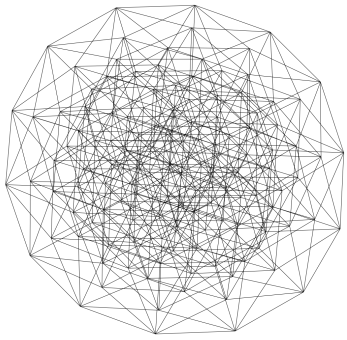
Though both follow the triangle inequality ($d(A, C) \leqslant d(A, B) + d(B, C)$), which in 3-d Euclidean distance may be loosely interpreted as "if A is close to B, and B is close to C, then A is also close to C" — $d(A, B) \leqslant r$ and $d(B, C) \leqslant r \Rightarrow d(A, C) \leqslant 2r$ —, but in SDM, although the inequality is also valid, two bitstrings would be close when, for instance, $r = 430$, so $2r = 860$ would cover all other bitstrings. Hence, it makes no sense to say that A is also close to C.

There are numerous, beautiful, counter-intuitive notions involved in this space. This difference in intuition may trick even experienced researchers when analyzing some situations.

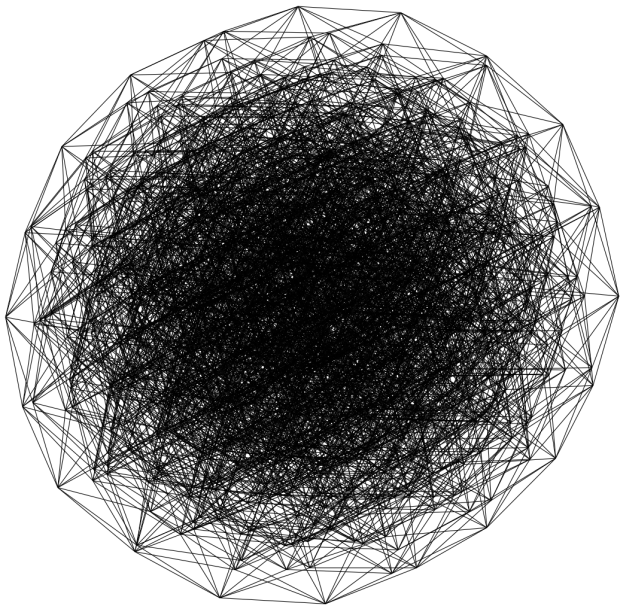
Unlike traditional memory used by computers, SDM performs read and write operations in a multitude of addresses, also called neurons. That is, the data is not written, or it is not read in a single address



(a) Q_3



(b) Q_7



(c) Q_{10}

Figure 11: Here we have Q_n, for $n \in \{3, 7, 10\}$. Each node corresponds to a bitstring in $\{0,1\}^n$, and two nodes are linked iff the bitstrings differ by a single dimension. A number of observations can be made here. First, the number of nodes grows as $O(2^n)$; which makes the space rapidly intractable. Another interesting observation, better seen in the figures below, is that most of the space lies 'at the center', at a distance of around $n/2$ from any given vantage point.

spot, but in many addresses. These are called activated addresses, or activated neurons.

The activation of addresses takes place according to their distances from the datum. Suppose one is writing datum η at address ξ, then all addresses inside a circle with center ξ and radius r are activated. So, η will be stored in all these activated addresses, which are around address ξ, such as in Figure 12. An address ξ' is inside the circle if its Hamming distance to the center ξ is less than or equal to the radius r, i.e. $distance(\xi, \xi') \leqslant r$.

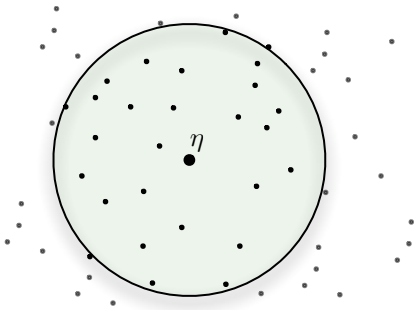


Figure 12: Activated addresses inside access radius r around the center address.

Every write or read in SDM memory activates a number of addresses with close distance. The data is written in these activated addresses or read from them. These issues will be addressed in due detail further on, but a major difference from a traditional computer memory is that the data are always stored and retrieved in a multitude of addresses. This way SDM memory has robustness against loss of addresses (e.g., death of a neuron).

In traditional memory, each datum is stored in an address and every lookup of a specific datum requires a search through the memory. In spite of computer scientists having developed beautiful algorithms to perform fast searches, almost all of them do a precise search. That is, if you have an imprecise clue of what you need, these algorithms will simply fail.

In SDM, the data space is the same as the address space, which amounts to a vectorial, binary space, that is, a $\{0,1\}^n$ space. This way, the addresses where the data will be written are the same as the data themselves. For example, the datum $\eta = 00101_b \in \{0,1\}^5$ will be written to the address $\xi = \eta = 00101_b$. If one chooses a radius of 1, the SDM will activate all addresses one bit away or less from the center address. So, the datum 00101_b will be written to the addresses 00101_b, 10101_b, 01101_b, 00001_b, 00111_b, and 00100_b.

In this case, when one needs to retrieve the data, one could have an imprecise cue at most one bit away from η, since all addresses one bit away have η stored in themselves. Extending this train of thought

for larger dimensions and radius, exponential numbers of addresses are activated and one can see why SDM is a distributed memory.

When reading a cue η_x that is x bits away from η, the cue shares many addresses with η. The number of shared addresses decreases as the cue's distance to η increases, in other words, as x increases. This is shown in Figure 13. The target datum η was written in all shared addresses, thus they will bias the read output in the direction of η. If the cue is sufficiently near the target datum η, the read output will be closer to η than η_x was. Repeating the read operation increasingly gets results closer to η, until it is precisely the same. So, it may be necessary to perform more than one read operation to converge to the target data η.

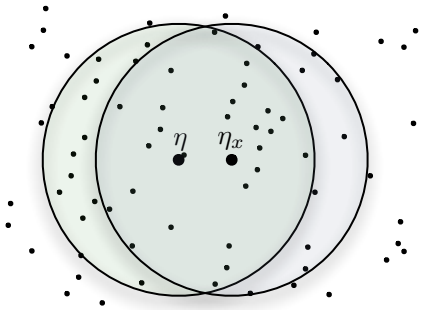


Figure 13: Shared addresses between the target datum η and the cue η_x.

The addresses of the $\{0,1\}^n$ space grow exponentially with the number of dimensions n, i.e., $N = 2^n$. For $n = 100$ we have $N \approx 10^{30}$, which is incredibly large when related to computer memory. Furthermore, Kanerva [54] suggests n between 100 and 10,000. Recently he has postulated 10,000 as a desirable minimum N (personal communication). To solve the feasibility problem of implementing this memory, Kanerva made a random sample of $\{0,1\}^n$, in his work, having N' elements. All these addresses in the sample are called hard locations. Other elements of $\{0,1\}^n$, not in N', are called virtual neurons. This is represented in Figure 14. All properties of reading and write operations presented before remain valid but limited to hard locations. Kanerva suggests taking a sample of about one million hard locations.

Using this sample of binary space, our data space does not exist completely. That is, the binary space has 2^n addresses, but the memory is far away from having these addresses available. In fact, only a fraction of this vectorial space is actually instantiated. Following Kanerva's suggestion of one million hard locations, for $n = 100$, only $100 \cdot 10^6/2^{100} = 7 \cdot 10^{-23}$ percent of the whole space exists, and for $n = 1,000$ only $100 \cdot 10^6/2^{1000} = 7 \cdot 10^{-294}$ percent.

Kanerva also suggests the selection of a radius that will activate, on average, one-thousandth of the sample, which is 1,000 hard locations for a sample of one million addresses. In order to achieve his suggestion, a 1,000-dimension memory uses an access radius $r = 451$, and a 256-dimensional memory, $r = 103$. We think that a 256-dimensional memory may be important because it presents conformity to Miller's magic number [66].

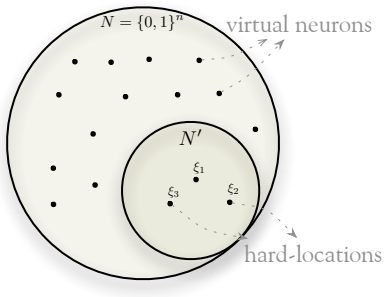


Figure 14: Hard-locations randomly sampled from binary space.

Since a cue η_x near the target bitstring η shares many hard locations with η, SDM can retrieve data from imprecise cues. Despite this feature, it is very important to know how imprecise this cue could be while still giving accurate results. What is the maximum distance from our cue to the original data that still retrieves the right answer? An interesting approach is to perform a read operation with a cue η_x, that is x bits away from the target η. Then measure the distance from the read output and η. If this distance is smaller than x we are converging. Convergence is simple to handle, just read again and again, until it converges to the target η. If this distance is greater than x we are diverging. Finally, if this distance equals x we are in a tip-of-the-tongue process. A tip-of-the-tongue psychologically happens when you know that you know, but you can't say what exactly it is. In SDM mathematical model, a tip-of-the-tongue process takes infinite time to converge. Kanerva [54] called this x distance, where the read's output averages x, the critical distance. Intuitively, it is the distance from which smaller distances converge and greater distances diverge. In Figure 15, the circle has radius equal to the critical distance and every η_x inside the circle should converge. The figure also shows convergence in four readings.

The $\{0,1\}^n$ space has $N = 2^n$ locations from which we instantiate N' samples. Each location in our sample is called a hard location. On these hard locations, we do operations of read and write. One of the insights of SDM is exactly the way we read and write: using data as addresses in a distributed fashion. Each datum η is written in every activated hard location inside the access radius centered on the address, that equals datum, $\xi = \eta$. Kanerva suggested using an

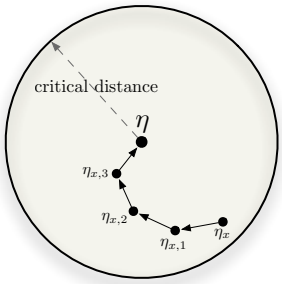


Figure 15: In this example, four iterative readings were required to converge from η_x to η.

η	0	1	1	0	1	0	0
ξ_{before}	6	-3	12	-1	0	2	4
	⇓ -1	⇓ +1	⇓ +1	⇓ -1	⇓ +1	⇓ -1	⇓ -1
ξ_{after}	**5**	**-2**	**13**	**-2**	**1**	**1**	**3**

Table 1: Write operation example in a 7-dimensional memory of data η being written to ξ, one of the activated addresses.

access radius r having about one-thousandth of N'. As an imprecise cue η_x shares hard locations with the target bitstring η, it is possible to retrieve η correctly. (Actually, probably more than one read is necessary to retrieve exactly η.). Moreover, if some neurons are lost, only a fraction of the datum is lost and it is possible that the memory can still retrieve the right datum.

A random bitstring is generated with equal probability of 0's and 1's for each bit. One can readily see that the average distance between two random bitstrings follows the binomial distribution with mean $n/2$ and standard deviation $\sqrt{n/4}$. For a large n, most of the space lies close to the mean and has fewer shared hard locations. As two bitstrings with distance far from $n/2$ are very improbable, Kanerva [54] defined that two bitstrings are orthogonal when their distance is $n/2$.

The write operation needs to store, for each dimension bit which happened more (0's or 1's). This way, each hard location has n counters, one for each dimension. The counter is incremented for each bit 1 and decremented for each bit 0. Thus, if the counter is positive, there have been more 1's than 0's, if the counter is negative, there have been more 0's than 1's, and if the counter is zero, there have been an equal number of 1's and 0's. Table 1 shows an example of a write operation being performed in a 7-dimensional memory.

The read is performed polling each activated hard location and statistically choosing the most written bit for each dimension. It consists of adding all n counters from the activated hard locations

and, for each bit, choosing bit 1 if the counter is positive, choosing bit 0 if the counter is negative, and randomly choose bit 0 or 1 if the counter is zero.

11.1 NEURONS AS POINTERS

One interesting view is that neurons in SDM work like pointers. As we write bitstrings in memory, the hard locations' counters are updated and some bits are flipped. Thus, the activated hard locations do not necessarily point individually to the bitstring that activated it, but together they point correctly. In other words, the read operation depends on many hard locations to be successful. This effect is represented in Figure 16: where all hard locations inside the circle are activated and they, individually, do not point to η. But, like vectors, adding them up points to η. If another datum ν is written into the memory near η, the shared hard locations will have information from both of them and would not point to either. All hard locations outside of the circle are also pointing somewhere (possibly other data points). This is not shown, however, in order to keep the picture clean and easily understandable.

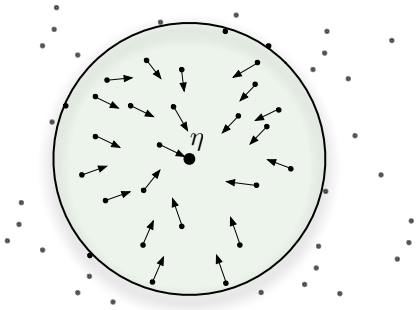


Figure 16: Hard-locations pointing, approximately, to the target bitstring.

11.2 CONCEPTS

Although Kanerva does not mention concepts directly in his book [54], the author's interpretation is that each bitstring may be mapped to a concept. Thus, unrelated concepts are orthogonal and concepts could be linked through a bitstring near both of them. For example, "beauty" and "woman" have distance $n/2$, but a bitstring that means "beautiful woman" could have distance $n/4$ to both of them. As a bitstring with distance $n/4$ is very improbable, it is linking those concepts together. Linhares et al. [66] approached this concept via "chunking through averaging".

Due to the distribution of hard locations between two random bitstrings, the vast majority of concepts is orthogonal to all others. Consider a non-scientific survey during a cognitive science seminar, where students asked to mention ideas unrelated to the course brought up terms like birthdays, boots, dinosaurs, fever, executive order, x-rays, and so on. Not only are the items unrelated to cognitive science, the topic of the seminar, but they are also unrelated to each other.

For any two memory items, one can readily find a stream of thought relating two such items ("Darwin gave dinosaurs the boot"; "she ran a fever on her birthday"; "isn't it time for the Supreme Court to x-ray that executive order?", ... and so forth). Robert French presents an intriguing example in which one suddenly creates a representation linking the otherwise unrelated concepts of "coffee cups" and "old elephants" [42].

This mapping from concepts to bitstrings brings us two central questions: (i) Suppose we have a bitstring that is linking two major concepts. How do we know which concepts are linked together? (ii) From a concept bitstring, how can we list all concepts that are somehow linked to it? This second question is called the problem of spreading activation.

11.3 READ OPERATION

In his work, Kanerva proposed and analyzed a read algorithm called here Kanerva's read. His read takes all activated hard locations counters and sum them. The resulting bitstring has bit 1 where the result is positive, bit 0 where the result is negative, and a random bit where the result is zero. In a word, each bit is chosen according to all written bitstrings in all hard locations, being equal to the bit more appeared. Table 2a shows an example of Kanerva's read result bitstring.

Daniel Chada, one member of our research group, proposed another way to read in SDM, in this work called Chada's read. Instead of summing all hard location counters, each hard location evaluates its resulting bitstring individually. Then, all resulting bitstrings are summed again, and the same rule as Kanerva applies. Table 2b shows an example of Chada's read result bitstring. The counter's values are normalized to 1, for positive ones, or -1, for negative ones, and the original values are the same as in Table 2a.

The main difference between Kanerva's read and Chada's is that, in the former, a hard location that has more bitstrings written has a greater weight in the decision of each bit. In the latter, all hard locations have the same weight, because they can contribute to the sum with only one bitstring.

It is important to say that Chada's read came from Anwar and Franklin [4] which gave a misguided description of the read operation. The original description is the following:

> With our datum distributively stored, the next question is how to retrieve it. With this in mind, let us ask first how one reads from a single hard location, x. Compute ζ, the bit vector read at x, by assigning its ith bit the value 1 or 0 according to the ith counter at x is positive or negative. Thus, each bit of ζ results from a majority rule decision of all the data that have been written at x. [...] Knowing how to read from a hard location allows us to read from any of the 2^{1000} arbitrary locations. Suppose ζ is any location. The bit vector, ξ, to be read at ζ, [...] Put another way, pool the bit vectors read from hard locations accessible from ζ, and let each of their ith bits vote on the ith bit of ξ.
> — Anwar and Franklin [4, p.342]

This fact just highlights how important it is to have a reference implementation that one may read the code to clarify one's understanding about the details of each operation.

11.3.1 *Generalized read operation*

A member of my Master's committee, Prof. Paulo Murilo[1], has proposed a generalized reading operation (personal communication), which covers both Kanerva's and Chada's read — and opens a new venue of potential discoveries. He proposed summing all hard location counters raised to the power of z while holding the original sign of the counter (positive or negative). Thus, Kanerva's read would be the same as $z = 1$, while Chada's would be the same as $z = 0$. Hence, we will here explore how SDM would behave with other values of z, such as 0.5, 2, and 3. Mathematically, let A be the set of the counters of the activated hard location, and c_i be the counter of the ith bit. Then,

$$s_i = \sum_{c \in A} \frac{c_i}{|c_i|} |c_i|^z$$

Finally, the ith bit of the resulting bitstring is 1 if $s_i > 0$, or 0 if $s_i < 0$, or random if $s_i = 0$. Notice that when $z = 1$, then $s_i = \sum_{c \in A} c_i$, which is the Kanerva's read; and when $z = 0$, then $s_i = \frac{c_i}{|c_i|} = \text{sign}(c_i)$, which is the Chada's read.

1 Universidade Federal Fluminense's Physics Professor Paulo Murilo

ξ_1	-2	12	4	0	-3
ξ_2	-5	-4	2	8	-2
ξ_3	-1	0	-1	-2	-1
ξ_4	3	2	-1	3	1
$\sum$	**-5**	**10**	**4**	**3**	**-5**
	⇓	⇓	⇓	⇓	⇓
	0	1	1	1	0

(a) Kanerva's read example

ξ_1	-1	1	1	1	-3
ξ_2	-1	-1	1	1	-1
ξ_3	-1	1	-1	-1	-1
ξ_4	1	1	-1	-1	1
$\sum$	**-2**	**1**	**0**	**0**	**-2**
	⇓	⇓	⇓	⇓	⇓
	0	1	1	1	0

(b) Chada's read example

Table 2: Comparison of Kanerva's read and Chada's read. Each ξ_i is an activated hard location and the values come from their counters. Gray cells' value is obtained randomly with probability 50%.

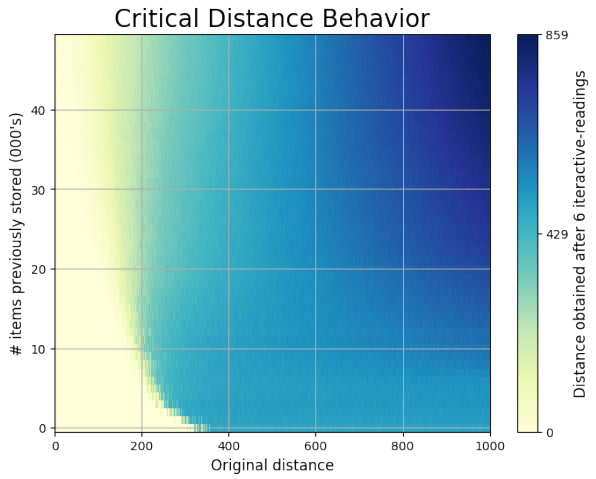
11.4 CRITICAL DISTANCE

Kanerva describes the critical distance as the threshold of convergence of a sequence of read words. It is "the distance beyond which divergence is more likely than convergence"[54]. Furthermore, Kanerva explains that "a very good estimate of the critical distance can be obtained by finding the distance at which the arithmetic mean of the new distance to the target equals the old distance to the target"[54]. In other words, the critical distance can be equated as the edge to our memory, the limit of human recollection.

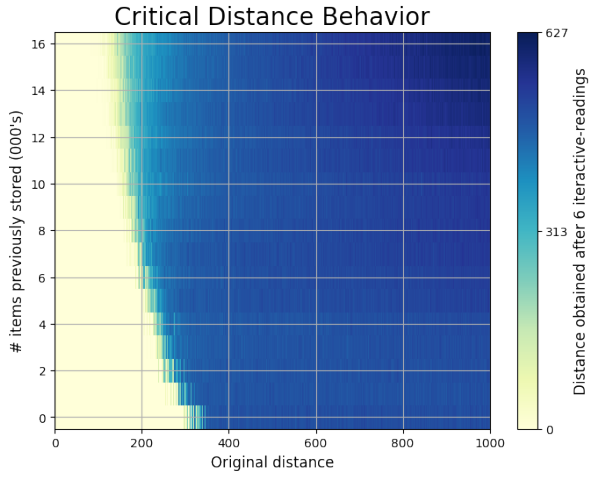
In his book, Kanerva analyzed a specific situation with $n = 1000$ ($N = 2^{1000}$), 1 million hard locations ($N' = 1,000,000$), an access-radius of 451 (within 1,000 hard locations in each circle), and 10 thousand writes of random bitstrings in the memory. As computer resources were very poor those days, Kanerva couldn't make a more generic analysis.

Starting from the premise of SDM as a faithful model of human short-term memory, a better understanding of the critical distance may shed light on our understanding of the thresholds that bind our own memory.

Figure 17 compares the critical distance behavior under different scenarios. This replicates our previous results [18, 20] and is a first part of the process of framework validation, to which we throw our attention next.



(a) Kanerva's original model



(b) Chada's read

Figure 17: How far, in Hamming distance, is a read item from the original stored item? Kanerva demonstrated that, after a small number of iterative readings (6 here), a critical distance behavior emerges. Items read at close distance converge rapidly; whereas farther items do not converge. Most striking is the point in which the system displays the tip-of-tongue behavior. Described by psychological moments when some features of the item are prominent in one's thoughts, yet the item still cannot be recalled (but an additional cue makes convergence 'immediate'). Mathematically, this is the precise distance in which, despite having a relatively high number of cues (correct bits) about the desired item, the time to convergence is infinite. Heatmap colors display the Hamming distance the associative memory is able to cleanly converge to—or not. In the x-axis, the distance from the desired item is displayed. In the y-axis, we display the read operation's behavior as the number of items registered in the memory grows. These graphs are computing intensive, yet they can be easily tested by readers in our provided Jupyter notebooks. Note the different scales.

12

FRAMEWORK ARCHITECTURE

The framework implements the basic operations in a Sparse Distributed Memory which may be used to create more complex operations. It is developed in C language and the OpenCL parallel framework — which may be loaded in many platforms and programming languages — with a wrapper in Python. The Python module makes it easy to create and execute simulations in a Sparse Distributed Memory and works properly in Jupyter Notebook [58]. It works in both Python 2 and Python 3.

The SDM memory has been split into two parts: the hard location addresses and the hard location counters. Thus, the addresses (bitstrings) of the hard locations are stored in one array, while their counters in another. This makes possible to create multiple SDMs using the same address space, which would save computational effort to scan a bitstring in all the SDMs — since they share the same address space, the activated hard locations will be the same in all of them. As the slowest part of reading and writing operations is scanning the address space, the performance benefits are significant.

Each part may be stored either in the RAM memory or in a file. The RAM memory is interesting for quick experiments, automated tests, and others scenarios in which the SDM may be lost, while the file is interesting for a long-term SDM, like creating an SDM file with 10,000 random writes, which will be copied over and over to run multiple experiments. The file may also be sent to another researcher or may be published within the paper to let others run their own checks and verify the results. In summary, the framework fits many different uses and necessities.

Let a SDM memory with n dimensions and H hard locations. Then, in a 64-bit computer, the array of hard location addresses will use $H \cdot 8 \cdot \lceil n/64 \rceil$ bytes of memory, and there will be $H \cdot n$ hard location counters. For example, in a SDM memory with 1,000 dimensions and 1,000,000 hard locations, using 32-bit integers for the counters, the array of addresses will use 122MB of memory and the counters will use 3.8 GB of memory.

Basic operations were grouped into four sets: (i) for bitstrings, (ii) for addresses, (iii) for counters, and (iv) for memories (SDMs). Operations include creating new bitstrings, flipping bits, generating a bitstring with a specific distance from a given bitstring, scanning the address space using different algorithms, writing a bitstring to a counter, writing in an SDM, reading from an SDM, and iteratively reading from an SDM until convergence.

12.1 BITSTRING

Bitstrings are the main structure of SDM. The addresses are represented in bitstrings, as well as the data. A bitstring is stored as an array of integers. Each integer may be 16-bit, 32-bit, or 64-bit long, depending on the configuration. By default, each integer is 64-bit long.

For instance, a 1,000-bit bitstring will have $\lceil 1000/64 \rceil = 16$ integers. These integers will have a total of $16 \cdot 64 = 1,024$ bits. The remaining 24 bits are always zero, so they do not affect the result of any operation. The memory usage efficiency is $1 - 24/1024 = 97.65\%$. Bitstrings store neither how many bits they have nor the array length. These pieces of information are only stored in the address space.

12.1.1 *The distance between two bitstrings*

The distance between two bitstrings is calculated by the Hamming distance, which is the number of different bits between them. It is calculated counting the number of ones in the exclusive or (xor) between the bitstrings, i.e., $d(x, y) =$ number of ones in $x \oplus y$.

There are several algorithms to calculate the number of ones [101], but the performance depends on the processor. So, we have implemented three different algorithms and one may be selected through compiling flags. The default algorithm is to use a built-in __popcnt() instruction from the compiler.

There is also the naive algorithm, which really counts the number of ones checking bit by bit. It is available only for testing purposes and should never be used.

The other algorithm available is the lookup. It pre-calculates a table with the number of ones of all possible 16-bit integers. This table is accessed a few times to calculate the number of ones of a 64-bit integer, i.e., to calculate the distance between two bitstrings, it sums the distance of each 16-bit part of the bitstrings, i.e., $d(x[0:63], y[0:63]) = d(x[0:15], y[0:15]) + d(x[16:31], y[16:31]) + d(x[32:47], y[32:47]) + d(x[48:63], y[48:63])$ where $x[i:i+15]$ and $y[i, i+15]$ are the 16-bit integers formed by the bits between i and $i+15$ of x and y, respectively. Each 16-bit distance is calculated through a single table access. As each distance is calculated in O(1), this algorithm runs in $O(\lceil bits/16 \rceil)$. This table uses 65MB of RAM. One may change the table from 16-bit integers to 32-bit integers, which would halve the number of accesses at the expense of 4GB of RAM (instead of 65MB).

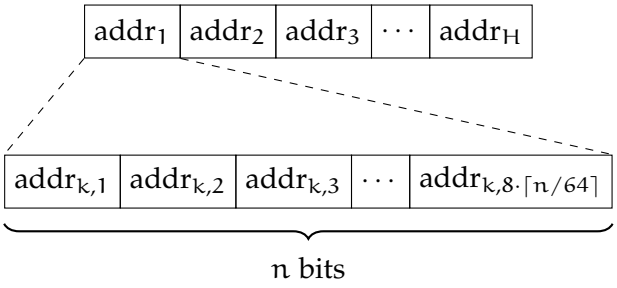


Figure 18: Address space's bitstrings are stored in a contiguous array. In a 64-bit computer, each bitstring is stored in a sub-array of 64-bit integers, with length $8 \cdot \lceil n/64 \rceil$.

12.2 ADDRESS SPACE

An address space is a fixed collection of bitstrings, and each bitstring represents a hard location address. They store the number of bitstrings, as well as the number of bits, number of integers per bitstring, and the number of remaining bits.

Bitstrings are stored in a contiguous array of 64-bit integers, as shown in Figure 18. Hence, basic pointer arithmetic provides us with performance improvements in their access, as processors realize fetches of contiguous chunks of memory [81].

The scan for activated hard locations is performed in an address space. It returns the indexes of the bitstrings which were inside the circle (and their distances). Then, each operation uses these pieces of information in a different way.

12.2.1 *Scanning for activated hard locations*

Scanning for the activated hard locations is a problem similar to well-known problems in computational geometry called "range reporting in higher dimensions". In this case, none of the known algorithms is able to solve our problem faster than $O(H)$. The algorithm which seems to best fit in our problem consumes $O(H)$ space and runs in $O(\log^n(H))$ [26], which is significantly slower than $O(H)$ when, for instance, $H = 1,000,000$ and $n = 1,000$. For a review of the range reporting algorithms, see Chan et al. [25].

In 2014, Norouzi et al. [78] published a solution to fast search in hamming space which seems applicable to our problem. It provides a fast search when $r/n < 0.11$ or $r/n < 0.06$, where r is the radius and n is the number of bits. But, in our case, for a 1,000 bits SDM, $r/n = 0.451$, which changes the runtime to $O(H^{0.993})$. This is very close to $O(H)$, but with a larger constant. Unfortunately, $O(H)$ is still faster.

It is intriguing that none of those algorithms is able to solve our scanning problem. The roughly idea behind those computational

geometry algorithms is to split the search space in half each step, which would take $O(\log(H))$ to go through the whole space. But this approach does not work because of the high number of dimensions (i.e., 1,000) and because the hard locations' addresses are randomly sampled from the $\{0,1\}^n$ space. Although each addresses' bit itself splits the hard locations in half, it does not split the search space in half since both halves still must be covered by the algorithm. For instance, let's say we have $n = 1,000$ dimensions with $H = 1,000,000$ hard locations, and we are scanning within a circle with radius $r = 451$, then after checking the first bit we have two cases: (i) for the half with the same first bit, we must keep scanning with radius 451; and (ii) for the half with a different first bit, we must keep scanning with radius 450. Hence, the search space has not been split in half because both halves have been covered (and one of them should have been skipped).

Finally, as our best approach is to scan through all hard locations, we may distribute the scan into many tasks which will be executed independently. The tasks may be executed in different processes, threads, or even computers. They may also run in the CPU or in the GPU. In this case, we may take into account both the time required to distribute the tasks and the time to receive their results.

The framework implements three main scanner algorithms: linear scanner, thread scanner, and OpenCL scanner. The linear scanner runs in a single core, is the slowest one, and was developed only for testing purposes; the thread scanner runs at the CPU in multiple threads sharing memory (and our recommendation is to use the number of threads equals to twice the number of CPU cores); and the OpenCL scanner runs in multiple GPU cores and support multiple devices. The speed of a scan depends on the CPU and GPU devices, thus the best approach to choose which scanner is best for one's setup is to run a benchmark.

The OpenCL must be initialized, which basically copies the address space's bitstrings to the GPU's memory. Then, many scans may be executed with no necessity to upload the bitstrings again. The OpenCL scanner supports running on multiple devices.

12.2.2 *OpenCL kernels*

There are eight OpenCL kernels which differently explore the GPU architecture to improve performance. It is necessary because there are several GPU microarchitectures and a single kernel will never be optimal for all of them. In simplified form, OpenCL splits the tasks into workgroups which, in turn, split their part of the task into workers. The works are like threads in a computer. OpenCL specifies four levels of memory hierarchy for the GPUs: global memory, read-only memory, local memory, and private memory. The global memory

and read-only memory are accessible by all workgroups, while each workgroup has its own local memory, shared by its workers. Finally, each worker has its own private memory. The number of workers per workgroup is defined by user and must be multiple of the number of tasks.

All eight kernels do the same thing: calculate the exclusive OR (XOR) between two 64-bit integers and count the number of bits one in the result. They just do it with different approaches. For instance, `single_scan0` calculates one distance between bitstrings per worker (Listing 1); while `single_scan2` uses a whole workgroup to calculate each distance, distributing each element of the 64-bit integer array per worker (Listing 3).

The OpenCL kernels `single_scan3` (Listing 4), `single_scan4` (Listing 5), `single_scan5` (Listing 6), `single_scan5_unroll` (Listing 7), `single_scan6` (Listing 8) explore the GPU architecture to improve the sum of the partial distances. Each workgroup calculates the distance of several bitstrings. During the distance calculation, each worker calculates the exclusive OR (XOR) between two 64-bit integers and use the built-in popcount function to count the number of ones. Then, they update an array of partial distances with their results. This array is stored in the local memory and is shared between all workers of the same workgroup. This whole step happens simultaneously in the GPU. Then, a reduction algorithm is used to sum the partial distances array in order to calculate the total distance. This reduction algorithm is also distributed among the workers and runs in $O(\log_2(\text{bs\_step}))$. Finally, the first worker of each workgroup checks whether the distance is less than or equal to the radius to include the bitstring index into the resulting array.

Some of the optimizations may not work in some GPUs because not all their premises are valid. Before choosing a kernel, one should check whether it works property for one's specific GPU device.

12.3 COUNTERS

A counter is an array of integers which stores the data of all hard locations. Each hard location has one integer of data per bit. For instance, each hard location of a 1,000 bits SDM has 1,000 integers, totalizing 1,000,000 integers. Those integers are stored in a counter. So, the counter's array has $n \cdot H$ integers.

When two counters are added in a third counter, there may occur an overflow. It is not supposed to be a problem because, by default, each counter is a signed 32-bit integer that can store any number between -2,147,483,648 and 2,147,483,647, which means they will not overflow with fewer writes than $2^{31} - 1$ divided by the average number of activated hard locations. For instance, when $n = 1,000$, $H = 1,000,000$, and $r = 451$, the average number of activated hard

```
__kernel
void single_scan0(
                __constant const uchar *bitcount_table,
                __global const ulong *bitstrings,
                const uint bs_len,
                const uint sample,
                __constant const ulong *bs,
                const uint radius,
                __global uint *counter,
                __global uint *selected,
                __local uint *partial_dist)
{
        uint id = get_global_id(0);

        if (id < sample) {
                ulong a;
                uint dist;

                const __global ulong *row = bitstrings + id*bs_len;

                dist = 0;
                for(uint j=0; j<bs_len; j++) {
                        a = row[j] ^ bs[j];
                        dist += popcount(a);
                }
                if (dist <= radius) {
                        selected[atomic_inc(counter)] = id;
                }
        }
}
```

Listing 1: OpenCL kernel `single_scan0`. It calculates one distance per worker and lets the GPU decide how to distribute this task between workgroups and workers. It is the most straightforward kernel and does not explore any details of the GPU architecture.

```
1  __kernel
2  void single_scan1(
3          __constant const uchar *bitcount_table,
4          __global const ulong *bitstrings,
5          const uint bs_len,
6          const uint sample,
7          __constant const ulong *bs,
8          const uint radius,
9          __global uint *counter,
10         __global uint *selected,
11         __local uint *partial_dist)
12 {
13     uint id;
14     ulong a;
15     uint dist;
16     const __global ulong *row;
17
18     for (id=get_global_id(0); id < sample; id += get_global_size(0)) {
19
20         row = bitstrings + id*bs_len;
21
22         dist = 0;
23         for(uint j=0; j<bs_len; j++) {
24             a = row[j] ^ bs[j];
25             dist += popcount(a);
26         }
27         if (dist <= radius) {
28             selected[atomic_inc(counter)] = id;
29         }
30
31     }
32 }
```

Listing 2: OpenCL kernel `single_scan1`. It is just like `single_scan0`, but it distributes several distances per workgroup, which, in turn, distributes the distances among their workers.

```
__kernel
void single_scan2(
        __constant const uchar *bitcount_table,
        __global const ulong *bitstrings,
        const uint bs_len,
        const uint sample,
        __constant const ulong *bs,
        const uint radius,
        __global uint *counter,
        __global uint *selected,
        __local uint *partial_dist)
{
    uint dist;
    ulong a;
    uint j;

    for (uint id = get_group_id(0); id < sample; id += get_num_groups(0)) {

        const __global ulong *row = bitstrings + id*bs_len;

        dist = 0;
        j = get_local_id(0);
        if (j < bs_len) {
            a = row[j] ^ bs[j];
            dist += popcount(a);
        }
        partial_dist[get_local_id(0)] = dist;

        barrier(CLK_LOCAL_MEM_FENCE);

        if (get_local_id(0) == 0) {
            dist = 0;
            for(uint t = 0; t < bs_len; t++) {
                dist += partial_dist[t];
            }
            if (dist <= radius) {
                selected[atomic_inc(counter)] = id;
            }
        }

        barrier(CLK_LOCAL_MEM_FENCE);
    }
}
```

Listing 3: OpenCL kernel `single_scan2`. It calculates one distance per workgroup, distributing each 64-bit integer operation per worker, and then summing the results obtained by the workers. The sum algorithm is done by only the first worker of each workgroup.

```
__kernel
void single_scan3(
        __constant const uchar *bitcount_table,
        __global const ulong *bitstrings,
        const uint bs_len,
        const uint sample,
        __constant const ulong *bs,
        const uint radius,
        __global uint *counter,
        __global uint *selected,
        __local uint *partial_dist)
{
    uint dist;
    ulong a;
    uint j;

    for (uint id = get_group_id(0); id < sample; id += get_num_groups(0)) {

        const __global ulong *row = bitstrings + id*bs_len;

        dist = 0;
        j = get_local_id(0);
        if (j < bs_len) {
            a = row[j] ^ bs[j];
            dist = popcount(a);
        }
        partial_dist[get_local_id(0)] = dist;

        // Parallel reduction to sum all partial_dist array.
        for(uint stride = get_local_size(0)/2; stride > 0; stride /= 2) {
            barrier(CLK_LOCAL_MEM_FENCE);
            if (get_local_id(0) < stride) {
                partial_dist[get_local_id(0)] +=
                    partial_dist[get_local_id(0) + stride];
            }
        }

        if (get_local_id(0) == 0) {
            if (partial_dist[0] <= radius) {
                selected[atomic_inc(counter)] = id;
            }
        }

        barrier(CLK_LOCAL_MEM_FENCE);
    }
}
```

Listing 4: OpenCL kernel `single_scan3`. It calculates one distance per workgroup, distributing each 64-bit integer operation per worker, and then summing the results obtained by the workers. The sum algorithm is a parallel reduction, in which the workers split the array into two parts and sum the second part in the first part every loop. So, the sum is calculated in $O(\log_2(\text{number of workers per workgroup}))$. This kernel only works when the number of workers per workgroup is a power-of-2.

```
__kernel
void single_scan4(
        __constant const uchar *bitcount_table,
        __global const ulong *bitstrings,
        const uint bs_len,
        const uint sample,
        __constant const ulong *bs,
        const uint radius,
        __global uint *counter,
        __global uint *selected,
        __local uint *partial_dist)
{
    uint dist;
    ulong a;
    uint j;

    for (uint id = get_group_id(0); id < sample; id += get_num_groups(0)) {

        const __global ulong *row = bitstrings + id*bs_len;

        dist = 0;
        j = get_local_id(0);
        if (j < bs_len) {
            a = row[j] ^ bs[j];
            dist = popcount(a);
        }
        partial_dist[get_local_id(0)] = dist;

        uint old_stride = get_local_size(0);
        __local uint extra;
        extra = 0;
        for(uint stride = get_local_size(0)/2; stride > 0; stride /= 2) {
            barrier(CLK_LOCAL_MEM_FENCE);
            if ((old_stride&1) == 1 && get_local_id(0) == old_stride-1) {
                extra += partial_dist[get_local_id(0)];
            }
            if (get_local_id(0) < stride) {
                partial_dist[get_local_id(0)] +=
                    partial_dist[get_local_id(0) + stride];
            }
            old_stride = stride;
        }

        if (get_local_id(0) == 0) {
            if (partial_dist[0] + extra <= radius) {
                selected[atomic_inc(counter)] = id;
            }
        }

        barrier(CLK_LOCAL_MEM_FENCE);
    }
}
```

Listing 5: OpenCL kernel `single_scan4`. This kernel is just like `single_scan3`, but it works with any number of workers per workgroup. The tradeoff is that it includes an aditional step in the parallel reduction algorithm.

```
__kernel
void single_scan5(
        __constant const uchar *bitcount_table,
        __global const ulong *bitstrings,
        const uint bs_len,
        const uint sample,
        __constant const ulong *bs,
        const uint radius,
        __global uint *counter,
        __global uint *selected,
        __local uint *partial_dist)
{
    uint dist;
    ulong a;
    uint j;

    for (uint id = get_group_id(0); id < sample; id += get_num_groups(0)) {
        const __global ulong *row = bitstrings + id*bs_len;

        dist = 0;
        j = get_local_id(0);
        if (j < bs_len) {
            a = row[j] ^ bs[j];
            dist = popcount(a);
        }
        partial_dist[get_local_id(0)] = dist;

        uint stride;
        for(stride = get_local_size(0)/2; stride > 32; stride /= 2) {
            barrier(CLK_LOCAL_MEM_FENCE);
            if (get_local_id(0) < stride) {
                partial_dist[get_local_id(0)] +=
                    partial_dist[get_local_id(0) + stride];
            }
        }
        barrier(CLK_LOCAL_MEM_FENCE);
        for(/**/; stride > 0; stride /= 2) {
            if (get_local_id(0) < stride) {
                partial_dist[get_local_id(0)] +=
                    partial_dist[get_local_id(0) + stride];
            }
        }

        if (get_local_id(0) == 0) {
            if (partial_dist[0] <= radius) {
                selected[atomic_inc(counter)] = id;
            }
        }
        barrier(CLK_LOCAL_MEM_FENCE);
    }
}
```

Listing 6: OpenCL kernel `single_scan5`. This kernel is just like `single_scan3`, but it explores one more detail of many GPU microarchitecture: the size of the warp. As the workers in the same warp are always synchronized, there is no need to sync them using a barrier. This specific kernel only works when the number of workers per workgroup is a power-of-2.

```
__kernel
void single_scan5_unroll(
        __constant const uchar *bitcount_table,
        __global const ulong *bitstrings,
        const uint bs_len,
        const uint sample,
        __constant const ulong *bs,
        const uint radius,
        __global uint *counter,
        __global uint *selected,
        __local uint *partial_dist)
{
    uint dist;
    ulong a;
    uint j;

    for (uint id = get_group_id(0); id < sample; id += get_num_groups(0)) {
        const __global ulong *row = bitstrings + id*bs_len;

        dist = 0;
        j = get_local_id(0);
        if (j < bs_len) {
            a = row[j] ^ bs[j];
            dist = popcount(a);
        }
        partial_dist[get_local_id(0)] = dist;

        for(uint stride = get_local_size(0)/2; stride > 32; stride /= 2) {
            barrier(CLK_LOCAL_MEM_FENCE);
            if (get_local_id(0) < stride) {
                partial_dist[get_local_id(0)] += partial_dist[get_local_id(0) + stride];
            }
        }

        // We do not need to sync because they all run in the same warp.
        if (get_local_id(0) < 32 && get_local_size(0) >= 64) {
            partial_dist[get_local_id(0)] += partial_dist[get_local_id(0) + 32];
        }
        if (get_local_id(0) < 16 && get_local_size(0) >= 32) {
            partial_dist[get_local_id(0)] += partial_dist[get_local_id(0) + 16];
        }
        if (get_local_id(0) < 8 && get_local_size(0) >= 16) {
            partial_dist[get_local_id(0)] += partial_dist[get_local_id(0) + 8];
        }
        if (get_local_id(0) < 4 && get_local_size(0) >= 8) {
            partial_dist[get_local_id(0)] += partial_dist[get_local_id(0) + 4];
        }
        if (get_local_id(0) < 2 && get_local_size(0) >= 4) {
            partial_dist[get_local_id(0)] += partial_dist[get_local_id(0) + 2];
        }

        if (get_local_id(0) == 0) {
            partial_dist[0] += partial_dist[1];
            if (partial_dist[0] <= radius) {
                selected[atomic_inc(counter)] = id;
            }
        }
        barrier(CLK_LOCAL_MEM_FENCE);
    }
}
```

Listing 7: OpenCL kernel `single_scan5_unroll`. This kernel is exactly like `single_scan5`, but it unrolls the last `for` since it has at most 5 loops.

```
__kernel
void single_scan6(
        __constant const uchar *bitcount_table,
        __global const ulong *bitstrings,
        const uint bs_len,
        const uint sample,
        __constant const ulong *bs,
        const uint radius,
        __global uint *counter,
        __global uint *selected,
        __local uint *partial_dist)
{
    uint dist;
    ulong a;
    uint j;

    for (uint id = get_group_id(0); id < sample; id += get_num_groups(0)) {
        const __global ulong *row = bitstrings + id*bs_len;

        dist = 0;
        j = get_local_id(0);
        if (j < bs_len) {
            a = row[j] ^ bs[j];
            dist = popcount(a);
        }
        partial_dist[get_local_id(0)] = dist;

        uint old_stride = get_local_size(0);
        uint stride;
        __local uint extra;
        extra = 0;
        for(stride = get_local_size(0)/2; stride > 32; stride /= 2) {
            barrier(CLK_LOCAL_MEM_FENCE);
            if ((old_stride&1) == 1 && get_local_id(0) == old_stride-1) {
                extra += partial_dist[get_local_id(0)];
            }
            if (get_local_id(0) < stride) {
                partial_dist[get_local_id(0)] +=
                    partial_dist[get_local_id(0) + stride];
            }
            old_stride = stride;
        }
        barrier(CLK_LOCAL_MEM_FENCE);
        for(/**/; stride > 0; stride /= 2) {
            if ((old_stride&1) == 1 && get_local_id(0) == old_stride-1) {
                extra += partial_dist[get_local_id(0)];
            }
            if (get_local_id(0) < stride) {
                partial_dist[get_local_id(0)] +=
                    partial_dist[get_local_id(0) + stride];
            }
            old_stride = stride;
        }

        if (get_local_id(0) == 0) {
            if (partial_dist[0] + extra <= radius) {
                selected[atomic_inc(counter)] = id;
            }
        }
        barrier(CLK_LOCAL_MEM_FENCE);
    }
}
```

Listing 8: OpenCL kernel `single_scan6`. This kernel is just like `single_scan5`, but it works with any number of workers per work. The tradeoff is an additional step in the parallel reduction algorithm.

locations is 1,000 and it would require at least one million writes before any overflow is possible. Note also that it would be more likely to saturate the memory before any overflow.

Anyway, counters may have overflow protection depending on compiling options. By default, there is no overflow check for performance reasons (and because it does not seem necessary).

12.4 READ AND WRITE OPERATIONS

The reading and writing operations are executed in two steps: first, the address space is swept looking for the activated addresses; then, the operation is performed in the counters. Reading operation assemblies the bitstring according to the counters of the activated addresses, while the writing operation changes the counters.

The iterated reading keeps reading until it gets exactly the same bitstring (or the number of maximum interactions has been reached), i.e., it performs $\eta_{i+1} = \text{read}(\eta_i)$ and stops when $\eta_{k+1} = \eta_k$. If the initial bitstring is inside the critical distance of η, it will converge to η, but, if it is not, it will diverge and reach the maximum number of iterations.

The framework has both Kanerva's read and Murilo's generalized read. The generalization brings a parameter z, which is the exponent. In this case, the results are floating points instead of integers, which considerably reduces performance. When $z = 1$, it is precisely as the Kanerva's read. When $z = 0$, it is the Chada's read. We also explored how SDM would behave for different values of z.

There is another particular read operation: the weighted reading. In the weighted reading, the value of the counters is multiplied by a weight which depends only on the distance between the reading address and the hard location address. The weight is retrieved from a lookup table of integers indexed by the distance. The remaining of the read operation is just the same.

There is also a weighted writing operation. In this case, the weight is applied when the counters are updated, i.e., if the weight is 2, the counters are increased twice when bits are 1, and decreased twice when bits are 0. Just as in the weighted reading, the weights depend only on the distance between the writing address and the hard location address. The weights are retrieved from a lookup table of integers indexed by the distance.

13 RESULTS (I): FRAMEWORK VALIDATION

The framework has been validated comparing its results with the expected results from Kanerva [54]. Thus, we run simulations which were then compared to the theoretical analysis conducted some decades ago.

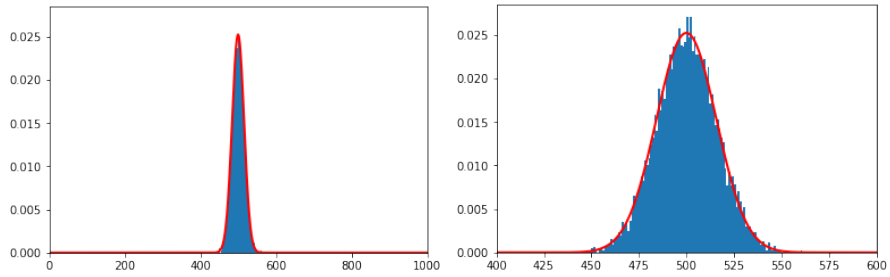
13.1 DISTANCE BETWEEN RANDOM BITSTRINGS

As shown by Kanerva [54], the distance between two bitstrings follows a binomial distribution with mean $\mu = n/2$ and standard deviation $\sigma = \sqrt{n}/2$. For large values of n, it may be approximated by a normal distribution with the same mean and standard deviation.

In order to validate our random bitstring generation algorithm, we have calculated 10,000 distances between two random bitstrings with $n = 1,000$ bits. In total, 20,000 random bitstrings have been generated during the simulation. The code is available in the "Distance between bitstrings" notebook [19].

In figure 19, we can notice that the theoretical model and the simulation results matches. Hence, it seems the random bitstring generation algorithm works appropriately.

This also validates the algorithm used to calculate the distance between two bitstrings. In this simulation, we have used the built-in __popcnt() function.



(a) Full histogram (b) Zoom in the interval $[400, 600]$

Figure 19: Histogram of 10,000 distances between two random bitstrings with 1,000 bits. The curve in red is the theoretical normal distribution with $\mu = 500$ and $\sigma = \sqrt{500}/2$.

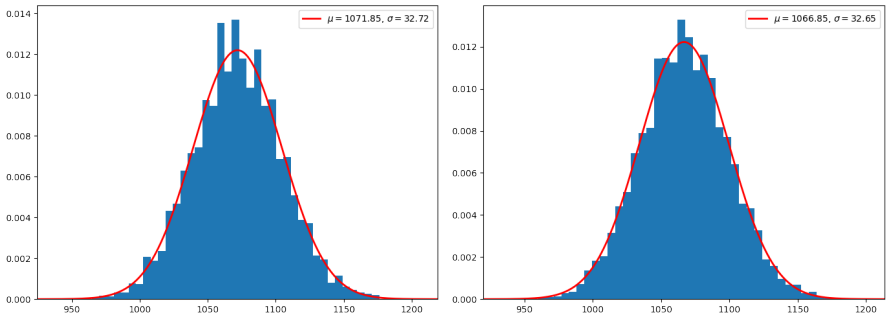
13.2 NUMBER OF ACTIVATED HARD LOCATIONS

In his seminal work, Kanerva proposed to use a sample of 1,000,000 hard locations in a 1,000 bits SDM. He also proposed to activate only 1,000 of them, on average. He calculated that an access radius of $r = 451$ would activate, on average, 0.00107185004892 of the whole space, or, in this case, 1,071.85 hard locations.

We extended his results, calculating the distribution of the number of activated hard locations. As each hard location has probability $p = 0.00107185004892$ of being activated, the probability of activating exactly a out of H hard locations follows a binomial distribution with mean $\mu = pH$ and standard deviation $\sigma = \sqrt{Hp(p-1)}$. In this case, $\mu = 1071.85$ and $\sigma = 32.72$.

In order to validate our scan algorithm, we have run 10,000 scans from a random bitstring and counted the number of activated hard locations. The code is available in the "Number of activated hard locations" notebook [19].

In figure 20, we can notice that the theoretical model and the simulation matches. Hence, it seems that both the address space generation algorithm and the scan algorithm work properly. Notice that the curve is almost the same for $n = 1,000$ and $n = 256$. This difference happens because the access radius is adjusted to get p as close as possible to 0.001, which mean they are very close, but not the same.



(a) $n = 1,000$, $H = 1,000,000$, $r = 451$, and $p = 0.00107185$

(b) $n = 256$, $H = 1,000,000$, $r = 103$, and $p = 0.00106684$

Figure 20: Histogram of the number of activated hard locations in 10,000 scans from a random bitstring. The curve in red is the theoretical normal distribution with $\mu = Hp$ and $\sigma = p(p-1)H$.

Besides the number of activated hard locations, we have also extended Kanerva's results to calculate the distribution of distances between the center of the circle and the activated hard locations. Let A be the set of activated hard locations, ξ be the center of the circle, and r be the access radius, then:

$$P(d(a,\xi) = x | a \in A) = \frac{P(d(a,\xi) = x)}{P(a \in A)} \tag{2}$$

$$= \frac{\binom{n}{x}}{\sum_{k=0}^{r} \binom{n}{k}} \tag{3}$$

In order to check Equation 3, we have calculated the distances of the activated hard locations to the center of 1,000 random circles. The code is available in the "Distances of activated hard locations" notebook [19].

In figure 21, we can notice that the theoretical model and the simulation matches.

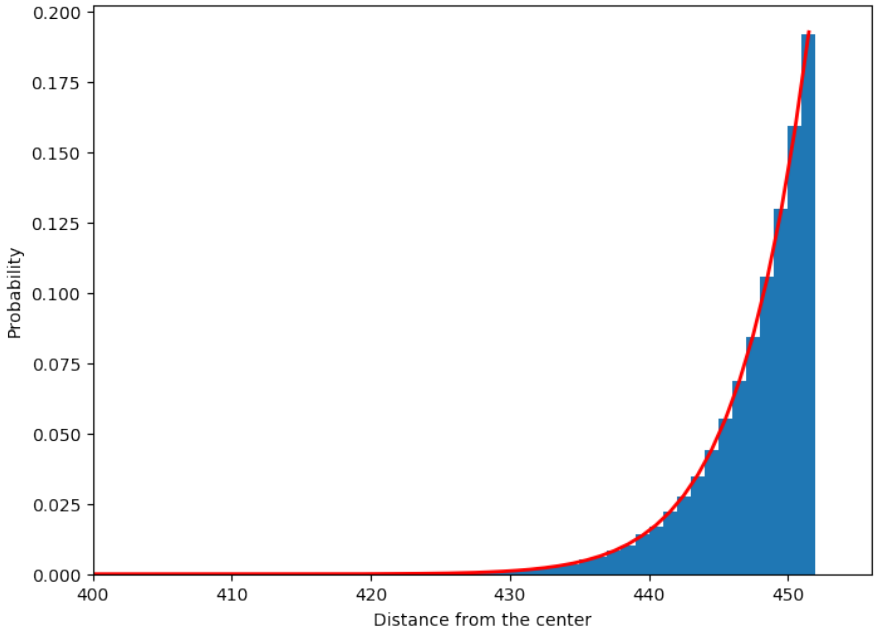


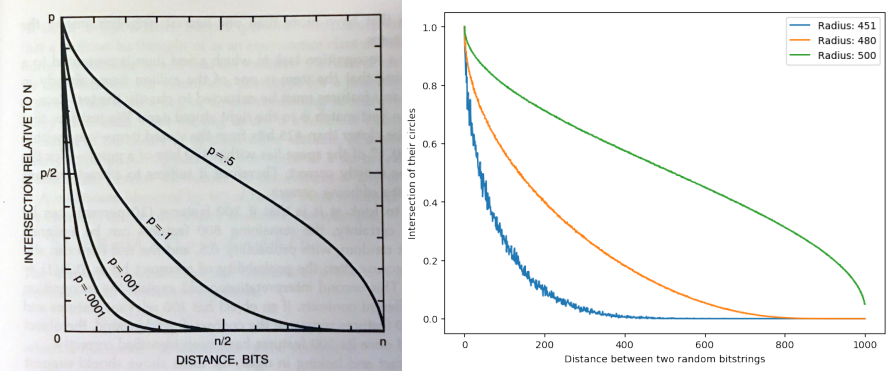
Figure 21: Histogram of the distances of activated hard locations to the center of the circles. The curve in red is the theoretical distribution of Equation 3

13.3 INTERSECTION OF TWO CIRCLES

Kanerva has calculated the intersection of two circles according to the distance between their centers. The intersection is important to understand how SDM works because it directly affects the critical distance. When η_d is inside the critical distance, then it will converge to η. In fact, it converges because they share a sufficient amount of hard locations, i.e., the intersection of the circle around η_d and η is enough to converge. For further information about the relation between the critical distance and the intersection, see Brogliato et al. [20].

We have calculated the intersection between a random bitstring (bs1) and another bitstring (bs2) exactly d bits away. The former (bs1) is just a random bitstring. The latter (bs2) was generated randomly flipping d bits of bs1. The code is available in the "Kanerva's Figure 1.2" notebook [19].

In Figure 22, we can notice that we have obtained the same results as Kanerva. It seems that the random flipping bits algorithm and the scan algorithm work properly.



(a) Kanerva [54, Figure 1.2, p.25] (b) Generated by SDM framework with $n = 1,000$

Figure 22: Number of hard locations in the intersection of circles around two bitstrings x bits away.

13.4 STORAGE AND RETRIEVAL OF SEQUENCES

Kanerva [54, Ch.8] presented an approach to store and retrieve sequences using k different SDMs, namely sdm_1, sdm_2, dots, sdm_k.

Let $a_0, a_1, a_2, \ldots, a_n$ be a sequence to be stored in a k-fold memory. So, all pointers of the form $a_i \rightarrow a_{i+k}$ will be written to sdm_k memory, i.e., in sdm_1, the following pointers will be written: $a_0 \rightarrow a_1$, $a_1 \rightarrow a_2$, $\ldots$, $a_{n-1} \rightarrow a_n$; while in sdm_2, the following pointers will be written: $a_0 \rightarrow a_2$, $a_2 \rightarrow a_3$, $\ldots$, $a_{n-2} \rightarrow a_n$; and so forth.

We have tested the same example presented in Kanerva [54], p.85. We wrote two sequences to a 3-fold memory: $< A, B, C, D >$ and $< E, B, C, F >$. Then, after reading the sequences $< A, B, C >$ and $< E, B, C >$, we have obtained D and F, respectively.

Each reading operation was performed summing the counters of all activated hard locations from all three memories. For instance, to read the sequence $< A, B, C >$, we have activated the hard locations around C in sdm_1, we have also activated the hard locations around D in sdm_2, and, finally, we have also activated the hard locations around A in sdm_3. After summing the counters of all those hard locations, we evaluate the resulting bitstring just as in the original read operation.

The code is available in the "Sequences (Kanerva Ch 8)" notebook [19].

The logic behind how it works is that, when reading the sequence $< A, B, C >$, we have A pointing to D, while both B and C point to D and F. Thus, D appeared more often than F and ended up being the result.

Hence, as we have replicated the theoretical results from Kanerva, we have one more evidence that our framework works appropriately.

13.4.1 k-*fold memory using only one SDM*

We have extended Kanerva's ideas to be able to store and retrieve sequences in k-fold memories using only one SDM (instead of k SDMs).

Our idea was to create k random bitstrings, one for each fold. We have performed writing and reading exactly as Kanerva's original idea, but, instead of writing to sdm_k, we have written a_{i+k} into the address $a_i \oplus \text{tag}_k$, and, instead of reading from sdm_k, we have read from address $a_i \oplus \text{tag}_k$, where $\oplus$ is the exclusive or (XOR) operator.

It worked as if we had split SDM into k regions with low intersection between two of them. So, as the interference is minimal, they work like independent SDMs. The major disadvantage of this approach is that memory capacity may be reached faster.

Splitting the memory into regions may be an interesting strategy to other sorts of problems, mostly the ones which would need many SDMs and, consequently, would use a lot of RAM.

14

RESULTS (II): CRITICAL DISTANCE

One particular analysis of Kanerva's interest is given by the limits of recovery. That is, given an item read at a distance x from a previously stored η, does this reading at a η_x recover the original? Suppose an SDM is trying to read an item written at η, but the cues received so far lead to a point of distance x from η. As one reads at η_x, a new bitstring β is obtained, leading to Kanerva's question: what is the new distance from η to β? Is it smaller or larger than x? That, of course, depends on the ratio between x and the number of dimensions of the memory.

Kanerva [54, p.70] originally predicted a ~500-bit distance after a point (Figure 23). The original prediction considered that the read distance would decline when inside the critical distance and increase afterward, converging to a ~500-bit distance. At this point, each read would lead to a different, orthogonal, ~500-bit distance bitstring. He analyzed specifically an SDM with 1,000 bits and 10,000 random bitstrings written into it.

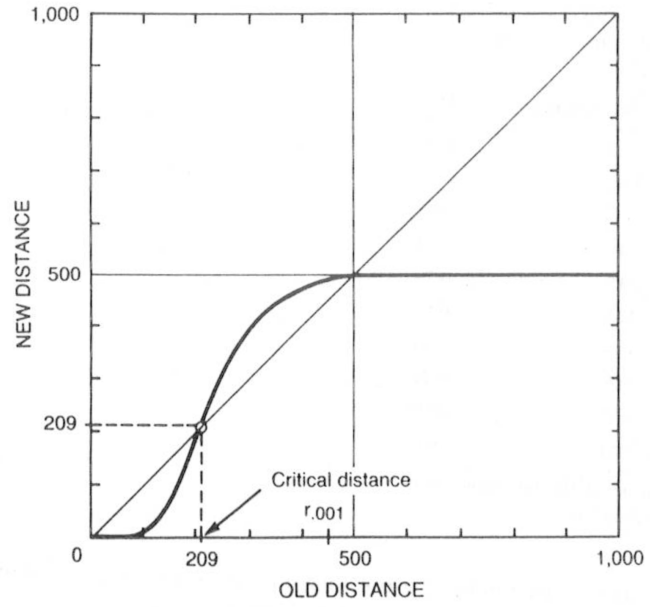
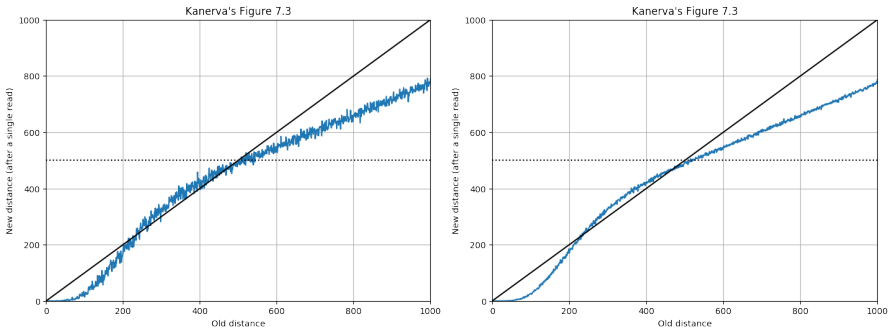


Figure 7.3
New distance to target as a function of old distance.

Figure 23: Kanerva's original Figure 7.3 (p. 70) predicting a ~500-bit distance after a point.

As we ran the simulations, this one, in particular, struck our attention: The new distances obtained after a read operation were not perfectly predicted by the theoretical model. We have strictly followed Kanerva's configuration and, even so, we have found out some deviations from Kanerva's original theoretical analysis and the results obtained by simulation.

In details, we have created a SDM with $n = 1,000$, $H = 1,000,000$, and $r = 451$. Then, we have generated 10,000 random bitstrings and written them into the memory. Then, we have generated a reference bitstring (bs_ref) and written it into the memory. Then, we have executed the following steps with x from 0 to 1,000: (i) copy bs_ref into a new bitstring; (ii) randomly flipped x bits of the copy; (iii) read from the memory in the copy address; and (iv) stored the distance between the returned bitstring and bs_ref. Finally, we have plotted Figure 24.



(a) 1 sample for each distance x (b) 6 samples for each distance x

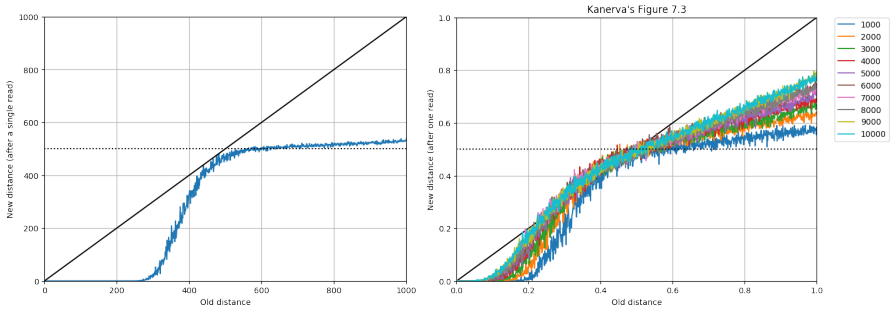
Figure 24: Results generated by the framework diverging from Kanerva's original Figure 7.3. Here we had a 1,000 bit, 1,000,000 hard location SDM with 10,000 random bitstrings written into it, which was also Kanerva's configuration.

Figure 24a has a lot of noise because we have read only once for each distance x and Kanerva has predicted the average distance. So, we have changed the steps to run k reads and store the average new distance. We run with $k = 6$, and the results can be seen in Figure 24b, which has much lower noise and still holds the divergence.

Our results show that the theoretical prediction is not accurate. There are interaction effects from one or more of the attractors created by the 10,000 writes, and these attractors seem to raise the distance beyond ~500 bits (Figure 24).

Obviously, these small deviations from Kanerva's original theoretical predictions deserve a qualification. Kanerva was working in the 1980s and the 1990s, and had no access to the immense computational power that we do today. It is no surprise that some small interaction effects should exist as machines allow us to explore the ideas of his monumental work.

However, when we reduced the number of random bitstrings written in the SDM from 10,000 to only 100, the results reflected very well the Kanerva's theoretical expectation (Figure 25a). This result strengthens our hypothesis that the disparities in the computational outcomes are due to the interaction effect of high numbers of different attractors. In Figure 25b we can notice that the more random bitstrings are written, the stronger the attractors.



(a) 100 writes

(b) Steps of 1,000 writes

Figure 25: Results generated by the framework similar to Kanerva's original Figure 7.3. Here we have a 1,000 bit, 1,000,000 hard location SDM with (a) just 100 random bitstrings written into it and (b) steps of 1,000 random bitstrings written into it.

To obtain the results from Figures 24 and 25, we had to write 10,000 random bitstrings to an SDM, and then randomly choose one of those bitstrings to be our origin. Finally, we randomly flipped some bits from the origin bitstring and executed a reading operation in the SDM. Thereby, in order to show the interaction effects more clearly, we changed the single read for a 15-iterative read. As we can see in Figure 26, after a distance of 500 bits, all bitstrings converged to 500-bit distance bitstrings, just as described by Kanerva.

Hence, our understanding is that the attractors are just preventing the bitstrings to converge directly to 500-bit distance bitstrings, requiring more reading steps to do so. They are in other orthogonal bitstrings' critical distance, but sufficiently far not to converge in a single read.

Going further in the analysis, we calculated the probability of missing a bit when reading from SDM. After all, that is how Kanerva has originally found the curve. To do this, we used the following equations from our previous work [20]. Let d be the distance to the target, h be the number of hard locations activated during reading and write operations, s be the number of total stored bitstrings, H be the number of total hard locations, w be the number of times the target was written into SDM, θ be the total random bitstrings in all h hard locations activated by read operation, and $\phi(d)$ be the average number of shared hard locations activated two bitstrings d bits away.

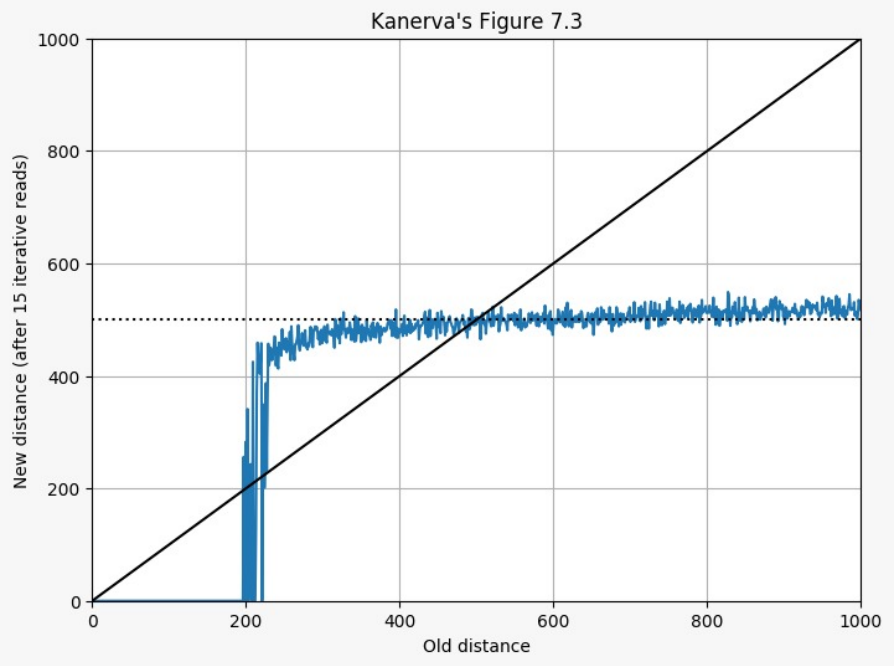


Figure 26: This graph shows the interaction effects more clearly. As we change the single read to a 6-iterative read, the effect has vanished, and all bitstrings above $x = 500$ have converged to 500-bit distance bitstrings. Here we have precisely the same configuration of Figure 24, except for the iterative read.

$$\theta = \frac{sh^2}{H} - w \cdot \phi(d) \tag{4}$$

$$P(\text{miss}|\text{bit} = 0) = 1 - P\left(\sum_{i=1}^{\theta} X_i < \frac{sh^2}{2H}\right) \tag{5}$$

$$P(\text{miss}|\text{bit} = 1) = P\left(\sum_{i=1}^{\theta} X_i < \frac{sh^2}{2H} - w \cdot \phi(d)\right) \tag{6}$$

$$P(\text{miss}) = \frac{1}{2} \cdot [P(\text{miss}|\text{bit} = 0) + P(\text{miss}|\text{bit} = 1)] \tag{7}$$

For details and the proof of this equation, see Brogliato et al. [20]. Although Kanerva has found a formula for $\phi(d)$ through an unsolved integral, and de Pádua Braga and Aleksander [32] have proposed another way to calculate $\phi(d)$, we have used our framework to estimate the values of d. In order to do that, we used a Monte Carlo approach, generating many pairs of random bitstrings d bits away from them and calculating the average number of shared hard locations between them. The code is available in the "Calculate critical distance" notebook [19].

Kanerva's settings according to the parameters of the equation were: $s = 10,000$, $H = 1,000,000$, and $w = 1$. We have calculated $\phi(d)$ as explained, and $h = H \cdot 2^{-n} \sum_{i=0}^{r} \binom{n}{i}$, where $n = 1,000$ and

$r = 451$. Finally, $h = 1,071.85$ and changing d from 0 to 1000, we got Figure 27.

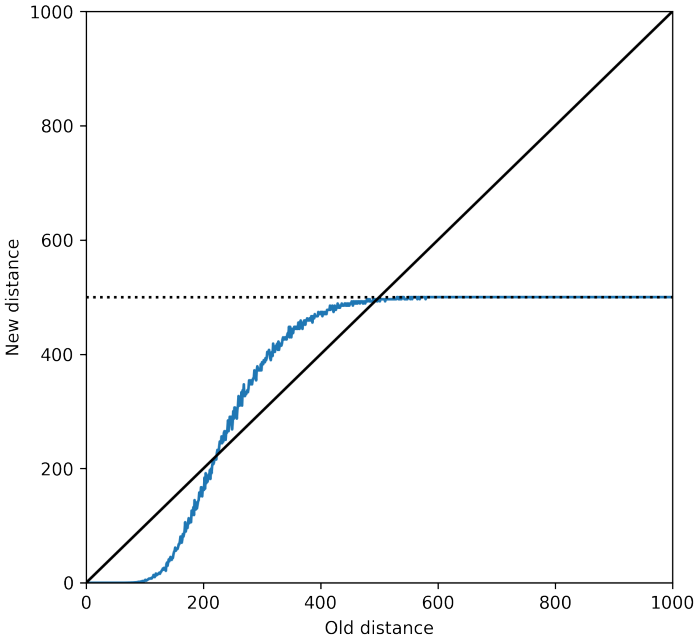


Figure 27: Kanerva's original Figure 7.3 generated using the equations from Brogliato et al. [20].

As one can easily notice, we have got exactly the same curve as Kanerva. Both his and our model expect that, after reading, say, from 550 bits of distance from a written bitstring, we should obtain the expected $n/2$ equator distance. This question has intrigued us, and here we look for a more analytic explanation than merely interference from the other written attractors. Let us turn back to mathematics to study this anomaly.

14.1 A DEVIATION FROM THE EQUATOR DISTANCE?

Kanerva writes[1]:

> You have done an incredibly thorough analysis of SDM. I like the puzzle in your message and believe that your simulations are correct and to be learned from. So what to make of the difference compared to my Figure 7.3 (and your Figure 27)? I think the difference comes from my not having accounted fully for the effect of the other 9,999 vectors that are stored in the memory. You say in it

1 Email thread 'SDM: A puzzling issue and an invitation', started March 16th 2018, in which we discussed the aforementioned discrepancy. To think that some centuries ago, all scientific publishing was the exchange of such letters.

> "Our results show that the theoretical prediction is not accurate. There are interaction effects from one or more of the attractors created by the 10,000 writes, and these attractors seem to raise the distance beyond 500 bits (Figure 24)."
>
> I think that is correct. It also brings to mind a comment Louis Jaeckel made when we worked at NASA Ames. He pointed out that autoassociative storage (each vector is stored with itself as the address) introduces autocorrelation that my formula for Figure 7.2 did not take into account. When we read from memory, each stored vector exerts a pull toward itself, which also means that each bit of a retrieved vector is slightly biased toward the same bit of the read address, regardless of the read address. We never worked out the math.

This is an important observation. A hard location is activated because it shares many dimensions with the items read from or written onto it. Imagine the 'counter's eye view': each individual counter 'likes' to write on its own corresponding bit-address value more than it likes the opposite; as each hard-location has a say in its own area — and nowhere else.

Let x and y be random bitstrings and n be the number of dimensions in the memory; let x_i and y_i be the i-th bit of x and y, respectively; and $d(x, y)$ be the Hamming distance. Whilst the probability of a shared bit-value between same dimension-bits in two random addresses is $1/2$, an address only activates hard-locations close to it. Let us call these shared bitvalues a *bitmatch in dimension* i.

So, what is the probability of bitmatches given that we know the access radius r between the address and a hard-location?

Theorem 13. Each dimension has a small pull bias, which can be measured by $P(x_i = y_i | d(x, y) \leqslant r) = \frac{\sum_{k=0}^{r} \binom{n-1}{k}}{\sum_{k=0}^{r} \binom{n}{k}}$.

Proof. The left-hand expression $P(x_i = y_i | d(x, y) \leqslant r)$ computes the probability of a bitmatch in i, given that we know that x and y are in the access radius defined by r, i.e., $d(x, y) \leqslant r$.

Applying the law of total probability to the left-hand expression we obtain

$$\sum_{k=0}^{r} P(x_i = y_i | d(x, y) = k \leqslant r) P(d(x, y) = k | d(x, y) \leqslant r) \tag{8}$$

We also know that

$$P(x_i = y_i | d(x,y) = k) = \frac{n-k}{n} \tag{9}$$

$$P(d(x,y) = k | d(x,y) \leqslant r) = \frac{\binom{n}{k}}{\sum_{j=0}^{r} \binom{n}{j}} \tag{10}$$

Hence,

$$P(x_i = y_i | d(x,y) \leqslant r) = \frac{\sum_{k=0}^{r} \frac{n-k}{n} \binom{n}{k}}{\sum_{j=0}^{r} \binom{n}{j}} \tag{11}$$

Finally, the combinatorial identity

$$\frac{n-k}{n}\binom{n}{k} = \frac{(n-k)}{n}\frac{n!}{(n-k)!k!} = \frac{(n-1)!}{k!(n-1-k)!} = \binom{n-1}{k} \tag{12}$$

closes the theorem.

□

Theorem 13 is valid for both "x written at x" (autoassociative memory) and "random written at x" (heteroassociative memory). When $n = 1,000$ and $r = 451$, $P(x_i = y_i | d(x,y) \leqslant r) = p = 0.552905498137$. Each bit of a hard location does indeed have a small pull bias. What is meant by this is that each particular dimension has a small preference toward positive values if its address bit is set to 1, and negative values if set to 0.

So far we have looked only at a single pair of bitstrings, the probability of a single bitmatch between bitstrings within the access radius distance. Now let us consider the number of activated hard locations exhibiting this bitmatch.

Let h be the number of activated hard locations. As the probability of activating a specific hard location is constant, $h \sim \text{Binomial}(H, p_1)$. Thus, $\mathbf{E}[h] = \mu_h = Hp_1$ and $\mathbf{V}[h] = \sigma_h^2 = Hp_1(1-p_1)$, where $p_1 = 2^{-n}\sum_{k=0}^{r}\binom{n}{k}$.

Let Z be the number of activated hard locations with the same bit as the reading address. Then, $Z = \sum_{i=1}^{h} X_i$, where $X_i \sim \text{Bernoulli}(p)$, where $p = P(x_i = y_i | d(x,y) \leqslant r)$.

Theorem 14. *Given a reading address x and a dimension i, the number of activated hard-locations with bitmatches at i follows a normal distribution with $\mathbf{E}[Z] = \mu_Z = p\mu_h$ and $\mathbf{V}[Z] = \sigma_Z^2 = p(1-p)\mu_h + p^2\sigma_h^2$.*

Proof. As $P(973 < h < 1170) = 0.997$, by the central limit theorem, Z may be approximated by a normal distribution.

By the central limit theorem, Z is normally distributed.

Applying the law of total averages and the law of total variance, $\mathbf{E}[Z] = \mathbf{E}[\mathbf{E}[Z|h]] = \mathbf{E}[ph] = p\mathbf{E}[h] = ph$, and $\mathbf{V}[Z] = \mathbf{E}[\mathbf{V}[Z|h]] + \mathbf{V}[\mathbf{E}[Z|h]] = \mathbf{E}[hp(1-p)] + \mathbf{V}[ph] = p(1-p)\mathbf{E}[h] + p^2\mathbf{V}[h] = hp(1-p) + p^2Hp_1(1-p_1)$.

Applying the law of total variance, $\mathbf{V}[Z] = \mathbf{E}[\mathbf{V}[Z|h]] + \mathbf{V}[\mathbf{E}[Z|h]] = \mathbf{E}[hp(1-p)] + \mathbf{V}[ph] = p(1-p)\mathbf{E}[h] + p^2\mathbf{V}[h] = p(1-p)\mu_h + p^2\sigma_h^2$. □

See Figure 28 for a comparison between the theoretical model and a simulation.

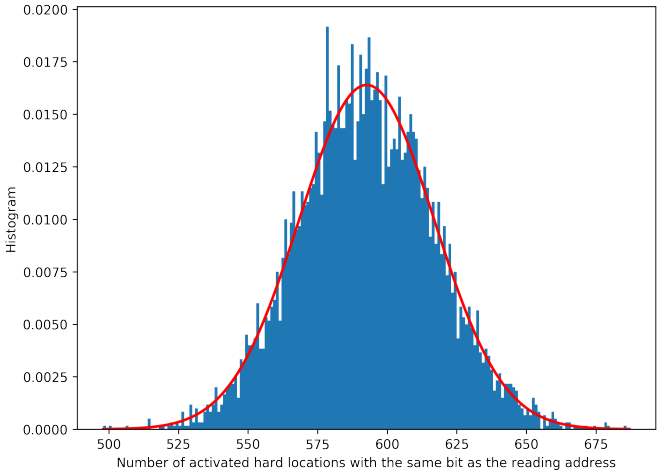


Figure 28: Given an address x and a dimension i, how many hard locations with bitmatches in i are activated by reading at x? The histogram was obtained through numerical simulation. The red curve is the theoretical normal distribution found in Theorem 14.

14.2 COUNTER BIAS

The bias begins in the counters. Let's analyze the ith counter of a hard location.

Let s be the number of bitstrings written into memory (in our case, $s = 10,000$) and addr_i be the ith bit of the hard location's address.

Let θ be the average number of bitstrings written in each hard location. As there are s bitstrings written into the memory, and the probability of activating a specific hard location is constant, $\theta \sim \text{Binomial}(s, p_1)$. Thus, $\mathbf{E}[\theta] = \mu_\theta = sp_1$ and $\mathbf{V}[\theta] = \sigma_\theta^2 = sp_1(1-p_1)$.

Let Y_i be the number of bitmatches in the $i$bit of a hard location's address after s written bitstrings. Then, $Y_i = \sum_{k=1}^{\theta} X_k$.

Theorem 15. *Giving the number of written bitstrings* s, $\mathbf{E}[Y_i] = \mu_Y = p\mu_\theta$ *and* $\mathbf{V}[Y_i] = \sigma_Y^2 = p(1-p)\mu_\theta + p^2\sigma_\theta^2$.

Proof. Applying the law of total expectation, $\mathbf{E}[Y] = \mathbf{E}[\mathbf{E}[Y|\theta]] = \mathbf{E}[p\theta] = p\mathbf{E}[\theta] = p\mu_\theta$.

Applying the law of total variance, $\mathbf{V}[Y] = \mathbf{E}[\mathbf{V}[Y|\theta]] + \mathbf{V}[\mathbf{E}[Y|\theta]] = \mathbf{E}[\theta p(1-p)] + \mathbf{V}[p\theta] = p(1-p)\mathbf{E}[\theta] + p^2\mathbf{V}[\theta] = p(1-p)\mu_\theta + p^2\sigma_\theta^2$. □

During a write operation, the counters are incremented for every bit 1 and decremented for every bit 0. So, after s writes, there will be θ bitstrings written in each hard location with Y_i bitmatches and $\theta - Y_i$ non-bitmatches. Thus, $[\text{cnt}_i|\text{addr}_i = 1] = (Y_i) - (\theta - Y_i) = 2Y_i - \theta$ and $[\text{cnt}_i|\text{addr}_i = 0] = \theta - 2Y_i$.

Theorem 16. $\mathbf{E}[cnt_i|addr_i = 1] = \mu_{cnt} = (2p-1)\mu_\theta$ *and* $\mathbf{V}[cnt_i|addr_i = 1] = \sigma^2_{cnt} = 4p(1-p)\mu_\theta + (2p-1)^2\sigma^2_\theta$.

Proof. $\mathbf{E}[\text{cnt}_i|\text{addr}_i = 1] = \mathbf{E}[2Y_i - \theta] = \mathbf{E}[2Y_i] - \mathbf{E}[\theta] = 2\mathbf{E}[Y_i] - \mu_\theta = 2p\mu_\theta - \mu_\theta = (2p-1)\mu_\theta$.

Applying the law of total variance, $\mathbf{V}[\text{cnt}_i|\text{addr}_i = 1] = \mathbf{V}[2Y_i - \theta] = \mathbf{E}[\mathbf{V}[2Y_i - \theta|\theta]] + \mathbf{V}[\mathbf{E}[2Y_i - \theta|\theta]]$.

Let us solve each part independently. Thus,
$\mathbf{V}[2Y_i - \theta|\theta] = \mathbf{V}[2Y_i|\theta] = 4\mathbf{V}[Y_i|\theta] = 4\mathbf{V}[\sum_{k=1}^{\theta} X_k] = 4\theta p(1-p)$.
$\mathbf{E}[\mathbf{V}[2Y_i - \theta|\theta]] = \mathbf{E}[4\theta p(1-p)] = 4p(1-p)\mathbf{E}[\theta] = 4p(1-p)\mu_\theta$.
Finally,
$\mathbf{E}[2Y_i - \theta|\theta] = 2\mathbf{E}[Y_i|\theta] - \mathbf{E}[\theta|\theta] = 2p\theta - \theta = (2p-1)\theta$.
$\mathbf{V}[\mathbf{E}[2Y_i - \theta|\theta]] = \mathbf{V}[(2p-1)\theta] = (2p-1)^2\mathbf{V}[\theta] = (2p-1)^2\sigma^2_\theta$. □

Theorem 17. $\mathbf{E}[cnt_i|addr_i = 0] = -\mu_{cnt}$ *and* $\mathbf{V}[cnt_i|addr_i = 1] = \sigma^2_{cnt}$.

Proof. Notice that $[\text{cnt}_i|\text{addr}_i = 0] = -[\text{cnt}_i|\text{addr}_i = 1]$. Thus, $\mathbf{E}[\text{cnt}_i|\text{addr}_i = 0] = -\mathbf{E}[\text{cnt}_i|\text{addr}_i = 1]$ and $\mathbf{V}[\text{cnt}_i|\text{addr}_i = 0] = \mathbf{V}[\text{cnt}_i|\text{addr}_i = 1]$. □

In summary,

$$[\text{cnt}_i|\text{addr}_i = 1] \sim \mathcal{N}(\mu_{\text{cnt}}, \sigma^2_{\text{cnt}}) \tag{13}$$

$$[\text{cnt}_i|\text{addr}_i = 0] \sim \mathcal{N}(-\mu_{\text{cnt}}, \sigma^2_{\text{cnt}}) \tag{14}$$

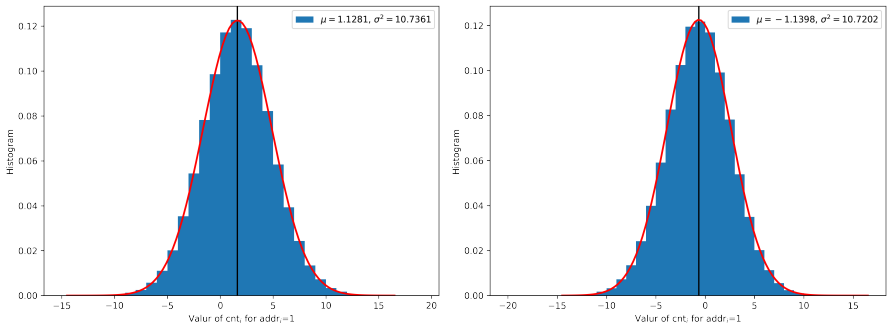
In our case, $p = 0.5529$, $s = 10{,}000$, and $H = 1{,}000{,}000$, so $[\text{cnt}_i|\text{addr}_i = 1] \sim \mathcal{N}(\mu = 1.1341, \sigma^2 = 10.7184)$. For "random at x", $p = 0.5$, so $\mu = 0$ and $\sigma^2 = 10.7185$. See Figure 29.

Finally,

$$P(\text{cnt}_i > 0|\text{addr}_i = 1) = P(\text{cnt}_i < 0|\text{addr}_i = 0) = 1 - \mathcal{N}.\text{cdf}(0) \tag{15}$$

For "random written at x", $p = 0.5$ implies $\mu_{\text{cnt}} = 0$, which implies $P(\text{cnt}_i > 0|\text{addr}_i = 1) = P(\text{cnt}_i < 0|\text{addr}_i = 0) = 0.5$, independently of the parameters because they will only affect the variance and the normal distribution is symmetrical around the average.

However, for "x written at x", $p = 0.5529$ and the probabilities depend on s. For $s = 10{,}000$, they are equal to 0.6354. For $s = 20{,}000$,



(a) $\text{addr}_i = 1$ (b) $\text{addr}_i = 0$

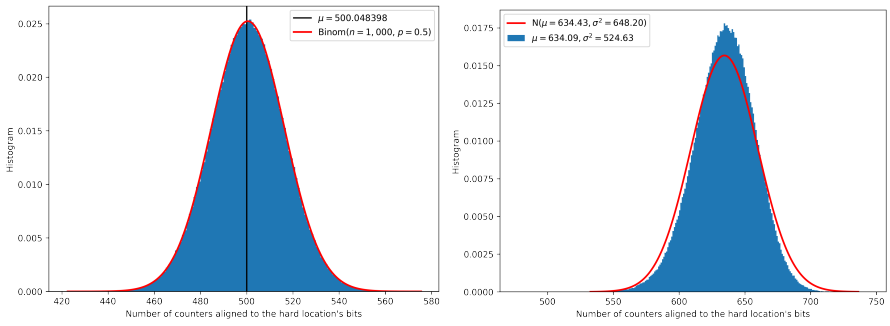
Figure 29: The value of the counters after $s = 10,000$ writes shows the autocorrelation in the counters in autoassociative memories ("x at x"). The histogram was obtained through simulation. The red curve is the theoretical normal distribution found in equations (13) and (14).

they are equal to 0.6867. For $s = 30,000$, they are equal to 0.7232. The more random bitstrings are written into the memory, the more the hard locations point to themselves.

Let D be the number of counters aligned with addr_i. The standard deviation was calculated using the fact that $[D|\theta] \sim \text{Binomial}(1000, q = P(\text{cnt}_i > 0|\text{addr}_i = 1, \theta))$.

Applying the law of total variance, $\mathbf{V}[D] = \mathbf{E}[\mathbf{V}[D|\theta]] + \mathbf{V}[\mathbf{E}[D|\theta]] = \mathbf{E}[1000q(1-q)] + \mathbf{V}[1000q] = 1000\mathbf{E}[q - q^2] + 1000^2\mathbf{V}[q] = 1000\mathbf{E}[q](1 - \mathbf{E}[q]) + 1000(1000-1)\mathbf{V}[q]$, where $\mathbf{E}[q] = \sum_\theta P(\text{cnt}_i > 0|\text{addr}_i = 1, \theta)P(\theta)$ and $\mathbf{E}[q^2] = \sum_\theta [P(\text{cnt}_i > 0|\text{addr}_i = 1, \theta)]^2 P(\theta)$.

Doing the math, $\mathbf{E}[q] = 0.402922$ and $\mathbf{E}[q^2] = 0.634433$. Thus, $\mathbf{V}[q] = \mathbf{E}[q^2] - (\mathbf{E}[q])^2 = 0.0004166$. Hence, $\mathbf{V}[D] = 648.2041$. See Figure 30 and notice that I still have to figure out why the mean is correct, but the standard deviation is not.



(a) "random at x" (b) "x at x"

Figure 30: Autocorrelation in the counters in autoassociative memories ("x written at x"). The histogram was obtained through simulation. The red curve is the theoretical distribution.

14.3 READ BIAS

Now that we know the distribution of $\text{cnt}_i|\text{addr}_i$, we may go to the read operation. During the read operation, on average, h hard locations are activated and their counters are summed up. So, for the ith bit,

$$\text{acc}_i = \sum_{k=1}^{h} \text{cnt}_k \tag{16}$$

Let η be the reading address and η_i the ith bit of it. Then, let's split the h activated hard locations into two groups: (i) the ones with the same bit as η_i with Z hard locations, and (ii) the ones with the opposite bit as η_i with $h - Z$ hard locations.

$$[\text{acc}_i|\eta_i] = \sum_{k=1}^{Z} [\text{cnt}_k|\text{addr}_k = \eta_i] + \sum_{k=1}^{h-Z} [\text{cnt}_k|\text{addr}_k \neq \eta_i] \tag{17}$$

Each sum is a sum of normally distributed random variables, so

$$\sum_{k=1}^{Z} [\text{cnt}_k|\text{addr}_k = \eta_1] \sim \mathcal{N}(\mu = \mu_{\text{cnt}}\mu_Z, \sigma^2 = \sigma^2_{\text{cnt}}\mu_Z + \mu^2_{\text{cnt}}\sigma^2_Z) \tag{18}$$

$$\sum_{k=1}^{h-Z} [\text{cnt}_k|\text{addr}_k \neq \eta_1] \sim \mathcal{N}(\mu = -\mu_{\text{cnt}}(1-p)\mu_h, \sigma^2 = \sigma^2_{\text{cnt}}(1-p)\mu_h + \mu^2_{\text{cnt}}\sigma^2_{h-Z}) \tag{19}$$

In our case, $\sum_{k=1}^{Z} [\text{cnt}_k|\text{addr}_k = 1] \sim \mathcal{N}(\mu = 672.12, \sigma^2 = 7113.87)$, and $\sum_{k=1}^{Z} [\text{cnt}_k|\text{addr}_k = 1] \sim \mathcal{N}(\mu = -543.49, \sigma^2 = 5752.54)$. See Figure 31 — we can notice that the average is correct but the variance is too small.

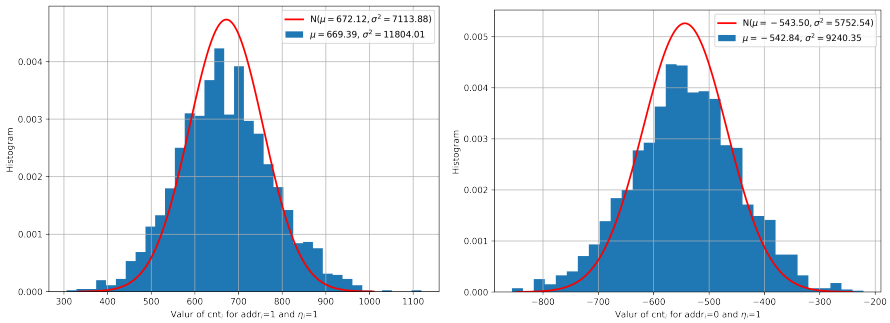
Hence,

$$[\text{acc}_i|\eta_i = 1] \sim \mathcal{N}(\mu = (2p-1)^2\mu_\theta\mu_h, \sigma^2 = \sigma^2_{\text{cnt}}\mu_h + 2\mu^2_{\text{cnt}}\sigma^2_h) \tag{20}$$

$$[\text{acc}_i|\eta_i = 0] \sim \mathcal{N}(\mu = -(2p-1)^2\mu_\theta\mu_h, \sigma^2 = \sigma^2_{\text{cnt}}\mu_h + 2\mu^2_{\text{cnt}}\sigma^2_h) \tag{21}$$

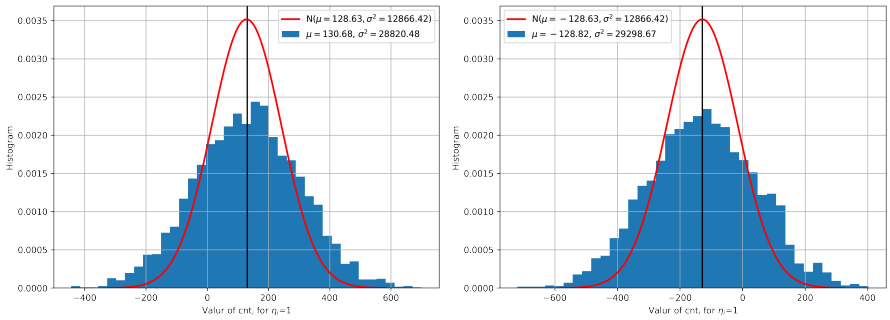
In our case, $[\text{acc}_i|\eta_i = 1] \sim \mathcal{N}(\mu = 128.62, \sigma^2 = 12865.69)$, and $[\text{acc}_i|\eta_i = 0] \sim \mathcal{N}(\mu = -128.62, \sigma^2 = 12865.69)$. See Figure 32 — we can notice that the variance issue from Figure 31 has propagated to these images.

Finally,



(a) Equation 18 ($addr_k = 1$) (b) Equation 19 ($addr_k = 0$)

Figure 31: The histogram was obtained through simulation. The red curve is the theoretical normal distribution.



(a) Equation 20 ($\eta_k = 1$) (b) Equation 21 ($\eta_k = 0$)

Figure 32: The histogram was obtained through simulation. The red curve is the theoretical normal distribution.

$$P(wrong) = P(acc_i < 0|\eta_i = 1) \cdot P(\eta_i = 1) + P(acc_i > 0|\eta_i = 0) \cdot P(\eta_i = 0) \tag{22}$$

$$= \frac{\mathcal{N}_{\eta_i=1}.cdf(0)}{2} + \frac{1 - \mathcal{N}_{\eta_i=0}.cdf(0)}{2} \tag{23}$$

$$= \frac{\mathcal{N}_{\eta_i=1}.cdf(0)}{2} + \frac{\mathcal{N}_{\eta_i=1}.cdf(0)}{2} \tag{24}$$

$$= \mathcal{N}_{\eta_i=1}.cdf(0) \tag{25}$$

Using the empirical variance of $\sigma^2 = 27838.3029124$, we calculate $P(wrong) = 0.22037771219874325$.

In order to check this probability, I have run a simulation reading from 1,000 random bitstrings (which have never been written into memory) and calculate the distance from the result of a single read. As the $P(wrong) = 0.22037$, I expected to get an average distance of 220.37 with a standard deviation of 13.10. See Figure 33 for the comparison between the simulated and the theoretical outcomes.

Figure 34 shows the new distance between η_d and $read(\eta_d)$, where η_d is d bits away from η. As for $d \geqslant 520$ there is no intersection

between η and η_d, our models applies and explains the horizontal line around distance 220.

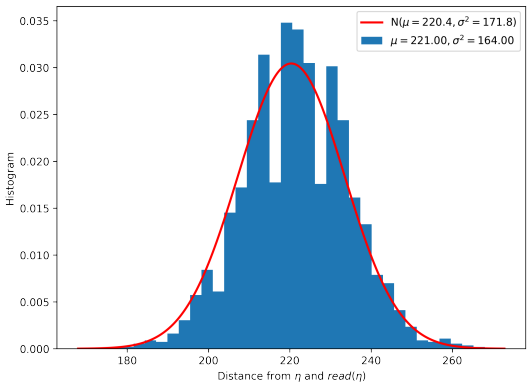


Figure 33: The histogram was obtained through simulation. The red curve is the theoretical normal distribution.

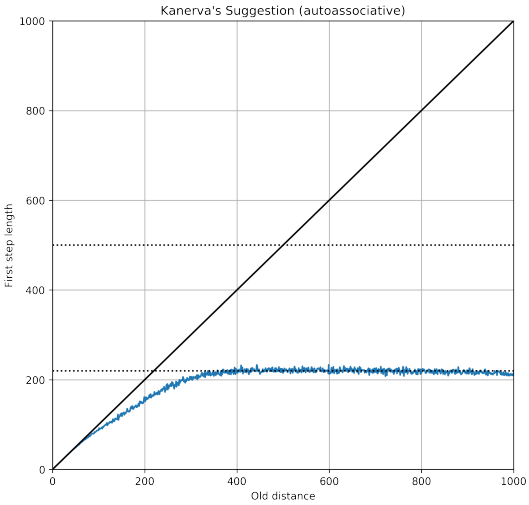


Figure 34: New distance after a single read operation in a bitstring η_d, which is d bits away from η. The new distance was calculated between η_d and $\text{read}(\eta_d)$. Notice that when $d \geqslant 520$, the intersection between η and η_d is zero, which means there is only random bitstrings written into the activated hard locations. The distance 220 equals $1000 \cdot 0.220$ which is the probability find in Figure 33.

14.4 CRITICAL DISTANCE OF 209

The critical distance is defined as d where $P(\text{miss}) = d/n$, or, in Figure 27, the point where the curve meets with the identity function (the black diagonal line). Thus, we plot a zoom-in of Figure 27 around $d = 209$ in Figure 35 using the same equations [20]. It was surprising that the meeting does not happen at $d = 209$, but around $d = 221$.

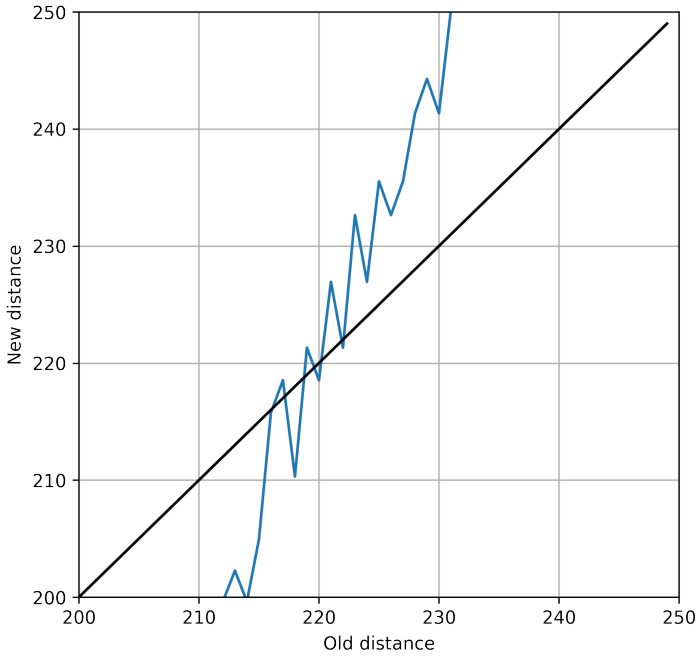


Figure 35: Zoom-in around $d = 209$ of Figure 27.

To confirm that the critical distance is not around 209, but around 221, we also plot a zoom-in of Figure 24 around $d = 209$ in Figure 36. In order to reduce the noise, we increased the samples to $k = 180$.

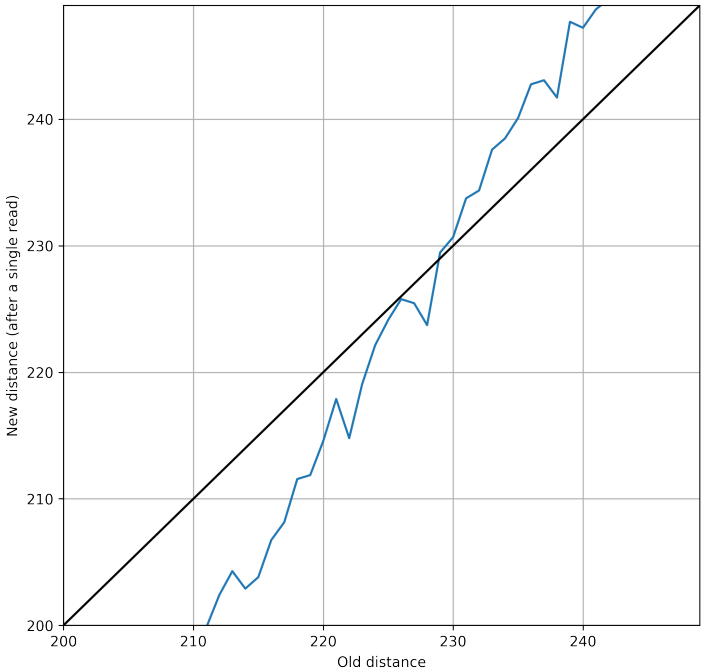


Figure 36: Zoom-in around $d = 209$ of Figure 24.

15

RESULTS (III): LOSS OF NEURONS

In SDM, the data is written distributed among millions of hard locations, which theoretically gives SDM robustness against loss of neurons. In other words, SDM should keep converging correctly even when some neurons are dead. The question is: how robust it really is? How many neurons may die before it starts to forget things? These questions have never been addressed before.

Looking for answers to these questions, we run simulations in which we kept killing some neurons and checking whether SDM remained converging to a given bitstring or not. In these simulations, 10,000 random bitstrings were written to a 1,000-bit SDM with 1,000,000 hard locations, and we choose one of them as our target. As the bitstrings were all written exactly once, we may generalize the results. The code is available in the "Resetting hard locations" notebook [19].

As neurons are hard locations in SDM, when we say that a neuron has been killed, we mean that its counters have been zeroed and a new random bitstring address has been assigned. During our simulations, no other bitstring has been written after the 10,000. Consequently, as their counters will remain zero, it is exactly like ignoring the dead hard locations in the subsequent reading operations.

In Figure 37, we can notice that SDM is robust up to 200,000 neuron deaths which are 20% of all hard locations. Its robustness is astonishing. In fact, SDM begins to be significantly affected by the loss of neurons after 600,000 neuron deaths (Figure 38) and obviously forgets everything when all neurons are dead.

It is interesting that 500,000 neuron deaths have a minor effect on SDM's recall capability (see Figure 39). It is analogous to do a hemispherectomy in a person and, after the procedure, the person being able to recall and learn almost just like before. In fact, there are clinical reports of children submitted to hemispherectomy who live an almost normal life with minor function problems.

An important observation is that around 800,000 neuron deaths (80% of all neurons) the critical distance becomes small, i.e., SDM recall capacity is hugely diminished. After 900,000 neuron deaths, the critical distance is zero, and everything has been lost.

Although there is some decrease in SDM recall after 600,000 neuron deaths, it is curious that there is a sudden change between 900,000 (90%) and 1,000,000 (100%). In Figure 40 we can see the details of this

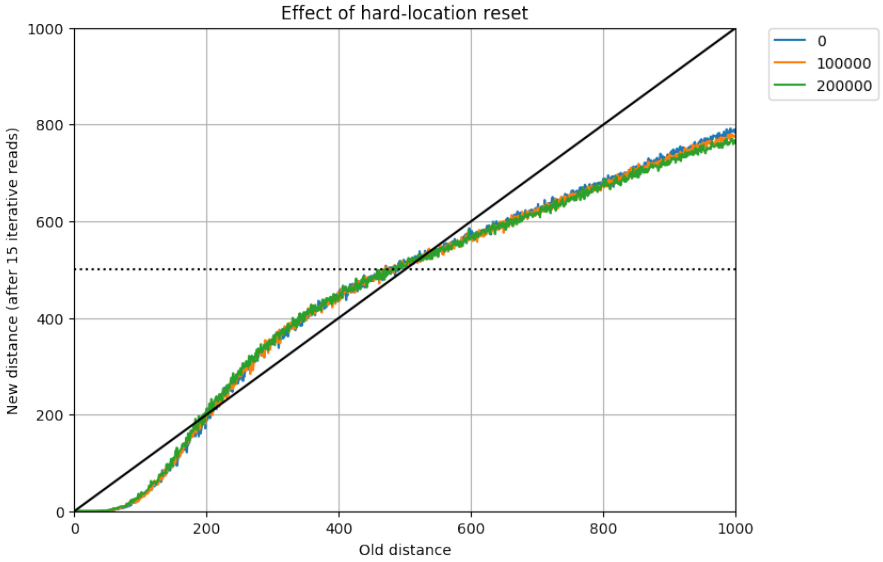


Figure 37: This graph shows the SDM's robustness against loss of neurons in a SDM with $n = 1,000$ and $H = 1,000,000$. It shows that a loss of 200,000 neurons, 20% of the total, does not seem to affect SDM whatsoever.

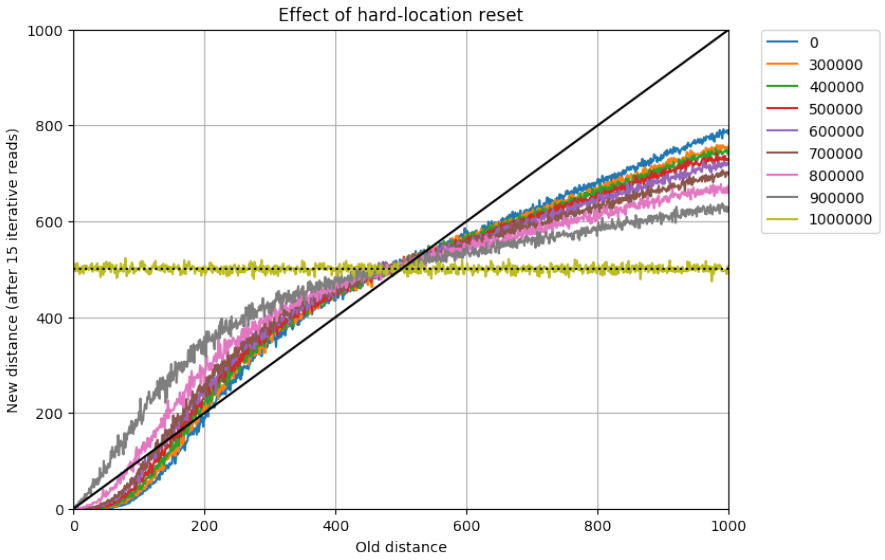


Figure 38: This graph shows the SDM's robustness against loss of neurons in a SDM with $n = 1,000$ and $H = 1,000,000$. The more neurons are lost, the smaller the critical distance, i.e., the worse the SDM recall.

non-linear change. Notice that after 950,000 even the exact clue η_0 does not converge to η.

We run the same simulation for a 256-bit SDM with 1,000,000 hard locations. The results were even more surprising, as the 256-bit SDM seems to be more robust to loss of neurons than the 1,000-bit SDM (see Figure 41). Notice that the loss of 50% of neurons barely affected the 256-bit SDM which remained functional even facing an enormous loss of 90% of neurons.

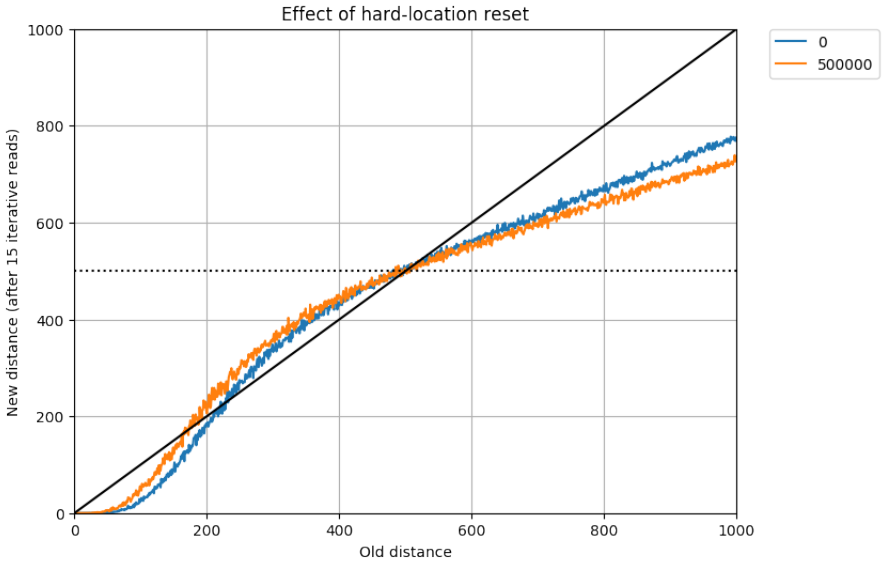


Figure 39: This graph shows the SDM's robustness against loss of neurons in a SDM with $n = 1,000$ and $H = 1,000,000$. Even when 50% of neurons are dead, SDM recall is barely affected, which is an impressive result and matches with some clinical results of children submitted to hemispherectomy.

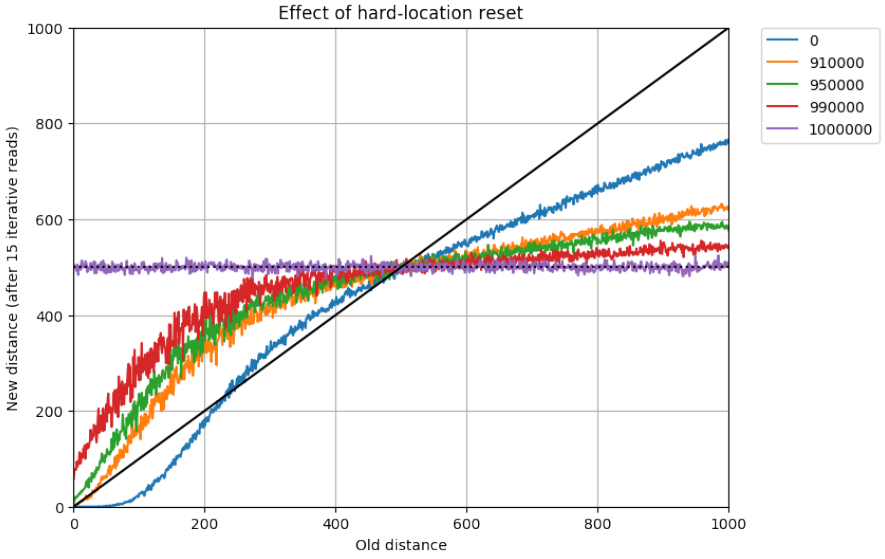
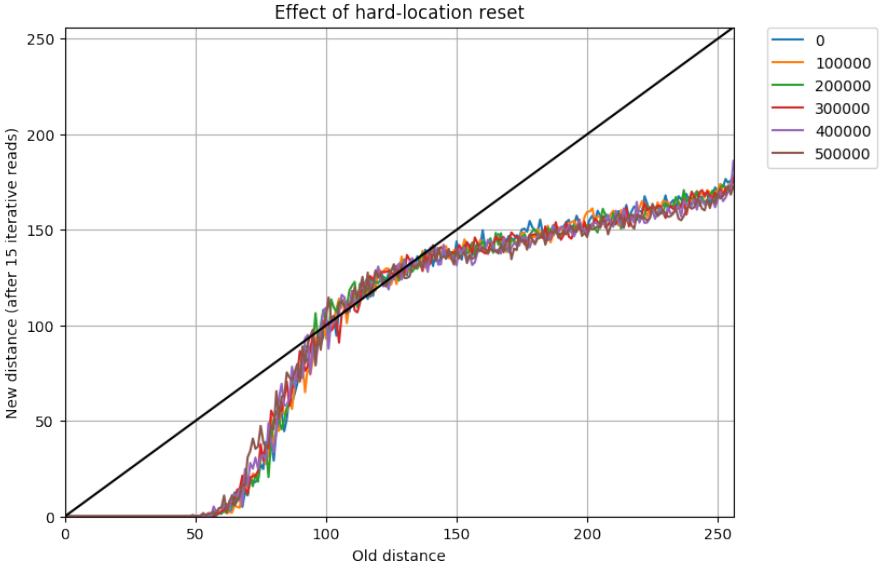
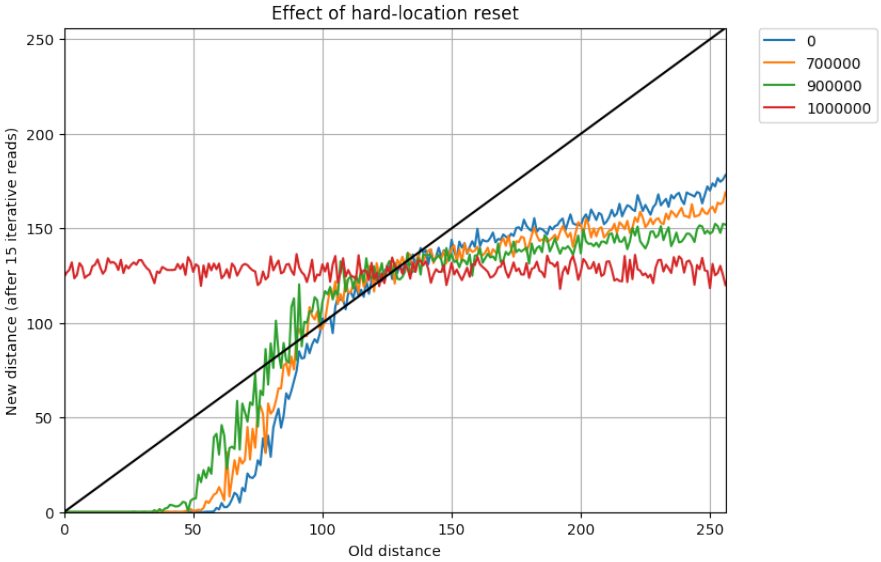


Figure 40: This graph shows the SDM's robustness against loss of neurons in a SDM with $n = 1,000$ and $H = 1,000,000$.



(a) Up to 500,000 neuron deaths



(b) From 600,000 to 1,000,000 neuron deaths

Figure 41: This graph shows the SDM's robustness against loss of neurons in a SDM with $n = 256$ and $H = 1,000,000$.

16

RESULTS (IV): GENERALIZED READ OPERATION

Dr. Murilo observed that the models of Kanerva's read ($z = 1$) and Chada's read ($z = 0$) were simple variations of a generalized read with an exponent z, which suggests experimenting with different values. Mathematically, let A be the set of the counters of the activated hard location, and c_i be the counter of the ith bit. Then,

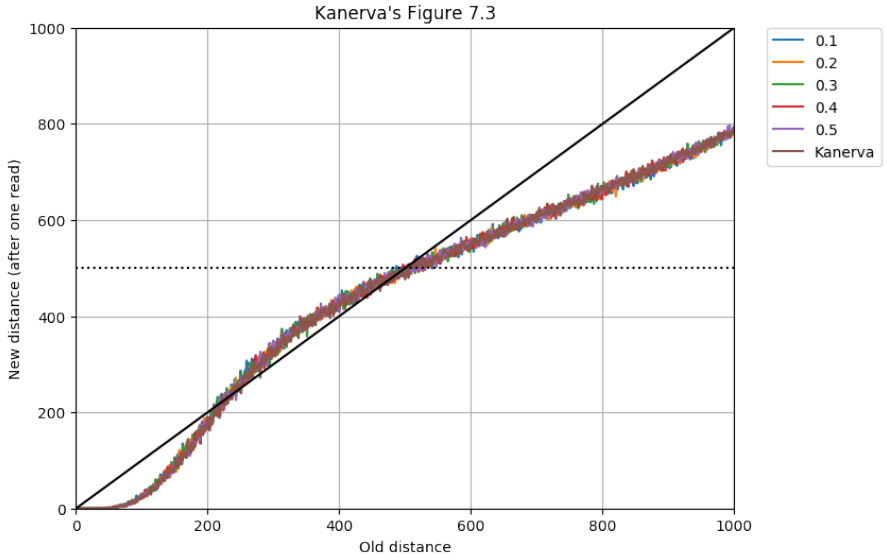
$$s_i = \sum_{c \in A} \frac{c_i}{|c_i|} |c_i|^z$$

The sum of $|c_i|^z$ turns the intermediate values from integers to floating point numbers. Thus, we have developed a specific read operation which stored the intermediate values in double variables.

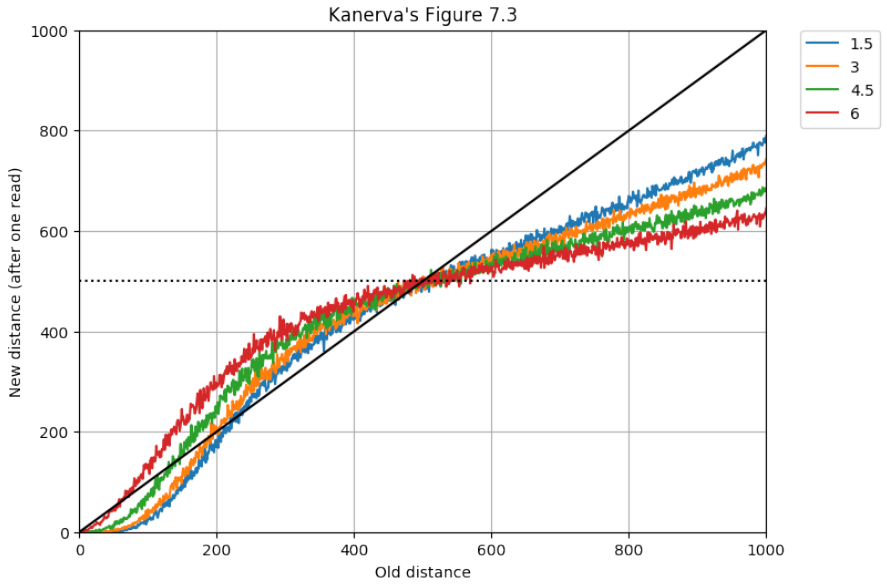
The results, however, have not yielded performance improvements. Though for $z \leqslant 1$ results are comparable to $z = 1$, for $z > 1$, the system shows an evident deterioration, with a smaller critical distance and faster divergence at large-distance reads. This is shown in Figures 42 and 43.

We understand that the critical distance is an important parameter of SDM. The bigger the critical distance, the better, because SDM is able to converge even with farther clues. For $z > 1$, the bigger the z, the smaller the critical distance. For $z = 6$, the critical distance almost reaches zero.

It is interesting that Kanerva has proposed $z = 1$ without realizing the generalized reading. Even so, he proposed the z with the highest critical distance.

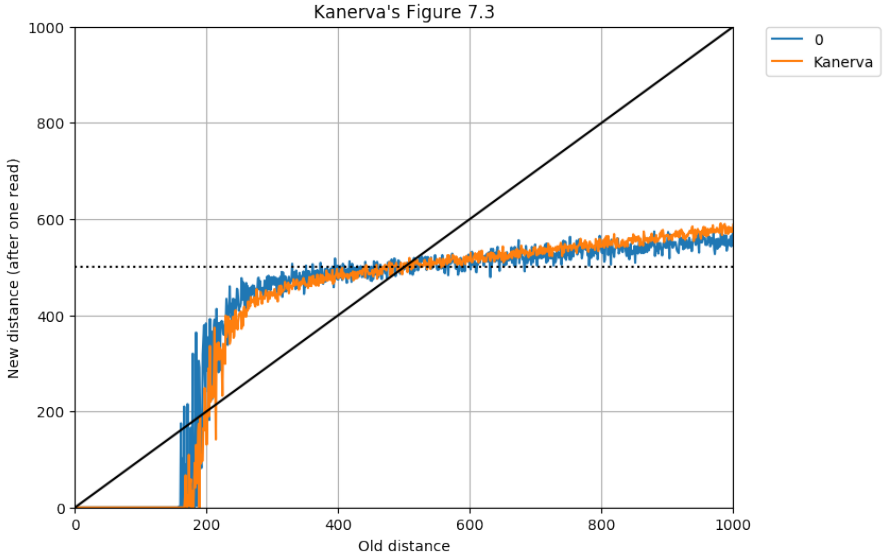


(a) SDM behavior when $z \in \{0.1, 0.2, 0.3, 0.4, 0.5, 1\}$

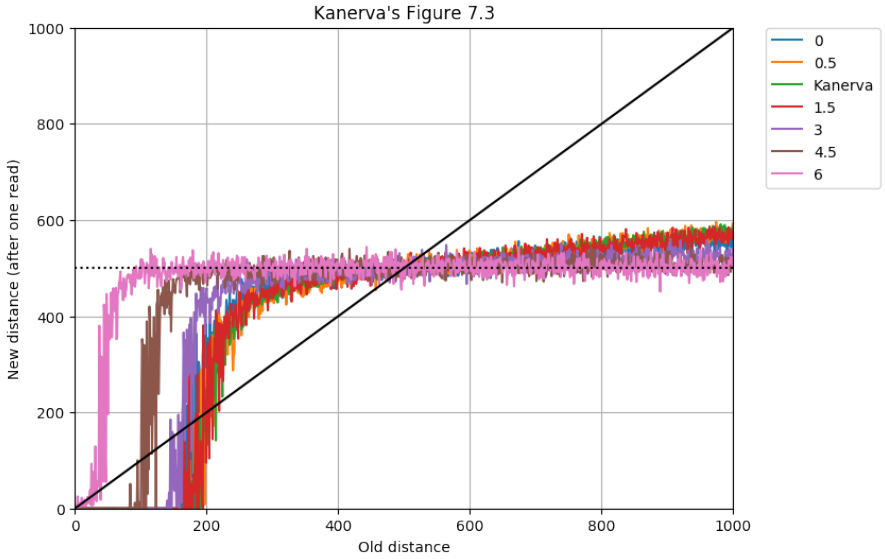


(b) SDM behavior when $z \in \{1.5, 3, 4.5, 6\}$

Figure 42: (a) and (b) show the behavior of a single read. As stated previously, we can see a deterioration of convergence, with lower critical distance as $z > 1$. Another observation can be made here, concerning the discrepancy of Kanerva's Fig 7.3 and our data. It seems that Kanerva may not have considered that a single read would only 'clean' a small number of dimensions *after the critical distance*. What we observe clearly is that with a single read, as the distance grows, the system only 'cleans' towards the orthogonal distance 500 after a number of iterative readings.



(a) $z \in \{0, 1\}$



(b) $z \in \{0, 0.5, 1, 1.5, 3, 4.5, 6\}$

Figure 43: (a) and (b) show the behavior of Figure 42, now executed with 6-iterative reads. What we observe clearly is that with a single read, as the distance grows, the system only 'cleans' towards the orthogonal distance 500 after a number of iterative readings.

17

RESULTS (V): PERFORMANCE

Performance matters — and has always mattered. If an experiment takes a few seconds, there is no point arguing whether we should try it. If it takes a few hours, maybe we should think first. If it takes a few days — or more —, it is important to devise a good plan. As SDM consumes large processor and memory resources, some experiments may take a long time.

Each scan on a 1,000-bits SDM with 1,000,000 hard locations executes 10^9 bit compares through $10^9/64 = 15,625,000$ XORs and calls to the built-in popcount. So, 10,000 writes execute 10^{13} bit compares, while a 6-iterative reading executes $6 \cdot 10^{12}$ bit compares. The heatmap of Figure 17a executed $3.05 \cdot 10^{15}$ bit compares. For comparison, the number of seconds since Jesus's birth is 63,639,648,000 = $6.36 \cdot 10^{10}$. The number of people who have ever lived on earth is estimated to be $1,08 \cdot 10^{11}$. There are approximately $1.8 \cdot 10^9$ websites on the internet. A modern laptop can increment a counter $5 \cdot 10^8$ times per second. Hence, a naive implementation of SDM may take several hours — or days — to simply write 10,000 random bitstrings.

Amazon EC2 p3.2xlarge has generated the heatmap of Figure 17a in 15 minutes and 3 seconds. It has compared $3.37 \cdot 10^{12}$ bits per second through $52.6 \cdot 10^9 = 52.6$ billion XORs and popcounts per second. It is a 60-fold improvement over the first versions of the code which took 16 hours to generate the same heatmap (and its memory use was already optimized and the computations were distributed in threads).

We have created a benchmark to be able to compare the performance of different devices. So, the same performance test was executable in our devices, with results reported in tables and figures. The benchmark has 3 parts: (i) the first part consists of comparing the available OpenCL kernels to find which works best for that device; than (ii) the second part consists of comparing the linear scanner, the thread scanner, and the OpenCL scanners with the best kernel found in part one; finally, (iii) the third part consists of comparing read and write operations using the thread scanner and the OpenCL scanner with the best kernel. Each part was run for three SDM setups: (i) $n = 1,000$, $r = 451$, and $H = 1,000,000$; (ii) $n = 256$, $r = 103$, and $H = 1,000,000$; and (iii) $n = 10,000$, $r = 4850$, and $H = 1,000,000$. The whole source code is available in the "Performance test" notebook [19]. We would like to invite the reader to run this benchmark and, if possible, share the results.

Our first device was a personal MacBook Pro Retina 13-inch Late 2013 with a 2.6GHz Intel core i5 processor, 6GB DDR3 RAM, and Intel Iris GPU. It was not possible to run the 10,000-bits on this device because there was no memory available — it needs 37.25 GB of memory. For its results, see Table 3

Our second device was an iMac Retina 5K 27-inch 2017 with a 3.8GHz Intel core i5 processor, 8GB DDR4 RAM, and a Radeon Pro 580 8G CPU. For its results, see Table 4.

Beyond that, we were also running in state-of-the-art devices: (i) an Amazon EC2 p2.xlarge with Intel Xeon E5-2686v4 processor, 61GB DDR3 RAM, and NVIDIA K80 GPU (see Table 5), and (ii) an Amazon EC2 p3.2xlarge with Intel Xeon E5-2686v4 processor, 488GB DDR3 RAM, and NVIDIA Tesla V100 GPU (see Table 6).

17.1 KERNELS COMPARISON

OpenCL is a framework for writing software that executes in *heterogeneous* devices [72], like CPUs, GPUs, FPGA and other co-processors for hardware acceleration. Because they are heterogeneous, they may differ a lot in architecture and performance, which means there is no one-size-fits-all kernel. A *kernel* is generally a small function on the code that runs in thousands of parallel threads, executing the same steps on different parts of a large vector or matrix. The slower kernel for one device may be the fastest for another device, as we will see happening in our case.

A total of 8 kernels have been developed for benchmarking in our framework: single_scan0, single_scan1, single_scan2, single_scan3, single_scan4, single_scan5, single_scan5_unroll, and single_scan6. Each scan uses a different algorithm to do exactly the same thing: calculate which hard locations are inside the circle of the given bitstring. They differ in how they split the work between work-groups and how they explore the GPU architecture to obtain performance gains.

The difference in which kernel is the best also depends on the SDM setup. The best kernel for the 1,000-bits SDM in the iMac 2017 was single_scan5_unroll with average scan time of 3.61ms; but, for the 256-bits SDM in the same device, it was single_scan0 with average scan time of 3.03ms; while, for the 10,000-bits SDM in the same device, it was single_scan6 with 10.96ms (see Table 4).

We recommend users to run a specific kernel comparison test for their own GPU and SDM settings. This is available in the Jupyter notebooks.

17.2 SCANNERS COMPARISON

In this section, we are comparing the OpenCL scanner (with the best kernel) with the linear scanner and the threads scanner. Again, which one is faster depends on the SDM settings. In the iMac 2017, the fastest scanner for a 1,000-bits SDM was the OpenCL scanner with `single_scan5_unroll`, but, for a 256-bit SDM, it was the threads scanner.

What happened here is that the OpenCL kernel chosen was a generic one which performs the scan for any SDM. It is always possible to optimize the OpenCL kernel to a specific setting, and it would be faster than the threads. By default, the framework chooses a generic kernel which we believe would be reasonable for the most common setups.

We can notice that Amazon EC2 p3.2xlarge and p2.xlarge's linear and thread scanners, both running on CPU, were much slower than the CPU of both the personal MacBook Pro and the iMac 2017. As Amazon EC2 are virtual machines with GPU devices, their CPU is shared with other virtual machines which significantly reduces CPU power. Hence, for both virtual machines we have tested, using the GPU seriously boosts performance, but using the CPU should be avoided (see Table 4)

17.3 READ AND WRITE OPERATIONS

Even though scanning is an important part of the operations, we are really interested in the performance of the entirety of operations themselves. Comparing the times of the thread and OpenCL scanners with the times of their respective operations (either read or write), we can notice that their difference remains almost constant, which means the operation time itself is negligible when compared to the scan time. In other words, in order to gain even more performance, we have to pursue ways to improve the scan.

17.3.1 *Summary of results*

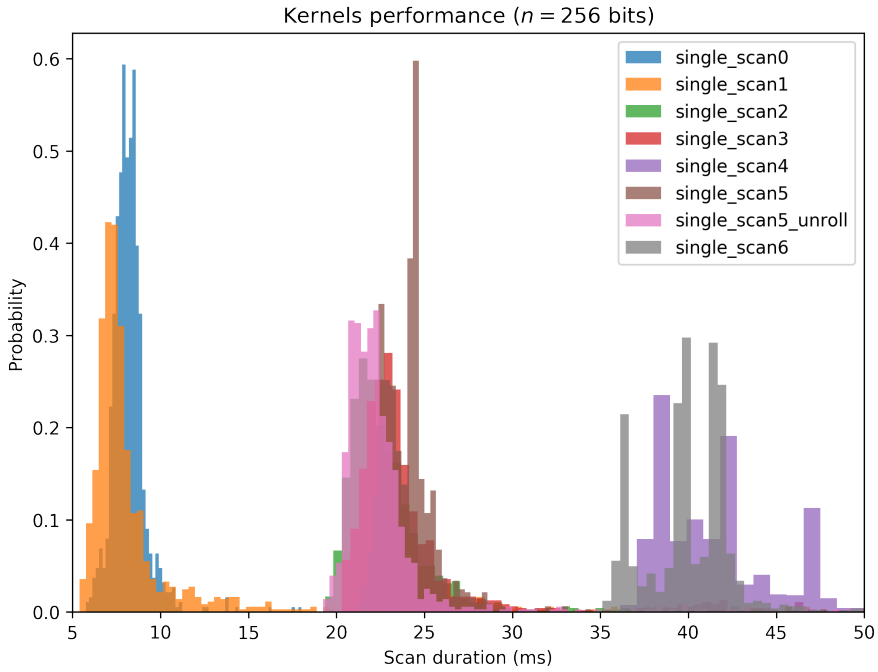
The results, beyond showing the obvious fact that consumer grade hardware is not comparable to the Amazon instances, indicate a non-trivial issue: The chosen kernel for scanning the memory is of crucial importance to performance, and this kernel speed is a function of both the hardware used and the particular parameters used in the SDM settings.

It is reasonable to consider that the performance results obtained are of particular merit, and one particular fact stands out: The scanning of 1,000,000 hard locations has become, in the desired

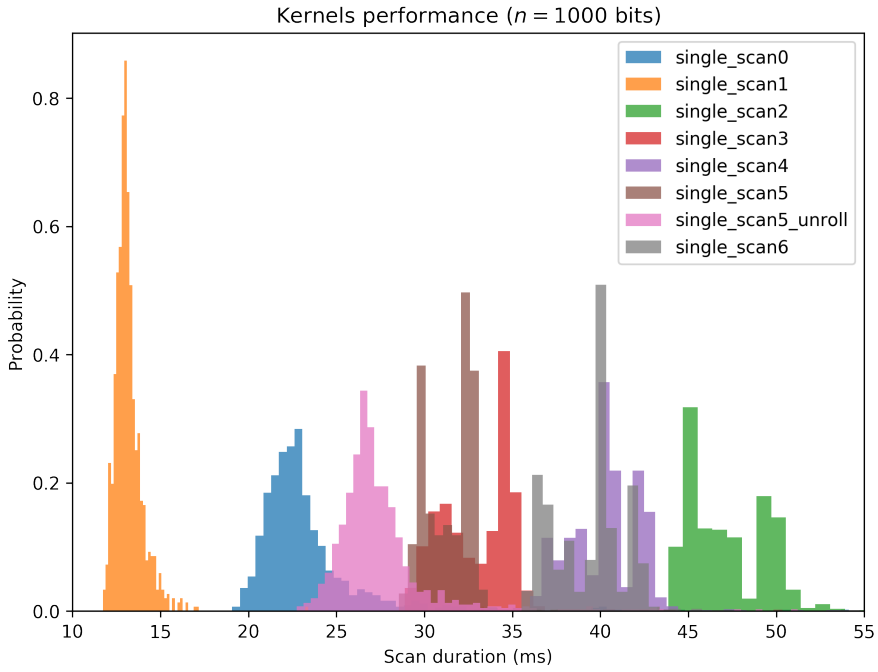
	256 bits	**1,000 bits**	**10,000 bits**
Kernel	Duration (ms)	Duration (ms)	Duration (ms)
single_scan0	8.36	23.60	
single_scan1	10.43	13.22	
single_scan2	23.48	47.28	
single_scan3	25.51	33.06	
single_scan4	42.39	40.32	
single_scan5	24.42	31.51	
single_scan5_unroll	22.77	27.55	
single_scan6	42.18	39.48	
Scanner	Duration (ms)	Duration (ms)	Duration (ms)
Linear scan	9.07	17.98	
Thread scan	5.14	10.17	
OpenCL scan	8.05	12.35	
Operation	Duration (ms)	Duration (ms)	Duration (ms)
Thread write	6.72	14.13	
Thread single read	5.88	11.12	
OpenCL write	6.39	18.55	
OpenCL single read	5.26	13.06	

Table 3: MacBook Pro Retina 13-inch Late 2013 with a 2.6GHz Intel core i5 processor, 6GB DDR3 RAM, and Intel Iris GPU. The SDM settings were: (i) $n = 256$, $r = 103$, and $H = 1,000,000$; (ii) $n = 1,000$, $r = 451$, and $H = 1,000,000$; and (iii) $n = 10,000$, $r = 4850$, and $H = 1,000,000$. There is no benchmark for $n = 10,000$ because memory is not enough on either RAM or GPU—it would consume 37.25 GB of RAM and 1.2GB of memory in the GPU. For the histogram of durations, see Figures 44, 45, 46, and 47.

professional-grade machines, *faster* than the updating of the 1,000 active locations.

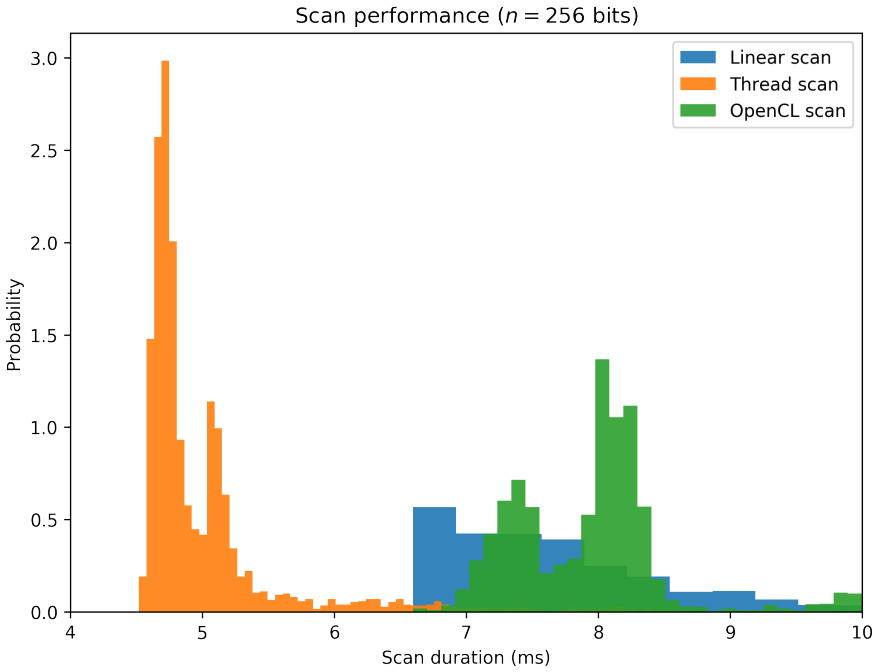


(a) $n = 256$, $r = 103$, and $H = 1,000,000$

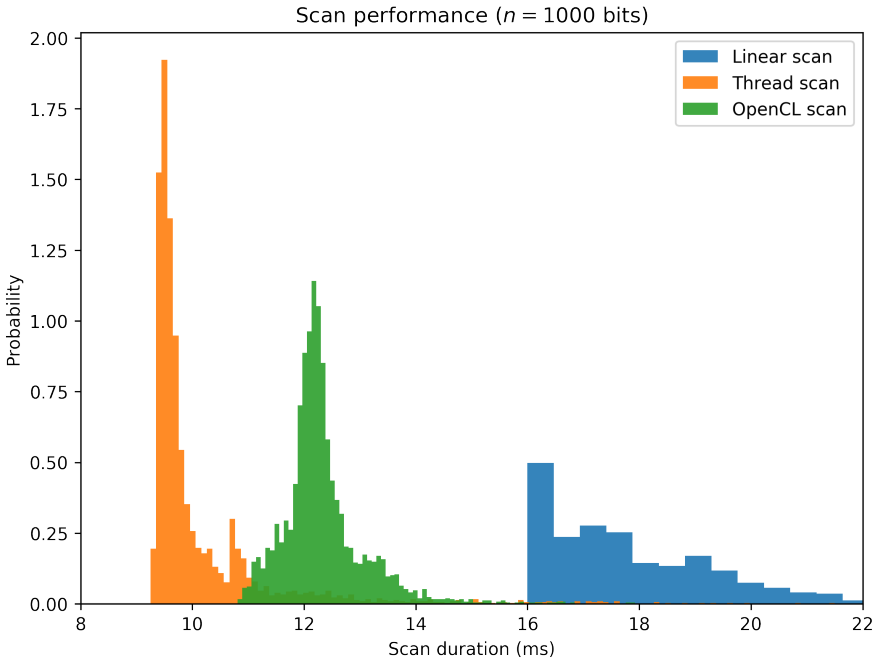


(b) $n = 1,000$, $r = 451$, and $H = 1,000,000$

Figure 44: Kernel comparisons for MacBook Pro Retina 13-inch Late 2013 with a 2.6GHz Intel core i5 processor, 6GB DDR3 RAM, and Intel Iris GPU.

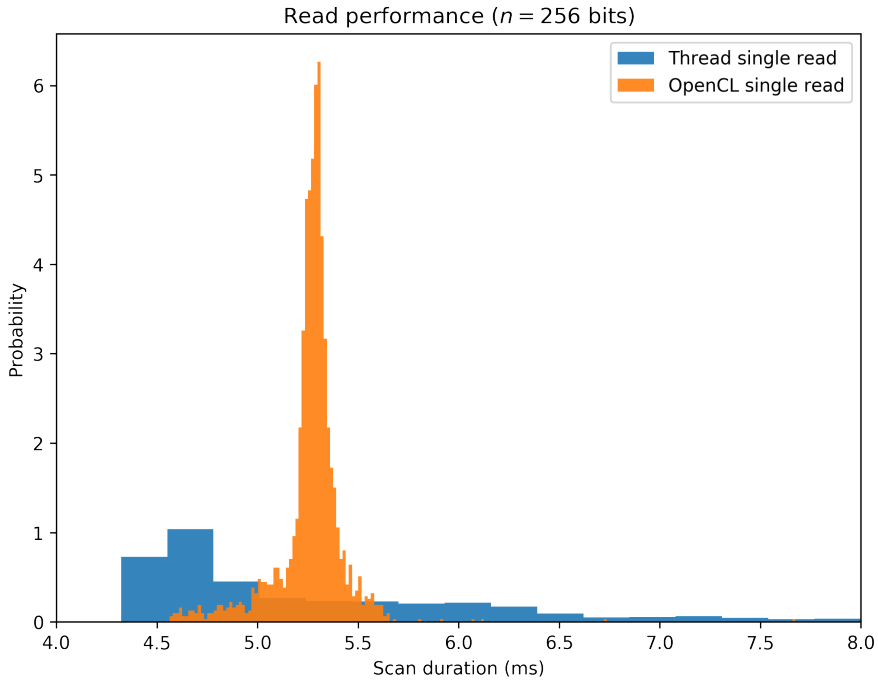


(a) $n = 1,000$, $r = 451$, and $H = 1,000,000$

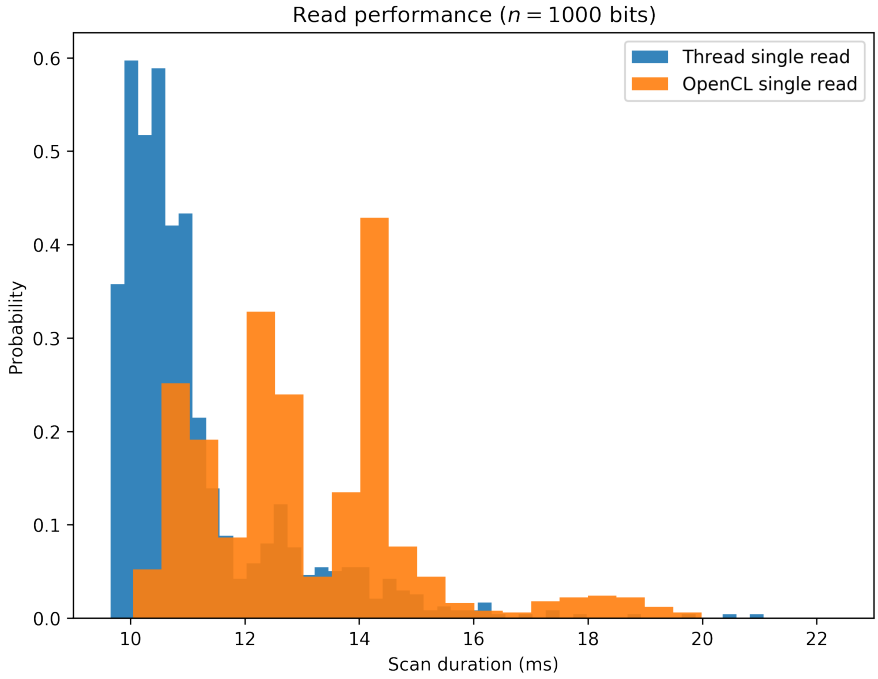


(b) $n = 1,000$, $r = 451$, and $H = 1,000,000$

Figure 45: Scanner comparisons for MacBook Pro Retina 13-inch Late 2013 with a 2.6GHz Intel core i5 processor, 6GB DDR3 RAM, and Intel Iris GPU.

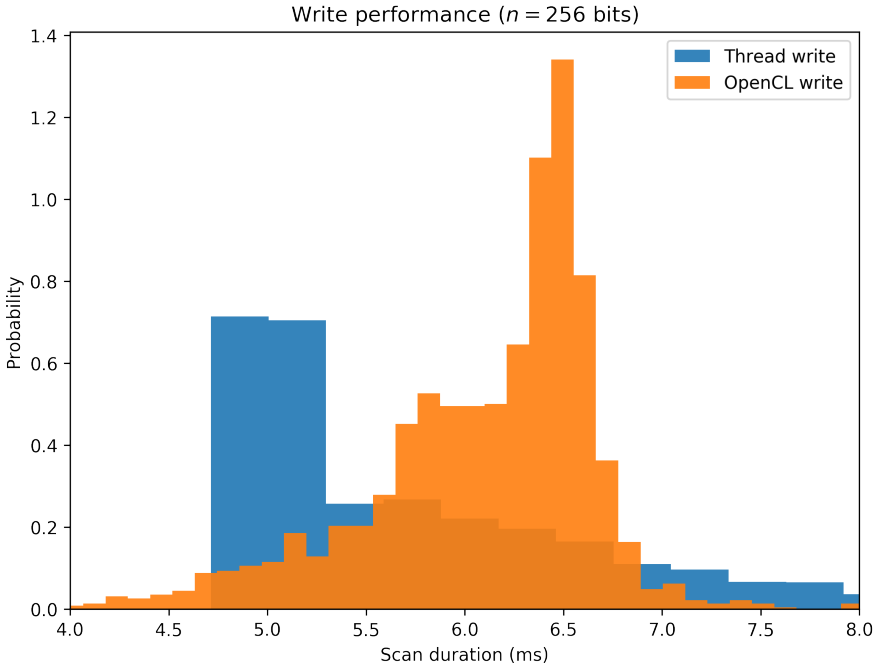


(a) $\mathrm{n} = 1,000$, $\mathrm{r} = 451$, and $\mathrm{H} = 1,000,000$

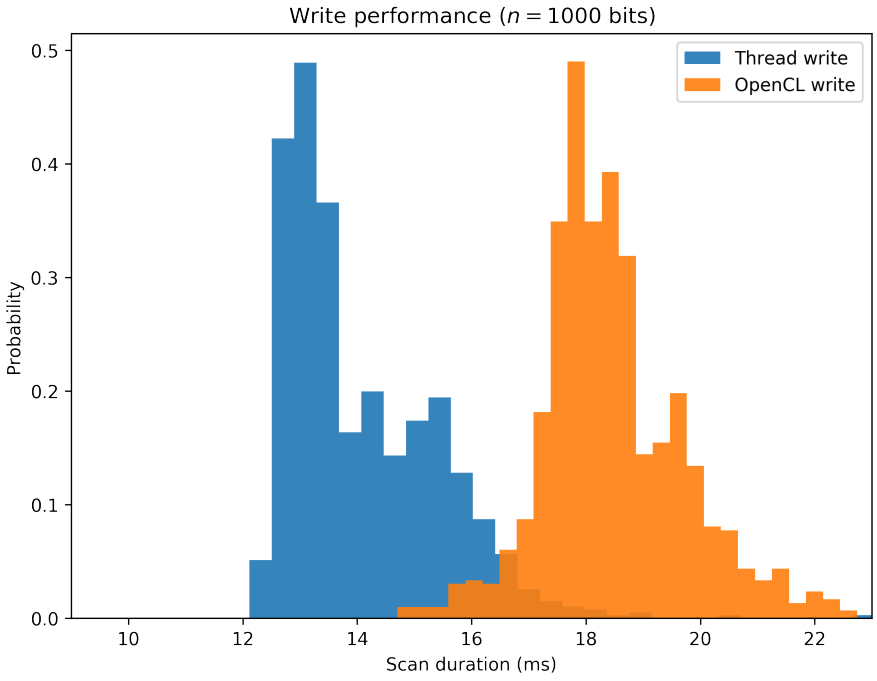


(b) $\mathrm{n} = 1,000$, $\mathrm{r} = 451$, and $\mathrm{H} = 1,000,000$

Figure 46: Read operation comparisons for MacBook Pro Retina 13-inch Late 2013 with a 2.6GHz Intel core i5 processor, 6GB DDR3 RAM, and Intel Iris GPU.



(a) $n = 1,000$, $r = 451$, and $H = 1,000,000$

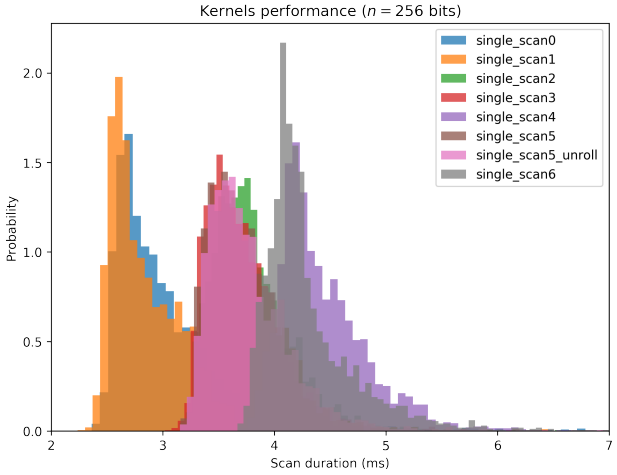


(b) $n = 1,000$, $r = 451$, and $H = 1,000,000$

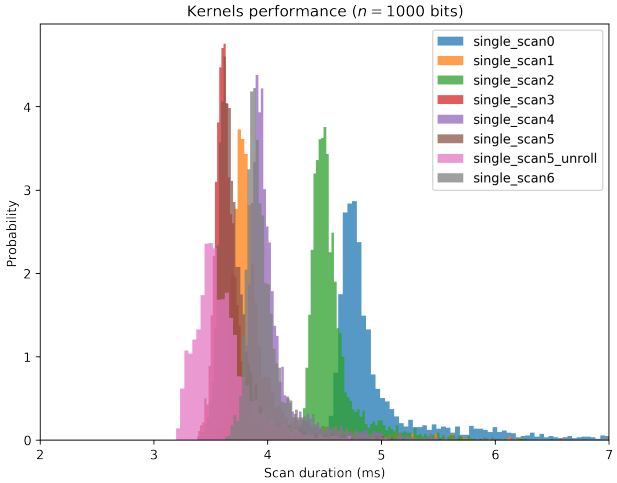
Figure 47: Write operation comparisons for MacBook Pro Retina 13-inch Late 2013 with a 2.6GHz Intel core i5 processor, 6GB DDR3 RAM, and Intel Iris GPU.

	256 bits	**1,000 bits**	**10,000 bits**
Kernel	Duration (ms)	Duration (ms)	Duration (ms)
single_scan0	3.03	5.00	61.06
single_scan1	2.87	3.95	44.96
single_scan2	3.82	4.57	44.98
single_scan3	3.72	3.68	12.67
single_scan4	4.48	4.04	11.45
single_scan5	3.76	3.72	12.58
single_scan5_unroll	3.79	3.61	11.37
single_scan6	4.36	4.02	10.96
Scanner	Duration (ms)	Duration (ms)	Duration (ms)
Linear scan	5.04	12.25	116.38
Thread scan	2.92	6.95	53.56
OpenCL scan	2.81	4.20	12.95
Operation	Duration (ms)	Duration (ms)	Duration (ms)
Thread write	3.28	13.34	
Thread single read	2.55	10.39	
OpenCL write	2.64	7.90	
OpenCL single read	2.14	5.25	

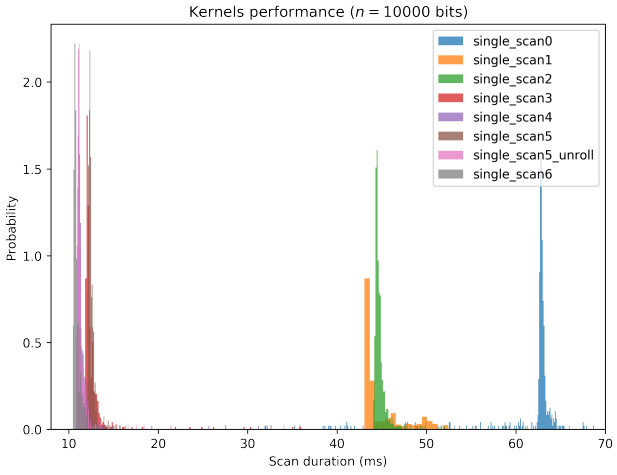
Table 4: iMac Retina 5K 27-inch 2017 with a 3.8GHz Intel core i5 processor, 8GB DDR4 RAM, and a Radeon Pro 580 8G GPU. The SDM settings were: (i) $n = 256$, $r = 103$, and $H = 1,000,000$; (ii) $n = 1,000$, $r = 451$, and $H = 1,000,000$; and (iii) $n = 10,000$, $r = 4850$, and $H = 1,000,000$. There is no benchmark for read and write operations with $n = 10,000$ because RAM is not enough to allocate the counters—it would consume 37.25 GB of RAM. For the histogram of durations, see Figures 48, 49, 50, and 51.



(a) $n = 256$, $r = 103$, and $H = 1,000,000$

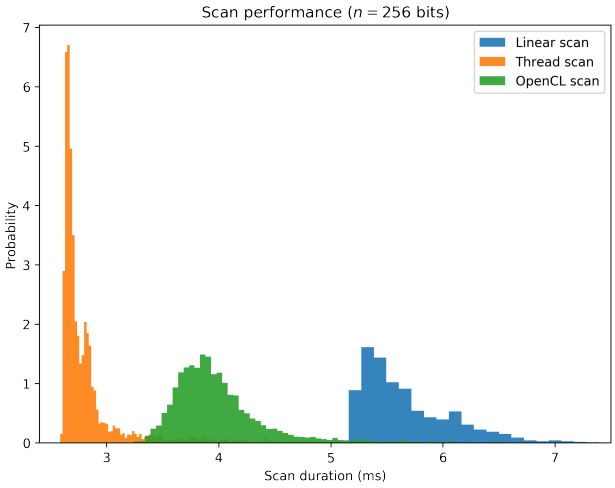


(b) $n = 1,000$, $r = 451$, and $H = 1,000,000$

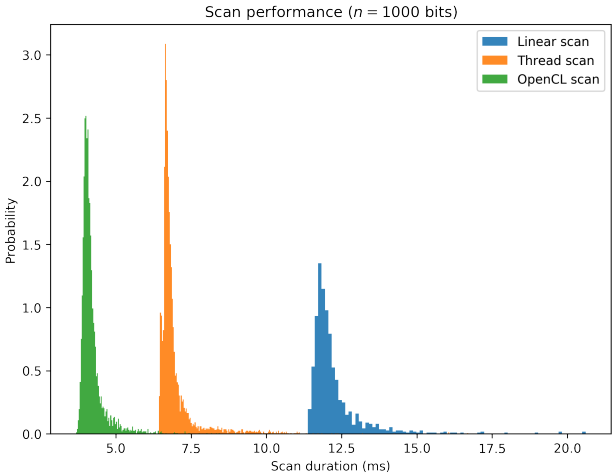


(c) $n = 10,000$, $r = 4805$, and $H = 1,000,000$

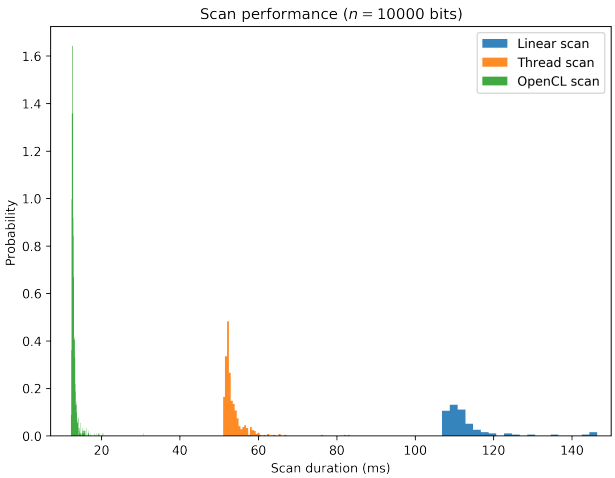
Figure 48: Kernel comparisons for iMac Retina 5K 27-inch 2017 with a 3.8GHz Intel core i5 processor, 8GB DDR4 RAM, and a Radeon Pro 580 8G GPU.



(a) $n = 1,000$, $r = 451$, and $H = 1,000,000$

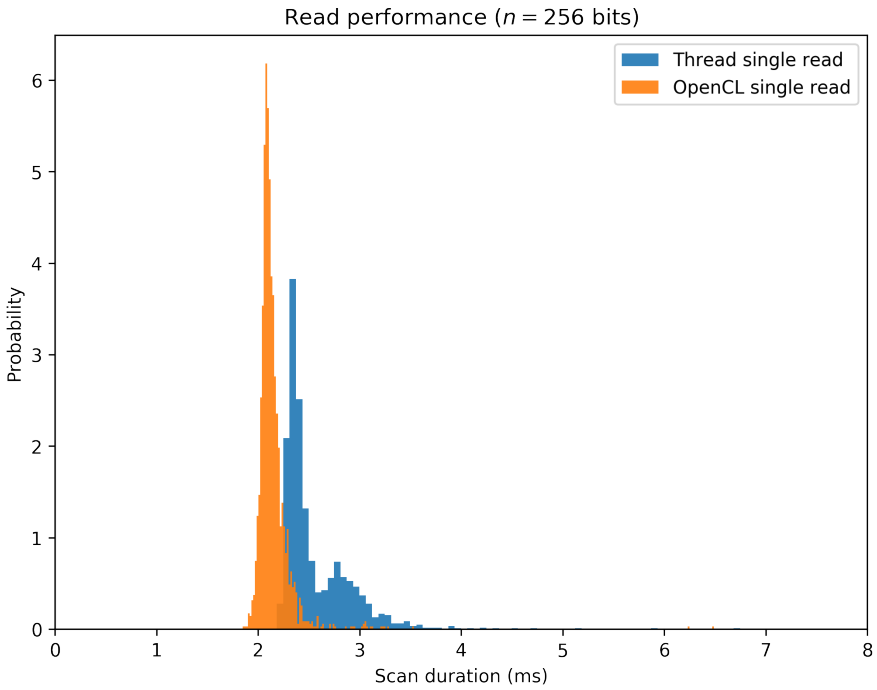


(b) $n = 1,000$, $r = 451$, and $H = 1,000,000$

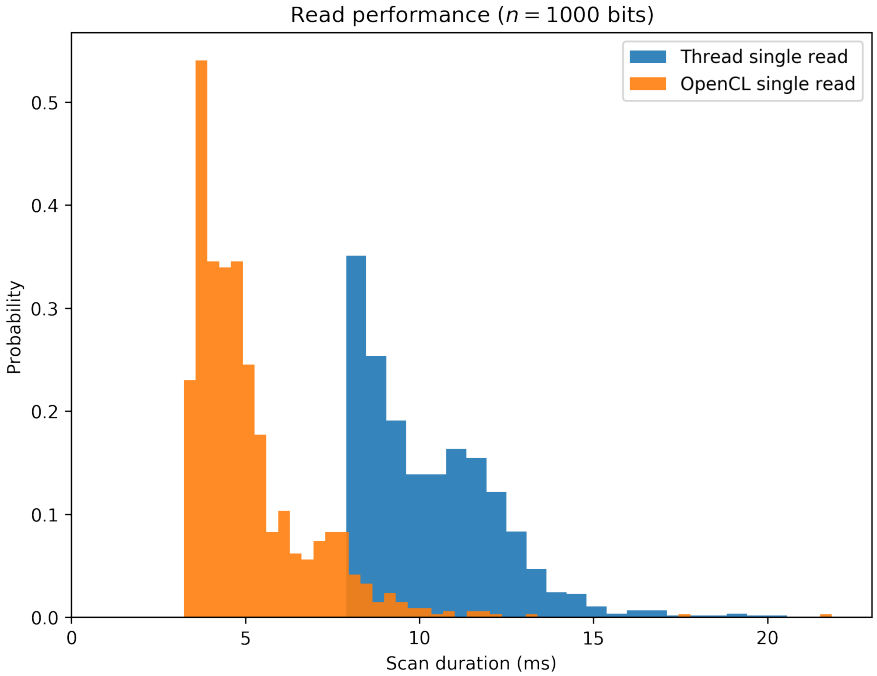


(c) $n = 10,000$, $r = 4805$, and $H = 1,000,000$

Figure 49: Scanner comparisons for iMac Retina 5K 27-inch 2017 with a 3.8GHz Intel core i5 processor, 8GB DDR4 RAM, and a Radeon Pro 580 8G GPU.

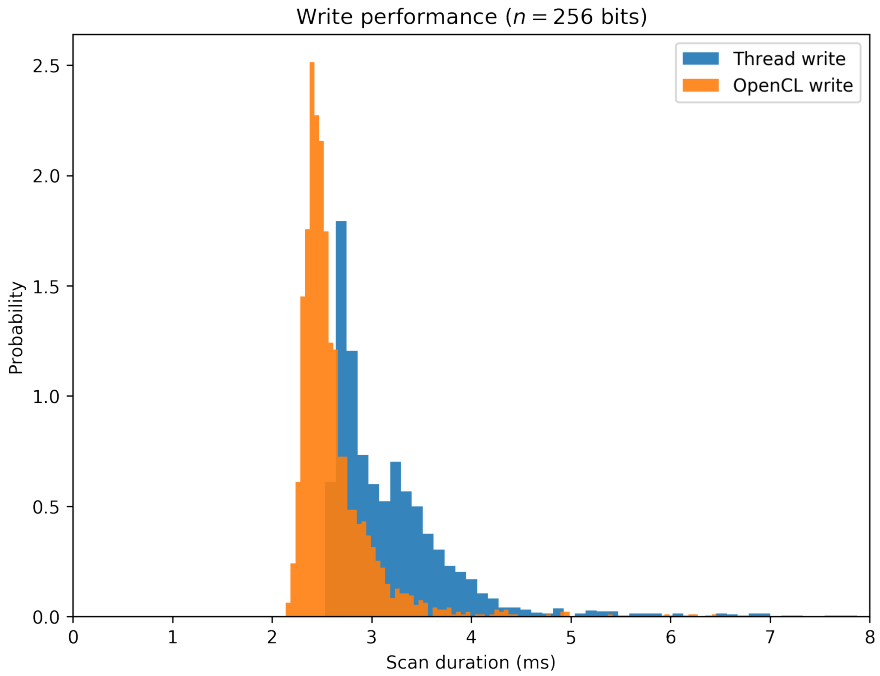


(a) $n = 1,000$, $r = 451$, and $H = 1,000,000$

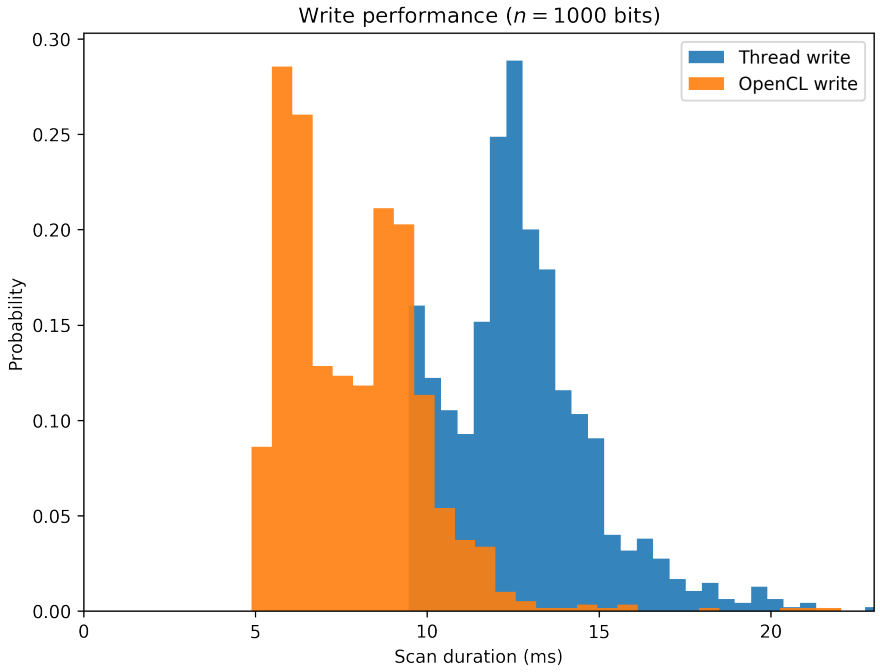


(b) $n = 1,000$, $r = 451$, and $H = 1,000,000$

Figure 50: Read operation comparisons for iMac Retina 5K 27-inch 2017 with a 3.8GHz Intel core i5 processor, 8GB DDR4 RAM, and a Radeon Pro 580 8G GPU.



(a) $n = 1,000$, $r = 451$, and $H = 1,000,000$

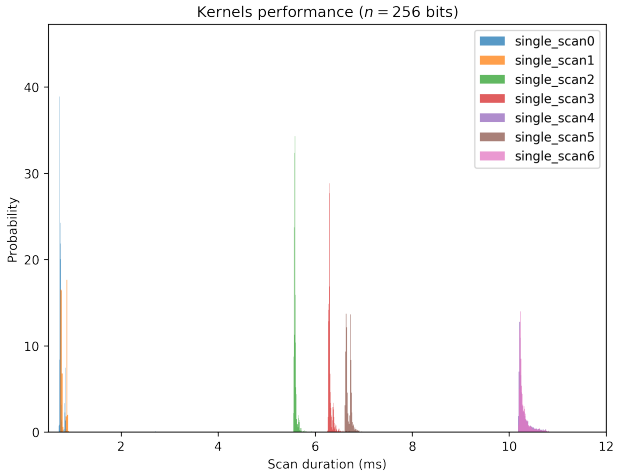


(b) $n = 1,000$, $r = 451$, and $H = 1,000,000$

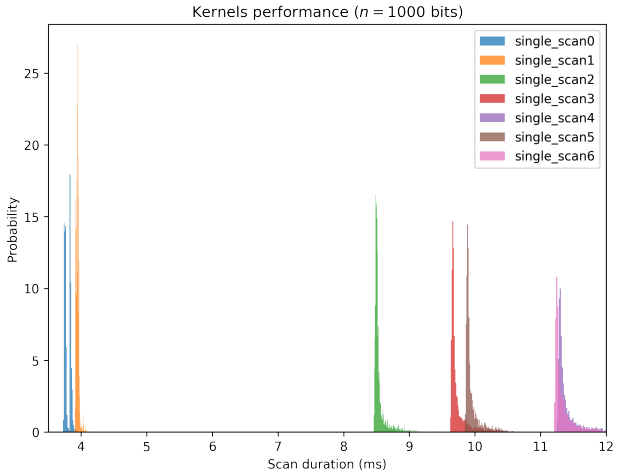
Figure 51: Write operation comparisons for iMac Retina 5K 27-inch 2017 with a 3.8GHz Intel core i5 processor, 8GB DDR4 RAM, and a Radeon Pro 580 8G GPU.

	256 bits	**1,000 bits**	**10,000 bits**
Kernels	Duration (ms)	Duration (ms)	Duration (ms)
single_scan0	0.76	3.79	35.45
single_scan1	0.80	3.94	57.80
single_scan2	5.59	8.54	63.71
single_scan3	6.31	9.73	39.92
single_scan4	10.29	11.38	45.49
single_scan5	6.69	9.95	43.51
single_scan5_unroll			
single_scan6	10.29	11.33	41.42
Scanners	Duration (ms)	Duration (ms)	Duration (ms)
Linear scan	19.09	64.73	600.75
Thread scan	9.95	32.81	296.56
OpenCL scan	6.88	10.67	46.73
Operations	Duration (ms)	Duration (ms)	Duration (ms)
Thread write	11.80	42.64	383.50
Thread single read	10.49	35.37	307.97
OpenCL write	8.84	19.31	122.17
OpenCL single read	7.50	11.72	55.47

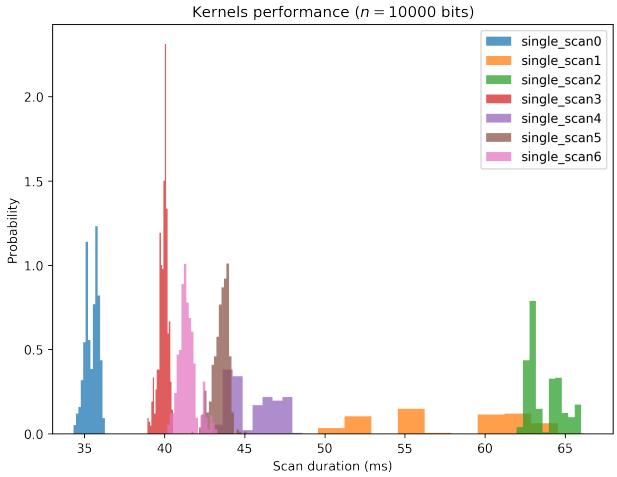
Table 5: Amazon EC2 p2.xlarge with Intel Xeon E5-2686v4 processor, 61GB DDR3 RAM, and NVIDIA K80 GPU. Running an SDM with $n = 256$ bits, $H = 1,000,000$, and $r = 103$. The SDM settings were: (i) $n = 256$, $r = 103$, and $H = 1,000,000$; (ii) $n = 1,000$, $r = 451$, and $H = 1,000,000$; and (iii) $n = 10,000$, $r = 4850$, and $H = 1,000,000$. There is no benchmark for kernel single_scan5_unroll because it returns the wrong result in this GPU. The problem is related to the premises of the optimization used by this kernel, which are not true for this GPU. For the histogram of durations, see Figures 52, 49, 50, and 51.



(a) $n = 256$, $r = 103$, and $H = 1,000,000$

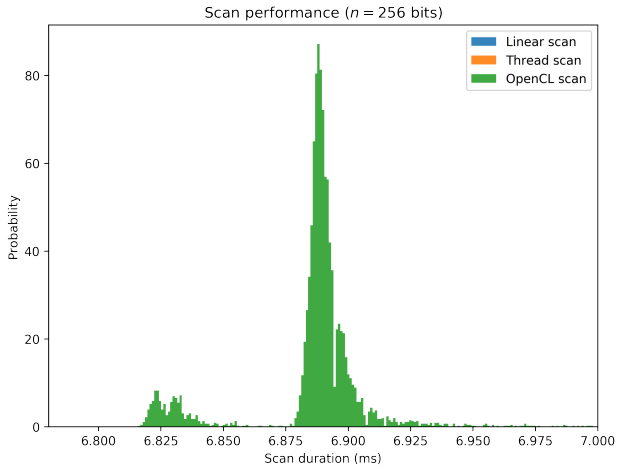


(b) $n = 1,000$, $r = 451$, and $H = 1,000,000$

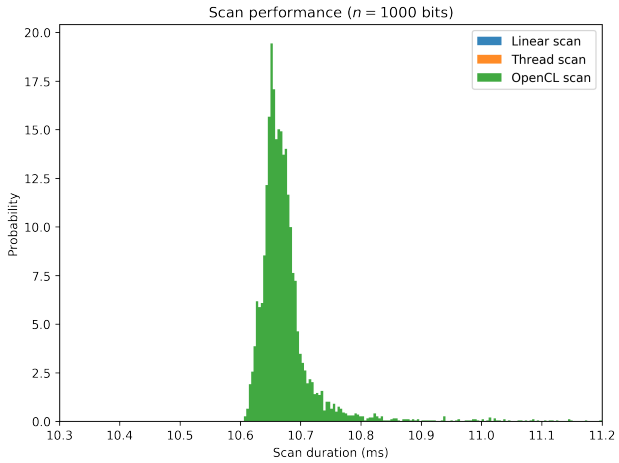


(c) $n = 10,000$, $r = 4805$, and $H = 1,000,000$

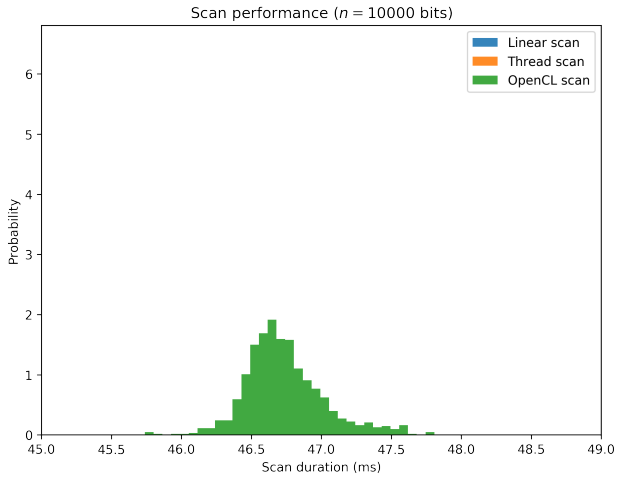
Figure 52: Kernel comparisons for Amazon EC2 p2.xlarge with Intel Xeon E5-2686v4 processor, 61GB DDR3 RAM, and NVIDIA K80 GPU.



(a) $n = 1,000$, $r = 451$, and $H = 1,000,000$

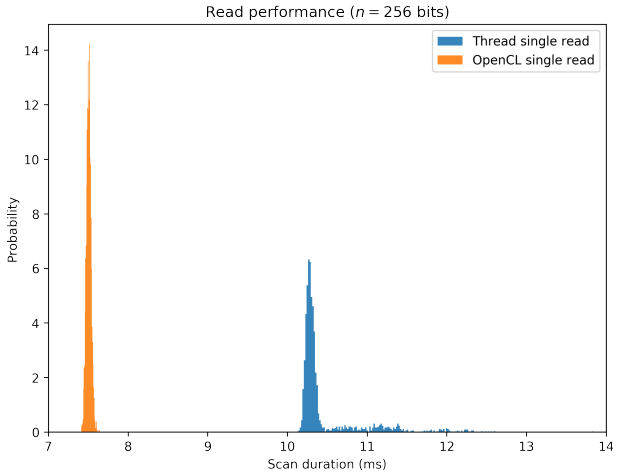


(b) $n = 1,000$, $r = 451$, and $H = 1,000,000$

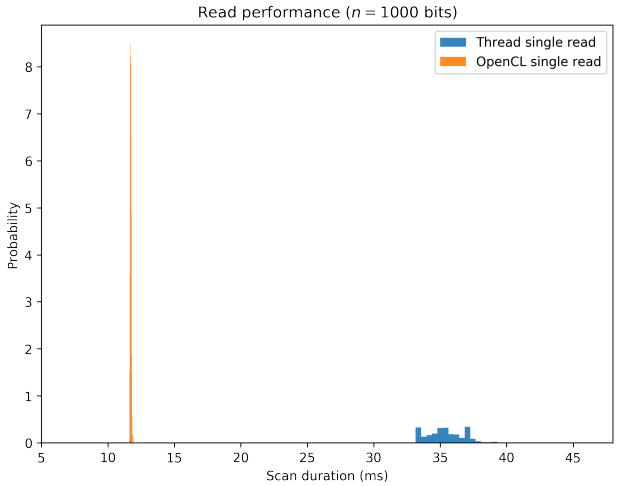


(c) $n = 10,000$, $r = 4805$, and $H = 1,000,000$

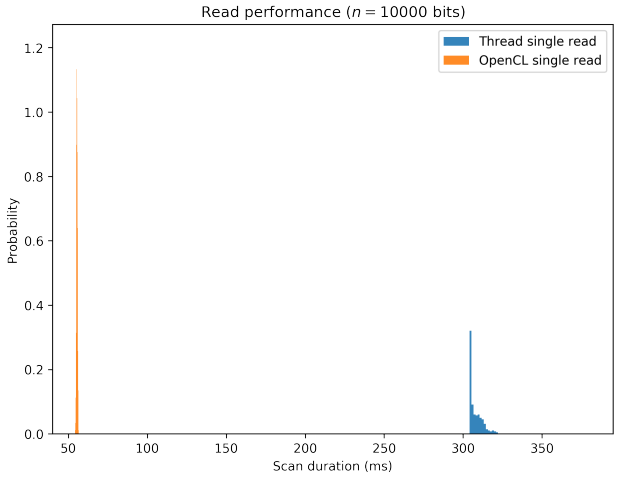
Figure 53: Scanner comparisons for Amazon EC2 p2.xlarge with Intel Xeon E5-2686v4 processor, 61GB DDR3 RAM, and NVIDIA K80 GPU.



(a) $n = 1,000$, $r = 451$, and $H = 1,000,000$

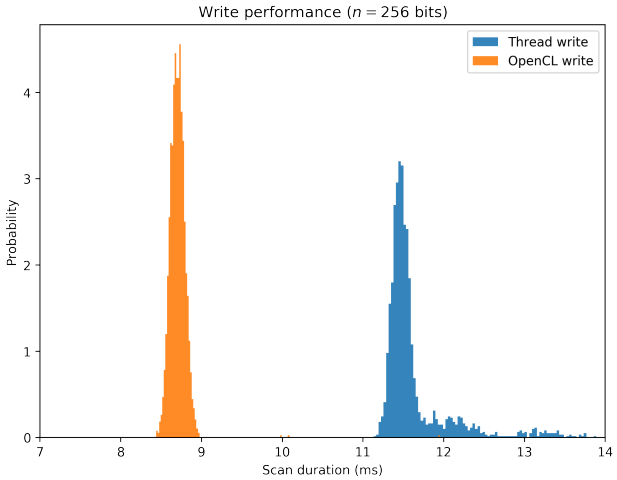


(b) $n = 1,000$, $r = 451$, and $H = 1,000,000$

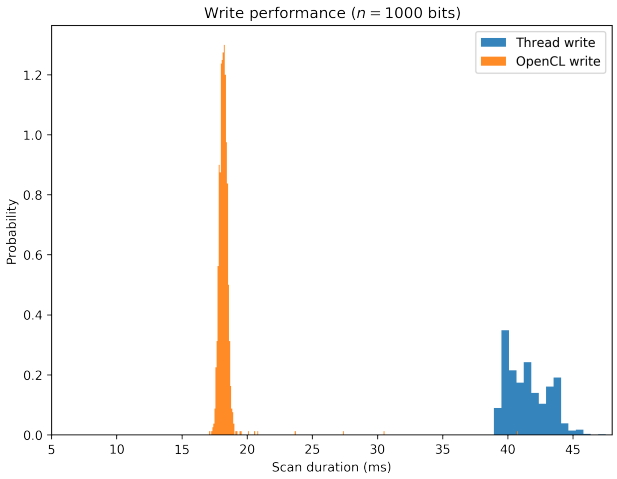


(c) $n = 10,000$, $r = 4805$, and $H = 1,000,000$

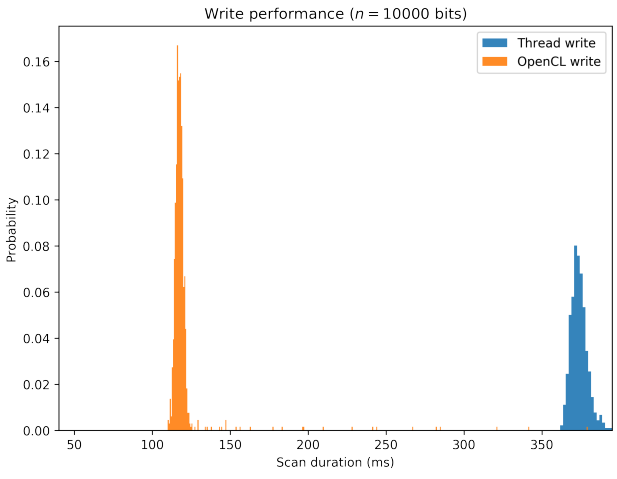
Figure 54: Read operation comparisons for Amazon EC2 p2.xlarge with Intel Xeon E5-2686v4 processor, 61GB DDR3 RAM, and NVIDIA K80 GPU.



(a) $n = 1,000$, $r = 451$, and $H = 1,000,000$



(b) $n = 1,000$, $r = 451$, and $H = 1,000,000$

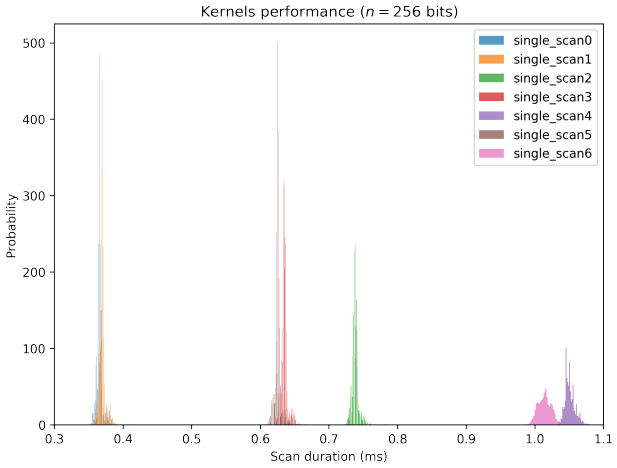


(c) $n = 10,000$, $r = 4805$, and $H = 1,000,000$

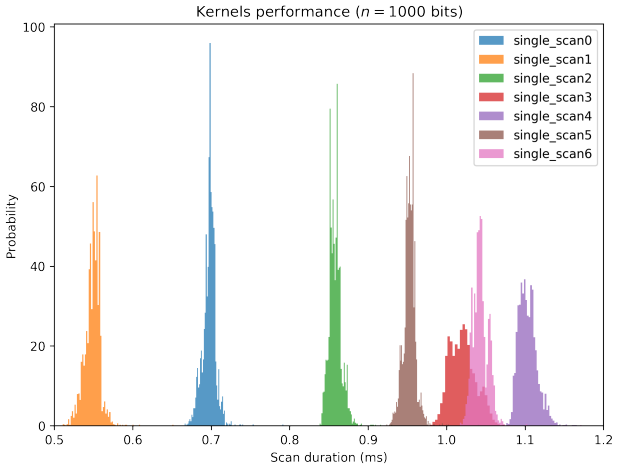
Figure 55: Write operation comparisons for Amazon EC2 p2.xlarge with Intel Xeon E5-2686v4 processor, 61GB DDR3 RAM, and NVIDIA K80 GPU.

	256 bits	**1,000 bits**	**10,000 bits**
Kernel	Duration (ms)	Duration (ms)	Duration (ms)
single_scan0	0.36	0.69	20.60
single_scan1	0.36	0.54	4.94
single_scan2	0.73	0.85	5.01
single_scan3	0.63	1.02	6.07
single_scan4	1.05	1.10	5.99
single_scan5	0.62	0.95	5.36
single_scan5_unroll			
single_scan6	1.01	1.04	5.96
Scanner	Duration (ms)	Duration (ms)	Duration (ms)
Linear scan	17.60	58.34	540.97
Thread scan	5.19	16.39	198.74
OpenCL scan	0.63	0.85	5.74
Operation	Duration (ms)	Duration (ms)	Duration (ms)
Thread write	7.59	28.47	222.08
Thread single read	6.17	17.44	145.01
OpenCL write	2.33	8.77	80.48
OpenCL single read	1.01	1.82	13.99

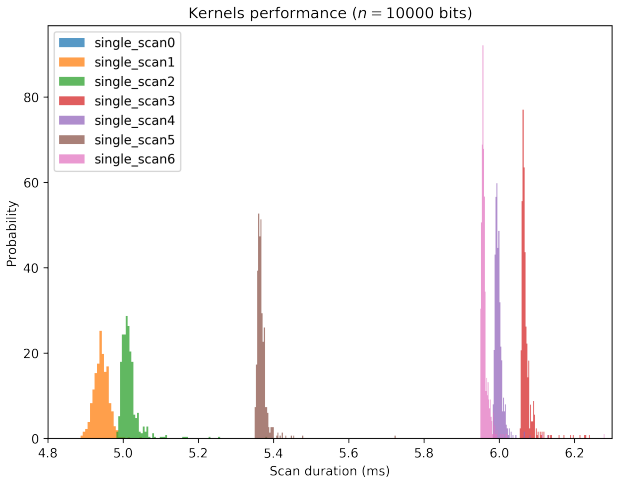
Table 6: Amazon EC2 p3.2xlarge with Intel Xeon E5-2686v4 processor, 488GB DDR3 RAM, and NVIDIA Tesla V100 GPU. The SDM settings were: (i) $n = 256$, $r = 103$, and $H = 1,000,000$; (ii) $n = 1,000$, $r = 451$, and $H = 1,000,000$; and (iii) $n = 10,000$, $r = 4850$, and $H = 1,000,000$. There is no benchmark for kernel single_scan5_unroll because it returns the wrong result in this GPU. The problem is related with the premises of the optimization used by this kernel, which are not true for this GPU. For the histogram of durations, see Figures 56, 57, 58, and 59.



(a) $n = 256$, $r = 103$, and $H = 1,000,000$

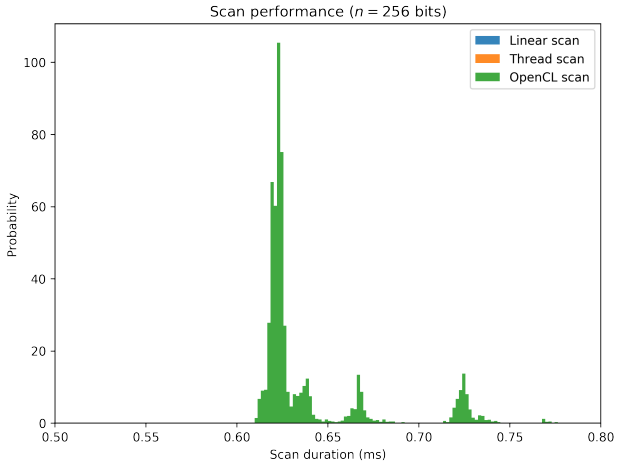


(b) $n = 1,000$, $r = 451$, and $H = 1,000,000$

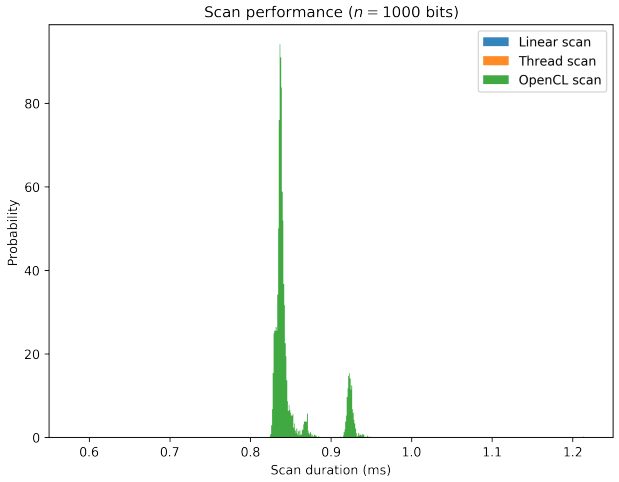


(c) $n = 10,000$, $r = 4805$, and $H = 1,000,000$

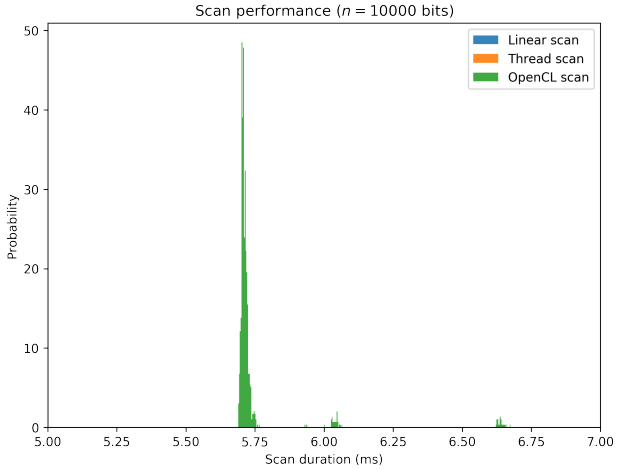
Figure 56: Kernel comparisons for Amazon EC2 p3.2xlarge with Intel Xeon E5-2686v4 processor, 488GB DDR3 RAM, and NVIDIA Tesla V100 GPU.



(a) $n = 1,000$, $r = 451$, and $H = 1,000,000$

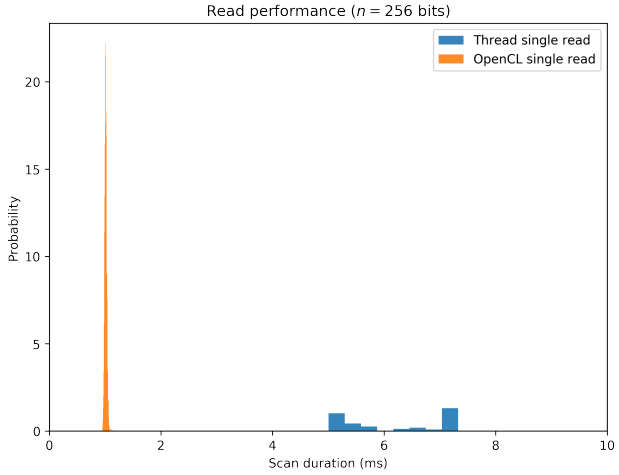


(b) $n = 1,000$, $r = 451$, and $H = 1,000,000$

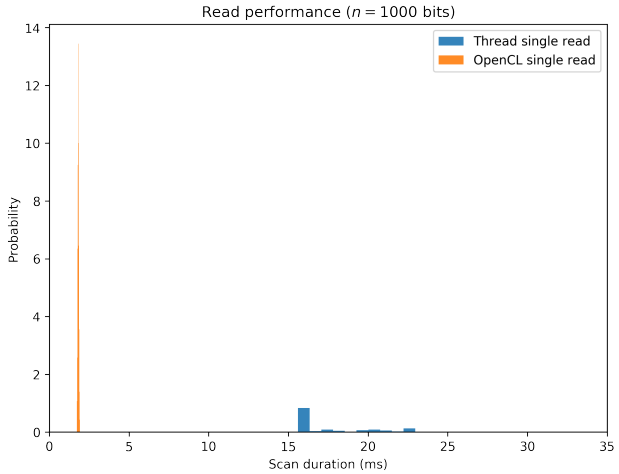


(c) $n = 10,000$, $r = 4805$, and $H = 1,000,000$

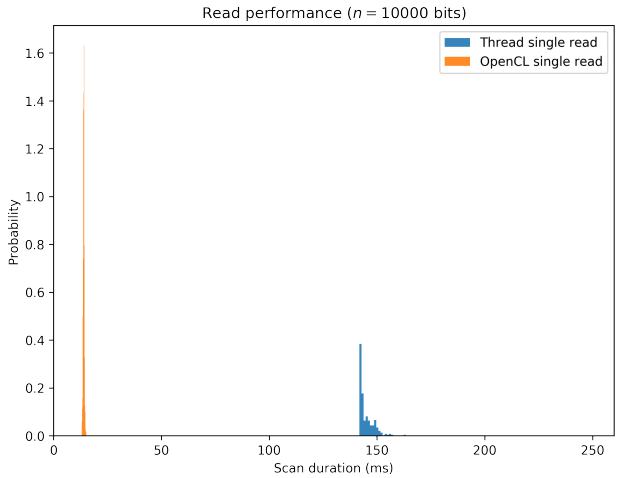
Figure 57: Scanner comparisons for Amazon EC2 p3.2xlarge with Intel Xeon E5-2686v4 processor, 488GB DDR3 RAM, and NVIDIA Tesla V100 GPU.



(a) $n = 1,000$, $r = 451$, and $H = 1,000,000$

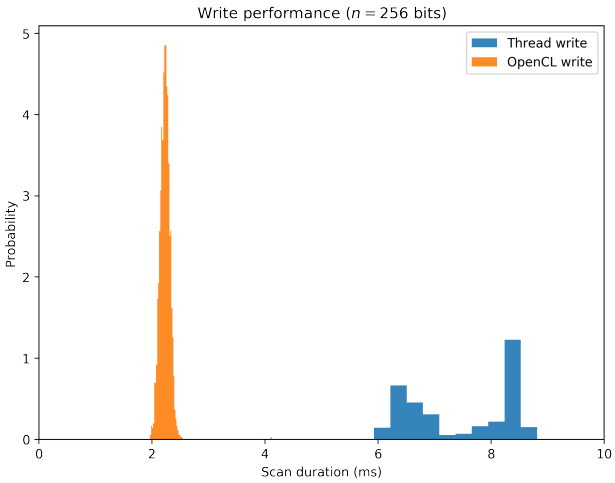


(b) $n = 1,000$, $r = 451$, and $H = 1,000,000$

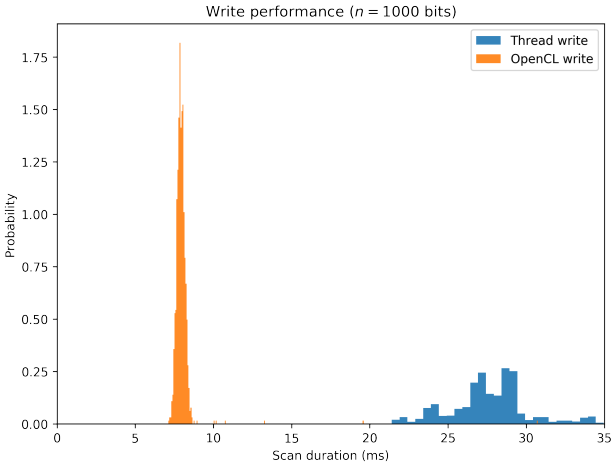


(c) $n = 10,000$, $r = 4805$, and $H = 1,000,000$

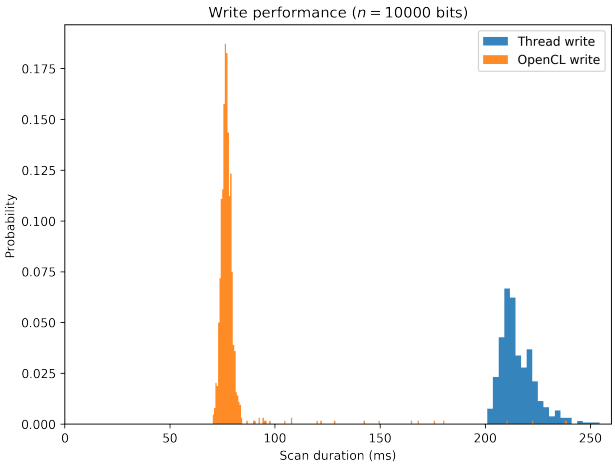
Figure 58: Read operation comparisons for Amazon EC2 p3.2xlarge with Intel Xeon E5-2686v4 processor, 488GB DDR3 RAM, and NVIDIA Tesla V100 GPU.



(a) $n = 1,000$, $r = 451$, and $H = 1,000,000$



(b) $n = 1,000$, $r = 451$, and $H = 1,000,000$



(c) $n = 10,000$, $r = 4805$, and $H = 1,000,000$

Figure 59: Write operation comparisons for Amazon EC2 p3.2xlarge with Intel Xeon E5-2686v4 processor, 488GB DDR3 RAM, and NVIDIA Tesla V100 GPU.

18

RESULTS (VI): SUPERVISED CLASSIFICATION APPLICATION

Supervised classification problem consists of categorizing data into groups after seeing some samples from each group. First, it is presented pieces of data with their categories. The algorithm learns from these data, which is known as the learning phase. Then, new pieces of data are presented and the algorithm must classify them into the already known groups. It is named "supervised" because the algorithm will not create the groups itself. It will learn the groups during the learning phase, in which the groups have already been defined and the pieces of data have already been classified into them.

Although this problem has already been studied (REF), our intention here is to show that a pure SDM may also be used to classify data. Fan and Wang [41] has used SDM to solve a classification problem, recognizing handwriting letters from images, but he used a mix of genetic algorithm with SDM, which is very different from the original SDM described by [54]. Even though his algorithm has classified properly, we were intrigued whether a pure SDM would also classify successfully.

Hence, we have developed a supervised classification algorithm based on a pure SDM as our main memory. Our goal was to classify noisy images into their respective letters (case sensitive) and numbers. For some examples, see Figure 60.

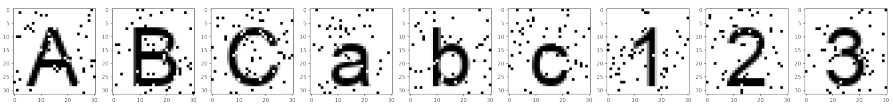


Figure 60: Examples of noisy images with uppercase letters, lowercase letters, and numbers.

The images had 31 pixels of width and 32 pixels of height, totaling 992 pixels per image. Each image was mapped into a 1,000-bit bitstring in which the bits were set according to the color of each pixel of the image. So, white pixels were equal to bit 0, and black pixels to bit 1. The eight remaining bits were all set to zero. This was a bijective mapping (or one-to-one mapping), i.e., there was only one bitstring for each image, and there was only one image for each bitstring.

A total of 62 classification groups have been trained in the SDM. For each of them, it was generated a random bitstring. Thus, the groups' bitstrings were orthogonal between any two of them. There is one

image for each of the 62 groups in Figure 61. Notice that the SDM has never seen a single image with no noise.

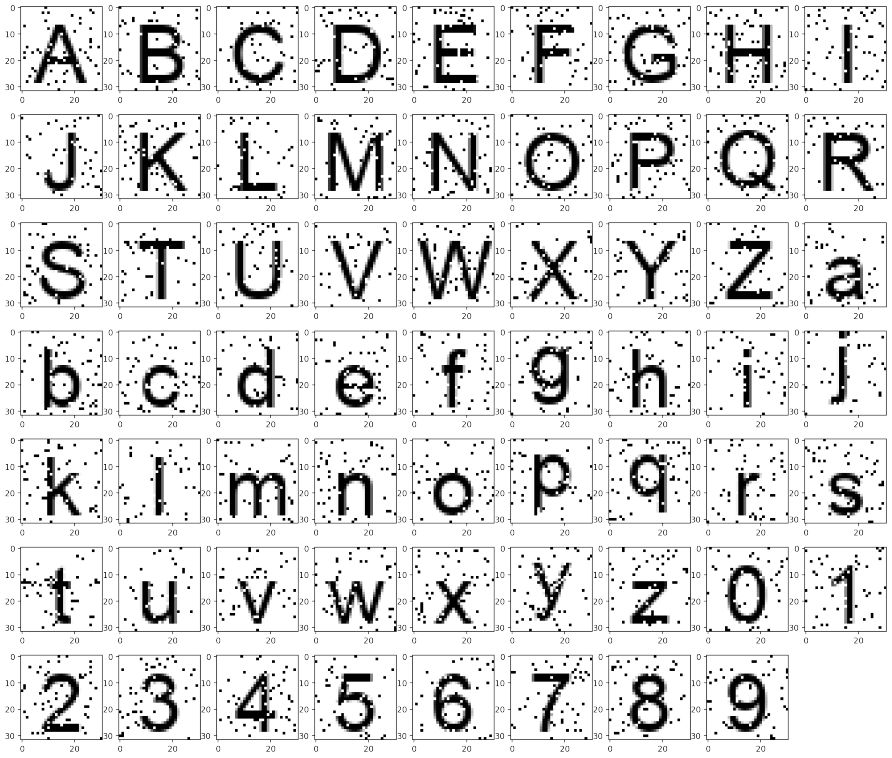


Figure 61: One noisy image for each of the 62 classification groups.

The association of images to groups was stored as sequences in SDM, as detailed by Kanerva [54] in Chapter 8. During the learning phase, the image bitstrings were stored pointing to their group's bitstrings, i.e., write(addr=bs_image, datum=bs_label). Thus, in order to classify an unknown image, we only had to read from its address and check which group has been found.

During the learning phase, we have generated 100 noisy images for each character. The images had 5% of noise, i.e., 5% of their pixels have been randomly flipped. For example, see the generated images for letter A in Figure 62. Then, we have written the classification group bitstring into the bitstring associated to each noisy image, i.e., write(bs_image, bs_label). For a complete image training set, see Appendix XYZ.

Finally, we have assessed the performance of our classifier. We had done it in three different scenarios: high noise (20%), low noise (5%) and no noise. See Figures 63 and 64 for images with 20% noise and no noise. The low noise scenario had the same noise as the training set. For each scenario, we had classified 620 unknown images with ten images per group.

The performance was calculated as the percentage of hits for each group. We did not expect the same performance for all groups

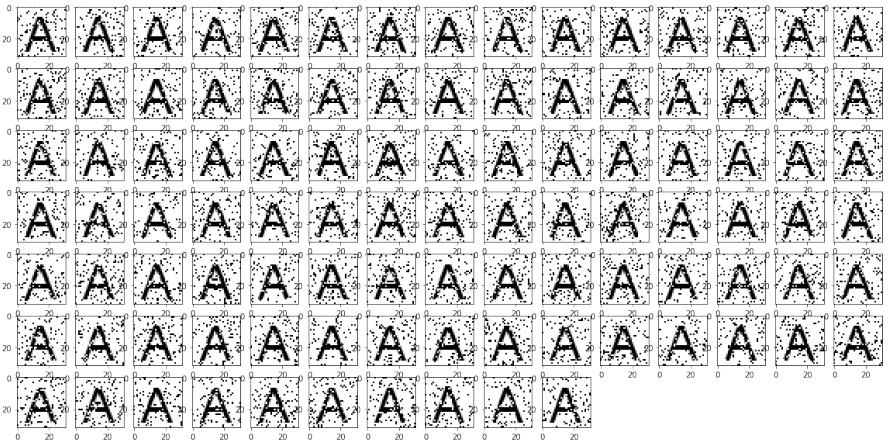


Figure 62: 100 noisy images generated to train label A.

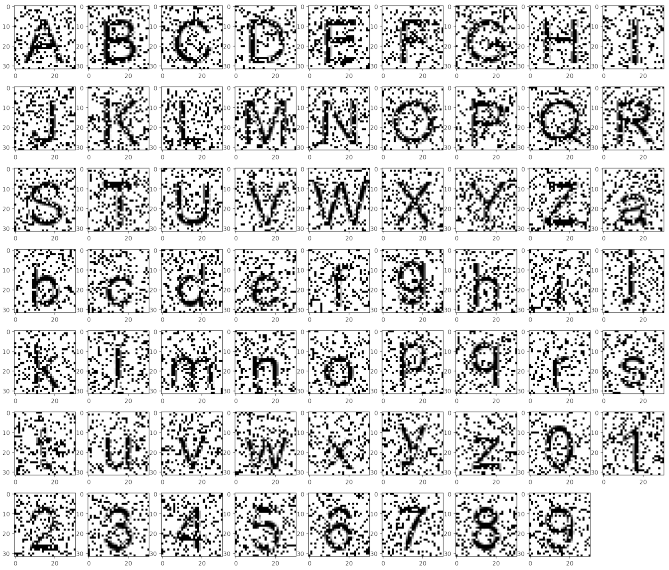


Figure 63: Images generated using a 20% noise for the high noise scenario.

because some groups become very similar to other depending on the noise level, and this similarity may even confuse a person (see Figure 65).

In the no noise scenario, the classifier has hit all characters, except letter "l" which was wrongly associated with the group of "i". We believe that it happened because the classifier had never seen an image with no noise and the difference between the images of "l" and "i" is smaller than the critical distance. So, both groups have been merged and it converged to only one of them. In our simulation, it happened to be the group of "i".

In the low noise scenario, it has made few mistakes. It correctly classified all images but some from characters "b", "e", "f", "l", "t", and "9". It completely classified "l" images to the "i" group. In the

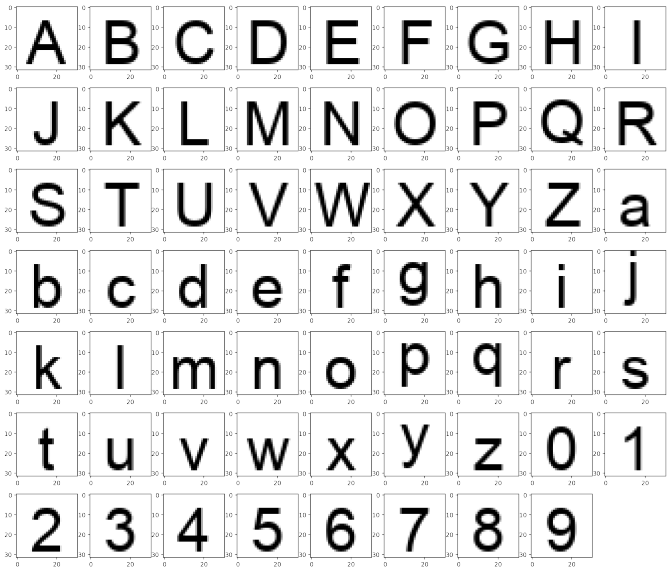


Figure 64: Images generated for the no noise scenario.

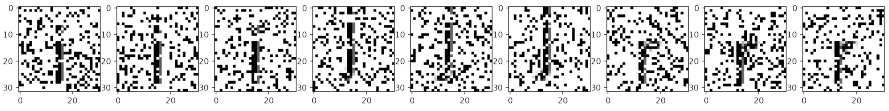
other cases, it made just a few mistakes. See Figure 66 to check the images and their classification.

The high noise scenario is the most interesting, because, even in a high noise level, the classifier has hit most of the characters. It has hit all images for 44 out of 62 groups and made at least one miss for the other 18 groups. The misses may be seen in details in Figure **??**.

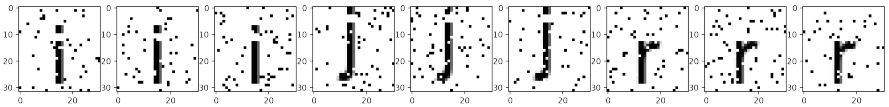
The critical distance plays an important role in the classification error. As we have 62 groups and each has been trained with 100 images, there were 6,200 writes to the memory. When an image is being classified, it will have to converge to a group, and the convergence depends on the distance between this image and the images from the training set, i.e., in the noise level.

In our simplified scenario, there is neither translation nor rotation. Future work may explore how sensible this classification algorithm is to these operations. We expect that, with proper training, the algorithm will remain classifying the images with a good hit rate.

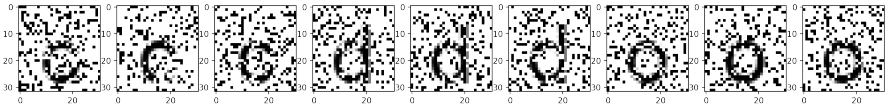
These results show that the SDM may be used as a supervised classification algorithm. Although we do not believe that the mapping between images and bitstrings are even close to the way human cognition deals with images, we believe the results are exciting and useful to many possible real-world problems.



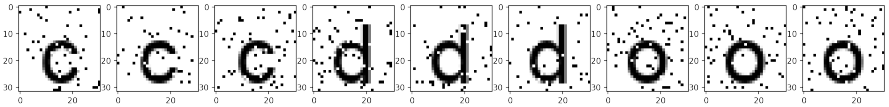
(a) “i”, “l”, and “r” with 20% noise.



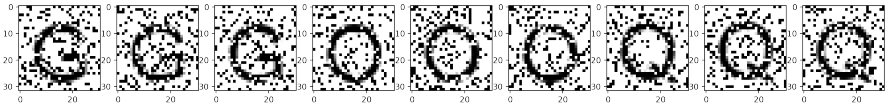
(b) “i”, “l”, and “r” with 5% noise.



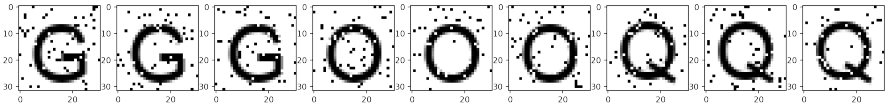
(c) “c”, “d”, and “o” with 20% noise.



(d) “c”, “d”, and “o” with 5% noise.

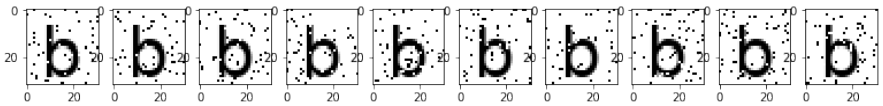


(e) “G”, “O”, and “Q” with 20% noise.

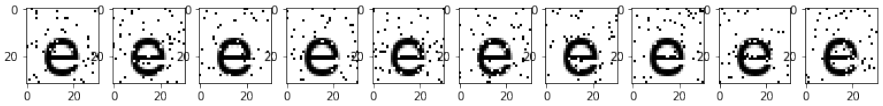


(f) “G”, “O”, and “Q” with 5% noise.

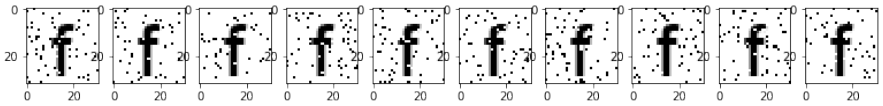
Figure 65: Images of different characters which may be confusing depending on the noise level.



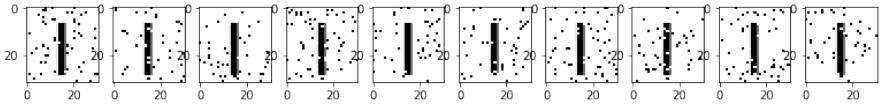
(a) Images from character "b which were classified as [b, b, b, h, b, o, b, h, b, b], respectively. It has made 3 misses.



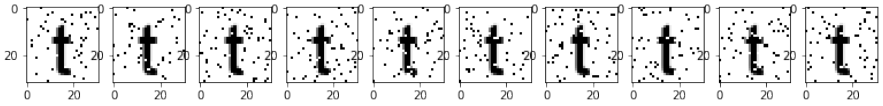
(b) Images from character "e which were classified as [e, e, e, e, e, e, e, e, o, e], respectively. It has made 1 miss.



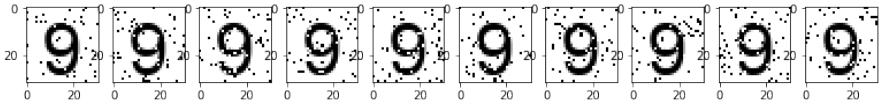
(c) Images from character "f which were classified as [i, f, f, I, I, I, f, f, f, f], respectively. It has made 4 misses.



(d) Images from character "l" which were classified as [i, i, i, i, i, i, i, i, i, i], respectively. It has missed them all as if both groups have been merged.

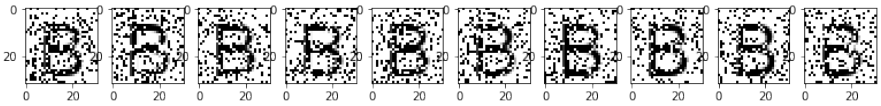


(e) Images from character "t" which were classified as [t, t, t, t, t, t, t, i, t, t], respectively. It has made 1 miss.

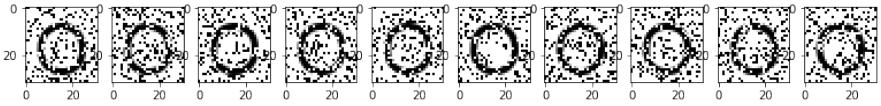


(f) Images from character "9" which were classified as [9, 9, o, 9, 9, 9, o, o, 9, 9], respectively. It has made 3 misses.

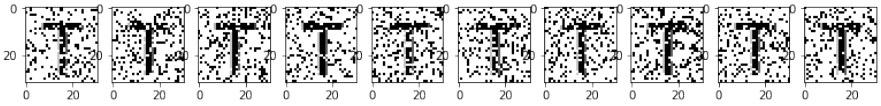
Figure 66: Characters in the low noise scenario in which the classifier has made at least one mistake. In all the other cases, it correctly classified the images. We may notice that the groups of "i" and "l" have been completely merged by the classifier, because it cannot distinguish them, not even with no noise.



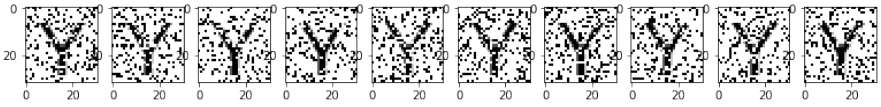
(a) Images from character “B” which were classified as [S, B, B, B, B, B, B, B, B, B]. It has made 1 mistake.



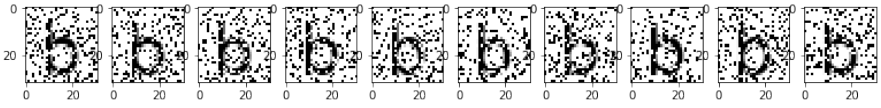
(b) Images from character “O” which were classified as [G, G, O, O, O, O, O, O, O, O]. It has made 2 mistakes.



(c) Images from character “T” which were classified as [T, T, T, T, T, I, T, T, T, T]. It has made 1 mistake.



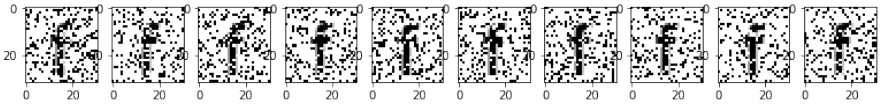
(d) Images from character “Y” which were classified as [Y, I, Y, Y, Y, Y, Y, Y, Y, Y]. It has made 1 mistake.



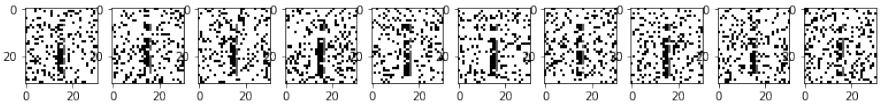
(e) Images from character “b” which were classified as [o, o, o, b, o, h, h, b, b, o]. It has made 7 mistakes.

(f) Images from character “c” which were classified as [c, c, c, c, c, o, c, c, c, o]. It has made 2 mistakes.

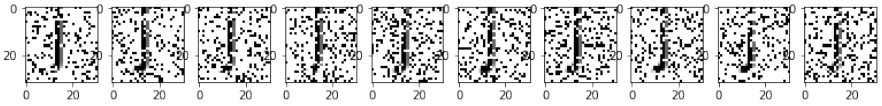
(g) Images from character “e” which were classified as [e, o, e, o, o, o, e, o, o, e]. It has made 6 mistakes.



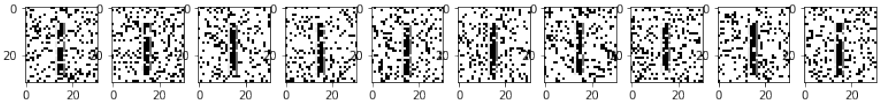
(h) Images from character “f” which were classified as [I, I, I, I, i, I, I, I, I, I]. It has missed them all.



(i) Images from character "i" which were classified as [i, i, i, I, i, i, i, i, I, i]. It has made 2 mistakes.

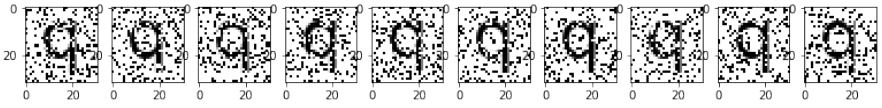


(j) Images from character "j" which were classified as [j, j, j, I, I, j, j, j, j, I]. It has made 3 mistakes.

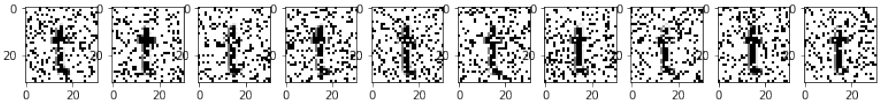


(k) Images from character "l" which were classified as [I, i, I, I, I, I, i, I, I, i]. It has missed them all.

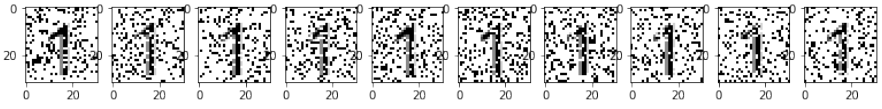
(l) Images from character "n" which were classified as [u, n, n, n, n, n, u, u, u, h]. It has made 5 mistakes.



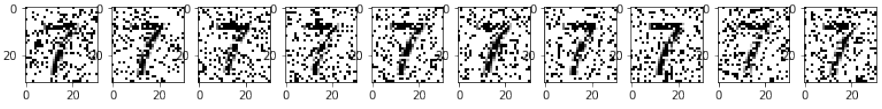
(m) Images from character "q" which were classified as [q, q, q, q, q, q, q, q, q, g]. It has made 1 mistake.



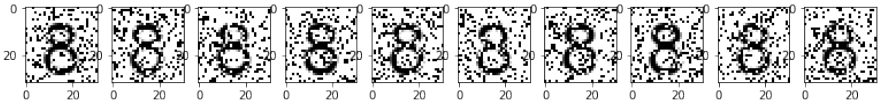
(n) Images from character "t" which were classified as [I, r, I, i, I, i, i, I, I, i]. It has missed them all.



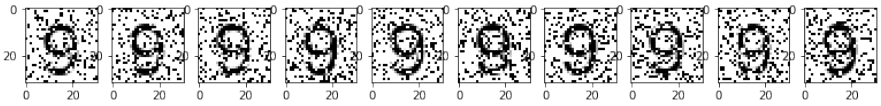
(o) Images from character "1" which were classified as [1, I, 1, I, 1, 1, I, I, 1, I]. It has made 5 mistakes.



(p) Images from character "7" which were classified as [7, 7, 7, I, 7, I, I, 7, 7, 7]. It has made 3 mistakes.



(q) Images from character "8" which were classified as [8, 6, 6, 6, 8, d, 8, 8, d, 6]. It has made 6 mistakes.



(r) Images from character "9" which were classified as [9, 0, 6, 0, 9, 0, 0, 9, 0, 0]. It has made 7 mistakes.

Figure 65: Characters in the high noise scenario in which the classifier has made at least one mistake. In all the other cases, it correctly classified the images.

19

RESULTS (VII): IMAGE NOISE FILTERING APPLICATION

Image noise filtering consists in removing the noise from an input, in our case an image. Our images are black & white images, and the noise is generated randomly flipping some of their pixels from black to while and vice versa. In Figure 66, we may see an image with different levels of noise, from 0% to 45% in steps of 5%. It makes no sense to apply 50% of noise because it would absolutely randomize the image.

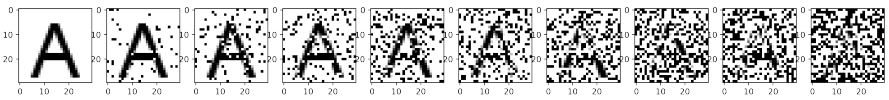


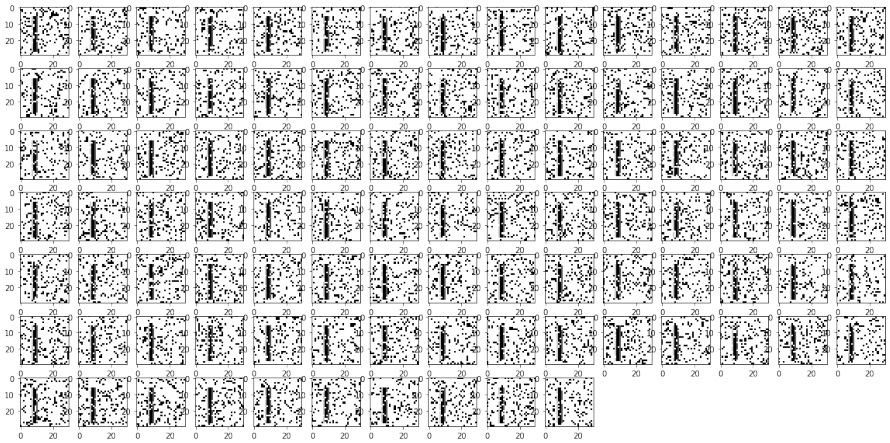
Figure 66: Progressive noise into letter "A", from 0% to 45% in steps of 5%.

The images have 30 x 30 pixels, totaling 900 pixels per image. Each image is mapped into a bitstring of 1,000 bit in which the bits are set according to the color of each pixel of the image. White pixels are assigned to bit 0, and black pixels to bit 1. The 100 remaining bits are all set to zero. This is a bijective mapping (or one-to-one) from images and bitstrings, i.e., there is one, and only one, bitstring for each image, and vice versa.

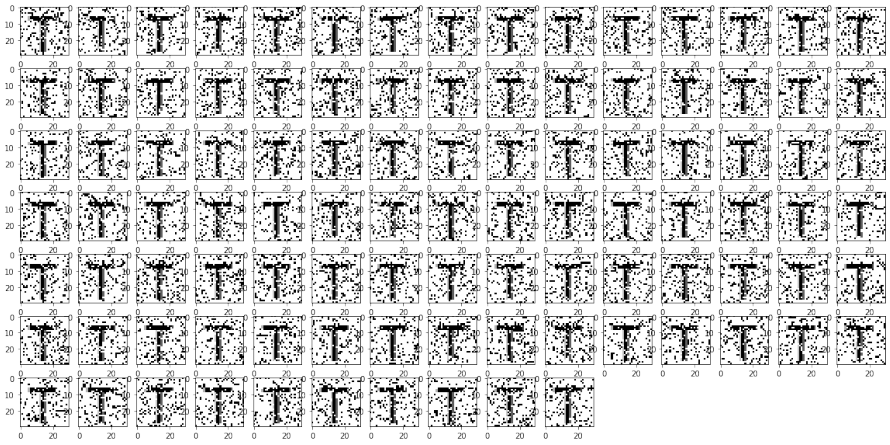
In the learning phase, 200 noisy images were generated and written into SDM. Half of the letter "I" and half of the letter "T" (see Figure 67). They were written into their own addresses, i.e., `write(address=bs_image, datum=bs_image)`.

Then, in order to test the filtering, we just have read from noisy images, and the results were remarkable. We were able to clean images up to 42% of noise (see Figure 68). While SDM has never seen a clean version of the letters, it just learned from the learning phase which pixels have appeared more frequently and choose them.

A simplified mathematical analysis would be: During the learning phase, 200 images with 15% of noise were written to SDM, so, the average distance between them and the clean image was 150 bits. Thus they shared, on average, 175 hard locations with the clean image. In these 175 hard locations, the counter's value for a black pixel mapped to bit 1 was $(1 - 0.15) \cdot 200 - 0.15 \cdot 200 = 140$. Finally, let's analyze the reading. When reading from a noisy image with 42% of noise, the average distance between the noisy image and its clean image is 420, which means they share, on average, 6 hard locations. As the average number of activated hard locations is 1,072, the sum of their counters will be, on average,

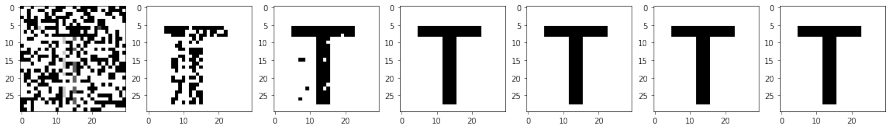


(a) Letter "I" with 15% of noise.

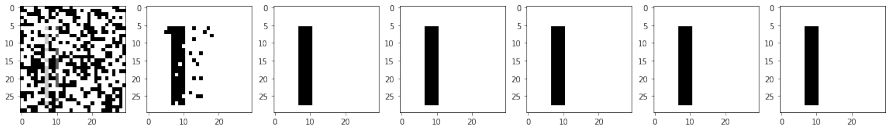


(b) Letter "T" with 15% of noise.

Figure 67: Training images written into the SDM. They were written in their own addresses — write(address=bs_image, datum=bs_image).



(a) Steps of reading from letter "T" with 42% of noise



(b) Steps of reading from letter "I" with 42% of noise

Figure 68: In order to test the SDM as a noise filter, we read from noisy images expecting to get a clean image. It is interesting to highlight that SDM has never seen a clean version letters "T" and "I".

$Y = 6 \cdot 140 - \sum_{i=1}^{1072-6} X_i$, where X_i is a Bernoulli trial with probability 0.5. Hence, $P(\text{black pixel}) = P(Y > 0) =$

$P(6 \cdot 140 - \sum_{i=1}^{1072-6} X_i > 0) = P(\sum_{i=1}^{1066} X_i < 840) = 99.99\%$. But, when reading from a noisy image with 45% of noise, the average number of hard locations shared with the clean image is only 3. Thus the sum of the activated hard locations' counters will be, on average, $3 \cdot 140 - \sum_{i=1}^{1072-3} X_i$, and $P(\text{black pixel}) = P(\sum_{i=1}^{1069} X_i < 420) = 1.28 \cdot 10^{-12}$. The probability abruptly drops from 100% to 0% when the noise goes from 42% to 45% (see Figure 69). The analysis for while pixels is exactly the same, but with opposite signs. The code to calculate this probability is available in "Noise filter - Math analysis" notebook [19].

This is a simplified analysis because it does not take into consideration the hard locations shared by the different letters. It works fine for our example because letters "I" and "T" are almost orthogonal and share, on average, only one hard location.

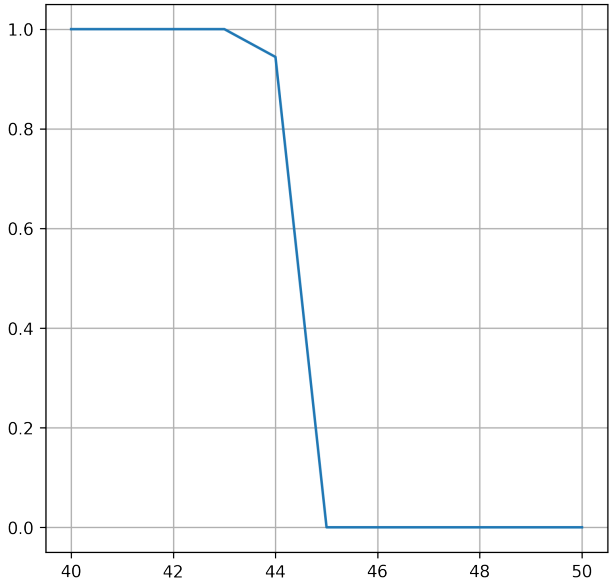
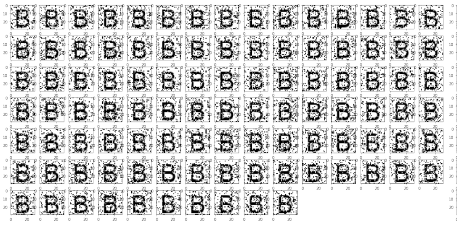


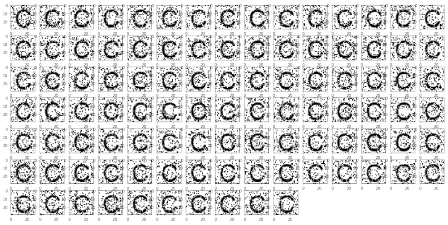
Figure 69: Probability of getting the right pixel when reading from an image with noise p. It assumes that SDM was trained with 200 images with 15% noise.

If the intersection between images becomes too high, the noise filter stops working properly. We have confirmed it writing the letters "B", "C", "D" and the numeral "8". They share a high number of hard locations, and our noise filter could not filter their noise correctly. The training sets can be seen in Figure 70 and the results in Figure 71.

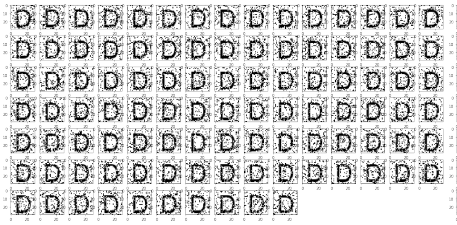
A possible solution to this interference problem is to use labels. Each label has a random bitstring, which will be chunked with the images before writing into SDM. Hence, before reading, we also have to chunk the image with the label — which means we need to know the label of each image. The chunk was done using the exclusive OR (XOR) operator, i.e., $bs\_chunck = bs\_image \oplus bs\_label$. In other



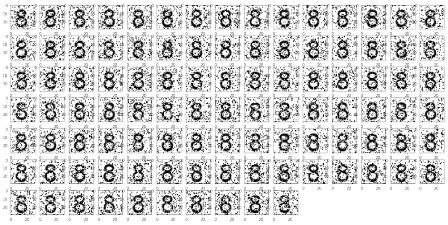
(a) Letter “B” with 15% of noise.



(b) Letter “C” with 15% of noise.

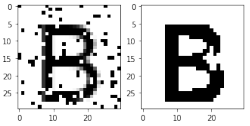


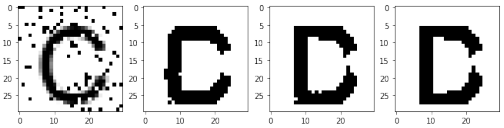
(c) Letter “D” with 15% of noise.

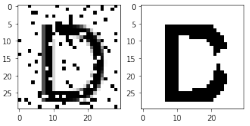


(d) Letter “8” with 15% of noise.

Figure 70: Training images in which the intersection between images is too high. They were written in their own addresses — write(address=bs_image, datum=bs_image).







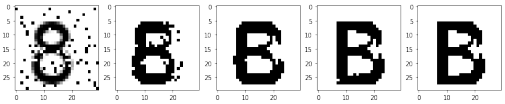


Figure 71: When the intersection between images becomes too high, there appears some interference in the resulting image. All cases have 10% noise. We can notice that the empty space on the right side of the “C” letter generates some white pixels on the right side of both “B” and “D” letters.

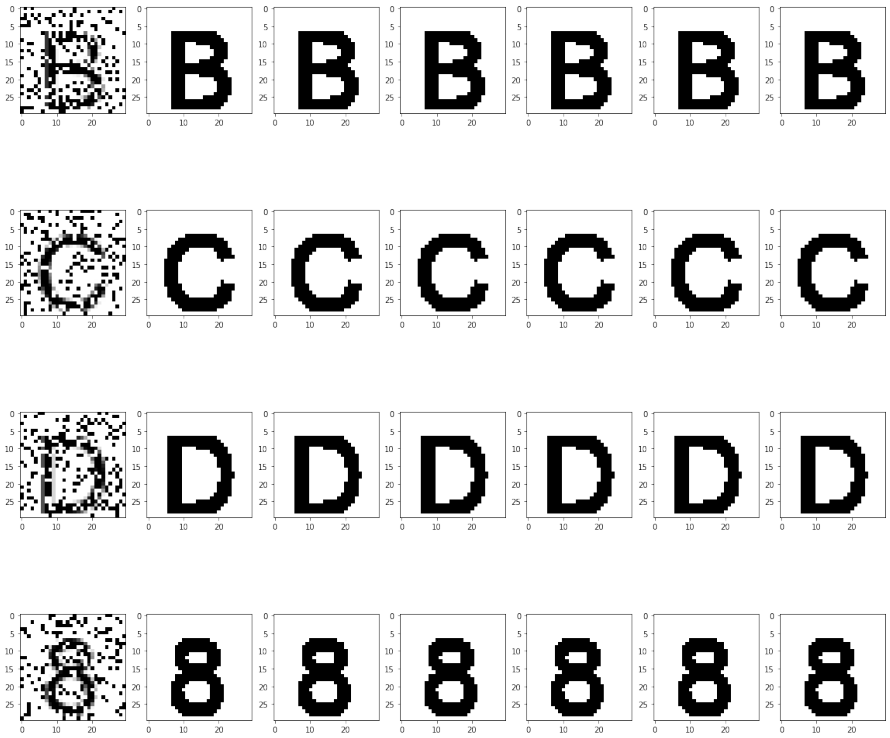


Figure 72: Using labels solves the interference problem when the intersection between images becomes too high. All cases have 20% noise.

words, we run `write(address=bs_chunk, datum=bs_label)` during the training, and `read(address=bs_chunk)` during the testing. We used the same training set as before, and the results can be seen in Figure 72.

The chunk through exclusive OR (XOR) works because of Theorem 18, which says that chunking the images with labels will generate, on average, orthogonal bitstrings. Thus, these orthogonal bitstrings will not interfere with each other because they share, on average, only one hard location.

The disadvantage of using labels is that it requires classification of the images. In our example, we just used the correct label with each image, but we could have used our classification algorithm as a pre-processing step, and only then run the noise filter.

Theorem 18. *If* ν_1 *and* ν_2 *are random bitstrings, then* $\forall a, b, \mathbf{E}[d(a \oplus \nu_1, b \oplus \nu_2)] = n/2$.

Proof. Let $\mathbb{A} = \{i | a^i = b^i\}$ be the indexes in which the bits of a are equal to the bits of b, and $\mathbb{B} = \{i | a^i \neq b^i\}$ be indexes in which the bits of a are different from the bits of b. Thus,

$$\begin{aligned} d(a \oplus v_1, b \oplus v_2) &= \sum_{i=1}^{n} d(a^i \oplus v_1^i, b^i \oplus v_2^i) \\ &= \sum_{i=1}^{n} (a^i \oplus v_1^i) \oplus (b^i \oplus v_2^i) \\ &= \sum_{i=1}^{n} (a^i \oplus b^i) \oplus (v_1^i \oplus v_2^i) \\ &= \sum_{i \in \mathbb{A}} (a^i \oplus b^i) \oplus (v_1^i \oplus v_2^i) + \sum_{i \in \mathbb{B}} (a^i \oplus b^i) \oplus (v_1^i \oplus v_2^i) \end{aligned}$$

For $i \in \mathbb{A}$, $a^i \oplus b^i = 0$, and follows:

$$\begin{aligned} (a^i \oplus b^i) \oplus (v_1^i \oplus v_2^i) &= 0 \oplus (v_1^i \oplus v_2^i) \\ &= v_1^i \oplus v_2^i \\ &= d(v_1^i, v_2^i) \end{aligned}$$

Hence, $\mathbf{E}\left[\sum_{i \in \mathbb{A}} d(a^i \oplus v_1^i, b^i \oplus v_2^i)\right] = \mathbf{E}\left[\sum_{i \in \mathbb{A}} d(v_1^i, v_2^i)\right] = |\mathbb{A}|/2$, because v_1 and v_2 are random bitstrings and their average distance is half the number of bits.

For $i \in \mathbb{B}$, $a^i \oplus b^i = 1$, and follows:

$$\begin{aligned} (a^i \oplus b^i) \oplus (v_1^i \oplus v_2^i) &= 1 \oplus (v_1^i \oplus v_2^i) \\ &= d(1, v_1^i \oplus v_2^i) \end{aligned}$$

Hence,
$\mathbf{E}\left[\sum_{i \in \mathbb{B}} d(a^i \oplus v_1^i, b^i \oplus v_2^i)\right] = \mathbf{E}\left[\sum_{i \in \mathbb{B}} d(1, v_1^i \oplus v_2^i)\right] = |\mathbb{B}|/2$.

Finally, $\mathbf{E}\left[d(a \oplus v_1, b \oplus v_2)\right] = |\mathbb{A}|/2 + |\mathbb{B}|/2 = n/2$, since $|\mathbb{A}| + |\mathbb{B}| = n$ □

A final note is in order. In no way is it claimed here that these bitstrings should form a plausible representation of letters in the human mind (See, for instance, Hofstadter [51] for a marvelous discussion of the subtleties and fluidity involved in that process). These letters should merely be seen as *invariant patterns* that are processed by the memory, which is able to capture their underlying invariant structure even when presented only under heavy noise.

20

RESULTS (VIII): THE POSSIBILITY OF UNSUPERVISED REINFORCEMENT LEARNING

Reinforcement learning has increasing prominence in the media after AlphaZero has won all games from both the best chess grandmasters in the world and the best chess engines. What is incredible about these victories is that AlphaZero has almost no knowledge about chess game and has learned all its movement playing against itself for 4 hours. Basically, it knows only the valid movements and had to learn everything from scratch, which it did using a reinforcement learning algorithm.

Reinforcement learning is a machine learning algorithm which learns from the rewards of its actions. So, it receives the game state as input, it decides which action will be taken, and then it learns from the rewards of all the actions it has chosen. In theory, it learns after each reward feedback it receives, improving its decision over time and presenting intelligent behavior. A positive reward would indicate that the chosen action should be encouraged, while a negative reward would indicate the opposite. In some algorithms, there may be a neutral reward which would indicate that the chosen action was neither positive nor negative. How each type of reward should be handled depends on each algorithm.

We have done some experiments with an SDM as a memory for a TicTacToe player. Basically, it receives the current board state and returns which action should be played. At the end of the game, it receives both the sequences of boards and the winner and is supposed to learn from them.

In our approach, there were 9 possible actions: one for each cell of the TicTacToe game. Action 1 means playing in the first cell of the first row; action 2 means playing in the second cell of the first row; and so one. Figure 73 shows the TicTacToe board numbering and the link between each cell and its respective action.

Our algorithm to decide what should be played is simple: it reads the current board from SDM and then it chooses the valid action with the highest score. To calculate the scores, the bitstring is split into 9 parts, one per action (Figure 73). The number of bits 1 in each part indicates the score of its respective action.

After a game has finished, it is time to learn from its decisions. Our algorithm has three rewards: positive, negative, and neutral. It always learns from both players, no matter who wins or if it was a draw. The winner's sequence of actions feeds our positive reward learning. The

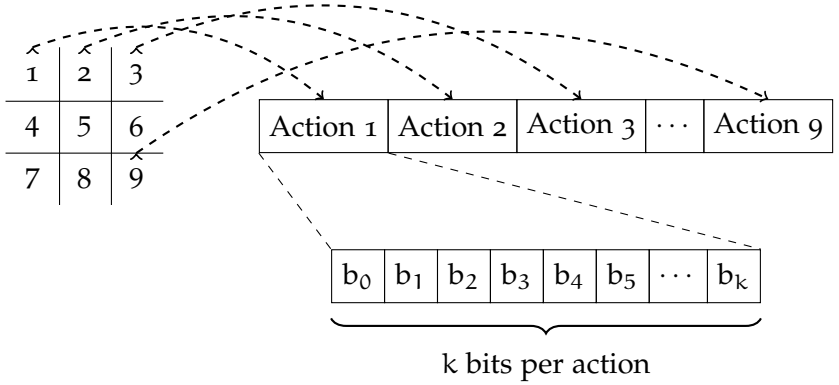


Figure 73: Each action is a cell in the TicTacToe board and is mapped to slice of the bitstring.

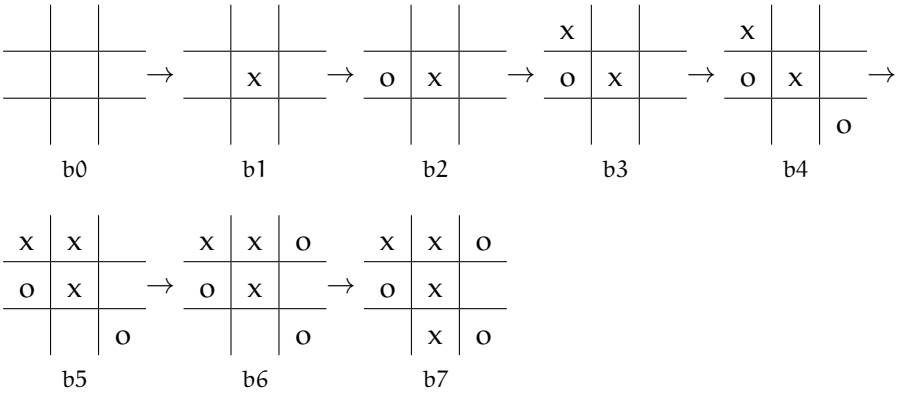


Figure 74: Example of a game with 7 movements in which X wins.

loser's sequence of actions feeds our negative reward learning. If it is a draw, both sequences feed our neutral reward learning.

Each board state has a unique random bitstring. For instance, each board in Figure 74 has its own unique random bitstring. Thus, if a specific board state is reached again, SDM will return the scores for each action.

Let $b_0, b_1, b_2, \ldots, b_n$ be the board sequence of the game (see Figure 74). In order to learn, our algorithm will map each action which goes from b_k to b_{k+1} to a reward bitstring and then will write a pointer from b_k to this reward bitstring, i.e., `write(bs_board, bs_reward)`.

The reward bitstring may be a positive, negative, or neutral bitstring. The positive reward bitstring is randomly generated and then only the bits related to the action will be set to 1 (see Figure **??**). The negative reward bitstring is also randomly generated and then only the bits related to the action will be set to 0. Finally, the neutral reward bitstring is only a random bitstring.

The idea behind these reward bitstrings is to increase the score of positive rewards (all bits set to 1), and to decrease the score of

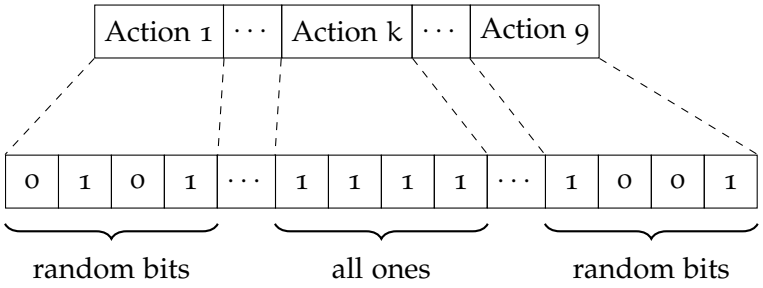


Figure 75: Positive reward bitstrings used in our reinforcement learning algorithm.

negative rewards (all bits set to 0). The neutral reward will have, on average, half of its bits 1 and the other half 0. Thus, it is in the middle of a positive reward and a negative reward.

Let the b be the sequence of boards and a be the sequence of actions, then $b_0 \xrightarrow{a_0} b_1 \xrightarrow{a_1} b_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} b_n$. Suppose there is a winner, thus the winner's actions will be the sequence $a_{n-1}, a_{n-3}, a_{n-5}, \ldots$, while the loser's actions will be the sequence $a_{n-2}, a_{n-4}, a_{n-6}, \ldots$.

The positive reward learning will be writing the following pointers in SDM: $b_{n-1} \to a_{n-1}$, $b_{n-3} \to a_{n-3}$, and so on. The reward bitstring of a_{n-1} will have all bits set to one in a_{n-1}'s slice. All other bits will be random.

The negative reward learning will be writing the following pointers in SDM: $b_{n-2} \to a_{n-2}$, $b_{n-3} \to a_{n-3}$, and so on. The reward bitstring of a_{n-2} will have all bits set to zero in a_{n-2}'s slice. All other bits will be random.

If it is a draw, all actions will be mapped to the neutral reward bitstring, which is simply a random bitstring.

There are also weights associated with positive, negative, and neutral rewards. They are used to indicate what goal is more important. For instance, if SDM should try to win in the first place, no matter if it may lead to losing, the weight of the positive reward should be higher than the others. But, if it is more important not to lose, then the weight of the negative reward should be higher.

In the very beginning, SDM is empty, and its counters are zeroed. So, any reading will result in a random bitstring, because it will flip a coin for all counters. Thus, the chosen action will be random, as their scores will follow a binomial distribution. It is precisely the desired behavior — we play randomly until we learn. In fact, it will happen every time an unknown board is seen.

Internally, every board is mapped into a random bitstring and passed to SDM. As every two boards are, on average, orthogonal, SDM knows nothing about the boards themselves, neither whether they are consecutive or not. It knows only the score of the actions

according to the games it has seen and learned from. Hopefully, the actions will lead to a victory or a draw.

In more details, the next movement decision consists in one read from SDM, resulting in a bitstring. Then the scores of the actions are calculated counting the number of 1 in each part of the bitstring. The chosen action is the one which is valid and has the highest score.

20.1 TRAINING

Our algorithm learned playing games against opponents. We had four types of opponents: (i) another SDM player, (ii) a random player whose actions are always random; (iii) a smart player whose actions wins when it can, block the opponent when it can, or are random; and (iv) a human player.

The weights of the rewards were chosen to prevent losing. Thus the weight of the negative reward was 5, while the weight of the positive reward was 2 and the neutral was 1.

Every learn cycle had two parts: (i) 100 games learning and (ii) 100 games testing. So, it has never learned during the testing phase and has not affected the measure of the statistics.

20.2 RESULTS

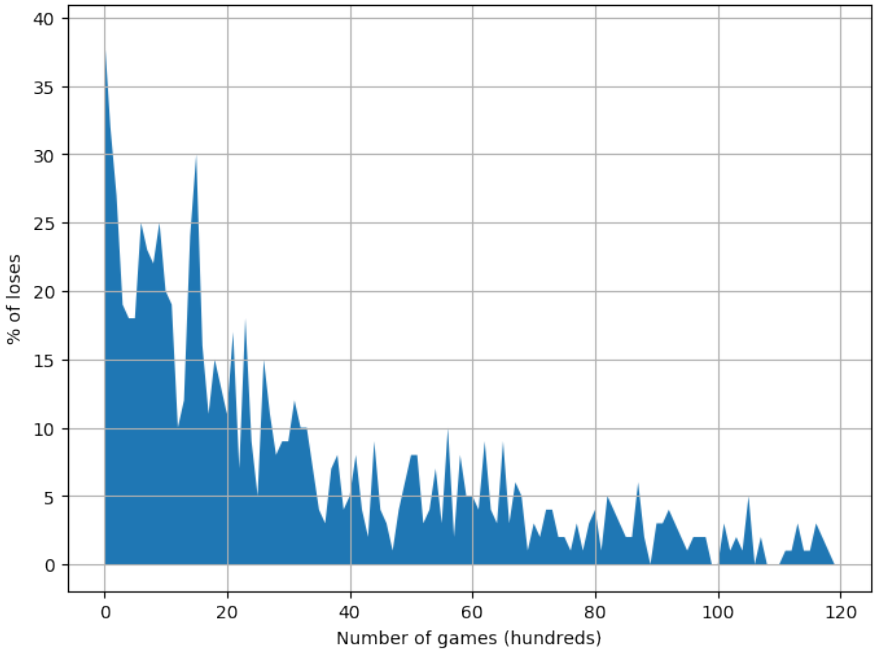
When playing against the random player, SDM has already started improving after the first 100 training games. Its winning rate converges quickly to around 80%, while the drawing rate starts to grow after 2,000 games and keeps rising until the end. The losing rate keeps decreasing until it reaches cycles of 100 games without any loss. See Figure 76.

When playing against the smart player, SDM has started learning how not to lose during the first training cycles. The drawing rate grows quickly in the first 500 games and slowly since then. The winning rate grew to around 20% and remained there until 6,000 games, while the drawing rate kept increasing. Then, after learning how not to lose, SDM started learning how to win, since, after 7,000 games, the winning rate started to increase. See Figure 77.

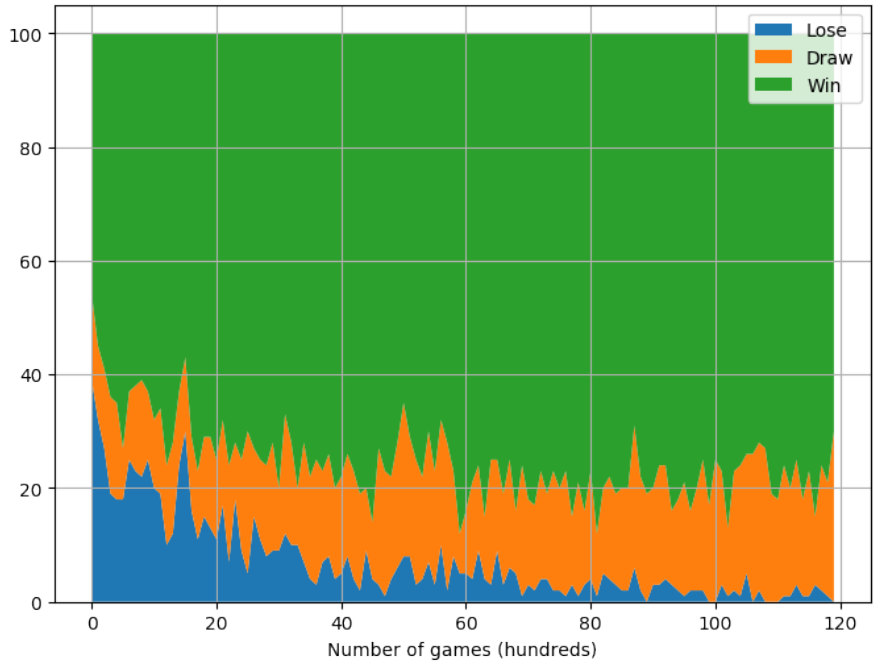
When playing against another SDM player, both player quickly learns how not to lose. During the first 100 testing games, without any learning game, they have behaved like two random players. After the 100 training games, they have learned how not to lose, and the drawing rate grows quickly, reaching 100%. See Figure 78.

When playing against mixed players, SDM has also adapted. In Figure 79, it has played 6,000 games against the random player, the smart player, and another SDM player. In each cycle, one of them was randomly chosen. The number of losses over time is decreasing, whereas the number of wins and draws change a lot. It is easy to

notice when the other SDM player was chosen since all games in that cycle have drawn.

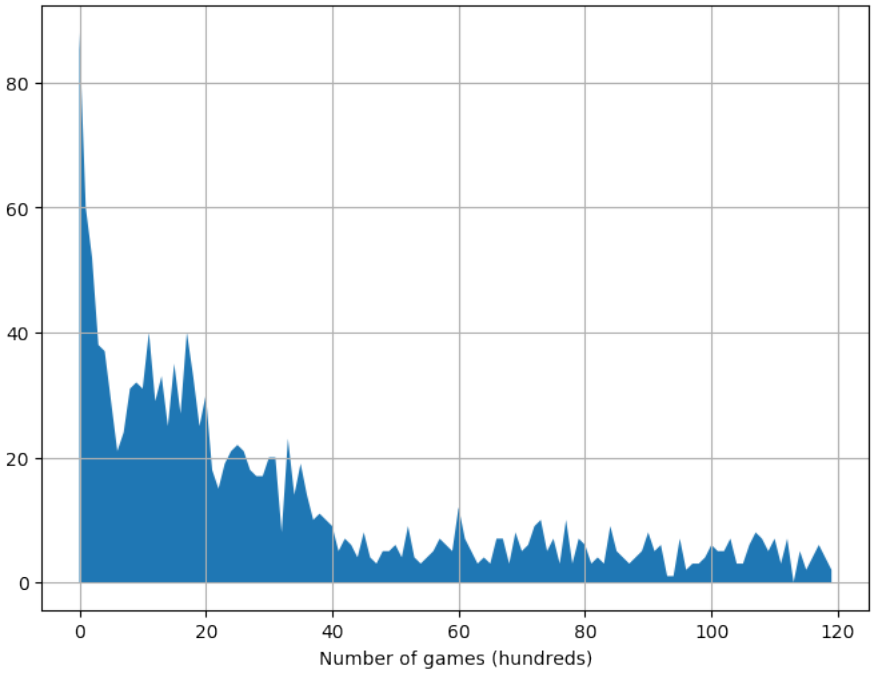


(a) Losses over time

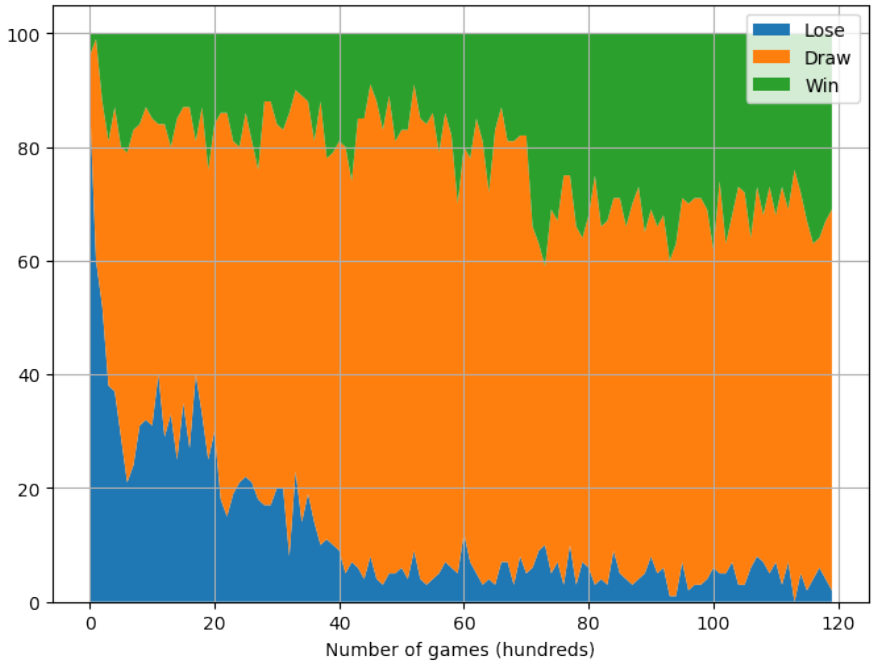


(b) Results over time

Figure 76: Results playing against the random player. Each cycle was made of 100 games for training, and then 100 games for measuring statistics.

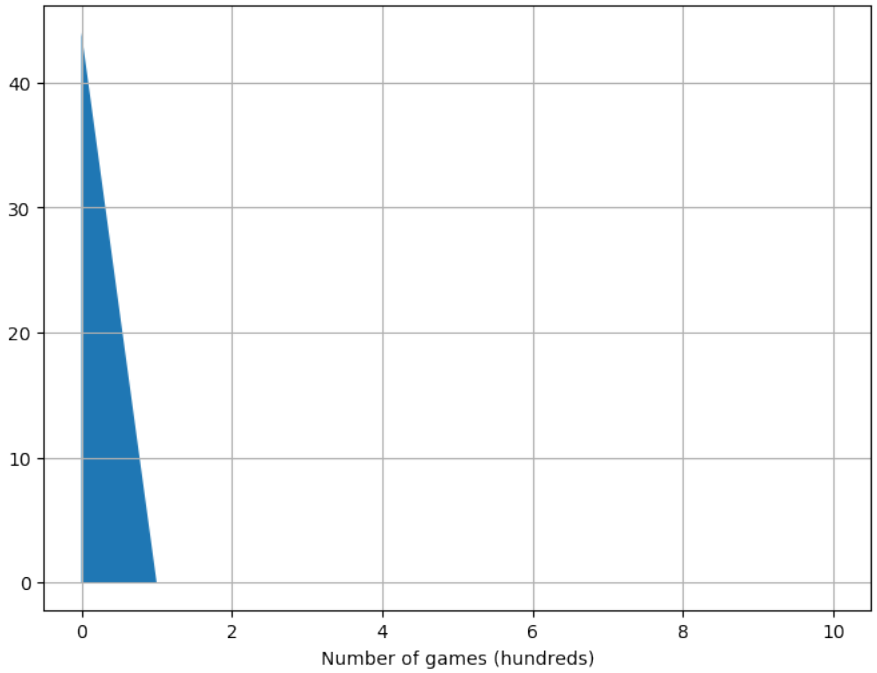


(a) Losses over time

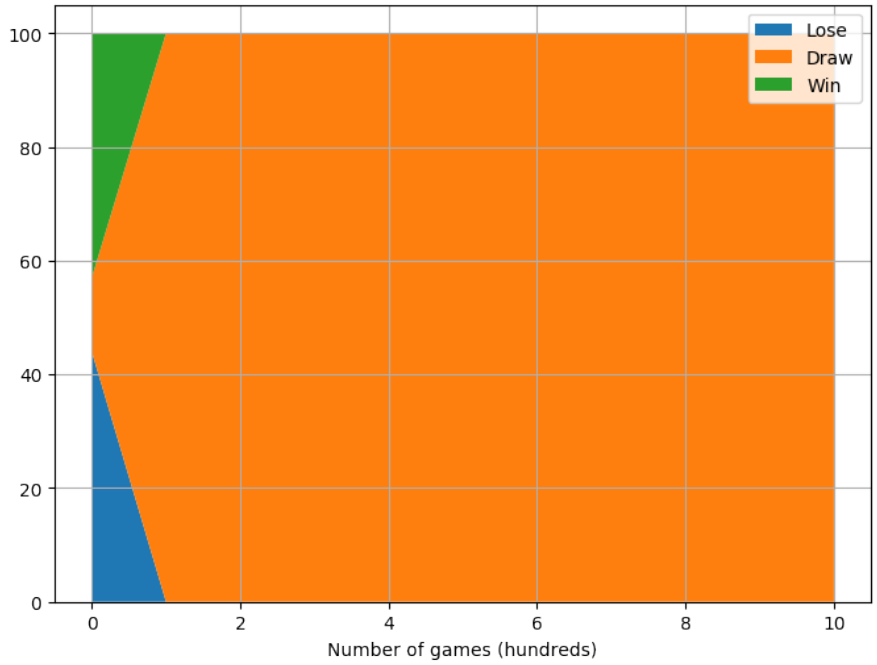


(b) Results over time

Figure 77: Results playing against the smart player. Each cycle was made of 100 games for training, and then 100 games for measuring statistics.

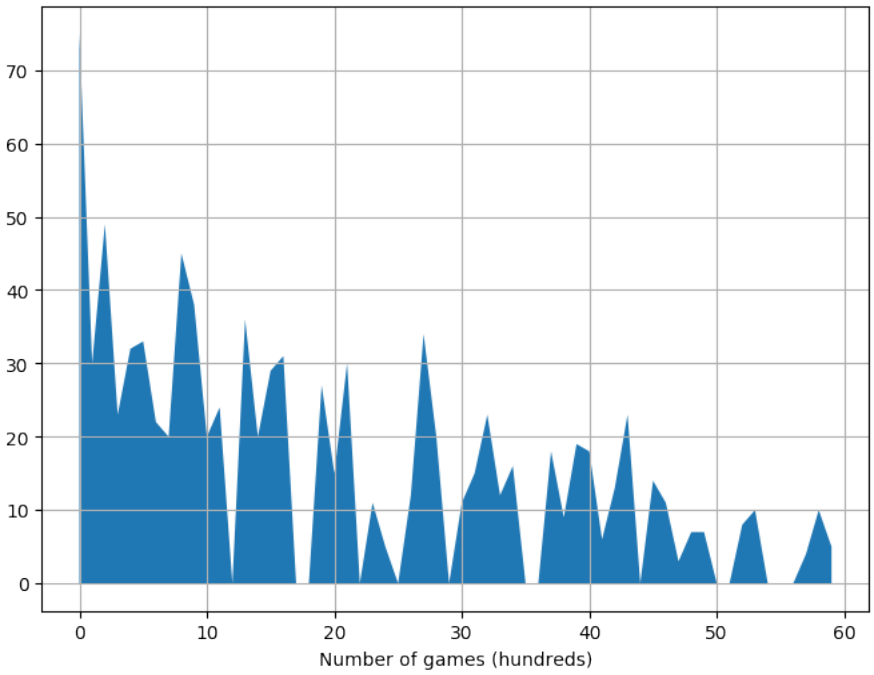


(a) Losses over time

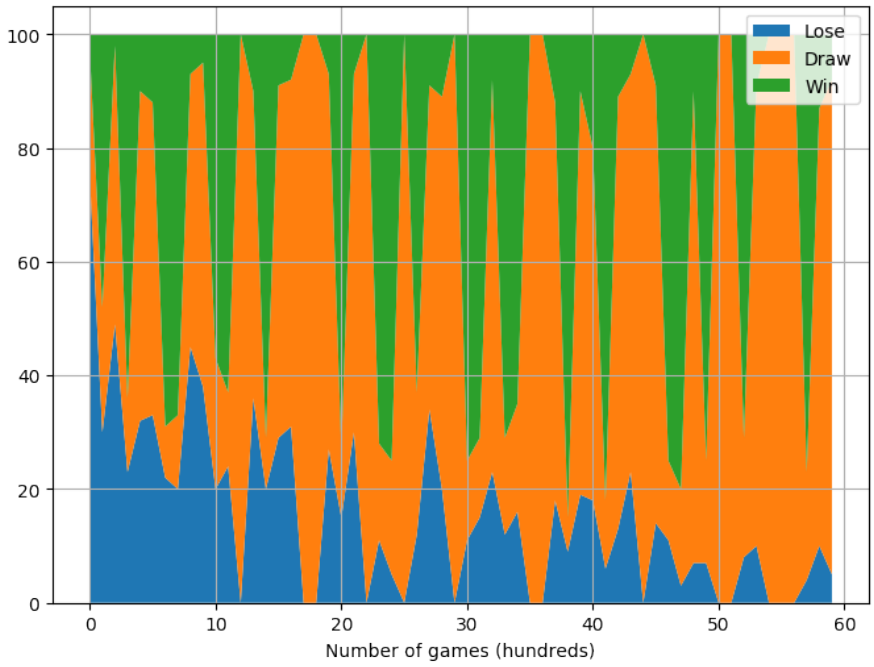


(b) Results over time

Figure 78: Results playing against another SDM player. Each cycle was made of 100 games for training, and then 100 games for measuring statistics.



(a) Losses over time



(b) Results over time

Figure 79: Results playing against a randomly chosen player between random player, smart player, and another SDM player. Each cycle was made of 100 games for training, and then 100 games for measuring statistics.

21

RESULTS (IX): INFORMATION-THEORETICAL WRITE OPERATION

My advisor, Alexandre Linhares, has proposed another write operation: an information-theoretical weighted write. In it, the sum of the counter's value is weighted based on the distance between each hard location's address and the reading address. The logic behind it is to vary the importance of each hard location inside the circle. It is only natural that one encodes an item in closer hard locations with a stronger signal, and a natural candidate for this signal function is the amount of information contained in the distance between the item and each hard location. Closer hard locations have lower probabilities and therefore should encode more information.

Note that this is not the first time in which a weighted function has been applied to writing in SDM — Hely et al. [50] suggest a rather complex spreading model based on floating point signals in the interval [0.05, 1.0] — they were, however, only able to test their model with 1,000 hard locations.

Consider the following. Information Theory [30] let us compute the precise amount of information in an event when given its probability p, through the measure of *self-information*:

$$I(p) = -\log_2(p)$$

Now, given any two n-sized bitstrings, the probability of their Hamming distance being exactly d is given by $P(X = d) = 2^{-n}\binom{n}{d}$, and the probability of it being at most d is:

$$P(X \leqslant d) = 2^{-n} \sum_{i=0}^{d} \binom{n}{i}$$

But we must consider that not all hard locations are activated in each write operation, which changes our probability function. Thus, let r be the access radius then:

$$\begin{aligned} P(X = d | X \leqslant r) &= \frac{P((X = d) \cap (X \leqslant r))}{P(X \leqslant r)} \\ &= \frac{P(X = d)}{P(X \leqslant r)}, \quad \text{as } d \leqslant r \\ &= \frac{2^{-n}\binom{n}{d}}{2^{-n}\sum_{i=0}^{r}\binom{n}{i}} \\ &= \frac{\binom{n}{d}}{\sum_{i=0}^{r}\binom{n}{i}}, \quad d \leqslant r \end{aligned}$$

And the probability of it being at most d is:

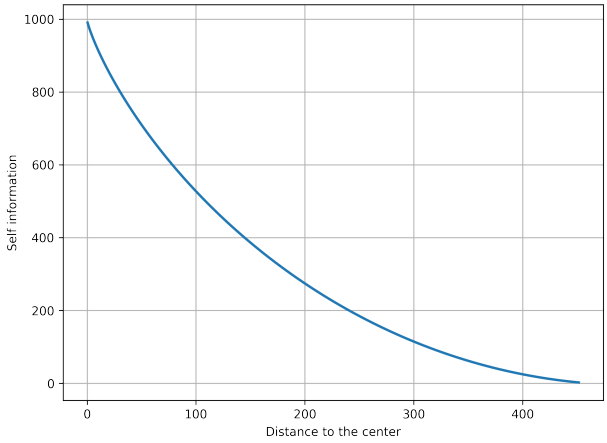
$$P(X \leqslant d | X \leqslant r) = \frac{\sum_{i=0}^{d}\binom{n}{i}}{\sum_{i=0}^{r}\binom{n}{i}}, \quad d \leqslant r$$

As expected, $P(X \leqslant d | X \leqslant r) = 1$ when $d = r$.

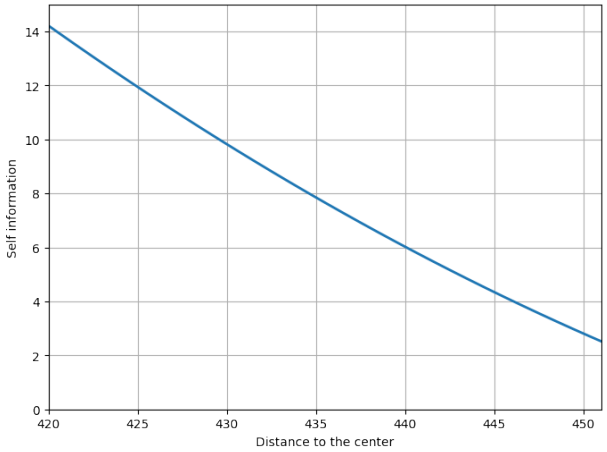
Hence the weighted write would, on each hard location, sum (or subtract) using the following weights, as seen in Figure 80:

$$w(d) = -\log_2 (P(X = d | X \leqslant r)) = -\log_2 \binom{n}{d} + \log_2 \sum_{i=0}^{r} \binom{n}{i}, \quad d \leqslant r$$

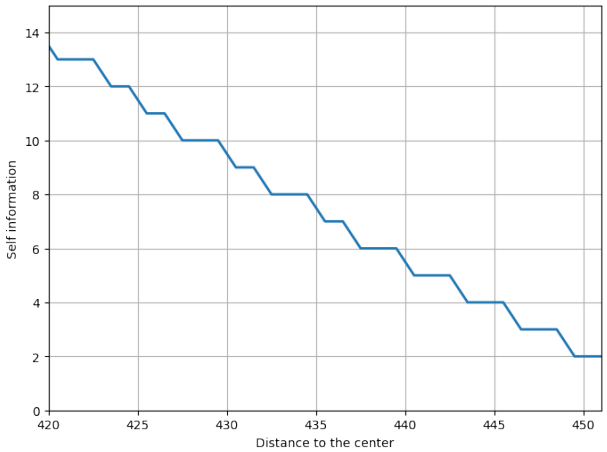
The initial results of this *Shannon write* operation can be seen in Figure 81 and seem promising. It seems that, when $n = 1{,}000$, $H = 1{,}000{,}000$, $r = 451$, and 10,000 written random bitstrings, the critical distance increased from around 221 to around 250. This increase may be interpreted as an improvement in SDM, because it would converge to the correct bitstring even for farther bitstrings. Note that 29 additional bits imply an attractor area 2^{29} times larger than the original. This Shannon write may affect memory capacity — possibly increasing it. Another point to keep in mind is that, since the modulus of the vectors are not uniform in this approach, the shape of the attractor may have asymmetries. Whereas these are just some initial tests, the idea seems meritorious so far. As for future research, we will execute all tests in the thesis and compare this Shannon write with the original Kanerva model.



(a) $w(d), d \in \{1, 2, ..., n\}$.

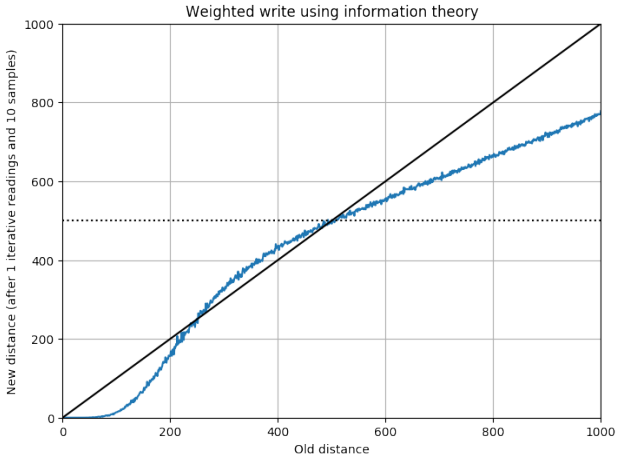


(b) $w(d)$ for the desired range.

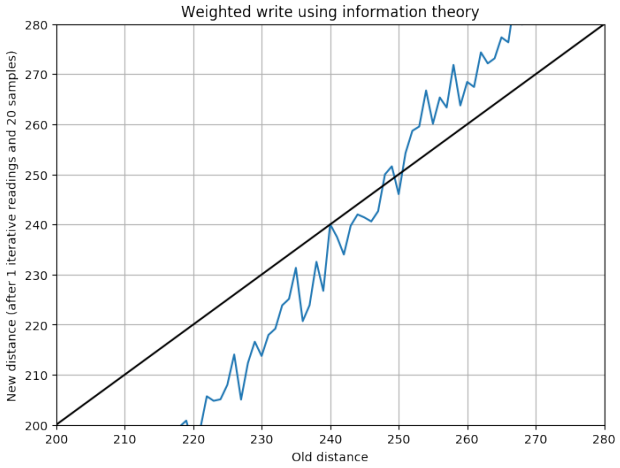


(c) Stepwise $\lfloor w(d) \rfloor$ for fast integer computation.

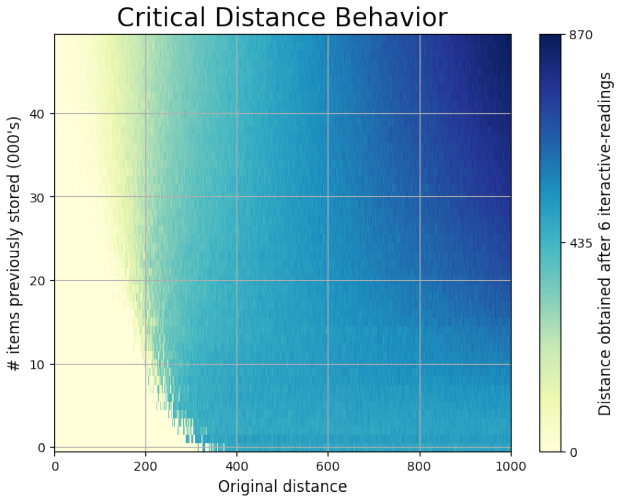
Figure 80: Shannon write operation: Computing the amount of information of a signal to each hard location in its access radius. (a) entirety of the space; (b) region of interest; (c) Fast integer computation is possible through a stepwise function.



(a) Write process weighted by the amount of information contained in the distance between the written bitstring and each hard location



(b) Zoom in Figure 81a



(c) Behavior of weighted write operation according to the distance from the center and the number of items previously stored in the memory

Figure 81: Behavior of the critical distance under the information-theoretic weighted write operation when $n = 1,000$, $H = 1,000,000$ and $r = 451$.

22 CONCLUSION

Sparse Distributed Memory is a viable model of human memory, yet it does require researchers to (re-)implement a number of parallel algorithms in different architectures.

We provide a new, open-source, cross-platform, highly parallel framework in which researchers may be able to create hypotheses and test them computationally with minimal effort. The framework is well-documented for public release at this time (http://sdm-framework.readthedocs.io), it has already served as the backbone of Chada's Ph.D. thesis [24]. The single-line command "pip install sdm" will install the framework on posix-like systems, and single-line commands will let users test the framework, generate some of the figures from Kanerva's theoretical predictions in their own machines, and — if interested enough —, test their own theories and improve the framework, and the benchmarks used to evaluate the framework, in open-source fashion. It is our belief that such work is a necessary component towards accelerating research in this promising field.

Here are interesting questions that have been considered during this work, but have had to be left for future research.

22.1 "I" versus "l"

The classification algorithm had some problems classifying the patterns "i" and "l", due to the low distance between them. In this case, SDM could not discriminate the differences — it has only considered the big picture. Although this behavior is close to how humans see things, we also have the ability to zoom and focus on the details, clearly discerning letter "i" from letter "l".

I have run the classification algorithm under the MNIST database of handwritten digits [37]. First, SDM has been trained with the 60,000 training images, and then it classified the 10,000 testing images. In these initial tests, the memory has given the correct classification for 79.22% of the images, which is inferior to the specialized algorithms. For instance, in 1998, LeCun et al. [60] have developed algorithms which achieve from 88% through a linear classifier, to 99.7% through a convolutional net. For a review of algorithms' performance in the MNIST database, see [59].

Looking into the reason behind images incorrectly classified, I have found that the issue is very related to the "i" versus "l" issue. Some handwritten digits are very close to others, and a "2" or a "7" may

look like a "1", for instance. So, how can we solve this issue without using anything specific to images? Machine learning algorithms use specific techniques to improve performance. I would like to unveil a solution psychologically closer to how we behave — even if that eventually leads to lower performance.

An unexplored idea is to use multiple SDMs which communicate. A first SDM would write the whole picture, just like we have done. Another SDM would write specific regions of the image, just like our eye focusing on specific regions. When reading, they may compose the counters and give a more precise classification.

22.2 MAGIC NUMBERS

Kanerva suggests, in his book, the use of 1,000 dimensions and 1,000,000 hard locations. More recently, he suggested the use of 10,000 dimensions, and on personal discussions suggested that this should be a minimum; as he has been concerned in latent semantic analysis and seems to be the proper scale in that application.

Each parameter set choice like this will lead to particular numbers — many of them emergent —, such as the access radius size, critical distance, memory capacity, and so forth. For example, the number of dimensions n leads to the number of standard deviations available on the space: $2\sqrt{n}$. One then must choose a proper multiple $m \times \sigma$: Kanerva (and most in the literature) have been using $m = 3$, as $\mu - 3\sigma$ selects approximately 1/1000 of the space. These constants will determine the size of the access radius. Then one must choose the number of hard locations, which will determine how many hard-locations are activated on average; and so on and so forth...

One intriguing question here is: is there a 'better' number of dimensions and hard locations? If so, can such numbers be better studied algebraically or numerically? How should these parameters be compared? What are the tradeoffs that should be considered? What are the 'best' benchmarks possible?

22.3 SYMMETRICAL, RAPIDLY ACCESSIBLE, HARD LOCATIONS

A hypercube with n dimensions can be divided by two hypercubes with $n - 1$ dimensions. Is there an algorithm that separates the area of each hard location in such a form that there exists a function mapping each bitstring in $\{0, 1\}^n$ to the set of hard locations it 'belongs to'? Though this would break Kanerva's assumption of a randomly yet uniformly distributed set of hard locations — for a perfectly symmetrical set of hard locations —, there could be large performance gains if such a mapping function from a bitstring to its

corresponding set of nearest hard locations exists. Note that others have attempted heuristic approaches to this[1].

Consider the hypercube with n dimensions. We want to select a subset of its vertices with cardinality 2^{20} that is symmetrically distributed over the space. Afterward, $\forall b \in \{0,1\}^n$, we want an algorithm A that yields the particular list of hard locations for b and all hard locations respect the desired properties of the memory.

A reduction from measuring the distance to 2^{20} hard locations to a computation of 2^{10} hard locations might yield astonishing performance gains, depending, of course, on our optimistic assumptions concerning existence and complexity of such algorithm. At large scales of computing, the very ability to perform some experiments is a function of sheer performance. The horizon of experiments — and possibly of knowledge — expands *as a function of computational demands*. A little more on this in my closing words.

22.4 ILLUMINATING, PAVING, CLEARING

Let us revisit, in these concluding thoughts, the emphasis employed over speed of computation. At first sight, that might seem like a typical objective of efficiency in computer science. But we are not only interested in the computer science effects here — the ambition is different. More important than this 'computer-sciency' goal, i.e., a beautiful, clean, efficient algorithm with the primary effect of enhanced speed, however, is the secondary effect on the sociology of science: *We can see farther*.

Beyond speed, I have also strived for *ease of use*. All the simulations and graphics generated in this thesis are promptly available to be re-executed and explored by those interested. I have generated a Docker image, which makes it even easier to explore the framework. After running the container, a Jupyter Notebook is available with sdm-framework and other tools already installed. We invite the reader to take a look and explore a little bit.

The overarching intention here is to not only provide a starting point, but to provide a documented Framework in which SDM research can be conducted. Consider having the ability to compare the results of a new ('forked') model to the previous 'best' (under a particular benchmark set). For example, some of the benchmarks that we plan to develop in future research are: how fast is convergence through iterative reading? How large is the attractor of the critical distance? How well does the system filter noise? How well does the system work under the supervised learning task? And

1 Kanerva [55] has recently mentioned that 'The computationally most efficient implementation of it, by Karlsson [57], is equivalent to the RAM-based WISARD of Aleksander et al. [2].' Note, however, that Karlsson's model may introduce non-uniform asymmetries in the space, and seems to require a staggering $O(2^n)$ hard-locations.

other authors may be able to improve this benchmark set themselves, as is usual in open source development. It is perhaps this facility of ease to build on top of previous work that seems most exciting at this stage.

Consider the misunderstanding concerning the SDM read operation: Dr. Stan Franklin describes Kanerva's read operation in a way that each hard location, at each dimension, provides only a single bit of information to the read operation (instead of Kanerva's full counter). We have referred to this modified read operation as Chada read[2]. Having an open, testable, codebase reduces the possibilities of such misunderstandings in the long run. Indeed, a high-quality codebase seems to have become a scientific community's form of unequivocally standing behind a consensus. For example, the journal Nature analyzed the top-100 cited papers in history, to find:

> ... some surprises, not least that it takes a staggering 12,119 citations to rank in the top 100 — and that many of the world's most famous papers do not make the cut. A few that do, such as the first observation of carbon nanotubes (number 36) are indeed classic discoveries. But the vast majority describe experimental methods or *software that have become essential in their fields. [...] The list reveals just how powerfully research has been affected by computation and the analysis of large data sets.*
>
> — Van Noorden et al. [99], emphasis mine.

It is no coincidence that scientific journals such as BMC Neuroscience, or the Journal of Machine Learning Research have specific sections on open-source software. The journal Neurocomputing states, bluntly: "software is scientific method by machine".

Of course, for the skeptical reader who may consider software a less worthy pursuit, there is also new work here. The mathematics of the model has been shown to be correct numerically (with a single, small, anomaly); we have shown how to execute unsupervised learning with nothing besides operations original to the SDM; we have studied the generalized Murilo read; we have seen noise filtering; the death of neurons; how information-theory may be of use; and finally, we have reproduced numerous of the original propositions put forth by Kanerva. The emphasis might have been on the *breadth of topics*, in detriment of depth here or there. The work on, say, reinforcement learning, is most definitely not the last word we will see on the subject, but a challenge left for readers to

2 Legend has it that my friend & colleague, Dr. Daniel de Magalhães Chada, along with Linhares, did not consult and re-check with Kanerva's book and only discovered the discrepancy in code and ideas a couple of years afterward.

contemplate. But this is due to our research group's enthusiasm for the topic; we do indeed believe that SDM is — if not correct — extremely close to a full scientific understanding of human long-term memory. If so, it is such a monumental achievement that we want readers to be able to see all of what we see and imagine the vastness of possibilities.

Ralph Waldo Emerson once said: *do not go where the path may lead. Go, instead, where there is no path, and leave a trail.* Professor Pentti Kanerva has left the trail. It is my job to illuminate it and to pave it and to clear it; to try to deliver an easier pathway for the next generation. Some essays completely shut the door close at the end; this one intends to leave it wide open. As the reader might have noticed, this final section does not read as an analysis of the work done; it reads, instead, as a *desideratum*, a prologue, a yearning for others to join me in imagining the shape of things to come.

23

APPENDIX

LIST OF JUPYTER NOTEBOOKS

Part IV

DIFFUSION AND DISMISSAL OF INNOVATION: FORECASTING THE NUMBER OF FACEBOOK'S ACTIVE USERS

24

INTRODUCTION

Why do entrepreneurs appear, not continuously, that is singly in every appropriately chosen interval, but in clusters? Exclusively because the appearance of one or a few entrepreneurs facilitates the appearance of others, and these the appearance of more, in ever-increasing numbers.

— Joseph Schumpeter

The way innovations diffuse through markets is an important and useful topic in marketing, which was made popular through Rogers' work [90] and has influenced many marketing researchers. Within innovation diffusion, Bass [9] proposed a model which forecasts how many people will have adopted new products or technologies by a given point in time. INFORMS members have voted this model as one of the Top 10 Most Influential Papers published in the 50-year history of Management Science in connection with the 50th anniversary of the journal [10].

The Bass [9] model was designed to forecast only innovation adoption, which is the first time one consumes the innovation. In the model, either the consumer has or has not consumed the innovation by a given point in time. Thus, recurrent customers are considered only once, because they have already consumed the innovation before. The model has only three parameters, which are estimated through the number of adoptors of the innovation. Though very simple, it is considered very robust.

Although it is clear that the marketing investment, the product prices, the economy itself, and many other variables affect the diffusion process, the Bass [9] model does not contemplate these variables and yet it is still able to describe the empirical adoption curve of a large number of new products and technological innovations. In order to explain the robustness of the model, Bass et al. [11] developed a general model including these variables, and they showed that this general model reduces to the Bass model as a special case. They also showed that the shape of the diffusion of innovation process is always the same, an S-curve. Norton and Bass [79] analyzed the diffusion of innovation for product substitution, explaining some unexpected changes in innovation adoption which were still unclear.

The motivation behind the present work is that there may be people who reject a particular innovation. Such people do not

recommend the innovation, on the contrary, they may publicly complain about it and bad-mouth it. This negative word-of-mouth effect of rejection has always existed [89, 15, 94, 22] but it is becoming more and more important as information can spread more easily and faster among people through the usage of new technologies [52, 48, 6]. Nowadays, before making a decision, people may search the internet for reviews and feedbacks about the innovation, and what they find affects their decision [27, 34, 40, 35, 61, 83]. A number of firms have already perceived this change caused in large part by the social media and have adapted to this new condition, like Starbucks [46], for example.

There is an extensive literature on innovation diffusion. Numerous extensions to the Bass [9] model have been proposed (for reviews, see Meade and Islam [68] and Peres et al. [82]), among which Mahajan and colleagues were the first to propose a model that includes the negative word-of-mouth [67]. The latter and other extensions that include negative word-of-mouth are much more complex than the Bass [9] model, and their parameters must be estimated using the number of people who have already adopted the innovation at a given time. A problem is that most recent innovations, like Facebook, Twitter, and Netflix do not disclose their total number of users. Actually, they deem this number confidential. They only disclose the number of active users, not including the number of users who have rejected them.

In this work, I propose an extension to the Bass model which (i) includes the negative effect of the rejections, (ii) is as simple as the Bass [9] model, and (iii) its parameters can be estimated using the number of people who have adopted and have not posteriorly rejected the innovation, i.e., the number of *active adopters*. An important difference between the number of *active adopters* and the *total adopters* is that first may decrease over time, while the latter cannot. First, I discuss the Bass [9] model and its parameters; then I describe the model extension and analyze four models of rejection; next, I detail the estimation method; after that, I estimate the models by using Facebook's active users dataset; and finally, I discuss the results and conclude.

The main contribution of this work is that the proposed extension is the first to include the effect of the rejections that can be estimated using the number of *active adopters* instead of the number of *total adopters*. The model can be applied to forecast the number of active adopters of these companies in the next quarters, which becomes an important tool for analysts, investors, and the companies themselves. As the number of active adopters is related to the market cap of these companies, these forecasts may be useful to estimate the future value

of the firms. At the time of this writing, Facebook's market cap is $222.69B, Twitter's is $23.18B, and Netflix's is $20.49B[1].

1 Data obtained from Yahoo! Finance website on December 1st, 2016.

25

THE BASS MODEL

The Bass [9] model is a simplification of the diffusion of innovation process proposed by Rogers [90]. The Rogers' classification of adopters has five classes: (i) innovators; (ii) early adopters; (iii) early majority; (iv) late majority; and (v) laggards. Bass simplified them to only two classes: innovators and imitators. The innovators are the ones who start using an innovation regardless of who else and how many other people are already using it. The imitators are the ones who concern themselves about who is using the innovation and, as long as many other people are using it, they are more inclined to adopt it. Thus, at the beginning of the diffusion of innovation, the majority of adopters are innovators. As more and more people adopt the innovation, the majority of new adopters shift to imitators.

Mathematically, the model presents itself with the following set of equations:

$$S(t) = mf(t) \tag{26}$$

$$Y(t) = mF(t) \tag{27}$$

$$\frac{f(t)}{1 - F(t)} = p + qF(t), F(0) = 0 \tag{28}$$

Both $S(t)$ and $Y(t)$ are related to the absolute amount of adopters; while $S(t)$ is the number of new adopters at time t, $Y(t)$ is the number of people who had already adopted the innovation by time t, thus $S(t) = \frac{d}{dt}Y(t)$. The model also has three positive parameters: (i) the potential market size m; (ii) the innovators parameter p; and (iii) the imitators parameter q. There are also $f(t)$ and $F(t)$ which are related to the percentage of the potential market: $f(t)$ is the percentage of the potential market which is adopting the innovation at time t, while $F(t)$ is the percentage of the potential market which had already adopted it at time t, thus $f(t) = \frac{d}{dt}F(t)$.

Therefore, $S(t)$ and $f(t)$ are related to the adoption rate at time t, while $Y(t)$ and $F(t)$ are related to the accumulated adopters at time t. In the beginning of the diffusion of innovation, there are no adopters at all, thus $Y(0) = F(0) = 0$. As the number of adopters can only increase over time, both $Y(t)$ and $F(t)$ are monotonically increasing functions, and both $S(t)$ and $f(t)$ are always greater or equal to zero.

The non-linear ordinary differential equation 28 is the main equation in the Bass model. Its left side is known as the hazard function and it expresses the probability of someone adopting the innovation, provided that he/she has not chosen to adopt it yet, i.e.,

the rate of adoption. Its right side means that this probability is at least p and increases linearly with the percentage of people who have already adopted the innovation, i.e., $F(t)$.

Solving the differential equation, Bass found a closed formula for the diffusion, which is $F(t) = \frac{1-e^{-(p+q)t}}{1+\frac{q}{p}e^{-(p+q)t}}$. This closed formula always has the famous shape of the S-curve, regardless of the values of $p > 0$ and $q > 0$. From the closed formula of $F(t)$, it is trivial to obtain the equations of $S(t)$, $Y(t)$, and $f(t)$.

The potential market size m is the unknown number of people who will have adopted the innovation after a very long time. It is not exactly the target market of the innovation, but a subset of it, as no product diffuses over its entire target market. If a company estimates and updates the model more than once for their product, the change in m is a change in the potential market and could help the company to understand whether their decisions in the meantime have increased or decreased the number of future adopters. As $\lim_{t\to\infty} Y(t) = m$, the whole potential market will have adopted the innovation at some point.

The innovator parameter p is related to the proportion of people in the potential market who adopt the innovation regardless the others. In other words, their decision to adopt is not influenced by the social system, but by other external factors. The bigger the p, the larger the number of innovators, thus the faster the diffusion at the beginning.

The imitator parameter q is related to the influence of those actually using the innovation on those who are not using it yet. This is why this parameter multiplies $F(t)$, which is the proportion of the market which had already adopted it. This influence is mainly understood as a result of the word-of-mouth recommendation. In other words, the more people use the innovation, the more other people will adopt it. The bigger the q, the larger the imitator effect, thus, the faster the diffusion.

Practitioners have been using this model to forecast future demand. First, they measure the number of adopters over time. Then, they estimate the parameters m, p, and q. Finally, they extrapolate $S(t)$ out of the measured time window and use its value as the forecast demand. They also calculate $m - Y(t)$ as a forecast of how many people have not adopted the innovation yet.

26

THE EXTENDED MODEL

I propose adding a new term in the differential equation 28 in order to include the effect of rejection, as in equation 29. Hereafter I will refer to: (i) the people who have adopted the innovation as *total users*; (ii) the people who have adopted the innovation and remain using it as *active users*; and (iii) the people who rejected the innovation as *inactive users*. Clearly, the function $Y(t)$ is the number of *total users*, which is equal to the sum of the number of *active users* with the number of *inactive users*.

$$\frac{f(t)}{1-F(t)} = p + qF(t) - wR(t), F(0) = 0, R(0) = 0 \tag{29}$$

The function $R(t)$ is the percentage of accumulated *inactive users* at time t. Thus, $A(t) = F(t) - R(t)$ is the percentage of accumulated *active users* at time t. Multiplying by m, $mA(t) = Y(t) - mR(t)$ is the number of *active users* at time t. Since $A(t) \geqslant 0$, $F(t) \geqslant R(t)$, which makes sense because it is not possible to have more *inactive users* than *total users*.

The negative word-of-mouth parameter w is related to how much the *inactive users* really affect the new adopters decision in the diffusion process. The bigger the w, the greater the negative influence of these *inactive users* on the new adopters. Another possible understanding is that w is related to how much the *inactive users* are bad-mouthing the innovation. The bigger the w, the more they bad-mouth the innovation.

Equation 29 could be rewritten as $\frac{f(t)}{1-F(t)} = p + qA(t) + (q-w)R(t)$. This form is useful in order to understand the impact of *active users* and *inactive users* on the rate of adoption at time t.

If $w = q$, then the rate of adoption increases linearly with the *active users*, since $\frac{f(t)}{1-F(t)} = p + qA(t)$. In other words, the positive word-of-mouth has exactly the same influence as the negative word-of-mouth on the new adopters. It also means that the rate of adoption is always greater than zero, thus the whole potential market will have adopted the innovation sooner or later.

If $w < q$, then the new adopters are more influenced by the number of *total users* than by the number of *inactive users*. It is just as if the *inactive users* do not bad-mouth the innovation so much, or at all. Again, the rate of adoption is always greater than zero, thus the whole potential market will have adopted the innovation sooner or later.

If $w > q$, then the proposed extension really differs from the Bass model. In this case, the influence of the *inactive users* is greater than

the influence of *total users*. So, if the rate of adoption were equal to zero ($wR(t) = p + qF(t)$), the innovation might not be adopted by the whole potential market, i.e., $\lim_{t\to\infty} F(t) < 1$.

The main contribution of this work is to have the choice to use the number of *active users*, which is the information that most of the companies disclose, in order to estimate the parameters of the proposed extension model. With that, both the number of *inactive users* and *total users* could be forecast. It is important to notice that the number of *active users* ($mA(t)$) could decrease over time. In fact, forecasting when this is going to happen may be crucial for corporations.

27

MODELS FOR R(t)

The proposed extended model already includes the effect of rejection through the $R(t)$ function and the w parameter. In order to complete the model, $R(t)$ has to be well defined. Let $r(t) = \frac{d}{dt}R(t)$ be the rate of new *inactive users* at time t. The proposed differential equations for $R(t)$ are the following:

$$\text{Model 1: } r(t) = \nu f(t) \tag{30}$$

$$\text{Model 2: } \frac{r(t)}{1 - R(t)} = \nu f(t) \tag{31}$$

$$\text{Model 3: } r(t) = \nu[F(t) - R(t)] \tag{32}$$

$$\text{Model 4: } \frac{r(t)}{1 - R(t)} = \nu[F(t) - R(t)] \tag{33}$$

These four models can be grouped into two families, one for the equations 30 and 31, and another for the equations 32 and 33. The former relates the rejection to the rate of new people adopting the innovation, as if people decide whether they will use or reject the innovation when they try it. Then, they do not change their position anymore. The latter assumption relates the rejection to the number of *active users*, as if the *active users* first adopt the innovation and then they continuously reject it.

The rejection parameter ν has a different interpretation in each family. In equations 30 and 31, it is the proportion of new adopters who will reject the innovation. In equations 32 and 33, it is the proportion of active users who are continuously rejecting the innovation.

Therefore, for all these models of rejection, the complete diffusion of innovation model has five parameters to be estimated, namely m, p, q, w, and ν.

27.1 MODEL 1

In this model, the rate of new *inactive users* at time t is proportional to the percentage of people adopting the innovation at time t, as if people decide whether they will use or reject the innovation when they are adopting it.

The differential equation 30 can be easily solved integrating both sides. Thus, $R(t) = \nu F(t)$, $A(t) = (1 - \nu)F(t)$, and $f(t)/[1 - F(t)] = p + (q - w\nu)F(t)$. As the imitator coefficient must be positive, we must have $w\nu < q$.

The solution shows that this model has exactly the same explanatory power as the Bass model, neither better nor worse. This happens because the model's solution has exactly the same equation after the linear transformation $q^* = q - w\nu$.

As $\lim_{t\to\infty} F(t) = 1$, thus $\lim_{t\to\infty} R(t) = \nu$. Hence, the proportion of *inactive users* is exactly equal to the rejection parameter.

Solving the differential equation for $F(t)$, it gets $F(t) = (1 - e^{-(p+q-w\nu)t})/(1 + \frac{(q-w\nu)}{p}e^{-(p+q-w\nu)t})$. Finally, as $R(t) = \nu F(t)$, thus $R(t) = \nu(1 - e^{-(p+q-w\nu)t})/(1 + \frac{(q-w\nu)}{p}e^{-(p+q-w\nu)t})$.

Unfortunately, it is not possible to estimate this model. The problem is that $\forall\nu \in \mathbb{R}^+$, $\exists\hat{\nu} \in \mathbb{R}^+$ such as the set of parameters (m, p, q, w, ν) and $(m, p, q, w\nu/\hat{\nu}, \hat{\nu})$ have exactly the same residuals when estimated. That is, the model can be estimated for any value arbitrarily set for ν. Intuitively, as both $F(t)$, $A(t)$, and $R(t)$ have the same shape, the parameters can be estimated with an empirical *active users* dataset and then you can slide up or down $F(t)$ just changing the values of ν and w.

This result is interesting because it shows that the Bass model can already explain the diffusion of innovations which follows this model of rejection. Hence, it just confirms the robustness of the Bass model.

27.2 MODEL 2

The right side of the differential equation 31 is the rate of rejection, i.e., the probability of someone rejecting the innovation, provided that he/she has not rejected it yet. Thus, in this model, the rate of rejection is proportional to the percentage of people adopting the innovation at time t, i.e., the more people adopt the innovation, the more they reject it. But if no one is adopting, there would be no rejection also, which would hold the number of *active users* the same.

The differential equation 31 can also be solved for $R(t)$ algebraically. Using the fact that $-\frac{d}{dt}\log[1 - R(t)] = r(t)/[1 - R(t)] = \nu f(t)$, and integrating both sides of this equation yields:

$$-\frac{d}{dt}\int_0^t \log[1 - R(\tau)]d\tau = \nu\int_0^t f(\tau)d\tau \tag{34}$$

$$-\log[1 - R(t)] = \nu F(t) \tag{35}$$

$$1 - R(t) = e^{-\nu F(t)} \tag{36}$$

$$R(t) = 1 - e^{-\nu F(t)} \tag{37}$$

Finally, $f(t)/[1 - F(t)] = p + qF(t) - w[1 - e^{-\nu F(t)}]$ and $\lim_{t\to\infty} R(t) = 1 - e^{-\nu}$.

As $0 \leqslant F(t) \leqslant 1 \Rightarrow 0 \leqslant \nu F(t) \leqslant \nu$, we can do a good approximation of $e^{-\nu F(t)}$ using a Taylor series around the point $\nu/2$ for small values of ν.

From the Taylor series centered around $\nu/2$, we know that $e^{-x} \approx e^{-\nu/2}(1+\nu/2-x)$. Thus, we have $1-e^{-\nu F(t)} \approx 1-e^{-\nu/2}(1+\nu/2)+e^{-\nu/2}\nu F(t)$, and, finally, $f(t)/[1-F(t)] = [p-w-we^{-\nu/2}(1+\nu/2)]+(q-we^{-\nu/2}\nu)F(t)$.

Therefore, for small values of ν, this model has approximately the same explanation power as the Bass model and we can write $f(t) = p^* + q^*F(t)$, where $p^* = p-w-we^{-\nu/2}(1+\nu/2)$ and $q^* = q-we^{-\nu/2}\nu$.

In contrast to model 1, no parameter could be arbitrarily set in this model, thus it can be estimated using an empirical *active users* dataset.

27.3 MODEL 3

In this model, the rate of new *inactive users* increases linearly with the number of *active users*, since $A(t) = F(t) - R(t)$. Thus, while there are *active users*, a fraction ν of them will be rejecting the innovation. Hence, everyone will have rejected the innovation sooner or later.

The differential equation 32 can be rewritten as the following first order linear differential equation, which has to be solved:

$$\frac{d}{dt}R(t) + \nu R(t) = \nu F(t) \tag{38}$$

The solution to this differential equation is:

$$R(t) = \nu e^{-\nu t}\int_0^t e^{\nu\tau}F(\tau)d\tau \tag{39}$$

$$= \nu\left[F(u) * e^{-\nu u}\right](t) \tag{40}$$

Or, as $\frac{d}{dt}\left(e^{\nu t}F(t)\right) = \nu e^{\nu t}F(t) + e^{\nu t}f(t)$, $R(t)$ can be rewritten as:

$$R(t) = F(t) - e^{-\nu t}\int_0^t e^{\nu\tau}f(\tau)d\tau \tag{41}$$

$$= F(t) - \left[f(u) * e^{-\nu u}\right](t) \tag{42}$$

As $F(t) = \int_0^t f(\tau)d\tau$ and $e^{\nu\tau} \geqslant 1$, we have that:

$$\int_0^t e^{\nu\tau}f(\tau)d\tau \geqslant F(t) \tag{43}$$

$$-e^{-\nu t}\int_0^t e^{\nu\tau}f(\tau)d\tau \leqslant -e^{-\nu t}F(t) \tag{44}$$

$$F(t) - e^{-\nu t}\int_0^t e^{\nu\tau}f(\tau)d\tau \leqslant F(t) - e^{-\nu t}F(t) \tag{45}$$

From equation 41:

$$R(t) \leqslant F(t) - e^{-\nu t}F(t) \tag{46}$$
$$R(t) \leqslant F(t)(1 - e^{-\nu t}) \tag{47}$$

Finally, $R(t) < F(t)$, $r(t) > 0$, and $\lim_{t\to\infty} R(t) \leqslant 1$.

I did not manage to prove that $\lim_{t\to\infty} R(t) = 1$, but this result appeared in all performed simulations. If that is true, then all people will reject the innovation at some point in time - a fact that makes sense.

Unfortunately, it seems that there is no closed formula for $F(t)$. Using equation 39, the final differential equation is:

$$\frac{f(t)}{1 - F(t)} = p + qF(t) - w\nu e^{-\nu t}\int_0^t e^{\nu\tau}F(\tau)d\tau \tag{48}$$

Or, using 41, it becomes:

$$\frac{f(t)}{1 - F(t)} = p + (q - w)F(t) + we^{-\nu t}\int_0^t e^{\nu\tau}f(\tau)d\tau \tag{49}$$

The condition $w \leqslant p + q$ is sufficient to ensure $f(t) \geqslant 0$, since $w \leqslant p + q \Rightarrow wR(t) \leqslant pR(t) + qR(t) \leqslant p + qR(t) \leqslant p + qF(t) \Rightarrow p + qF(t) - wR(t) = f(t)/[1 - F(t)] \geqslant 0 \Rightarrow f(t) \geqslant 0$. Assuming that $\lim_{t\to\infty} R(t) = 1$, then it is easy to prove that this condition is also necessary.

27.4 MODEL 4

In this model, the rate of rejection increases linearly with the number of *active users*. Thus, while there are *active users*, the rate of rejection will be greater than zero. Hence, everyone will have rejected the innovation sooner or later.

Although equation 33 is a Ricatti equation [13], none of the available techniques could solve the differential equation and it seems that there is no closed formula for $F(t)$. Hence, the equation will be analyzed through a linearization around the fixed points.

$$\begin{cases} f(F, R) = (p + qF - wR)(1 - F) \\ r(F, R) = \nu(F - R)(1 - R) \end{cases} \tag{50}$$

Solving the system $f(F, R) = r(F, R) = 0$, the following solutions are found:

$$u_1^* = (1, 1) \tag{51}$$

$$u_2^* = \left(\frac{p}{w-q}, \frac{p}{w-q}\right) \tag{52}$$

$$u_3^* = \left(\frac{w-p}{q}, 1\right) \tag{53}$$

The only valid solutions are u_1^* and u_2^*. The solution u_3^* is not valid because $p + q - w > 0 \Rightarrow (w-p)/q < 1 \Rightarrow F < R$ which is not possible because it would imply a negative number of *active users*.

Finally, the linearization around u^* is:

$$\begin{bmatrix} f(R,T) - f(u^*) \\ r(R,T) - r(u^*) \end{bmatrix} = J|_{u^*} \left(\begin{bmatrix} R \\ T \end{bmatrix} - u^* \right) \tag{54}$$

$$J = \begin{bmatrix} q(1-F) - (p + qF - wR) & -w(1-F) \\ \nu(1-R) & -\nu[(1-R) + (F-R)] \end{bmatrix} \tag{55}$$

Now, let's analyze the jacobian matrices and their eigenvalue for each valid solution.

$$J|_{u_1^*} = \begin{bmatrix} -(p+q-w) & 0 \\ 0 & 0 \end{bmatrix} \tag{56}$$

As the eigenvalues of $J|_{u_1^*}$ are 0 and $-(p+q-w) < 0$, the point $u_1^* = (1, 1)$ is a sink, i.e., the neighborhood converges to u_1^* when $t \to \infty$. It may be interpreted that all users will have rejected the innovation after a long time.

28

ESTIMATION METHOD

The parameters of the model should be estimated using empirical data in order to check the explanation power of the model. Bass [9] used a discrete version of his differential equation with ordinary least squares. While it worked well for him, it has not in the present work. There are several well-known problems in the estimation of parameters, most of them related to approximation of derivatives and instability of the estimators. These problems have been noticed by many authors [91, 97, 103].

In the present work, the parameters of the models were estimated using a maximum likelihood function on the residuals between $A(t) = F(t) - R(t)$ from the model and the empirical values from the dataset. The residuals were assumed to be normally distributed with $\mu = 0$, which leads to the same results as the ordinary least square method. In order to calculate the residuals, $F(t)$ and $R(t)$ were calculated based on their differential equations using the 4th order Runge Kutta (RK4) numerical method [21] with $\Delta t = 0.01$.

The empirical data format was (t_i, x_i), where t_i is the time and x_i is the value, and the measurements were not equally spaced over time. The following Log Likelihood equation was used:

$$\text{residual}(t_i, x_i \mid m, p, q, w, \nu) = x_i - m \cdot [F(t_i \mid p, q, w) - R(t_i \mid \nu)] \tag{57}$$

$$\text{Log Likelihood}(\vec{t}, \vec{x} \mid m, p, q, w, \nu) = -\sum_{i=1}^{N} [\text{residual}(t_i, x_i \mid m, p, q, w, \nu)]^2 \tag{58}$$

During the evaluation of $F(t)$ and $R(t)$ using the Runge Kutta numerical method, sometimes the exact value of t_i was not reached because of the chosen Δt. In these cases, the value of x_i was calculated using a linear approximation with the nearest points. Let $(\hat{t}_k, \hat{x}_k)$ and $(\hat{t}_{k+1}, \hat{x}_{k+1})$ be values calculated from the Runge Kutta method, such that $\hat{t}_k < t_i < \hat{t}_{k+1}$. Then, the calculated value at t_i was $\hat{x}_k + \left(\frac{\hat{x}_{k+1} - \hat{x}_k}{\hat{t}_{k+1} - \hat{t}_k}\right) \cdot (t_i - \hat{t}_k)$.

Since there is no explicit solution for the parameters m, p, q, w, and ν which maximize the LogLikelihood function, the parameters were estimated using the Truncated Newton Constrained (TNC) method [36, 75, 74] from the SciPy Python Library [53]. The constraints were $m > 0$, $p > 0$, $q > 0$, $w \geqslant 0$, and $\nu \geqslant 0$ for all models. The initial guess for the TNC method was the same for all models estimation.

Sometimes, the method did not converge and another initial guess had to be used.

Even though it has not been done yet, the confidence interval, estimator average and estimator variance for each parameter will be calculated using the bootstrap method.

29

PRELIMINARY RESULTS

The models' parameters were estimated using the number of Facebook's *active users* from December 2004 to March 2013 [98]. In the dataset, x_i was the number of Facebook's *active users* at time t_i. The dataset had 23 non-equally time spaced measures. The users who have accessed Facebook at least once in each month were counted in the number of *active users* in that month.

The model 1 has not been estimated because it is not possible to estimate it.

The estimated models can be seen at figures 82, 83, and 84. These figures also have forecasts for the number of *active users*, *inactive users*, and *total users* of Facebook for the next 4 years (from $t = 100$ to $t = 140$).

In spite of the favorable goodness of fit using model 2 (see figure 82), the model does not seem to provide a plausible forecast, because it would mean that Facebook is reaching a stable number of *active users* and the rejections are near the end.

Models 3 and 4 have very similar outcomes (see figures 83 and 84). Their Bayesian Information Criterion (BIC) are also close, but model 4 has a better fit with the data. Their forecast makes more sense than the forecast of model 2. It predicts that Facebook is very close to the peak of *active users* and, in approximately 3 years, it is going to decline. It is also interesting to notice that, according to these outcomes, Facebook may not reach its total potential market.

The difference between the outcomes of model 2 and models 3 and 4 could be explained by the fact that they have different rationales behind their models of rejection. While differential equation of the model 2 uses the rate of new *total users*, the differential equations of the models 2 and 3 use the proportion of *active users*.

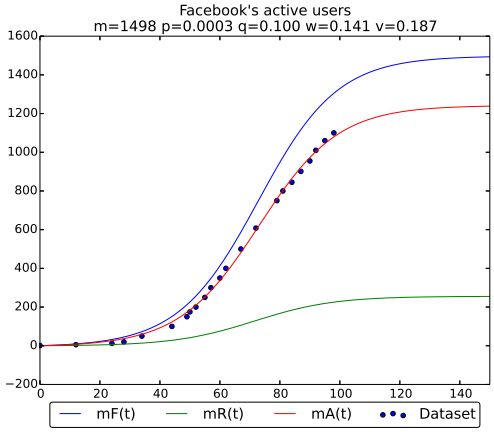


Figure 82: Fit of Model 2 with Facebook's active users dataset. $mF(t)$ is the total users, $mR(t)$ is the inactive users, and $mA(t)$ is the active users. The unit of these functions are thousands of people. The parameters are $m = 1,497.50$, $p = 0.000331$, $q = 0.100088$, $w = 0.140595$, and $\nu = 0.187188$. The goodness of fit are $R^2 = 99.84\%$ and BIC=10,566.52.

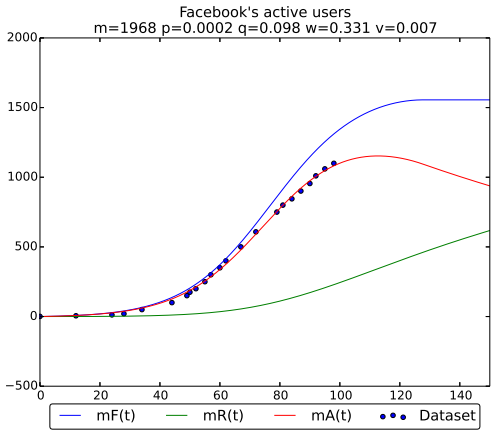


Figure 83: Fit of Model 3 with Facebook's active users dataset. $mF(t)$ is the total users, $mR(t)$ is the inactive users, and $mA(t)$ is the active users. The unit of these functions are thousands of people. The parameters are $m = 1,967.64$, $p = 0.000184$, $q = 0.097867$, $w = 0.330511$, and $\nu = 0.006912$. The goodness of fit are $R^2 = 99.83\%$ and BIC=11,485.68

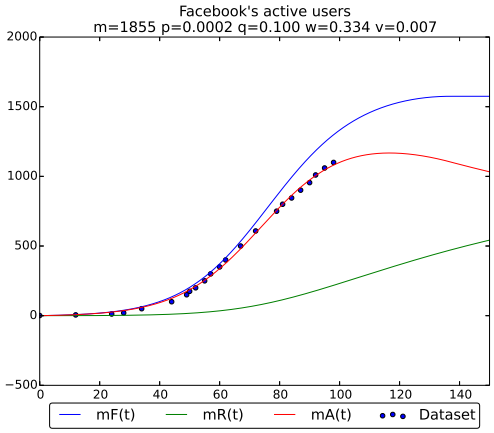


Figure 84: Fit of Model 4 with Facebook's active users dataset. $mF(t)$ is the total users, $mR(t)$ is the inactive users, and $mA(t)$ is the active users. The unit of these functions are thousands of people. The parameters are $m = 1,854.85$, $p = 0.000183$, $q = 0.099738$, $w = 0.334454$, and $v = 0.007007$. The goodness of fit are $R^2 = 99.84\%$ and BIC=10,724.55

30 CONCLUSION

The main contribution of this work is the parameter estimation through the empirical number of *active users* dataset, forecasting the number of *total users*, the number of *active users*, and the number of *inactive users*.

If the adopters who have rejected the innovation follow the equations of model 1 and 2, then the proposed extended model is transformed into the Bass model through a linear transformation of the parameters. This confirms the Bass model robustness.

Model 2 does not seem to be a good model of rejection, since the number of *active users* never decreases which does not seem to be plausible.

Models 3 and 4 had very similar results when fitting the Facebook dataset. The lack of analytical solutions for them, however, is a barrier to better understand their behavior, and to know whether they will always have similar outcomes or they will diverge depending on the data. Model 3 seems to be more analytically manageable.

It is important to notice that, in models 3 and 4, the innovation may not be adopted by the whole potential market, but it would in the Bass model. Whether it will be adopted by the whole potential market or not depends on the parameters w and ν. For instance, it seems that $mF(t)$ is not converging to m at figures 83 and 84. It is a major difference between the proposed extended model and the Bass model.

As this is a working paper, it is also intended to include the analyses of other datasets, like either Twitter's number of *active users*, or WhatsApp's, or Netflix's, or Reddit's, or Dropbox's, or Waze's, etc. This would enhance the proposed model power of forecasting.

It is also intended to run a backtest with the available data. First, the parameters of the model are estimated using a subset of the dataset. Then, through extrapolation, the number of *active users* is forecast. Finally, it is compared to this part of the dataset - which must not have been used in the estimation.

The main limitation of this work is that it has no theory to support which of the models of rejection best fit with empirical data. Although there is an extensive literature on negative word-of-mouth, this literature does not predict which model would be the best. But, if any of these companies discloses the number of *total users* and *active users*, the estimation method could be adapted to estimate the parameters using both pieces of information at the same time, which

would make possible to verify which of the models of rejection provides the best fit.

Future work could explore other models of rejection and also other estimation methods, like nonlinear least squares [97] and Kalman filter [103].

Part V

CONCLUSION

31

CONCLUSION

Software is eating the World.

— Mark Andreesen [3]

Modern management and high technology interact in multiple, profound, ways. Software — given the widespread availability of general-purpose Turing-complete hardware — seems to have an immense power of entering arenas which seemed, at some point, to require either specific hardware or the skill of humans. One of the members of this thesis committee, Dr. Nichols, will participate through teleconferencing over the open web, with no use of hardware specific for the task. The corporate biography of Tonny Martins, President of IBM Brasil, mentions his successes with blockchain, AI, and cognitive technology... as an executive, not as a research scientist or specialized engineer [16]. Professor Andrew Ng tells students at Stanford's Graduate School of Business that "AI is the new electricity" [76], as his hyperbolic way to emphasize the potential transformational power of the technology.

It is not impossible that a purely digital form of money may exist. It is not impossible that machines may become intelligent. Moreover, it is not impossible that these two processes may have already begun.

It is worthwhile, in this concluding section, to reflect on some ideas on what this thesis is and what it is that we, as computational management scientists, can obtain from this sort of study. Clemenceau once said that "war is too important to be left to the generals". I believe it is not far-fetched to state that technology has become too important to be isolated to the realm of computer science, or engineering, or applied mathematics, or any single discipline. The emergence of scientific journals with names such as Computational Management Science; INFORMS Journal on Computing; Ledger; Computational Statistics; ACM Transactions on Economics and Computation, and so forth, show that there are growing communities deeply interested in the intersection of business and the computer sciences. To whom, for instance, does the OpenAI project belong? To computing or to business? Recall that the project was created as a risk-management strategy against the far-fetched, science-fiction sounding — but not impossible —, possibility of having machines yielding too much power. What about corporations like Uber? AirBnB? Imagine a new method that

increases profits by 50% at a tech company. Should this method, if implementable as an algorithm (like PageRank [17]) or a data structure (like a blockchain) be discussed in conferences of 'computer science' or 'business'? It seems quite arbitrary to name a single group, as a whole new ecosystem seems to have emerged within those two. That is why this thesis is computational and why it is business. This work explores topics that seem, on the surface, to belong to computer science, but their applicability and impact to businesses seem too large, too central, to be delegated away, something "for those nerds in the fifth floor". As technical decisions become central to the organization of man's life, the technician becomes the visionary, the innovator, the decision-making arbiter, sometimes the billionaire.

The possibility that there will be some form of purely digital money has become very real; and we have started this study with two possible forms of organization of a purely digital money system; a blockchain and a directed acyclic graph. Consider, just as a matter of comparison, Brazil's most important company: Petrobras. As of this writing, the "market cap" of Ethereum exceeds that of Petrobras by ten billion dollars, while Bitcoin's is valued at more than double of Petrobras (195B usd vs 83B usd). Prices change, of course; but these technologies should, at a minimum, be taken seriously.

We have explored Kanerva's Sparse Distributed Memory. In AI, SDM seems to be a particularly interesting area for study. The model plausibly reflects a number of well-known aspects of psychology and neuroscience. For example, neurons can easily compute the address decoding scheme of the system. Neurons are fragile and may be lost, whereas the information remains preserved, due to the distributed character of the model. The "tip-of-the-tongue" behavior emerges naturally, and so does Miller's magic number.

There are at least three contributions[1] made on SDM: First, I have illuminated a discrepancy between Kanerva's theoretical model and the real system dynamics; Also, we have seen that pattern classification through supervised learning is possible without presuming any new SDM mechanism. This is in contrast with the literature, that presumes additional mechanisms, like genetic algorithms, to account for supervised learning. Finally, we now have a tested open-source framework that offers parallelism and can become a de-facto standard in SDM research. The framework (i) carefully reproduces crucial figures from Kanerva's theoretical book; (ii) shows how noise filtering and (iii) supervised learning can be done, and, through the use of (iv) Jupyter Notebooks, enables the reader to easily reproduce all the results on their own machines. This respects all constraints posed by Robert M. French in his article

1 As many issues have been explored in less detail, it might be advisable to leave it to history to decide whether these explorations were actually contributions.

on 'Computational Modeling in Cognitive Science: A Manifesto for Change' [1].

The ability to rapidly reproduce results, and to build on prior work, is, I believe, fundamental to modern science. Consider, for instance, the groundbreaking successes in the arena of deep learning. Having standard computer libraries to work with has brought together a community, which reinforces the system, as users also gradually improve these libraries. It may be possible to achieve new results with multiple layers of a SDM, yet, having to start development from scratch takes a large opportunity cost from most scientists — especially those who are less concentrated on the computer science aspects, but still would be able to contribute meaningfully.

Finally, we have studied how variations of the Bass Model may reflect systems or technologies that may wither in time. Though some innovations, such as the radio, have gained widespread use in a sustainable form... One may want to review the Bass model when one is concerned with rapidly-evolving technological ecosystems. Hardly anyone remembers the names AskJeeves, World Wide Web Worm, Lycos, WebCrawler, or AltaVista, early web search engines; later replaced, in the market and by the market, by the almost unnoticed url `http://google.stanford.edu` [17].

Another possibility would be to compare the proposed model with a computation of the momentum of Metcalfe's law in between competitors. As the reader may remember, Metcalfe's law states that the value of a network grows $O(n^2)$ with n being the number of network nodes. If the proposed model and Metcalfe's network effects reflect reality, then there could be an integrated mathematical model that explains and represents both Metcalfe's law and the variation of the Bass model presented herein.

With this, I submit this thesis in the hope that all readers, present and future, may find the aforementioned studies as useful, genuine, and legitimate contributions to the thriving field of Computational Management Science.

Part VI

APPENDIX

A

RECENT RESULTS IN THEORY OF COMPUTING - I

A.1 THE HALTING PROBLEM IS SOLVABLE

A fundamental question in the graduate computer science curriculum can be posed as follows: Given an average grad student doing a Ph.D, will the student ever complete his dissertation? This problem has been termed the "Halting Problem," and it has been an open problem thus far. In the following, we show that the halting problem is solvable. Furthermore, the problem can be solved within the time stipulated by the Graduate College for Ph.Ds or, in the worst case, with only a constant number of petitions for extensions.

The halting problem was first formulated by Alan Turing, who observed a number of his graduate students being apparently busy all the time but never graduating. Turing tried to solve the problem by first stopping all assistantships after the sixth year and then by purging all games from the research computers. Needless to say, his efforts were fruitless. Later, Church almost succeeded in solving the problem when he placed notices in grad students' mailboxes indicating attractive jobs in industry with several orders of magnitude higher remuneration. The so called Church's thesis was that the halting problem is solvable, given enough financial motivation. Church's idea backfired when grads found out that they have to actually work to earn money in the outside world. Thus, far from solving the halting problem, Church aggravated it (After this, we are not sure whether Church himself graduated). Recently, Cook et al have shown that the halting problem falls under a new complexity class, "NP Hairy." (NP hairy is the class of hopelessly complicated problems with no known solutions. The hardest problem in NP hairy has been shown to be the problem of trying to claim standard deductions in the 1040 form).

In the following, we show that the halting problem is indeed solvable. For this, we assume the existence of a "Super Grad," who is capable of working in any area in CS (except possibly numerical analysis). For notational convenience, we call this super grad, S sub G sup i,j sub * (written using a funky theoretical CS font). The property of Super grad is that, given the description of any grad (mostly in terms of the number of newsfiles he/she reads every day) and a description of his/her thesis topic, Super grad will either halt with a dissertation or keep publishing technical reports indefinitely. Now, we give Super grad a description of himself and his own thesis topic. If Super grad halts, we are done (and so is he) otherwise we

get a stream of technical reports. But by the "fundamental research theorem" of CS Departments (refer to the graduate study manual) any five arbitrary technical reports on unrelated topics can be compiled into a Ph.D thesis. Thus, we are done in the second case too.

Finally, how long does it take for a dissertation to be completed? The time is either less than or equal to the duration allowed by the Grad College for the completion of a Ph.D or it is greater. In the latter case, infinite number of petitions can be filed for extensions. Since the Grad College never remembers previous petitions, the total number of petitions received by the Grad College is always one, a small constant[1]. (QED)

1 'With apologies to non-cs types': from Bala Rajagopalan, posted on Internet Usenet rec.humor.funny in June 12 1989.

BIBLIOGRAPHY

[1] Caspar Addyman and Robert M. French. Computational modeling in cognitive science: A manifesto for change. *Topics in Cognitive Science*, 4(3):332–341, 2012. ISSN 1756-8765. doi: 10.1111/j.1756-8765.2012.01206.x. URL http://dx.doi.org/10.1111/j.1756-8765.2012.01206.x.

[2] Igor Aleksander, Thomas John Stonham, and BA Wilkie. Computer vision systems for industry. *Digital Systems for Industrial Automation*, 1(4):305–320, 1982.

[3] Mark Andreesen. Why software is eating the world. URL https://www.wsj.com/articles/SB10001424053111903480904576512250915629460.

[4] Ashraf Anwar and Stan Franklin. Sparse distributed memory for 'conscious' software agents. *Cognitive Systems Research*, 4(4): 339–354, 2003.

[5] Moshe Babaioff, Shahar Dobzinski, Sigal Oren, and Aviv Zohar. On bitcoin and red balloons. In *Proceedings of the 13th ACM conference on electronic commerce*, pages 56–73. ACM, 2012.

[6] Ana Babic, Francesca Sotgiu, Kristine de Valck, and Tammo HA Bijmolt. The effect of electronic word of mouth on sales: A meta-analytic review of platform, product, and metric factors. *Journal of Marketing Research*, 2015.

[7] Norman TJ Bailey. On queueing processes with bulk service. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 80–87, 1954.

[8] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better—how to make bitcoin a better currency. In *International Conference on Financial Cryptography and Data Security*, pages 399–414. Springer, 2012.

[9] Frank M Bass. A new product growth for model consumer durables. *Marketing science*, 15(5):215–227, 1969.

[10] Frank M Bass. Comments on "a new product growth for model consumer durables the bass model". *Management science*, 50 (12_supplement):1833–1840, 2004.

[11] Frank M Bass, Trichy V Krishnan, and Dipak C Jain. Why the bass model fits without decision variables. *Marketing science*, 13 (3):203–223, 1994.

[12] BitcoinStats. Data propagation, 2013-2017. http://bitcoinstats.com/network/propagation/.

[13] Sergio Bittanti, Alan J Laub, and Jan C Willems. *The Riccati Equation*. Springer-Verlag New York, Inc., 1991.

[14] Blockchain.info. Bitcoin blockchain size. https://blockchain.info/charts/blocks-size. Last accessed on July 14, 2017.

[15] Paula Fitzgerald Bone. Word-of-mouth effects on short-term and long-term product judgments. *Journal of business research*, 32(3):213–223, 1995.

[16] IBM Brasil. Antonio martins: Presidente da ibm brasil. URL https://www-03.ibm.com/press/br/pt/biography/53561.wss.

[17] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998. URL http://infolab.stanford.edu/~backrub/google.html.

[18] M. S. Brogliato. Understanding the critical distance in sparse distributed memory. Master's thesis, Escola Brasileira de Administração Pública e de Empresas - EBAPE, Fundação Getulio Vargas, 2011.

[19] Marcelo Brogliato. Sdm framework documentation. URL http://sdm-framework.readthedocs.io/.

[20] Marcelo S Brogliato, Daniel M Chada, and Alexandre Linhares. Sparse distributed memory: understanding the speed and robustness of expert memory. *Frontiers in Human Neuroscience*, 8: 222, 2014.

[21] John Charles Butcher. *The numerical analysis of ordinary differential equations: Runge-Kutta and general linear methods*. Wiley-Interscience, 1987.

[22] Francis A Buttle. Word of mouth: understanding and managing referral marketing. *Journal of strategic marketing*, 6(3):241–254, 1998.

[23] John Cannarella and Joshua A Spechler. Epidemiological modeling of online social network dynamics. *arXiv preprint arXiv:1401.4208*, 2014.

[24] Daniel de Magalhães Chada. *Are you experienced? Contributions towards experience recognition, cognition, and decision making*. PhD thesis, 2016.

[25] Timothy M Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the ram, revisited. In *Proceedings of the twenty-seventh annual symposium on Computational geometry*, pages 1–10. ACM, 2011.

[26] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17(3):427–462, 1988.

[27] Pei-Yu Chen, Shin-yi Wu, and Jungsun Yoon. The impact of online recommendations and consumer feedback on sales. *ICIS 2004 Proceedings*, page 58, 2004.

[28] CoinMarketCap. Bitcoin market capitalizations. http://coinmarketcap.com/currencies/bitcoin/. Last accessed on July 14, 2017.

[29] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

[30] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[31] N. Cowan. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, 24:87–185, 2000.

[32] Antônio de Pádua Braga and Igor Aleksander. Geometrical treatment and statistical modelling of the distribution of patterns in the n-dimensional boolean space. *Pattern Recognition Letters*, 16(5):507–515, 1995.

[33] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–10. IEEE, 2013.

[34] Chrysanthos Dellarocas. The digitization of word of mouth: Promise and challenges of online feedback mechanisms. *Management science*, 49(10):1407–1424, 2003.

[35] Chrysanthos Dellarocas, Xiaoquan Michael Zhang, and Neveen F Awad. Exploring the value of online product reviews in forecasting sales: The case of motion pictures. *Journal of Interactive marketing*, 21(4):23–45, 2007.

[36] Ron S Dembo and Trond Steihaug. Truncated-newtono algorithms for large-scale unconstrained optimization. *Mathematical Programming*, 26(2):190–212, 1983.

[37] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[38] Discussion. Dag, a generalized blockchain, 2014. https://nxtforum.org/proof-of-stake-algorithm/dag-a-generalized-blockchain/ (registration at nxtforum.org required).

[39] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. Ripemd-160: A strengthened version of ripemd. In *Fast Software Encryption*, pages 71–82. Springer, 1996.

[40] Wenjing Duan, Bin Gu, and Andrew B Whinston. Do online reviews matter?—an empirical investigation of panel data. *Decision Support Systems*, 45(4):1007–1016, 2008.

[41] Kuo-Chin Fan and Yuan-Kai Wang. A genetic sparse distributed memory approach to the application of handwritten character recognition. *Pattern Recognition*, 30(12):2015–2022, 1997.

[42] R. M. French. When coffee cups are like old elephants, or why representation modules dont make sense. In A. Riegler and M. Peschl, editors, *Proceedings of the 1997 International Conference on New Trends in Cognitive Science*, pages 158–163. Austrian Society for Cognitive Science, 1997.

[43] Robert M French. Subcognition and the limits of the turing test. *Mind*, 99(393):53–65, 1990.

[44] Robert M French. The turing test: the first 50 years. *Trends in cognitive sciences*, 4(3):115–122, 2000.

[45] Robert M French and Christophe Labiouse. Why co-occurrence information alone is not sufficient to answer subcognitive questions. *Journal of Experimental & Theoretical Artificial Intelligence*, 13(4):421–429, 2001.

[46] John Gallaugher and Sam Ransbotham. Social media and customer dialog management at starbucks. *MIS Quarterly Executive*, 9(4):197–212, 2010.

[47] Henri Gilbert and Helena Handschuh. Security analysis of sha-256 and sisters. In *International workshop on selected areas in cryptography*, pages 175–193. Springer, 2003.

[48] David Godes and Dina Mayzlin. Using online conversations to study word-of-mouth communication. *Marketing science*, 23(4): 545–560, 2004.

[49] Frank Harary, John P Hayes, and Horng-Jyh Wu. A survey of the theory of hypercube graphs. *Computers & Mathematics with Applications*, 15(4):277–289, 1988.

[50] Tim A Hely, David J Willshaw, and Gillian M Hayes. A new approach to kanerva's sparse distributed memory. *IEEE transactions on Neural Networks*, 8(3):791–794, 1997.

[51] Douglas R Hofstadter. On seeing a's and seeing as. *Stanford Humanities Review*, 4(2):109–121, 1995. URL https://web.stanford.edu/group/SHR/4-2/text/hofstadter.html.

[52] Bernard J Jansen, Mimi Zhang, Kate Sobel, and Abdur Chowdury. Twitter power: Tweets as electronic word of mouth. *Journal of the American society for information science and technology*, 60(11):2169–2188, 2009.

[53] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL http://www.scipy.org/. [Online; accessed 2014-12-16].

[54] P. Kanerva. *Sparse Distributed Memory*. MIT Press, 1988.

[55] P. Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, 1:139–159, 2009.

[56] Ghassan Karame, Elli Androulaki, and Srdjan Capkun. Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin. *IACR Cryptology ePrint Archive*, 2012(248), 2012.

[57] Roland Karlsson. A fast activation mechanism for the kanerva sdm memory. In *SICS Research Report R95:10, Swedish Institute of Computer Science*, pages 69–70, 1995. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.44.5112&rep=rep1&type=pdf.

[58] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. Jupyter notebooks-a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90, 2016.

[59] Yann LeCun, Corinna Cortes, and Christopher J. C. Burges. The mnist database. URL http://yann.lecun.com/exdb/mnist/.

[60] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[61] Mira Lee and Seounmi Youn. Electronic word of mouth (ewom) how ewom platforms influence consumer product judgement. *International Journal of Advertising*, 28(3):473–499, 2009.

[62] Sergio Demian Lerner. Dagcoin: a cryptocurrency without blocks, 2015. Available at https://bitslog.wordpress.com/2015/09/11/dagcoin/.

[63] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols, 2015. Available at http://www.cs.huji.ac.il/~avivz/pubs/15/inclusivebtc.pdf.

[64] Alexandre Linhares. Only one life. URL https://www.slideshare.net/linhares/you-only-get-one-life.

[65] Alexandre Linhares. A glimpse at the metaphysics of bongard problems. *Artificial Intelligence*, 121(1-2):251–270, 2000.

[66] Alexandre Linhares, Daniel M. Chada, and Christian N. Aranha. The emergence of miller's magic number on a sparse distributed memory. *PLOS One*, 6(1):e15592, Jan 2011. doi: 10.1371/journal.pone.0015592.

[67] Vijay Mahajan, Eitan Muller, and Roger A Kerin. Introduction strategy for new products with positive and negative word-of-mouth. *Management Science*, 30(12):1389–1404, 1984.

[68] Nigel Meade and Towhidul Islam. Modelling and forecasting the diffusion of innovation–a 25-year review. *International Journal of forecasting*, 22(3):519–545, 2006.

[69] Mateus Mendes, Manuel Crisóstomo, and A Paulo Coimbra. Robot navigation using a sparse distributed memory. In *Robotics and automation, 2008. ICRA 2008. IEEE international conference on*, pages 53–58. IEEE, 2008.

[70] Meng et al. A modified sparse distributed memory model for extracting clean patterns from noisy inputs. *Proceedings of International Joint Conference on Neural Networks*, June 2009.

[71] G. A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63:81–97, 1955.

[72] Aaftab Munshi. The opencl specification. In *Hot Chips 21 Symposium (HCS), 2009 IEEE*, pages 1–314. IEEE, 2009.

[73] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. Available at https://bitcoin.org/bitcoin.pdf.

[74] Stephen G Nash. Newton-type minimization via the lanczos method. *SIAM Journal on Numerical Analysis*, 21(4):770–788, 1984.

[75] Stephen G Nash. A survey of truncated-newton methods. *Journal of Computational and Applied Mathematics*, 124(1):45–59, 2000.

[76] Andrew Ng. Artificial intelligence is the new electricity, 2017. URL https://www.youtube.com/watch?v=21EiKfQYZXc.

[77] Kenneth A Norman and Randall C O'reilly. Modeling hippocampal and neocortical contributions to recognition memory: a complementary-learning-systems approach. *Psychological review*, 110(4):611, 2003.

[78] Mohammad Norouzi, Ali Punjani, and David J Fleet. Fast exact search in hamming space with multi-index hashing. *IEEE transactions on pattern analysis and machine intelligence*, 36(6):1107–1119, 2014.

[79] John A Norton and Frank M Bass. A diffusion theory model of adoption and substitution for successive generations of high-technology products. *Management science*, 33(9):1069–1086, 1987.

[80] A Pinar Ozisik, George Bissias, and Brian N Levine. Estimation of miner hash rates and consensus on blockchains. Technical report, Tech. rep., PDF available from arxiv. org and https://www. cs. umass. edu/ brian/status-reports. pdf (June 2017).

[81] Ram Pai, Badari Pulavarty, and Mingming Cao. Linux 2.6 performance improvement through readahead optimization. In *Proceedings of the Linux Symposium*, volume 2, pages 105–116, 2004.

[82] Renana Peres, Eitan Muller, and Vijay Mahajan. Innovation diffusion and new product growth models: A critical review and research directions. *International Journal of Research in Marketing*, 27(2):91–106, 2010.

[83] Jürgen Pfeffer, T Zorbach, and KM Carley. Understanding online firestorms: Negative word-of-mouth dynamics in social media networks. *Journal of Marketing Communications*, 20(1-2):117–128, 2014.

[84] Serguei Popov and Jinn Labs. The tangle. 2016. Available at https://iota.org/IOTA_Whitepaper.pdf.

[85] United Nations World Food Programme. The year in review report. 2016.

[86] United Nations World Food Programme. Blockchain against hunger: Harnessing technology in support of syrian refugees. 2017. URL https://www.wfp.org/news/news-release/.

[87] Rajesh Rao and Olac Fuentes. Hierarchical learning of navigational behaviors in an autonomous robot using a predictive

sparse distributed memory. *Machine Learning*, pages 87–113, 1998.

[88] Rajesh PN Rao and Dana H Ballard. Natural basis functions and topographic memory for face recognition. In *IJCAI*, pages 10–19, 1995.

[89] Marsha L Richins. Negative word-of-mouth by dissatisfied consumers: A pilot study. *The journal of marketing*, pages 68–78, 1983.

[90] Everett M Rogers. *Diffusion of innovations*. The Free Press, New York, 1st edition, 1962.

[91] David C Schmittlein and Vijay Mahajan. Maximum likelihood estimation for an innovation diffusion model of new product acceptance. *Marketing science*, 1(1):57–78, 1982.

[92] Devavrat Shah et al. Gossip algorithms. *Foundations and Trends® in Networking*, 3(1):1–125, 2009.

[93] Helen Shen. Interactive notebooks: Sharing the code. *Nature News*, 515(7525):151, 2014.

[94] Robert E Smith and Christine A Vogt. The effects of integrating advertising and negative word-of-mouth communications on message processing and response. *Journal of Consumer Psychology*, 4(2):133–151, 1995.

[95] Javier Snaider and Stan Franklin. Extended sparse distributed memory. Paper presented at the Biological Inspired Cognitive Architectures 2011, Washington D.C. USA.

[96] Yonatan Sompolinsky and Aviv Zohar. Accelerating bitcoin's transaction processing. fast money grows on trees, not chains, 2013. Available at https://eprint.iacr.org/2013/881.pdf.

[97] V Srinivasan and Charlotte H Mason. Technical note-nonlinear least squares estimation of new product diffusion models. *Marketing science*, 5(2):169–178, 1986.

[98] The Associated Press. Number of active users at facebook over the years, May 2013. URL http://news.yahoo.com/number-active-users-facebook-over-230449748.html.

[99] Richard Van Noorden, Brendan Maher, and Regina Nuzzo. The top 100 papers. *Nature*, 514(7524):550, 2014.

[100] David Vorick. Getting rid of blocks, 2015. Available at slides.com/davidvorick/braids.

[101] Henry S Warren. *Hacker's delight*. Pearson Education, 2013.

[102] Elizabeth Woyke. How blockchain can bring financial services to the poor. *MIT Technology Review*, 2017.

[103] Jinhong Xie, X Michael Song, Marvin Sirbu, and Qiong Wang. Kalman filter estimation of new product diffusion models. *Journal of Marketing Research*, pages 378–393, 1997.