



# Contents

<b>1</b>	<b>Fountain code</b>	<b>1</b>
1.1	Applications . . . . .	1
1.2	Fountain codes in standards . . . . .	1
1.3	Fountain codes for data storage . . . . .	2
1.4	See also . . . . .	2
1.5	Notes . . . . .	2
1.6	References . . . . .	2
<b>2</b>	<b>Forward error correction</b>	<b>3</b>
2.1	How it works . . . . .	3
2.2	Averaging noise to reduce errors . . . . .	3
2.3	Types of FEC . . . . .	4
2.4	Concatenated FEC codes for improved performance . . . . .	4
2.5	Low-density parity-check (LDPC) . . . . .	5
2.6	Turbo codes . . . . .	5
2.7	Local decoding and testing of codes . . . . .	5
2.8	Interleaving . . . . .	5
2.8.1	Example . . . . .	6
2.8.2	Disadvantages of interleaving . . . . .	6
2.9	List of error-correcting codes . . . . .	6
2.10	See also . . . . .	7
2.11	References . . . . .	7
2.12	Further reading . . . . .	7
2.13	External links . . . . .	8
<b>3</b>	<b>List of algebraic coding theory topics</b>	<b>9</b>
<b>4</b>	<b>Coding theory</b>	<b>10</b>
4.1	History of coding theory . . . . .	10
4.2	Source coding . . . . .	10
4.2.1	Definition . . . . .	11
4.2.2	Properties . . . . .	11
4.2.3	Principle . . . . .	11

4.2.4	Example . . . . .	11
4.3	Channel coding . . . . .	11
4.3.1	Linear codes . . . . .	11
4.4	Cryptographical coding . . . . .	13
4.5	Line coding . . . . .	13
4.6	Other applications of coding theory . . . . .	13
4.6.1	Group testing . . . . .	13
4.6.2	Analog coding . . . . .	14
4.7	Neural coding . . . . .	14
4.8	See also . . . . .	14
4.9	Notes . . . . .	14
4.10	References . . . . .	15
<b>5</b>	<b>Alternant code</b>	<b>16</b>
5.1	Definition . . . . .	16
5.2	Properties . . . . .	16
5.3	References . . . . .	16
<b>6</b>	<b>Arbitrarily varying channel</b>	<b>17</b>
6.1	Capacities and associated proofs . . . . .	17
6.1.1	Capacity of deterministic AVCs . . . . .	17
6.1.2	Capacity of AVCs with input and state constraints . . . . .	18
6.1.3	Capacity of randomized AVCs . . . . .	18
6.2	See also . . . . .	19
6.3	References . . . . .	19
<b>7</b>	<b>Bar product</b>	<b>20</b>
7.1	Properties . . . . .	20
7.1.1	Rank . . . . .	20
7.1.2	Hamming weight . . . . .	20
7.2	See also . . . . .	20
7.3	References . . . . .	21
<b>8</b>	<b>BCH code</b>	<b>22</b>
8.1	Definition and illustration . . . . .	22
8.1.1	Primitive narrow-sense BCH codes . . . . .	22
8.1.2	General BCH codes . . . . .	23
8.1.3	Special cases . . . . .	23
8.2	Properties . . . . .	23
8.3	Encoding . . . . .	23
8.4	Decoding . . . . .	23
8.4.1	Calculate the syndromes . . . . .	23
8.4.2	Calculate the error location polynomial . . . . .	23

8.4.3	Factor error locator polynomial . . . . .	24
8.4.4	Calculate error values . . . . .	24
8.4.5	Decoding based on extended Euclidean algorithm . . . . .	26
8.4.6	Correct the errors . . . . .	27
8.4.7	Decoding examples . . . . .	27
8.5	Citations . . . . .	29
8.6	References . . . . .	29
8.6.1	Primary sources . . . . .	29
8.6.2	Secondary sources . . . . .	29
8.7	Further reading . . . . .	29
<b>9</b>	<b>Belief propagation</b>	<b>30</b>
9.1	Description of the sum-product algorithm . . . . .	30
9.2	Exact algorithm for trees . . . . .	31
9.3	Approximate algorithm for general graphs . . . . .	31
9.4	Related algorithm and complexity issues . . . . .	31
9.5	Relation to free energy . . . . .	32
9.6	Generalized belief propagation (GBP) . . . . .	32
9.7	Gaussian belief propagation (GaBP) . . . . .	32
9.8	References . . . . .	33
9.9	Further reading . . . . .	34
<b>10</b>	<b>Berger code</b>	<b>35</b>
10.1	Unidirectional error detection . . . . .	35
10.2	References . . . . .	35
<b>11</b>	<b>Berlekamp–Welch algorithm</b>	<b>36</b>
11.1	History on decoding Reed–Solomon codes . . . . .	36
11.2	Error locator polynomial of Reed–Solomon codes . . . . .	36
11.3	The Berlekamp–Welch decoder and algorithm . . . . .	37
11.4	Example . . . . .	38
11.5	See also . . . . .	38
11.6	References . . . . .	39
11.7	External links . . . . .	39
<b>12</b>	<b>Binary erasure channel</b>	<b>40</b>
12.1	Description . . . . .	40
12.2	Definition . . . . .	40
12.2.1	Capacity of the BEC . . . . .	40
12.3	Deletion channel . . . . .	40
12.4	See also . . . . .	41
12.5	References . . . . .	41

<b>13 Binary Goppa code</b>	<b>42</b>
13.1 Construction and properties . . . . .	42
13.2 Decoding . . . . .	42
13.3 Properties and usage . . . . .	43
13.4 References . . . . .	43
13.5 See also . . . . .	43
<b>14 Binary symmetric channel</b>	<b>44</b>
14.1 Description . . . . .	44
14.2 Definition . . . . .	44
14.2.1 Capacity of $BSC_p$ . . . . .	44
14.3 Shannon's channel capacity theorem for $BSC_p$ . . . . .	44
14.3.1 Noisy coding theorem for $BSC_p$ . . . . .	44
14.4 Converse of Shannon's capacity theorem . . . . .	46
14.5 Codes for $BSC_p$ . . . . .	46
14.6 Forney's code for $BSC_p$ . . . . .	46
14.6.1 Decoding error probability for $C^*$ . . . . .	47
14.7 See also . . . . .	47
14.8 Notes . . . . .	47
14.9 References . . . . .	47
14.10 External links . . . . .	47
<b>15 Blackwell channel</b>	<b>48</b>
15.1 Definition . . . . .	48
15.1.1 Capacity of the Blackwell channel . . . . .	48
15.2 References . . . . .	48
<b>16 Blahut–Arimoto algorithm</b>	<b>49</b>
16.1 Algorithm . . . . .	49
16.2 References . . . . .	49
<b>17 Block code</b>	<b>50</b>
17.1 The block code and its parameters . . . . .	50
17.1.1 The alphabet $\Sigma$ . . . . .	50
17.1.2 The message length $k$ . . . . .	50
17.1.3 The block length $n$ . . . . .	50
17.1.4 The rate $R$ . . . . .	51
17.1.5 The distance $d$ . . . . .	51
17.1.6 Popular notation . . . . .	51
17.2 Examples . . . . .	51
17.3 Error detection and correction properties . . . . .	51
17.4 Lower and upper bounds of block codes . . . . .	52
17.4.1 Family of codes . . . . .	52

17.4.2	Hamming bound . . . . .	52
17.4.3	Singleton bound . . . . .	52
17.4.4	Plotkin bound . . . . .	52
17.4.5	Gilbert–Varshamov bound . . . . .	53
17.4.6	Johnson bound . . . . .	53
17.4.7	Elias–Bassalygo bound . . . . .	53
17.5	Sphere packings and lattices . . . . .	53
17.6	See also . . . . .	53
17.7	References . . . . .	53
17.8	External links . . . . .	53
<b>18</b>	<b>Burst error-correcting code</b>	<b>54</b>
18.1	Definitions . . . . .	54
18.1.1	Burst description . . . . .	54
18.2	Cyclic codes for burst error correction . . . . .	55
18.3	Burst error correction bounds . . . . .	56
18.3.1	Upper bounds on burst error detection and correction . . . . .	56
18.3.2	Further bounds on burst error correction . . . . .	57
18.4	Fire codes <sup>[3][4][5]</sup> . . . . .	57
18.4.1	Proof of Theorem . . . . .	58
18.4.2	Example: 5-burst error correcting fire code . . . . .	58
18.5	Binary Reed–Solomon codes . . . . .	59
18.5.1	An example of a binary RS code . . . . .	59
18.6	Interleaved codes . . . . .	59
18.6.1	Burst error correcting capacity of interleaver . . . . .	59
18.6.2	Block interleaver . . . . .	59
18.6.3	Convolutional interleaver . . . . .	60
18.7	Applications . . . . .	60
18.7.1	Compact disc . . . . .	60
18.8	See also . . . . .	61
18.9	References . . . . .	62
<b>19</b>	<b>Canonical Huffman code</b>	<b>63</b>
19.1	Algorithm . . . . .	63
19.1.1	As a fractional binary number . . . . .	63
19.2	Encoding the codebook . . . . .	64
19.3	Pseudo code . . . . .	64
19.4	Algorithm . . . . .	64
<b>20</b>	<b>Coding gain</b>	<b>65</b>
20.1	Example . . . . .	65
20.2	Power-limited regime . . . . .	65

20.3 Example . . . . .	65
20.4 Bandwidth-limited regime . . . . .	65
20.5 See also . . . . .	66
20.6 References . . . . .	66
<b>21 Comma code</b>	<b>67</b>
21.1 Examples . . . . .	67
21.2 See also . . . . .	67
21.3 References . . . . .	67
<b>22 Comma-free code</b>	<b>68</b>
22.1 References . . . . .	68
22.2 External links . . . . .	68
<b>23 Computationally bounded adversary</b>	<b>69</b>
23.1 Comparison to other models . . . . .	69
23.1.1 Worst-case model . . . . .	69
23.1.2 Stochastic noise model . . . . .	69
23.2 Applications . . . . .	69
23.2.1 Comparison to stochastic noise channel . . . . .	69
23.3 Specific applications . . . . .	70
23.4 References . . . . .	70
<b>24 Concatenated error correction code</b>	<b>71</b>
24.1 Background . . . . .	71
24.2 Description . . . . .	71
24.3 Properties . . . . .	72
24.4 Decoding concatenated codes . . . . .	72
24.4.1 Remarks . . . . .	72
24.5 Applications . . . . .	73
24.6 Turbo codes: A parallel concatenation approach . . . . .	73
24.7 See also . . . . .	73
24.8 References . . . . .	73
24.9 Further reading . . . . .	74
24.10 External links . . . . .	74
<b>25 Coset leader</b>	<b>75</b>
25.1 References . . . . .	75
<b>26 Covering code</b>	<b>76</b>
26.1 Definition . . . . .	76
26.2 Example . . . . .	76
26.3 Covering problem . . . . .	76
26.4 Football pools problem . . . . .	76

26.5 Applications . . . . .	76
26.6 References . . . . .	76
26.7 External links . . . . .	77
<b>27 Cyclic code</b>	<b>78</b>
27.1 Definition . . . . .	78
27.2 Algebraic structure . . . . .	78
27.3 Examples . . . . .	78
27.3.1 Trivial examples . . . . .	79
27.4 Quasi-cyclic codes and shortened codes . . . . .	79
27.4.1 Definition . . . . .	79
27.4.2 Definition . . . . .	79
27.5 Cyclic codes for correcting errors . . . . .	79
27.5.1 For correcting two errors . . . . .	79
27.6 Hamming code . . . . .	80
27.6.1 Hamming code for correcting single errors . . . . .	80
27.7 Cyclic codes for correcting burst errors . . . . .	80
27.7.1 Fire codes as cyclic bounds . . . . .	80
27.8 Cyclic codes on Fourier transform . . . . .	81
27.8.1 Fourier transform over finite fields . . . . .	81
27.8.2 Spectral description of cyclic codes . . . . .	81
27.8.3 Quadratic residue codes . . . . .	82
27.9 Generalizations . . . . .	82
27.10 See also . . . . .	82
27.11 Notes . . . . .	82
27.12 References . . . . .	82
27.13 Further reading . . . . .	82
27.14 External links . . . . .	82
<b>28 Decoding methods</b>	<b>84</b>
28.1 Notation . . . . .	84
28.2 Ideal observer decoding . . . . .	84
28.2.1 Decoding conventions . . . . .	84
28.3 Maximum likelihood decoding . . . . .	84
28.4 Minimum distance decoding . . . . .	85
28.5 Syndrome decoding . . . . .	85
28.6 Partial response maximum likelihood . . . . .	85
28.7 Viterbi decoder . . . . .	86
28.8 See also . . . . .	86
28.9 Sources . . . . .	86
28.10 References . . . . .	86



<b>29 Deletion channel</b>	<b>87</b>
29.1 Formal description . . . . .	87
29.2 Capacity . . . . .	87
29.3 External links . . . . .	87
29.4 References . . . . .	87
<b>30 Distributed source coding</b>	<b>88</b>
30.1 History . . . . .	88
30.2 Theoretical bounds . . . . .	89
30.2.1 Slepian–Wolf bound . . . . .	89
30.2.2 Wyner–Ziv bound . . . . .	89
30.3 Virtual channel . . . . .	89
30.4 Asymmetric DSC vs. symmetric DSC . . . . .	89
30.5 Practical distributed source coding . . . . .	89
30.5.1 Slepian–Wolf coding – lossless distributed coding . . . . .	90
30.5.2 Wyner–Ziv coding – lossy distributed coding . . . . .	91
30.5.3 Large scale distributed quantization . . . . .	91
30.6 Non-asymmetric DSC . . . . .	92
30.7 Non-asymmetric DSC for more than two sources . . . . .	92
30.8 See also . . . . .	92
30.9 References . . . . .	92
<b>31 Dual code</b>	<b>94</b>
31.1 Self-dual codes . . . . .	94
31.2 References . . . . .	94
31.3 External links . . . . .	94
<b>32 Elias Bassalygo bound</b>	<b>95</b>
32.1 Definition . . . . .	95
32.2 Proof . . . . .	95
32.3 See also . . . . .	96
32.4 References . . . . .	96
<b>33 Enumerator polynomial</b>	<b>97</b>
33.1 Basic properties . . . . .	97
33.2 MacWilliams identity . . . . .	97
33.3 Distance enumerator . . . . .	97
33.4 References . . . . .	97
<b>34 Erasure code</b>	<b>99</b>
34.1 Optimal erasure codes . . . . .	99
34.1.1 Parity check . . . . .	99
34.1.2 Polynomial oversampling . . . . .	99

34.2	Near-optimal erasure codes . . . . .	100
34.3	Examples . . . . .	100
34.3.1	Near optimal erasure codes . . . . .	100
34.3.2	Near optimal fountain (rateless erasure) codes . . . . .	100
34.3.3	Optimal erasure codes . . . . .	101
34.3.4	Other . . . . .	101
34.4	See also . . . . .	101
34.5	References . . . . .	101
34.6	External links . . . . .	101
<b>35</b>	<b>Even code</b>	<b>102</b>
<b>36</b>	<b>Expander code</b>	<b>103</b>
36.1	Expander codes . . . . .	103
36.2	Definition . . . . .	103
36.3	Rate . . . . .	103
36.4	Distance . . . . .	103
36.4.1	Proof . . . . .	103
36.5	Encoding . . . . .	104
36.6	Decoding . . . . .	104
36.6.1	Proof . . . . .	104
36.7	See also . . . . .	105
36.8	Notes . . . . .	105
36.9	References . . . . .	105
<b>37</b>	<b>Factorization of polynomials over finite fields</b>	<b>106</b>
37.1	Background . . . . .	106
37.1.1	Finite field . . . . .	106
37.1.2	Irreducible polynomials . . . . .	106
37.1.3	Complexity . . . . .	107
37.2	Factoring algorithms . . . . .	107
37.2.1	Berlekamp's algorithm . . . . .	107
37.2.2	Square-free factorization . . . . .	107
37.2.3	Distinct-degree factorization . . . . .	108
37.2.4	Equal-degree factorization . . . . .	109
37.3	Time complexity . . . . .	110
37.4	Rabin's test of irreducibility . . . . .	110
37.5	See also . . . . .	110
37.6	References . . . . .	110
37.7	External links . . . . .	111
37.8	Notes . . . . .	111
<b>38</b>	<b>Second Johnson bound</b>	<b>112</b>

38.1 Definition . . . . .	112
38.2 See also . . . . .	112
38.3 References . . . . .	112
<b>39 Folded Reed–Solomon code</b>	<b>114</b>
39.1 History . . . . .	114
39.2 Definition . . . . .	114
39.2.1 Graphic description . . . . .	114
39.3 Folded Reed–Solomon codes and the singleton bound . . . . .	115
39.3.1 Why folding might help? . . . . .	115
39.4 How folded Reed–Solomon (FRS) codes and Parvaresh Vardy (PV) codes are related . . . . .	115
39.5 Brief overview of list-decoding folded Reed–Solomon codes . . . . .	116
39.6 Linear-algebraic list decoding algorithm . . . . .	116
39.6.1 Step 1: The interpolation step . . . . .	116
39.6.2 Step 2: The root-finding step . . . . .	117
39.6.3 Step 3: The prune step . . . . .	117
39.6.4 Summary . . . . .	118
39.7 See also . . . . .	118
39.8 References . . . . .	118
<b>40 Forney algorithm</b>	<b>119</b>
40.1 Procedure . . . . .	119
40.2 Formal derivative . . . . .	119
40.3 Derivation . . . . .	119
40.4 Erasures . . . . .	120
40.5 See also . . . . .	120
40.6 References . . . . .	120
40.7 External links . . . . .	120
<b>41 Fuzzy extractor</b>	<b>121</b>
41.1 Motivation . . . . .	121
41.2 Basic definitions . . . . .	121
41.2.1 Predictability . . . . .	121
41.2.2 Min-entropy . . . . .	121
41.2.3 Statistical distance . . . . .	121
41.2.4 Definition 1 (strong extractor) . . . . .	122
41.2.5 Secure sketch . . . . .	122
41.2.6 Definition 2 (secure sketch) . . . . .	122
41.2.7 Fuzzy extractor . . . . .	122
41.2.8 Definition 3 (fuzzy extractor) . . . . .	122
41.2.9 Secure sketches and fuzzy extractors . . . . .	122
41.2.10 Lemma 1 (fuzzy extractors from sketches) . . . . .	122

41.2.11 Corollary 1 . . . . .	122
41.3 Basic constructions . . . . .	122
41.3.1 Hamming distance constructions . . . . .	123
41.3.2 Set difference constructions . . . . .	123
41.3.3 Edit distance constructions . . . . .	123
41.3.4 Other distance measure constructions . . . . .	123
41.4 Improving error-tolerance via relaxed notions of correctness . . . . .	124
41.4.1 Random errors . . . . .	124
41.4.2 Input-dependent errors . . . . .	124
41.4.3 Computationally bounded errors . . . . .	124
41.5 Privacy guarantees . . . . .	124
41.5.1 Correlation between helper string and biometric input . . . . .	124
41.5.2 $\text{Gen}(W)$ as a probabilistic map . . . . .	124
41.5.3 Uniform fuzzy extractors . . . . .	125
41.5.4 Uniform secure sketches . . . . .	125
41.5.5 Applications . . . . .	125
41.6 Protection against active attacks . . . . .	125
41.6.1 Robust fuzzy extractors . . . . .	125
41.7 References . . . . .	126
<b>42 Generalized minimum-distance decoding</b>	<b>127</b>
42.1 Setup . . . . .	127
42.2 Randomized algorithm . . . . .	127
42.3 Modified randomized algorithm . . . . .	128
42.4 Deterministic algorithm . . . . .	129
42.5 See also . . . . .	129
42.6 References . . . . .	129
<b>43 Generator matrix</b>	<b>130</b>
43.1 Terminology . . . . .	130
43.2 Equivalent Codes . . . . .	130
43.3 See also . . . . .	130
43.4 Notes . . . . .	131
43.5 References . . . . .	131
43.6 Further reading . . . . .	131
43.7 External links . . . . .	131
<b>44 Gibbs' inequality</b>	<b>132</b>
44.1 Gibbs' inequality . . . . .	132
44.2 Proof . . . . .	132
44.3 Alternative proofs . . . . .	133
44.4 Corollary . . . . .	133

44.5 See also . . . . .	133
44.6 References . . . . .	133
<b>45 Gilbert–Varshamov bound</b>	<b>134</b>
45.1 Statement of the bound . . . . .	134
45.2 Proof . . . . .	134
45.3 An improvement in the prime power case . . . . .	134
45.4 See also . . . . .	134
45.5 References . . . . .	135
<b>46 Goppa code</b>	<b>136</b>
46.1 Construction . . . . .	136
46.2 Function code . . . . .	136
46.3 Residue code . . . . .	137
46.4 References . . . . .	137
46.5 External links . . . . .	137
<b>47 Grammar-based code</b>	<b>138</b>
47.1 Examples and characteristics . . . . .	138
47.2 Practical algorithms . . . . .	138
47.3 See also . . . . .	138
47.4 References . . . . .	138
47.5 External links . . . . .	138
<b>48 Gray isometry</b>	<b>139</b>
48.1 Name . . . . .	139
48.2 Motivation . . . . .	139
48.3 History and practical application . . . . .	140
48.3.1 Position encoders . . . . .	140
48.3.2 Towers of Hanoi . . . . .	140
48.3.3 Genetic algorithms . . . . .	140
48.3.4 Karnaugh maps . . . . .	141
48.3.5 Error correction . . . . .	141
48.3.6 Communication between clock domains . . . . .	141
48.3.7 Cycling through states with minimal effort . . . . .	141
48.4 Constructing an $n$ -bit Gray code . . . . .	142
48.5 Converting to and from Gray code . . . . .	143
48.6 Special types of Gray codes . . . . .	143
48.6.1 $n$ -ary Gray code . . . . .	143
48.6.2 Balanced Gray code . . . . .	144
48.6.3 Monotonic Gray codes . . . . .	144
48.6.4 Beckett–Gray code . . . . .	145
48.6.5 Snake-in-the-box codes . . . . .	145

48.6.6 Single-track Gray code . . . . .	145
48.6.7 2-dimensional Gray code . . . . .	146
48.7 Gray isometry . . . . .	146
48.8 See also . . . . .	146
48.9 References . . . . .	147
48.10 External links . . . . .	148
<b>49 Griesmer bound</b>	<b>149</b>
49.1 Statement of the bound . . . . .	149
49.2 Proof . . . . .	149
49.3 The bound for the general case . . . . .	149
49.4 See also . . . . .	150
49.5 References . . . . .	150
<b>50 Group code</b>	<b>151</b>
50.1 Construction . . . . .	151
50.2 References . . . . .	151
<b>51 Guruswami–Sudan list decoding algorithm</b>	<b>152</b>
51.1 Algorithm 1 (Sudan’s list decoding algorithm) . . . . .	152
51.1.1 Problem statement . . . . .	152
51.1.2 Algorithm . . . . .	152
51.1.3 Analysis . . . . .	153
51.2 Algorithm 2 (Guruswami–Sudan list decoding algorithm) . . . . .	153
51.2.1 Definition . . . . .	153
51.2.2 Multiplicity . . . . .	153
51.2.3 Algorithm . . . . .	154
51.2.4 Analysis . . . . .	154
51.2.5 Factorization step . . . . .	154
51.3 References . . . . .	155
<b>52 GV-linear-code</b>	<b>156</b>
52.1 Gilbert–Varshamov bound theorem . . . . .	156
52.2 Comments . . . . .	157
52.3 See also . . . . .	157
52.4 References . . . . .	157
<b>53 Hadamard code</b>	<b>158</b>
53.1 History . . . . .	159
53.2 Constructions . . . . .	159
53.2.1 Construction using inner products . . . . .	159
53.2.2 Construction using a generator matrix . . . . .	160
53.2.3 Construction using general Hadamard matrices . . . . .	160

53.3	Distance . . . . .	160
53.4	Local decodability . . . . .	161
53.4.1	Proof of lemma 1 . . . . .	161
53.4.2	Proof of theorem 1 . . . . .	161
53.5	Optimality . . . . .	161
53.6	See also . . . . .	161
53.7	Notes . . . . .	162
53.8	References . . . . .	162
<b>54</b>	<b>Hamming bound</b>	<b>163</b>
54.1	Background on error-correcting codes . . . . .	163
54.2	Statement of the bound . . . . .	163
54.3	Proof . . . . .	163
54.4	Covering radius and packing radius . . . . .	164
54.5	Perfect codes . . . . .	164
54.6	See also . . . . .	164
54.7	Notes . . . . .	164
54.8	References . . . . .	164
<b>55</b>	<b>Hamming code</b>	<b>165</b>
55.1	History . . . . .	165
55.1.1	Codes predating Hamming . . . . .	165
55.2	Hamming codes . . . . .	166
55.2.1	General algorithm . . . . .	166
55.3	Hamming codes with additional parity (SECDED) . . . . .	167
55.4	[7,4] Hamming code . . . . .	167
55.4.1	Construction of $G$ and $H$ . . . . .	167
55.4.2	Encoding . . . . .	168
55.4.3	[7,4] Hamming code with an additional parity bit . . . . .	168
55.5	See also . . . . .	169
55.6	Notes . . . . .	169
55.7	References . . . . .	169
55.8	External links . . . . .	169
<b>56</b>	<b>Hamming distance</b>	<b>170</b>
56.1	Examples . . . . .	170
56.2	Properties . . . . .	170
56.3	Error detection and error correction . . . . .	171
56.4	History and applications . . . . .	171
56.5	Algorithm example . . . . .	171
56.6	See also . . . . .	172
56.7	Notes . . . . .	172

56.8	References . . . . .	172
56.9	Further reading . . . . .	172
<b>57</b>	<b>Hamming scheme</b>	<b>173</b>
57.1	References . . . . .	173
<b>58</b>	<b>Hamming space</b>	<b>174</b>
58.1	References . . . . .	174
<b>59</b>	<b>Hamming weight</b>	<b>175</b>
59.1	History and usage . . . . .	175
59.2	Efficient implementation . . . . .	175
59.3	Language support . . . . .	176
59.4	Processor support . . . . .	177
59.5	See also . . . . .	177
59.6	References . . . . .	177
59.7	External links . . . . .	177
<b>60</b>	<b>Hamming(7,4)</b>	<b>178</b>
60.1	Goal . . . . .	178
60.2	Hamming matrices . . . . .	178
60.3	Channel coding . . . . .	179
60.4	Parity check . . . . .	180
60.5	Error correction . . . . .	180
60.6	Decoding . . . . .	181
60.7	Multiple bit errors . . . . .	181
60.8	All codewords . . . . .	182
60.9	References . . . . .	182
60.10	External links . . . . .	182
<b>61</b>	<b>Hexacode</b>	<b>183</b>
61.1	References . . . . .	183
<b>62</b>	<b>Homomorphic signatures for network coding</b>	<b>184</b>
62.1	Network coding . . . . .	184
62.2	Decoding at the receiver . . . . .	184
62.3	History . . . . .	185
62.4	Advantages of homomorphic signatures . . . . .	185
62.5	Signature scheme . . . . .	185
62.5.1	Elliptic curves cryptography over a finite field . . . . .	185
62.5.2	Weil pairing . . . . .	185
62.5.3	Homomorphic signatures . . . . .	186
62.5.4	Signature verification . . . . .	186
62.6	System setup . . . . .	186



62.7 Proof of security . . . . .	186
62.8 See also . . . . .	186
62.9 References . . . . .	187
62.10 External links . . . . .	187
<b>63 Second Johnson bound</b>	<b>188</b>
63.1 Definition . . . . .	188
63.2 See also . . . . .	188
63.3 References . . . . .	188
<b>64 Justesen code</b>	<b>190</b>
64.1 Definition . . . . .	190
64.2 Property of Justesen code . . . . .	190
64.3 Theorem . . . . .	190
64.3.1 Proof . . . . .	190
64.4 Comments . . . . .	191
64.5 An example of a Justesen code . . . . .	191
64.6 See also . . . . .	191
64.7 References . . . . .	192
<b>65 Kraft–McMillan inequality</b>	<b>193</b>
65.1 Applications and intuitions . . . . .	193
65.2 Formal statement . . . . .	193
65.3 Example: binary trees . . . . .	193
65.4 Proof . . . . .	194
65.4.1 Proof for prefix codes . . . . .	194
65.4.2 Proof of the general case . . . . .	194
65.4.3 Alternative construction for the converse . . . . .	195
65.5 Notes . . . . .	195
65.6 References . . . . .	195
65.7 See also . . . . .	195
<b>66 Lee distance</b>	<b>196</b>
66.1 Example . . . . .	196
66.2 History and application . . . . .	196
66.3 References . . . . .	196
<b>67 Linear code</b>	<b>197</b>
67.1 Definition and parameters . . . . .	197
67.2 Generator and check matrices . . . . .	197
67.3 Example: Hamming codes . . . . .	198
67.4 Example: Hadamard codes . . . . .	198
67.5 Nearest neighbor algorithm . . . . .	198

67.6 Popular notation . . . . .	198
67.7 Singleton bound . . . . .	198
67.8 Examples . . . . .	199
67.9 Generalization . . . . .	199
67.10 See also . . . . .	199
67.11 References . . . . .	199
67.12 External links . . . . .	199
<b>68 List decoding</b>	<b>200</b>
68.1 Mathematical formulation . . . . .	200
68.2 Motivation for list decoding . . . . .	200
68.3 List-decoding potential . . . . .	201
68.4 $(p, L)$ -list-decodability . . . . .	201
68.5 Combinatorics of list decoding . . . . .	201
68.6 List-decoding capacity . . . . .	201
68.6.1 Sketch of proof . . . . .	201
68.7 List-decoding algorithms . . . . .	202
68.8 Applications in complexity theory and cryptography . . . . .	203
68.9 External links . . . . .	203
<b>69 Long code (mathematics)</b>	<b>204</b>
69.1 Definition . . . . .	204
69.2 Properties . . . . .	204
69.3 References . . . . .	204
<b>70 Low-density parity-check code</b>	<b>205</b>
70.1 History . . . . .	205
70.2 Applications . . . . .	205
70.3 Operational use . . . . .	205
70.4 Example Encoder . . . . .	206
70.5 Decoding . . . . .	207
70.5.1 Updating node information . . . . .	208
70.5.2 Lookup table decoding . . . . .	208
70.6 Code construction . . . . .	208
70.7 See also . . . . .	209
70.7.1 People . . . . .	209
70.7.2 Theory . . . . .	209
70.7.3 Applications . . . . .	209
70.7.4 Other capacity-approaching codes . . . . .	209
70.8 References . . . . .	209
70.9 External links . . . . .	210
<b>71 Luby transform code</b>	<b>211</b>

71.1 Why use an LT code? . . . . .	211
71.2 LT encoding . . . . .	211
71.3 LT decoding . . . . .	212
71.4 Variations . . . . .	212
71.5 Optimization of LT codes . . . . .	212
71.6 See also . . . . .	213
71.7 Notes and references . . . . .	213
71.8 External links . . . . .	213
<b>72 Minimum weight</b>	<b>214</b>
72.1 References . . . . .	214
<b>73 Multiple description coding</b>	<b>215</b>
73.1 References . . . . .	215
<b>74 Linear network coding</b>	<b>216</b>
74.1 Encoding and decoding . . . . .	216
74.2 A brief history . . . . .	216
74.3 The butterfly network example . . . . .	216
74.4 Random Linear Network Coding . . . . .	217
74.4.1 Open issues . . . . .	217
74.5 Wireless Network Coding . . . . .	217
74.6 Applications . . . . .	217
74.7 See also . . . . .	218
74.8 References . . . . .	218
74.9 External links . . . . .	218
<b>75 Noisy text</b>	<b>220</b>
75.1 Techniques for noise reduction . . . . .	220
75.2 See also . . . . .	220
75.3 References . . . . .	220
<b>76 Noisy-channel coding theorem</b>	<b>221</b>
76.1 Overview . . . . .	221
76.2 Mathematical statement . . . . .	221
76.3 Outline of proof . . . . .	222
76.3.1 Achievability for discrete memoryless channels . . . . .	222
76.3.2 Weak converse for discrete memoryless channels . . . . .	223
76.3.3 Strong converse for discrete memoryless channels . . . . .	223
76.4 Channel coding theorem for non-stationary memoryless channels . . . . .	223
76.4.1 Outline of the proof . . . . .	223
76.5 See also . . . . .	223
76.6 Notes . . . . .	223

76.7	References . . . . .	224
76.8	External links . . . . .	224
<b>77</b>	<b>Online codes</b>	<b>225</b>
77.1	Detailed discussion . . . . .	225
77.1.1	Outer encoding . . . . .	225
77.1.2	Inner encoding . . . . .	225
77.1.3	Decoding . . . . .	226
77.2	External links . . . . .	226
<b>78</b>	<b>Package-merge algorithm</b>	<b>227</b>
78.1	The coin collector's problem . . . . .	227
78.2	Description of the package-merge algorithm . . . . .	227
78.3	Reduction of length-limited Huffman coding to the coin collector's problem . . . . .	227
78.4	Performance improvements and generalizations . . . . .	227
78.5	References . . . . .	228
78.6	External links . . . . .	228
<b>79</b>	<b>Packet erasure channel</b>	<b>229</b>
79.1	See also . . . . .	229
79.2	References . . . . .	229
<b>80</b>	<b>Parity-check matrix</b>	<b>230</b>
80.1	Definition . . . . .	230
80.2	Creating a parity check matrix . . . . .	230
80.3	Syndromes . . . . .	230
80.4	See also . . . . .	230
80.5	Notes . . . . .	231
80.6	References . . . . .	231
<b>81</b>	<b>Parvaresh–Vardy code</b>	<b>232</b>
81.1	See also . . . . .	232
81.2	References . . . . .	232
<b>82</b>	<b>Plotkin bound</b>	<b>233</b>
82.1	Statement of the bound . . . . .	233
82.2	Proof of case $i$ ) . . . . .	233
82.3	See also . . . . .	234
82.4	References . . . . .	234
<b>83</b>	<b>Polar code (coding theory)</b>	<b>235</b>
83.1	Simulating Polar Codes . . . . .	235
83.2	Industrial Applications . . . . .	235
83.3	See also . . . . .	235

83.4	References	235
<b>84</b>	<b>Polynomial code</b>	<b>236</b>
84.1	Definition	236
84.2	Example	236
84.3	Encoding	236
84.3.1	Example	237
84.4	Decoding	237
84.5	Properties of polynomial codes	237
84.6	Specific families of polynomial codes	237
84.7	References	237
<b>85</b>	<b>Prefix code</b>	<b>238</b>
85.1	Techniques	238
85.2	Related concepts	239
85.3	Prefix codes in use today	239
85.3.1	Techniques	239
85.4	Notes	239
85.5	References	239
85.6	External links	240
<b>86</b>	<b>Preparata code</b>	<b>241</b>
86.1	Construction	241
86.2	Properties	241
86.3	References	241
<b>87</b>	<b>Puncturing</b>	<b>242</b>
87.1	See also	242
87.2	References	242
<b>88</b>	<b>Quadratic residue code</b>	<b>243</b>
88.1	Examples	243
88.2	Constructions	243
88.3	Weight	243
88.4	Extended code	243
88.5	References	243
<b>89</b>	<b>Rank error-correcting code</b>	<b>244</b>
89.1	Rank metric	244
89.2	Rank code	244
89.3	Generating matrix	244
89.4	Applications	244
89.5	See also	245
89.6	Notes	245

89.7	References . . . . .	245
89.8	External links . . . . .	245
<b>90</b>	<b>Raptor code</b>	<b>246</b>
90.1	Overview . . . . .	246
90.2	Decoding . . . . .	246
90.3	Computational complexity . . . . .	247
90.4	Legal complexity . . . . .	247
90.5	See also . . . . .	247
90.6	Notes . . . . .	247
90.7	References . . . . .	247
<b>91</b>	<b>Recursive indexing</b>	<b>248</b>
91.1	Encoding . . . . .	248
91.1.1	Example . . . . .	248
91.2	Decoding . . . . .	248
91.2.1	Example . . . . .	248
91.3	Uses . . . . .	248
91.4	References . . . . .	248
<b>92</b>	<b>Reed–Muller code</b>	<b>249</b>
92.1	Construction . . . . .	249
92.1.1	Building a generator matrix . . . . .	249
92.2	Example 1 . . . . .	250
92.3	Example 2 . . . . .	250
92.4	Properties . . . . .	250
92.4.1	Proof . . . . .	250
92.5	Alternative construction . . . . .	251
92.6	Construction based on low-degree polynomials over a finite field . . . . .	251
92.7	Table of Reed–Muller codes . . . . .	251
92.8	Decoding RM codes . . . . .	251
92.9	Notes . . . . .	251
92.10	References . . . . .	252
92.11	External links . . . . .	252
<b>93</b>	<b>Reed–Solomon error correction</b>	<b>253</b>
93.1	History . . . . .	253
93.2	Applications . . . . .	253
93.2.1	Data storage . . . . .	253
93.2.2	Bar code . . . . .	254
93.2.3	Data transmission . . . . .	254
93.2.4	Space transmission . . . . .	254
93.3	Constructions . . . . .	254

93.3.1	Reed & Solomon's original view: The codeword as a sequence of values . . . . .	254
93.3.2	The BCH view: The codeword as a sequence of coefficients . . . . .	256
93.3.3	Duality of the two views - discrete Fourier transform . . . . .	257
93.3.4	Remarks . . . . .	257
93.4	Properties . . . . .	257
93.5	Error correction algorithms . . . . .	258
93.5.1	Peterson–Gorenstein–Zierler decoder . . . . .	258
93.5.2	Berlekamp–Massey decoder . . . . .	260
93.5.3	Euclidean decoder . . . . .	260
93.5.4	Decoder using discrete Fourier transform . . . . .	261
93.5.5	Decoding beyond the error-correction bound . . . . .	261
93.5.6	Soft-decoding . . . . .	261
93.5.7	Matlab Example . . . . .	261
93.6	See also . . . . .	262
93.7	Notes . . . . .	262
93.8	References . . . . .	263
93.9	Further reading . . . . .	263
93.10	External links . . . . .	263
93.10.1	Information and Tutorials . . . . .	263
93.10.2	Code . . . . .	263
<b>94</b>	<b>Sardinas–Patterson algorithm</b>	<b>265</b>
94.1	Idea of the algorithm . . . . .	265
94.2	Precise description of the algorithm . . . . .	265
94.3	Termination and correctness of the algorithm . . . . .	266
94.4	See also . . . . .	266
94.5	Notes . . . . .	266
94.6	References . . . . .	266
<b>95</b>	<b>Second Johnson bound</b>	<b>267</b>
95.1	Definition . . . . .	267
95.2	See also . . . . .	267
95.3	References . . . . .	267
<b>96</b>	<b>Shannon's source coding theorem</b>	<b>269</b>
96.1	Statements . . . . .	269
96.1.1	Source coding theorem . . . . .	269
96.1.2	Source coding theorem for symbol codes . . . . .	269
96.2	Proof: Source coding theorem . . . . .	269
96.3	Proof: Source coding theorem for symbol codes . . . . .	270
96.4	Extension to non-stationary independent sources . . . . .	271
96.4.1	Fixed Rate lossless source coding for discrete time non-stationary independent sources . . .	271

96.5 See also . . . . .	271
96.6 References . . . . .	271
<b>97 Singleton bound</b>	<b>272</b>
97.1 Statement of the bound . . . . .	272
97.2 Proof . . . . .	272
97.3 Linear codes . . . . .	272
97.4 History . . . . .	272
97.5 MDS codes . . . . .	272
97.5.1 Arcs in projective geometry . . . . .	273
97.6 See also . . . . .	273
97.7 Notes . . . . .	273
97.8 References . . . . .	273
97.9 Further reading . . . . .	274
<b>98 Soliton distribution</b>	<b>275</b>
98.1 Ideal distribution . . . . .	275
98.2 Robust distribution . . . . .	275
98.3 See also . . . . .	275
98.4 References . . . . .	275
<b>99 Spherical code</b>	<b>276</b>
99.1 External links . . . . .	276
<b>100Srivastava code</b>	<b>277</b>
100.1Definition . . . . .	277
100.2Properties . . . . .	277
100.3References . . . . .	277
<b>101Standard array</b>	<b>278</b>
101.1Definition . . . . .	278
101.2Constructing a standard array . . . . .	278
101.2.1 Construction example . . . . .	278
101.3Decoding via standard array . . . . .	279
101.4See also . . . . .	279
101.5References . . . . .	279
<b>102Systematic code</b>	<b>280</b>
102.1Properties . . . . .	280
102.2Examples . . . . .	280
102.3Notes . . . . .	280
102.4References . . . . .	280
<b>103Tanner graph</b>	<b>281</b>



103.1	Origins	281
103.2	Tanner graphs for linear block codes	281
103.3	Bounds proved by Tanner	281
103.4	Computational complexity of Tanner graph based methods	281
103.5	Applications of Tanner graph	281
103.6	Notes	281
<b>104</b>	<b>Ternary Golay code</b>	<b>283</b>
104.1	Properties	283
104.1.1	Ternary Golay code	283
104.1.2	Extended ternary Golay code	283
104.2	History	283
104.3	See also	283
104.4	References	283
<b>105</b>	<b>Tornado code</b>	<b>285</b>
105.1	Overview	285
105.2	Patent issues	285
105.3	Citations	286
105.4	External links	286
105.5	See also	286
105.6	Notes	286
105.7	References	286
<b>106</b>	<b>Triangular network coding</b>	<b>287</b>
106.1	Coding and decoding	287
106.2	References	287
<b>107</b>	<b>Unary coding</b>	<b>288</b>
107.1	Unary code in use today	288
107.2	Unary coding in biological networks	288
107.3	Generalized unary coding	288
107.4	See also	288
107.5	References	289
<b>108</b>	<b>Variable-length code</b>	<b>290</b>
108.1	Codes and their extensions	290
108.2	Classes of variable-length codes	290
108.2.1	Non-singular codes	290
108.2.2	Uniquely decodable codes	290
108.2.3	Prefix codes	291
108.3	Advantages	291
108.4	Notes	291

108.5References . . . . .	291
<b>109Z-channel (information theory)</b>	<b>292</b>
109.1Definition . . . . .	292
109.2Capacity . . . . .	292
109.3Bounds on the size of an asymmetric-error-correcting code . . . . .	292
109.4References . . . . .	293
<b>110Zemor's decoding algorithm</b>	<b>294</b>
110.1Code construction . . . . .	294
110.1.1 Claim 1 . . . . .	294
110.1.2 Claim 2 . . . . .	295
110.2Zemor's algorithm . . . . .	295
110.2.1 Decoder algorithm . . . . .	295
110.2.2 Explanation of the algorithm . . . . .	296
110.2.3 Theorem . . . . .	296
110.3Drawbacks of Zemor's algorithm . . . . .	296
110.4See also . . . . .	296
110.5References . . . . .	296
<b>111Zigzag code</b>	<b>297</b>
111.1References . . . . .	297
<b>112Zyablov bound</b>	<b>298</b>
112.1Statement of the bound . . . . .	298
112.2Description . . . . .	298
112.3Remarks . . . . .	298
112.4See also . . . . .	299
112.5References and External Links . . . . .	299
112.6Text and image sources, contributors, and licenses . . . . .	300
112.6.1 Text . . . . .	300
112.6.2 Images . . . . .	307
112.6.3 Content license . . . . .	311

# Chapter 1

## Fountain code

In coding theory, **fountain codes** (also known as **rateless erasure codes**) are a class of **erasure codes** with the property that a potentially limitless sequence of **encoding symbols** can be generated from a given set of source symbols such that the original source symbols can ideally be recovered from any subset of the encoding symbols of size equal to or only slightly larger than the number of source symbols. The term *fountain* or *rateless* refers to the fact that these codes do not exhibit a fixed **code rate**.

A fountain code is optimal if the original  $k$  source symbols can be recovered from any  $k$  encoding symbols. Fountain codes are known that have efficient encoding and decoding **algorithms** and that allow the recovery of the original  $k$  source symbols from any  $k'$  of the encoding symbols with high probability, where  $k'$  is just slightly larger than  $k$ .

**LT codes** were the first practical realization of fountain codes. **Raptor codes** and **online codes** were subsequently introduced, and achieve linear time encoding and decoding **complexity** through a pre-coding stage of the input symbols.

### 1.1 Applications

Fountain codes are flexibly applicable at a fixed **code rate**, or where a fixed code rate cannot be determined a priori, and where efficient encoding and decoding of large amounts of data is required.

One example is that of a **data carousel**, where some large file is continuously broadcast to a set of receivers.<sup>[1]</sup> Using a fixed-rate erasure code, a receiver missing a source symbol (due to a transmission error) faces the **coupon collector's problem**: it must successfully receive an encoding symbol which it does not already have. This problem becomes much more apparent when using a traditional short-length erasure code, as the file must be split into several blocks, each being separately encoded: the receiver must now collect the required number of missing encoding symbols for *each* block. Using a fountain code, it suffices for a receiver to retrieve *any* subset of encoding symbols of size slightly larger than the set of source symbols. (In practice, the broadcast is typically scheduled for

a fixed period of time by an operator based on characteristics of the network and receivers and desired delivery reliability, and thus the fountain code is used at a code rate that is determined dynamically at the time when the file is scheduled to be broadcast.)

Another application is that of **hybrid ARQ** in **reliable multicast** scenarios: parity information that is requested by a receiver can potentially be useful for *all* receivers in the multicast group.

### 1.2 Fountain codes in standards

**Raptor codes** are the most efficient fountain codes at this time,<sup>[2]</sup> having very efficient linear time encoding and decoding algorithms, and requiring only a small constant number of **XOR** operations per generated symbol for both encoding and decoding.<sup>[3]</sup> **IETF RFC 5053** specifies in detail a **systematic** Raptor code, which has been adopted into multiple standards beyond the IETF, such as within the **3GPP MBMS** standard for broadcast file delivery and streaming services, the **DVB-H IPDC** standard for delivering IP services over **DVB** networks, and **DVB-IPTV** for delivering commercial TV services over an IP network. This code can be used with up to 8,192 source symbols in a source block, and a total of up to 65,536 encoded symbols generated for a source block. This code has an average relative reception overhead of 0.2% when applied to source blocks with 1,000 source symbols, and has a relative reception overhead of less than 2% with probability 99.9999%.<sup>[4]</sup> The relative reception overhead is defined as the extra encoding data required beyond the length of the source data to recover the original source data, measured as a percentage of the size of the source data. For example, if the relative reception overhead is 0.2%, then this means that source data of size 1 **megabyte** can be recovered from 1.002 megabytes of encoding data.

A more advanced Raptor code with greater flexibility and improved reception overhead, called RaptorQ, has been introduced into the IETF.<sup>[5]</sup> This code can be used with up to 56,403 source symbols in a source block, and a total of up to 16,777,216 encoded symbols generated for a source block. This code is able to recover a source block from any set of encoded symbols equal to the number of

source symbols in the source block with high probability, and in rare cases from slightly more than the number of source symbols in the source block.

### 1.3 Fountain codes for data storage

Erasure codes are used in data storage applications due to massive savings on the number of storage units for a given level of redundancy and reliability. The requirements of erasure code design for data storage, particularly for distributed storage applications, might be quite different relative to communication or data streaming scenarios. One of the requirements of coding for data storage systems is the systematic form, i.e., the original message symbols are part of the coded symbols. Systematic form enables reading off the message symbols without decoding from a storage unit. In addition, since the bandwidth and communication load between storage nodes can be a bottleneck, codes that allow minimum communication are very beneficial particularly when a node fails and a system reconstruction is needed to achieve the initial level of redundancy. In that respect, fountain codes are expected to allow efficient repair process in case of a failure: When a single encoded symbol is lost, it should not require too much communication and computation among other encoded symbols in order to resurrect the lost symbol. In fact, repair latency might sometimes be more important than storage space savings. Repairable fountain codes<sup>[6]</sup> are projected to address fountain code design objectives for storage systems. A detailed survey about fountain codes and their applications can be found at.<sup>[7]</sup>

### 1.4 See also

- Online codes
- Linear network coding
- Secret sharing
- Tornado codes, the precursor to *fountain codes*

### 1.5 Notes

- [1] J. Byers, M. Luby, M. Mitzenmacher, A. Rege (1998). "A Digital Fountain Approach to Reliable Distribution of Bulk Data" (PDF).
- [2] "Qualcomm Raptor Technology - Forward Error Correction".
- [3] (Shokrollahi 2006)
- [4] T. Stockhammer, A. Shokrollahi, M. Watson, M. Luby, T. Gasiba (March 2008). Furht, B.; Ahson, S., eds. "Application Layer Forward Error Correction for Mobile Multimedia Broadcasting". *Handbook of Mobile Broadcasting: DVB-H, DMB, ISDB-T and Media FLO*. CRC Press.

[5] (Luby et al. 2010)

[6] M. Asteris; A. G. Dimakis (2012). "'Repairable Fountain Codes'", In Proc. of 2012 IEEE International Symposium on Information Theory". arXiv:1401.0734v3 [cs.IT].

[7] Suayb S. Arslan (2014). "Incremental Redundancy, Fountain Codes and Advanced Topics" (PDF).

### 1.6 References

- M. Luby (2002). "LT Codes". *Proceedings of the IEEE Symposium on the Foundations of Computer Science*: 271–280.
- A. Shokrollahi (2006), "Raptor Codes" (PDF), *Transactions on Information Theory*, IEEE, **52** (6): 2551–2567.
- P. Maymounkov (November 2002). "Online Codes" (PDF). (*Technical Report*).
- David J. C. MacKay (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press. ISBN 0-521-64298-1.
- M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer (October 2007), "Raptor Forward Error Correction Scheme for Object Delivery", *Rfc 5053*, IETF.
- M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, L. Minder (May 2011), *RaptorQ Forward Error Correction Scheme for Object Delivery*, IETF.

## Chapter 2

# Forward error correction

“Interleaver” redirects here. For the fiber-optic device, see [optical interleaver](#).

In telecommunication, information theory, and coding theory, **forward error correction** (FEC) or **channel coding**<sup>[1]</sup> is a technique used for controlling errors in data transmission over unreliable or noisy communication channels. The central idea is the sender encodes the message in a redundant way by using an **error-correcting code** (ECC). The American mathematician Richard Hamming pioneered this field in the 1940s and invented the first error-correcting code in 1950: the Hamming (7,4) code.<sup>[2]</sup>

The redundancy allows the receiver to detect a limited number of errors that may occur anywhere in the message, and often to correct these errors without retransmission. FEC gives the receiver the ability to correct errors without needing a **reverse channel** to request retransmission of data, but at the cost of a fixed, higher forward channel bandwidth. FEC is therefore applied in situations where retransmissions are costly or impossible, such as one-way communication links and when transmitting to multiple receivers in **multicast**. FEC information is usually added to **mass storage** devices to enable recovery of corrupted data, and is widely used in **modems**.

FEC processing in a receiver may be applied to a digital bit stream or in the demodulation of a digitally modulated carrier. For the latter, FEC is an integral part of the initial **analog-to-digital conversion** in the receiver. The Viterbi decoder implements a **soft-decision algorithm** to demodulate digital data from an analog signal corrupted by noise. Many FEC coders can also generate a **bit-error rate** (BER) signal which can be used as feedback to fine-tune the analog receiving electronics.

The **noisy-channel coding theorem** establishes bounds on the theoretical maximum information transfer rate of a channel with some given noise level. Some advanced FEC systems come very close to the theoretical maximum.

The maximum fractions of errors or of missing bits that can be corrected is determined by the design of the FEC code, so different forward error correcting codes are suitable for different conditions.

## 2.1 How it works

FEC is accomplished by adding **redundancy** to the transmitted information using an algorithm. A redundant bit may be a complex function of many original information bits. The original information may or may not appear literally in the encoded output; codes that include the unmodified input in the output are **systematic**, while those that do not are **non-systematic**.

A simplistic example of FEC is to transmit each data bit 3 times, which is known as a (3,1) **repetition code**. Through a noisy channel, a receiver might see 8 versions of the output, see table below.

This allows an error in any one of the three samples to be corrected by “majority vote” or “democratic voting”. The correcting ability of this FEC is:

- Up to 1 bit of triplet in error, or
- up to 2 bits of triplet omitted (cases not shown in table).

Though simple to implement and widely used, this **triple modular redundancy** is a relatively inefficient FEC. Better FEC codes typically examine the last several dozen, or even the last several hundred, previously received bits to determine how to decode the current small handful of bits (typically in groups of 2 to 8 bits).

## 2.2 Averaging noise to reduce errors

FEC could be said to work by “averaging noise”; since each data bit affects many transmitted symbols, the corruption of some symbols by noise usually allows the original user data to be extracted from the other, uncorrupted received symbols that also depend on the same user data.

- Because of this “risk-pooling” effect, digital communication systems that use FEC tend to work well above a certain minimum **signal-to-noise ratio** and not at all below it.

- This *all-or-nothing tendency* — the **cliff effect** — becomes more pronounced as stronger codes are used that more closely approach the theoretical **Shannon limit**.
- Interleaving FEC coded data can reduce the all or nothing properties of transmitted FEC codes when the channel errors tend to occur in bursts. However, this method has limits; it is best used on narrowband data.

Most telecommunication systems use a fixed **channel code** designed to tolerate the expected worst-case bit error rate, and then fail to work at all if the bit error rate is ever worse. However, some systems adapt to the given channel error conditions: some instances of **hybrid automatic repeat-request** use a fixed FEC method as long as the FEC can handle the error rate, then switch to **ARQ** when the error rate gets too high; **adaptive modulation and coding** uses a variety of FEC rates, adding more error-correction bits per packet when there are higher error rates in the channel, or taking them out when they are not needed.

## 2.3 Types of FEC

Main articles: **Block code** and **Convolutional code**

The two main categories of FEC codes are **block codes** and **convolutional codes**.

- Block codes work on fixed-size blocks (packets) of bits or symbols of predetermined size. Practical block codes can generally be hard-decoded in **polynomial time** to their block length.
- Convolutional codes work on bit or symbol streams of arbitrary length. They are most often soft decoded with the **Viterbi algorithm**, though other algorithms are sometimes used. Viterbi decoding allows asymptotically optimal decoding efficiency with increasing constraint length of the convolutional code, but at the expense of **exponentially** increasing complexity. A convolutional code that is terminated is also a 'block code' in that it encodes a block of input data, but the block size of a convolutional code is generally arbitrary, while block codes have a fixed size dictated by their algebraic characteristics. Types of termination for convolutional codes include “tail-biting” and “bit-flushing”.

There are many types of block codes, but among the classical ones the most notable is **Reed-Solomon coding** because of its widespread use on the **Compact disc**, the **DVD**, and in **hard disk drives**. Other examples of classical block codes include **Golay**, **BCH**, **Multidimensional parity**, and **Hamming codes**.

Hamming ECC is commonly used to correct **NAND flash** memory errors.<sup>[3]</sup> This provides single-bit error correction and 2-bit error detection. Hamming codes are only suitable for more reliable **single level cell** (SLC) NAND. Denser **multi level cell** (MLC) NAND requires stronger multi-bit correcting ECC such as BCH or Reed–Solomon.<sup>[4][5]</sup> NOR Flash typically does not use any error correction.<sup>[4]</sup>

Classical block codes are usually decoded using **hard-decision** algorithms,<sup>[6]</sup> which means that for every input and output signal a hard decision is made whether it corresponds to a one or a zero bit. In contrast, convolutional codes are typically decoded using **soft-decision** algorithms like the Viterbi, MAP or **BCJR** algorithms, which process (discretized) analog signals, and which allow for much higher error-correction performance than hard-decision decoding.

Nearly all classical block codes apply the algebraic properties of **finite fields**. Hence classical block codes are often referred to as algebraic codes.

In contrast to classical block codes that often specify an error-detecting or error-correcting ability, many modern block codes such as **LDPC codes** lack such guarantees. Instead, modern codes are evaluated in terms of their bit error rates.

Most forward error correction correct only bit-flips, but not bit-insertions or bit-deletions. In this setting, the **Hamming distance** is the appropriate way to measure the **bit error rate**. A few forward error correction codes are designed to correct bit-insertions and bit-deletions, such as Marker Codes and Watermark Codes. The **Levenshtein distance** is a more appropriate way to measure the bit error rate when using such codes.<sup>[7]</sup>

## 2.4 Concatenated FEC codes for improved performance

Main article: **Concatenated error correction codes**

Classical (algebraic) block codes and convolutional codes are frequently combined in **concatenated** coding schemes in which a short constraint-length Viterbi-decoded convolutional code does most of the work and a block code (usually Reed-Solomon) with larger symbol size and block length “mops up” any errors made by the convolutional decoder. Single pass decoding with this family of error correction codes can yield very low error rates, but for long range transmission conditions (like deep space) iterative decoding is recommended.

Concatenated codes have been standard practice in satellite and deep space communications since **Voyager 2** first used the technique in its 1986 encounter with **Uranus**. The **Galileo** craft used iterative concatenated codes to compensate for the very high error rate conditions caused



by having a failed antenna.

## 2.5 Low-density parity-check (LDPC)

Main article: [Low-density parity-check code](#)

**Low-density parity-check (LDPC)** codes are a class of recently re-discovered highly efficient linear block codes made from many single parity check (SPC) codes. They can provide performance very close to the **channel capacity** (the theoretical maximum) using an iterated soft-decision decoding approach, at linear time complexity in terms of their block length. Practical implementations rely heavily on decoding the constituent SPC codes in parallel.

LDPC codes were first introduced by **Robert G. Gallager** in his PhD thesis in 1960, but due to the computational effort in implementing encoder and decoder and the introduction of **Reed–Solomon** codes, they were mostly ignored until recently.

LDPC codes are now used in many recent high-speed communication standards, such as **DVB-S2** (Digital video broadcasting), **WiMAX** (IEEE 802.16e standard for microwave communications), **High-Speed Wireless LAN** (IEEE 802.11n), **10GBase-T Ethernet** (802.3an) and **G.hn/G.9960** (ITU-T Standard for networking over power lines, phone lines and coaxial cable). Other LDPC codes are standardized for wireless communication standards within **3GPP MBMS** (see [fountain codes](#)).

## 2.6 Turbo codes

Main article: [Turbo code](#)

**Turbo coding** is an iterated soft-decoding scheme that combines two or more relatively simple convolutional codes and an interleaver to produce a block code that can perform to within a fraction of a decibel of the **Shannon limit**. Predating **LDPC codes** in terms of practical application, they now provide similar performance.

One of the earliest commercial applications of turbo coding was the **CDMA2000 1x** (TIA IS-2000) digital cellular technology developed by **Qualcomm** and sold by **Verizon Wireless**, **Sprint**, and other carriers. It is also used for the evolution of CDMA2000 1x specifically for Internet access, **1xEV-DO** (TIA IS-856). Like 1x, EV-DO was developed by **Qualcomm**, and is sold by **Verizon Wireless**, **Sprint**, and other carriers (Verizon's marketing name for 1xEV-DO is *Broadband Access*, Sprint's consumer and business marketing names for 1xEV-DO are *Power Vision* and *Mobile Broadband*, respectively).

## 2.7 Local decoding and testing of codes

Main articles: [Locally decodable code](#) and [Locally testable code](#)

Sometimes it is only necessary to decode single bits of the message, or to check whether a given signal is a codeword, and do so without looking at the entire signal. This can make sense in a streaming setting, where codewords are too large to be classically decoded fast enough and where only a few bits of the message are of interest for now. Also such codes have become an important tool in **computational complexity theory**, e.g., for the design of **probabilistically checkable proofs**.

**Locally decodable codes** are error-correcting codes for which single bits of the message can be probabilistically recovered by only looking at a small (say constant) number of positions of a codeword, even after the codeword has been corrupted at some constant fraction of positions. **Locally testable codes** are error-correcting codes for which it can be checked probabilistically whether a signal is close to a codeword by only looking at a small number of positions of the signal.

## 2.8 Interleaving

Interleaving is frequently used in digital communication and storage systems to improve the performance of forward error correcting codes. Many **communication channels** are not memoryless: errors typically occur in **bursts** rather than independently. If the number of errors within a **code word** exceeds the error-correcting code's capability, it fails to recover the original code word. Interleaving ameliorates this problem by shuffling source symbols across several code words, thereby creating a more **uniform distribution** of errors.<sup>[8]</sup> Therefore, interleaving is widely used for **burst error-correction**.

The analysis of modern iterated codes, like **turbo codes** and **LDPC codes**, typically assumes an independent distribution of errors.<sup>[9]</sup> Systems using LDPC codes therefore typically employ additional interleaving across the symbols within a code word.<sup>[10]</sup>

For turbo codes, an interleaver is an integral component and its proper design is crucial for good performance.<sup>[8][11]</sup> The iterative decoding algorithm works best when there are not short cycles in the **factor graph** that represents the decoder; the interleaver is chosen to avoid short cycles.

Interleaver designs include:

- rectangular (or uniform) interleavers (similar to the method using skip factors described above)

- convolutional interleavers
- random interleavers (where the interleaver is a known random permutation)
- S-random interleaver (where the interleaver is a known random permutation with the constraint that no input symbols within distance S appear within a distance of S in the output).<sup>[12]</sup>
- Another possible construction is a contention-free quadratic permutation polynomial (QPP).<sup>[13]</sup> It is used for example in the 3GPP Long Term Evolution mobile telecommunication standard.<sup>[14]</sup>

In multi-carrier communication systems, interleaving across carriers may be employed to provide frequency diversity, e.g., to mitigate frequency-selective fading or narrowband interference.<sup>[15]</sup>

### 2.8.1 Example

#### Transmission without interleaving:

Error-free message: aaaabbbbccccdddeeefffgggg  
Transmission with a burst error: aaaabbbbcccc\_\_\_\_deeefffgggg

Here, each group of the same letter represents a 4-bit one-bit error-correcting codeword. The codeword cccc is altered in one bit and can be corrected, but the codeword dddd is altered in three bits, so either it cannot be decoded at all or it might be decoded incorrectly.

#### With interleaving:

Error-free code words: aaaabbbbccccdddeeefffgggg  
Interleaved: abcdefgabcdefgabcdefgabcdefg  
Transmission with a burst error: abcdefgabcd\_\_\_\_bcdefgabcdefg  
Received code words after deinterleaving: aa\_abbbbccccddde\_eef\_ffg\_gg

In each of the codewords aaaa, eeee, ffff, gggg, only one bit is altered, so one-bit error-correcting code will decode everything correctly.

#### Transmission without interleaving:

Original transmitted sentence: ThisIsAnExampleOfInterleaving  
Received sentence with a burst error: ThisIs\_\_\_\_pleOfInterleaving

The term “AnExample” ends up mostly unintelligible and difficult to correct.

#### With interleaving:

Transmitted sentence: ThisIsAnExampleOfInterleaving...  
Error-free transmission: TIEpfeaghsxllrv.iAenli.snmOten.  
Received sentence with a burst error: TIEpfe\_\_\_\_Irv.iAenli.snmOten.  
Received sentence after deinterleaving: T\_isI\_AnE\_amp\_eOfInterle\_vin\_...

No word is completely lost and the missing letters can be recovered with minimal guesswork.

### 2.8.2 Disadvantages of interleaving

Use of interleaving techniques increases total delay. This is because the entire interleaved block must be received before the packets can be decoded.<sup>[16]</sup> Also interleavers hide the structure of errors; without an interleaver, more advanced decoding algorithms can take advantage of the error structure and achieve more reliable communication than a simpler decoder combined with an interleaver.

## 2.9 List of error-correcting codes

- AN codes
- BCH code, which can be designed to correct any arbitrary number of errors per code block.
- Berger code
- Constant-weight code
- Convolutional code
- Expander codes
- Group codes
- Golay codes, of which the Binary Golay code is of practical interest
- Goppa code, used in the McEliece cryptosystem
- Hadamard code
- Hagelbarger code
- Hamming code
- Latin square based code for non-white noise (prevalent for example in broadband over powerlines)
- Lexicographic code
- Long code
- Low-density parity-check code, also known as Gallager code, as the archetype for sparse graph codes
- LT code, which is a near-optimal rateless erasure correcting code (Fountain code)
- m of n codes
- Online code, a near-optimal rateless erasure correcting code
- Polar code (coding theory)




- Raptor code, a near-optimal rateless erasure correcting code
- Reed–Solomon error correction
- Reed–Muller code
- Repeat-accumulate code
- Repetition codes, such as Triple modular redundancy
- Spinal code, a rateless, nonlinear code based on pseudo-random hash functions <sup>[17]</sup>
- Tornado code, a near-optimal erasure correcting code, and the precursor to Fountain codes
- Turbo code
- Walsh–Hadamard code

## 2.10 See also

- Code rate
- Erasure codes
- Soft-decision decoder
- Error detection and correction
- Error-correcting codes with feedback
- Burst error-correcting code

## 2.11 References

- [1] Charles Wang; Dean Sklar; Diana Johnson (Winter 2001–2002). “Forward Error-Correction Coding”. *Crosslink — The Aerospace Corporation magazine of advances in aerospace technology*. The Aerospace Corporation. **3** (1). How Forward Error-Correcting Codes Work
- [2] Hamming, R. W. (April 1950). “Error Detecting and Error Correcting Codes” (PDF). *Bell System Tech. J.* USA: AT&T. **29** (2): 147–160. doi:10.1002/j.1538-7305.1950.tb00463.x. Retrieved 4 December 2012.
- [3] “Hamming codes for NAND flash memory devices”. EE Times-Asia. Apparently based on “Micron Technical Note TN-29-08: Hamming Codes for NAND Flash Memory Devices”. 2005. Both say: “The Hamming algorithm is an industry-accepted method for error detection and correction in many SLC NAND flash-based applications.”
- [4] “What Types of ECC Should Be Used on Flash Memory?”. (Spanion application note). 2011. says: “Both Reed-Solomon algorithm and BCH algorithm are common ECC choices for MLC NAND flash. ... Hamming based block codes are the most commonly used ECC for SLC.... both Reed-Solomon and BCH are able to handle multiple errors and are widely used on MLC flash.”
- [5] Jim Cooke. “The Inconvenient Truths of NAND Flash Memory”. 2007. p. 28. says “For SLC, a code with a correction threshold of 1 is sufficient. t=4 required ... for MLC.”
- [6] Baldi M.; Chiaraluce F. (2008). “A Simple Scheme for Belief Propagation Decoding of BCH and RS Codes in Multimedia Transmissions”. *International Journal of Digital Multimedia Broadcasting*. **2008**: 957846. doi:10.1155/2008/957846.
- [7] Shah, Gaurav; Molina, Andres; Blaze, Matt (2006). “Keyboards and covert channels” (PDF). *Proceedings of the 15th conference on USENIX Security Symposium*.
- [8] B. Vucetic; J. Yuan (2000). *Turbo codes: principles and applications*. Springer Verlag. ISBN 978-0-7923-7868-6.
- [9] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, V. Stemann (1997). “Practical Loss-Resilient Codes”. *Proc. 29th annual Association for Computing Machinery (ACM) symposium on Theory of computation*.
- [10] “Digital Video Broadcast (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other satellite broadband applications (DVB-S2)”. *En 302 307*. ETSI (V1.2.1). April 2009.
- [11] K. Andrews; et al. (November 2007). “The Development of Turbo and LDPC Codes for Deep-Space Applications”. *Proc. of the IEEE*. **95** (11).
- [12] S. Dolinar and D. Divsalar. Weight Distributions for Turbo Codes Using Random and Nonrandom Permutations. 1995.
- [13] Takeshita, Oscar (2006). “Permutation Polynomial Interleavers: An Algebraic-Geometric Perspective”. [arXiv:cs/0601048](https://arxiv.org/abs/cs/0601048) .
- [14] 3GPP TS 36.212, version 8.8.0, page 14
- [15] “Digital Video Broadcast (DVB); Frame structure, channel coding and modulation for a second generation digital terrestrial television broadcasting system (DVB-T2)”. *En 302 755*. ETSI (V1.1.1). September 2009.
- [16] “Explaining Interleaving - W3techie”. w3techie.com. Retrieved 2010-06-03.
- [17] Perry, Jonathan; Balakrishnan, Hari; Shah, Devavrat (2011). “Rateless Spinal Codes”. *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. doi:10.1145/2070562.2070568.

## 2.12 Further reading

- Clark, George C., Jr.; Cain, J. Bibb (1981). *Error-Correction Coding for Digital Communications*. New York: Plenum Press. ISBN 0-306-40615-2.
- Wicker, Stephen B. (1995). *Error Control Systems for Digital Communication and Storage*. Englewood Cliffs NJ: Prentice-Hall. ISBN 0-13-200809-2.

- Wilson, Stephen G. (1996). *Digital Modulation and Coding*. Englewood Cliffs NJ: Prentice-Hall. ISBN 0-13-210071-1.
- “Error Correction Code in Single Level Cell NAND Flash memories” 16 February 2007
- “Error Correction Code in NAND Flash memories” 29 November 2004
- Observations on Errors, Corrections, & Trust of Dependent Systems, by James Hamilton, February 26, 2012

## 2.13 External links

- Morelos-Zaragoza, Robert (2004). “The Error Correcting Codes (ECC) Page”. Retrieved 2006-03-05.

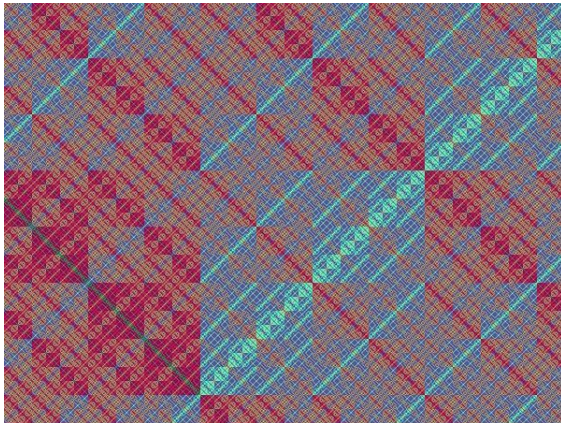
## Chapter 3

# List of algebraic coding theory topics

This is a list of **algebraic coding theory** topics.

## Chapter 4

# Coding theory



*A two-dimensional visualisation of the Hamming distance, a critical measure in coding theory.*

**Coding theory** is the study of the properties of codes and their respective fitness for specific applications. Codes are used for data compression, cryptography, error-correction, and networking. Codes are studied by various scientific disciplines—such as information theory, electrical engineering, mathematics, linguistics, and computer science—for the purpose of designing efficient and reliable data transmission methods. This typically involves the removal of redundancy and the correction or detection of errors in the transmitted data.

There are four types of coding:<sup>[1]</sup>

1. Data compression (or, *source coding*)
2. Error correction (or *channel coding*)
3. Cryptographic coding
4. Line coding

Data compression attempts to compress the data from a source in order to transmit it more efficiently. For example, Zip data compression makes data files smaller to reduce Internet traffic. Data compression and error correction may be studied in combination.

Error correction adds extra data bits to make the transmission of data more robust to disturbances present on the transmission channel. The ordinary user may not be

aware of many applications using error correction. A typical music CD uses the Reed-Solomon code to correct for scratches and dust. In this application the transmission channel is the CD itself. Cell phones also use coding techniques to correct for the fading and noise of high frequency radio transmission. Data modems, telephone transmissions, and NASA all employ channel coding techniques to get the bits through, for example the turbo code and LDPC codes.

### 4.1 History of coding theory

In 1948, Claude Shannon published "A Mathematical Theory of Communication", an article in two parts in the July and October issues of the *Bell System Technical Journal*. This work focuses on the problem of how best to encode the information a sender wants to transmit. In this fundamental work he used tools in probability theory, developed by Norbert Wiener, which were in their nascent stages of being applied to communication theory at that time. Shannon developed information entropy as a measure for the uncertainty in a message while essentially inventing the field of information theory.

The binary Golay code was developed in 1949. It is an error-correcting code capable of correcting up to three errors in each 24-bit word, and detecting a fourth.

Richard Hamming won the Turing Award in 1968 for his work at Bell Labs in numerical methods, automatic coding systems, and error-detecting and error-correcting codes. He invented the concepts known as Hamming codes, Hamming windows, Hamming numbers, and Hamming distance.

### 4.2 Source coding

Main article: Data compression

The aim of source coding is to take the source data and make it smaller.

### 4.2.1 Definition

Data can be seen as a **random variable**  $X : \Omega \rightarrow \mathcal{X}$ , where  $x \in \mathcal{X}$  appears with probability  $\mathbb{P}[X = x]$ .

Data are encoded by strings (words) over an **alphabet**  $\Sigma$ .

A code is a function

$C : \mathcal{X} \rightarrow \Sigma^*$  (or  $\Sigma^+$  if the empty string is not part of the alphabet).

$C(x)$  is the code word associated with  $x$ .

Length of the code word is written as

$l(C(x))$ .

Expected length of a code is

$$l(C) = \sum_{x \in \mathcal{X}} l(C(x)) \mathbb{P}[X = x]$$

The concatenation of code words  $C(x_1, \dots, x_k) = C(x_1)C(x_2)\dots C(x_k)$ .

The code word of the empty string is the empty string itself:

$$C(\epsilon) = \epsilon$$

### 4.2.2 Properties

1.  $C : \mathcal{X} \rightarrow \Sigma^*$  is **non-singular** if **injective**.
2.  $C : \mathcal{X}^* \rightarrow \Sigma^*$  is **uniquely decodable** if **injective**.
3.  $C : \mathcal{X} \rightarrow \Sigma^*$  is **instantaneous** if  $C(x_1)$  is not a prefix of  $C(x_2)$  (and vice versa).

### 4.2.3 Principle

**Entropy** of a source is the measure of information. Basically, source codes try to reduce the redundancy present in the source, and represent the source with fewer bits that carry more information.

Data compression which explicitly tries to minimize the average length of messages according to a particular assumed probability model is called **entropy encoding**.

Various techniques used by source coding schemes try to achieve the limit of Entropy of the source.  $C(x) \geq H(x)$ , where  $H(x)$  is entropy of source (bitrate), and  $C(x)$  is the bitrate after compression. In particular, no source coding scheme can be better than the entropy of the source.

### 4.2.4 Example

**Facsimile** transmission uses a simple **run length code**. Source coding removes all data superfluous to the need of the transmitter, decreasing the bandwidth required for transmission.

## 4.3 Channel coding

Main article: **Forward error correction**

The purpose of channel coding theory is to find codes which transmit quickly, contain many valid **code words** and can correct or at least **detect** many errors. While not mutually exclusive, performance in these areas is a trade off. So, different codes are optimal for different applications. The needed properties of this code mainly depend on the probability of errors happening during transmission. In a typical CD, the impairment is mainly dust or scratches. Thus codes are used in an interleaved manner. The data is spread out over the disk.

Although not a very good code, a simple repeat code can serve as an understandable example. Suppose we take a block of data bits (representing sound) and send it three times. At the receiver we will examine the three repetitions bit by bit and take a majority vote. The twist on this is that we don't merely send the bits in order. We interleave them. The block of data bits is first divided into 4 smaller blocks. Then we cycle through the block and send one bit from the first, then the second, etc. This is done three times to spread the data out over the surface of the disk. In the context of the simple repeat code, this may not appear effective. However, there are more powerful codes known which are very effective at correcting the "burst" error of a scratch or a dust spot when this interleaving technique is used.

Other codes are more appropriate for different applications. Deep space communications are limited by the **thermal noise** of the receiver which is more of a continuous nature than a bursty nature. Likewise, narrowband modems are limited by the noise, present in the telephone network and also modeled better as a continuous disturbance. Cell phones are subject to rapid fading. The high frequencies used can cause rapid fading of the signal even if the receiver is moved a few inches. Again there are a class of channel codes that are designed to combat fading.

### 4.3.1 Linear codes

Main article: **Linear code**

The term **algebraic coding theory** denotes the sub-field of coding theory where the properties of codes are expressed in algebraic terms and then further researched.

Algebraic coding theory is basically divided into two major types of codes:

1. Linear block codes
2. Convolutional codes

It analyzes the following three properties of a code – mainly:

- code word length
- total number of valid code words
- the minimum distance between two valid code words, using mainly the Hamming distance, sometimes also other distances like the Lee distance

### Linear block codes

Main article: [Block code](#)

Linear block codes have the property of **linearity**, i.e. the sum of any two codewords is also a code word, and they are applied to the source bits in blocks, hence the name linear block codes. There are block codes that are not linear, but it is difficult to prove that a code is a good one without this property.<sup>[2]</sup>

Linear block codes are summarized by their symbol alphabets (e.g., binary or ternary) and parameters  $(n, m, d_{min})$ <sup>[3]</sup> where

1.  $n$  is the length of the codeword, in symbols,
2.  $m$  is the number of source symbols that will be used for encoding at once,
3.  $d_{min}$  is the minimum hamming distance for the code.

There are many types of linear block codes, such as

1. Cyclic codes (e.g., Hamming codes)
2. Repetition codes
3. Parity codes
4. Polynomial codes (e.g., BCH codes)
5. Reed–Solomon codes
6. Algebraic geometric codes
7. Reed–Muller codes
8. Perfect codes

Block codes are tied to the **sphere packing** problem, which has received some attention over the years. In two dimensions, it is easy to visualize. Take a bunch of pennies flat on the table and push them together. The result is a hexagon pattern like a bee's nest. But block codes rely on more dimensions which cannot easily be visualized. The powerful (24,12) Golay code used in deep space communications uses 24 dimensions. If used as a binary code (which it usually is) the dimensions refer to the length of the codeword as defined above.

The theory of coding uses the  $N$ -dimensional sphere model. For example, how many pennies can be packed

into a circle on a tabletop, or in 3 dimensions, how many marbles can be packed into a globe. Other considerations enter the choice of a code. For example, hexagon packing into the constraint of a rectangular box will leave empty space at the corners. As the dimensions get larger, the percentage of empty space grows smaller. But at certain dimensions, the packing uses all the space and these codes are the so-called “perfect” codes. The only nontrivial and useful perfect codes are the distance-3 Hamming codes with parameters satisfying  $(2^r - 1, 2^r - 1 - r, 3)$ , and the [23,12,7] binary and [11,6,5] ternary Golay codes.<sup>[2][3]</sup>

Another code property is the number of neighbors that a single codeword may have.<sup>[4]</sup> Again, consider pennies as an example. First we pack the pennies in a rectangular grid. Each penny will have 4 near neighbors (and 4 at the corners which are farther away). In a hexagon, each penny will have 6 near neighbors. When we increase the dimensions, the number of near neighbors increases very rapidly. The result is the number of ways for noise to make the receiver choose a neighbor (hence an error) grows as well. This is a fundamental limitation of block codes, and indeed all codes. It may be harder to cause an error to a single neighbor, but the number of neighbors can be large enough so the total error probability actually suffers.<sup>[4]</sup>

Properties of linear block codes are used in many applications. For example, the syndrome-coset uniqueness property of linear block codes is used in trellis shaping,<sup>[5]</sup> one of the best known **shaping codes**. This same property is used in sensor networks for distributed source coding

### Convolutional codes

Main article: [Convolutional code](#)

The idea behind a convolutional code is to make every codeword symbol be the weighted sum of the various input message symbols. This is like **convolution** used in LTI systems to find the output of a system, when you know the input and impulse response.

So we generally find the output of the system convolutional encoder, which is the convolution of the input bit, against the states of the convolution encoder, registers.

Fundamentally, convolutional codes do not offer more protection against noise than an equivalent block code. In many cases, they generally offer greater simplicity of implementation over a block code of equal power. The encoder is usually a simple circuit which has state memory and some feedback logic, normally XOR gates. The **decoder** can be implemented in software or firmware.

The **Viterbi algorithm** is the optimum algorithm used to decode convolutional codes. There are simplifications to reduce the computational load. They rely on searching only the most likely paths. Although not optimum, they have generally been found to give good results in the lower



noise environments.

Convolutional codes are used in voiceband modems (V.32, V.17, V.34) and in GSM mobile phones, as well as satellite and military communication devices.

## 4.4 Cryptographical coding

Main article: [Cryptography](#)

**Cryptography** or cryptographic coding is the practice and study of techniques for **secure communication** in the presence of third parties (called **adversaries**).<sup>[6]</sup> More generally, it is about constructing and analyzing **protocols** that block adversaries;<sup>[7]</sup> various aspects in **information security** such as **data confidentiality**, **data integrity**, **authentication**, and **non-repudiation**<sup>[8]</sup> are central to modern cryptography. Modern cryptography exists at the intersection of the disciplines of **mathematics**, **computer science**, and **electrical engineering**. Applications of cryptography include **ATM cards**, **computer passwords**, and **electronic commerce**.

Cryptography prior to the modern age was effectively synonymous with **encryption**, the conversion of information from a readable state to apparent **nonsense**. The originator of an encrypted message shared the decoding technique needed to recover the original information only with intended recipients, thereby precluding unwanted persons from doing the same. Since **World War I** and the advent of the **computer**, the methods used to carry out cryptology have become increasingly complex and its application more widespread.

Modern cryptography is heavily based on mathematical theory and computer science practice; cryptographic algorithms are designed around **computational hardness assumptions**, making such algorithms hard to break in practice by any adversary. It is theoretically possible to break such a system, but it is infeasible to do so by any known practical means. These schemes are therefore termed **computationally secure**; theoretical advances, e.g., improvements in **integer factorization** algorithms, and faster computing technology require these solutions to be continually adapted. There exist **information-theoretically secure** schemes that provably cannot be broken even with unlimited computing power—an example is the **one-time pad**—but these schemes are more difficult to implement than the best theoretically breakable but computationally secure mechanisms.

## 4.5 Line coding

Main article: [Line code](#)

A **line code** (also called digital baseband modulation or

digital baseband transmission method) is a **code** chosen for use within a **communications system** for **baseband transmission** purposes. Line coding is often used for digital data transport.

Line coding consists of representing the **digital signal** to be transported by an amplitude- and time-discrete signal that is optimally tuned for the specific properties of the physical channel (and of the receiving equipment). The **waveform** pattern of voltage or current used to represent the 1s and 0s of a digital data on a transmission link is called **line encoding**. The common types of line encoding are **unipolar**, **polar**, **bipolar**, and **Manchester encoding**.

## 4.6 Other applications of coding theory

Another concern of coding theory is designing codes that help **synchronization**. A code may be designed so that a **phase shift** can be easily detected and corrected and that multiple signals can be sent on the same channel.

Another application of codes, used in some mobile phone systems, is **code-division multiple access** (CDMA). Each phone is assigned a code sequence that is approximately uncorrelated with the codes of other phones. When transmitting, the code word is used to modulate the data bits representing the voice message. At the receiver, a demodulation process is performed to recover the data. The properties of this class of codes allow many users (with different codes) to use the same radio channel at the same time. To the receiver, the signals of other users will appear to the demodulator only as a low-level noise.

Another general class of codes are the **automatic repeat-request** (ARQ) codes. In these codes the sender adds redundancy to each message for error checking, usually by adding check bits. If the check bits are not consistent with the rest of the message when it arrives, the receiver will ask the sender to retransmit the message. All but the simplest **wide area network** protocols use ARQ. Common protocols include **SDLC** (IBM), **TCP** (Internet), **X.25** (International) and many others. There is an extensive field of research on this topic because of the problem of matching a rejected packet against a new packet. Is it a new one or is it a retransmission? Typically numbering schemes are used, as in TCP. "RFC793". *RFCs*. Internet Engineering Task Force (IETF). September 1981.

### 4.6.1 Group testing

**Group testing** uses codes in a different way. Consider a large group of items in which a very few are different in a particular way (e.g., defective products or infected test subjects). The idea of group testing is to determine which items are "different" by using as few tests as possible. The origin of the problem has its roots in the **Second World**

War when the United States Army Air Forces needed to test its soldiers for syphilis. It originated from a groundbreaking paper by Robert Dorfman.

### 4.6.2 Analog coding

Information is encoded analogously in the neural networks of brains, in analog signal processing, and analog electronics. Aspects of analog coding include analog error correction,<sup>[9]</sup> analog data compression<sup>[10]</sup> and analog encryption.<sup>[11]</sup>

## 4.7 Neural coding

Neural coding is a neuroscience-related field concerned with how sensory and other information is represented in the brain by networks of neurons. The main goal of studying neural coding is to characterize the relationship between the stimulus and the individual or ensemble neuronal responses and the relationship among electrical activity of the neurons in the ensemble.<sup>[12]</sup> It is thought that neurons can encode both digital and analog information,<sup>[13]</sup> and that neurons follow the principles of information theory and compress information,<sup>[14]</sup> and detect and correct<sup>[15]</sup> errors in the signals that are sent throughout the brain and wider nervous system.


## 4.8 See also

- Coding gain
- Covering code
- Error-correcting code
- Group testing
- Hamming distance, Hamming weight
- Information theory
- Lee distance
- Spatial coding and MIMO in multiple antenna research
  - Spatial diversity coding is spatial coding that transmits replicas of the information signal along different spatial paths, so as to increase the reliability of the data transmission.
  - Spatial interference cancellation coding
  - Spatial multiplex coding
- Timeline of information theory, data compression, and error correcting codes
- List of algebraic coding theory topics
- Folded Reed–Solomon codes

## 4.9 Notes

- [1] James Irvine; David Harle (2002). “2.4.4 Types of Coding”. *Data Communications and Networks*. p. 18. There are four types of coding
- [2] Terras, Audrey (1999). *Fourier Analysis on Finite Groups and Applications*. Cambridge University Press. ISBN 0-521-45718-1.
- [3] Blahut, Richard E. (2003). *Algebraic Codes for Data Transmission*. Cambridge University Press. ISBN 0-521-55374-1.
- [4] Christian Schlegel; Lance Pérez (2004). *Trellis and turbo coding*. Wiley-IEEE. p. 73. ISBN 978-0-471-22755-7.
- [5] Forney, G.D., Jr. (March 1992). “Trellis shaping”. *IEEE Transactions on Information Theory*. **38** (2 Pt 2): 281–300. doi:10.1109/18.1196870.
- [6] Rivest, Ronald L. (1990). “Cryptology”. In J. Van Leeuwen. *Handbook of Theoretical Computer Science*. 1. Elsevier.
- [7] Bellare, Mihir; Rogaway, Phillip (21 September 2005). “Introduction”. *Introduction to Modern Cryptography*. p. 10.
- [8] Menezes, A. J.; van Oorschot, P. C.; Vanstone, S. A. *Handbook of Applied Cryptography*. ISBN 0-8493-8523-7.
- [9] Chen, Brian; Wornell, Gregory W. (July 1998). “Analog Error-Correcting Codes Based on Chaotic Dynamical Systems” (PDF). *IEEE Transactions on Communications*. **46** (7): 881–890. CiteSeerX 10.1.1.30.4093. doi:10.1109/26.701312.
- [10] Hvala, Franc Novak Bojan; Klavžar, Sandi (1999). “On Analog Signature Analysis”. *Proceedings of the conference on Design, automation and test in Europe*. CiteSeerX 10.1.1.142.5853. ISBN 1-58113-121-6.
- [11] Shujun Li; Chengqing Li; Kwok-Tung Lo; Guanrong Chen (April 2008). “Cryptanalyzing an Encryption Scheme Based on Blind Source Separation”. *IEEE Transactions on Circuits and Systems I*. **55** (4): 1055–63. doi:10.1109/TCSI.2008.916540.
- [12] Brown EN, Kass RE, Mitra PP (May 2004). “Multiple neural spike train data analysis: state-of-the-art and future challenges”. *Nat. Neurosci.* **7** (5): 456–61. doi:10.1038/nn1228. PMID 15114358.
- [13] Thorpe, S.J. (1990). “Spike arrival times: A highly efficient coding scheme for neural networks” (PDF). In Eckmiller, R.; Hartmann, G.; Hauske, G. *Parallel processing in neural systems and computers* (PDF). North-Holland. pp. 91–94. ISBN 978-0-444-88390-2. Retrieved 30 June 2013.
- [14] Gedeon, T.; Parker, A.E.; Dimitrov, A.G. (Spring 2002). “Information Distortion and Neural Coding”. *Canadian Applied Mathematics Quarterly*. **10** (1): 10. CiteSeerX 10.1.1.5.6365.



- [15] Stiber, M. (July 2005). “Spike timing precision and neural error correction: local behavior”. *Neural Computation*. **17** (7): 1577–1601. arXiv:q-bio/0501021 . doi:10.1162/0899766053723069.

## 4.10 References

- Elwyn R. Berlekamp (2014), *Algebraic Coding Theory*, World Scientific Publishing (revised edition), ISBN 978-9-81463-589-9.
- MacKay, David J. C.. *Information Theory, Inference, and Learning Algorithms* Cambridge: Cambridge University Press, 2003. ISBN 0-521-64298-1
- Vera Pless (1982), *Introduction to the Theory of Error-Correcting Codes*, John Wiley & Sons, Inc., ISBN 0-471-08684-3.
- Randy Yates, *A Coding Theory Tutorial*.

# Chapter 5

## Alternant code

In coding theory, **alternant codes** form a class of parameterised **error-correcting codes** which generalise the **BCH codes**.

### 5.1 Definition

An *alternant code* over  $\text{GF}(q)$  of length  $n$  is defined by a parity check matrix  $H$  of **alternant** form  $H_{i,j} = \alpha_j^i y_i$ , where the  $\alpha_j$  are distinct elements of the extension  $\text{GF}(q^m)$ , the  $y_i$  are further non-zero parameters again in the extension  $\text{GF}(q^m)$  and the indices range as  $i$  from 0 to  $\delta - 1$ ,  $j$  from 1 to  $n$ .

### 5.2 Properties

The parameters of this alternant code are length  $n$ , dimension  $\geq n - m\delta$  and minimum distance  $\geq \delta + 1$ . There exist long alternant codes which meet the **Gilbert-Varshamov bound**.

The class of alternant codes includes

- **BCH codes**
- **Goppa codes**
- **Srivastava codes**

### 5.3 References

- **F.J. MacWilliams**; **N.J.A. Sloane** (1977). *The Theory of Error-Correcting Codes*. North-Holland. pp. 332–338. ISBN 0-444-85193-3.

## Chapter 6

# Arbitrarily varying channel

An **arbitrarily varying channel (AVC)** is a communication channel model used in coding theory, and was first introduced by Blackwell, Breiman, and Thomasian. This particular channel has unknown parameters that can change over time and these changes may not have a uniform pattern during the transmission of a codeword.  $n$  uses of this channel can be described using a stochastic matrix  $W^n : X^n \times S^n \rightarrow Y^n$ , where  $X$  is the input alphabet,  $Y$  is the output alphabet, and  $W^n(y|x, s)$  is the probability over a given set of states  $S$ , that the transmitted input  $x = (x_1, \dots, x_n)$  leads to the received output  $y = (y_1, \dots, y_n)$ . The state  $s_i$  in set  $S$  can vary arbitrarily at each time unit  $i$ . This channel was developed as an alternative to Shannon's Binary Symmetric Channel (BSC), where the entire nature of the channel is known, to be more realistic to actual network channel situations.

## 6.1 Capacities and associated proofs

### 6.1.1 Capacity of deterministic AVCs

An AVC's capacity can vary depending on the certain parameters.

$R$  is an achievable rate for a deterministic AVC code if it is larger than 0, and if for every positive  $\varepsilon$  and  $\delta$ , and very large  $n$ , length- $n$  block codes exist that satisfy the following equations:  $\frac{1}{n} \log N > R - \delta$  and  $\max_{s \in S^n} \bar{e}(s) \leq \varepsilon$ , where  $N$  is the highest value in  $Y$  and where  $\bar{e}(s)$  is the average probability of error for a state sequence  $s$ . The largest rate  $R$  represents the capacity of the AVC, denoted by  $c$ .

As you can see, the only useful situations are when the capacity of the AVC is greater than 0, because then the channel can transmit a guaranteed amount of data  $\leq c$  without errors. So we start out with a theorem that shows when  $c$  is positive in an AVC and the theorems discussed afterward will narrow down the range of  $c$  for different circumstances.

Before stating Theorem 1, a few definitions need to be addressed:

- An AVC is *symmetric* if  $\sum_{s \in S} W(y|x, s)U(s|x') = \sum_{s \in S} W(y|x', s)U(s|x)$  for every  $(x, x', y, s)$ , where  $x, x' \in X$ ,  $y \in Y$ , and  $U(s|x)$  is a channel function  $U : X \rightarrow S$ .
- $X_r$ ,  $S_r$ , and  $Y_r$  are all random variables in sets  $X$ ,  $S$ , and  $Y$  respectively.
- $P_{X_r}(x)$  is equal to the probability that the random variable  $X_r$  is equal to  $x$ .
- $P_{S_r}(s)$  is equal to the probability that the random variable  $S_r$  is equal to  $s$ .
- $P_{X_r S_r Y_r}$  is the combined probability mass function (pmf) of  $P_{X_r}(x)$ ,  $P_{S_r}(s)$ , and  $W(y|x, s)$ .  $P_{X_r S_r Y_r}$  is defined formally as  $P_{X_r S_r Y_r}(x, s, y) = P_{X_r}(x)P_{S_r}(s)W(y|x, s)$ .
- $H(X_r)$  is the entropy of  $X_r$ .
- $H(X_r|Y_r)$  is equal to the average probability that  $X_r$  will be a certain value based on all the values  $Y_r$  could possibly be equal to.
- $I(X_r \wedge Y_r)$  is the mutual information of  $X_r$  and  $Y_r$ , and is equal to  $H(X_r) - H(X_r|Y_r)$ .
- $I(P) = \min_{Y_r} I(X_r \wedge Y_r)$ , where the minimum is over all random variables  $Y_r$  such that  $X_r$ ,  $S_r$ , and  $Y_r$  are distributed in the form of  $P_{X_r S_r Y_r}$ .

**Theorem 1:**  $c > 0$  if and only if the AVC is not symmetric. If  $c > 0$ , then  $c = \max_P I(P)$ .

*Proof of 1st part for symmetry:* If we can prove that  $I(P)$  is positive when the AVC is not symmetric, and then prove that  $c = \max_P I(P)$ , we will be able to prove Theorem 1. Assume  $I(P)$  were equal to 0. From the definition of  $I(P)$ , this would make  $X_r$  and  $Y_r$  independent random variables, for some  $S_r$ , because this would mean that neither random variable's entropy would rely on the other random variable's value. By using equation  $P_{X_r S_r Y_r}$ , (and remembering  $P_{X_r} = P$ ), we can get,

$$\begin{aligned}
P_{Y_r}(y) &= \sum_{x \in X} \sum_{s \in S} P(x) P_{S_r}(s) W(y|x, s) \\
&\equiv (\text{ since } X_r \text{ and } Y_r \text{ are independent random variables, } W(y|x, s) = W'(y|s) \text{ for some } W') \\
P_{Y_r}(y) &= \sum_{x \in X} \sum_{s \in S} P(x) P_{S_r}(s) W'(y|s) \\
&\equiv (\text{ because only } P(x) \text{ depends on } x \text{ now } ) \\
P_{Y_r}(y) &= \sum_{s \in S} P_{S_r}(s) W'(y|s) \left[ \sum_{x \in X} P(x) \right] \\
&\equiv (\text{ because } \sum_{x \in X} P(x) = 1 ) \\
P_{Y_r}(y) &= \sum_{s \in S} P_{S_r}(s) W'(y|s)
\end{aligned}$$

So now we have a probability distribution on  $Y_r$  that is independent of  $X_r$ . So now the definition of a symmetric AVC can be rewritten as follows:  $\sum_{s \in S} W'(y|s) P_{S_r}(s) = \sum_{s \in S} W'(y|s) P_{S_r}(s)$  since  $U(s|x)$  and  $W(y|x, s)$  are both functions based on  $x$ , they have been replaced with functions based on  $s$  and  $y$  only. As you can see, both sides are now equal to the  $P_{Y_r}(y)$  we calculated earlier, so the AVC is indeed symmetric when  $I(P)$  is equal to 0. Therefore,  $I(P)$  can only be positive if the AVC is not symmetric.

*Proof of second part for capacity:* See the paper “The capacity of the arbitrarily varying channel revisited: positivity, constraints,” referenced below for full proof.

### 6.1.2 Capacity of AVCs with input and state constraints

The next theorem will deal with the capacity for AVCs with input and/or state constraints. These constraints help to decrease the very large range of possibilities for transmission and error on an AVC, making it a bit easier to see how the AVC behaves.

Before we go on to Theorem 2, we need to define a few definitions and lemmas:

For such AVCs, there exists:

- An input constraint  $\Gamma$  based on the equation  $g(x) = \frac{1}{n} \sum_{i=1}^n g(x_i)$ , where  $x \in X$  and  $x = (x_1, \dots, x_n)$ .
- A state constraint  $\Lambda$ , based on the equation  $l(s) = \frac{1}{n} \sum_{i=1}^n l(s_i)$ , where  $s \in X$  and  $s = (s_1, \dots, s_n)$ .

$$- \Lambda_0(P) = \min_{x \in X, s \in S} \sum P(x) l(s)$$

-  $I(P, \Lambda)$  is very similar to  $I(P)$  equation mentioned previously,  $I(P, \Lambda) = \min_{Y_r} I(X_r \wedge Y_r)$ , but now any state  $s$  or  $S_r$  in the equation must follow the  $l(s) \leq \Lambda$  state restriction.

Assume  $g(x)$  is a given non-negative-valued function on  $X$  and  $l(s)$  is a given non-negative-valued function on  $S$  and that the minimum values for both is 0. In the literature I have read on this subject, the exact definitions of both  $g(x)$  and  $l(s)$  (for one variable  $x_i$ ) is never described formally. The usefulness of the input constraint  $\Gamma$  and the state constraint  $\Lambda$  will be based on these equations.

For AVCs with input and/or state constraints, the rate  $R$  is now limited to codewords of format  $x_1, \dots, x_N$  that satisfy  $g(x_i) \leq \Gamma$ , and now the state  $s$  is limited to all states that satisfy  $l(s) \leq \Lambda$ . The largest rate is still considered the capacity of the AVC, and is now denoted as  $c(\Gamma, \Lambda)$ .

**Lemma 1:** Any codes where  $\Lambda$  is greater than  $\Lambda_0(P)$  cannot be considered “good” codes, because those kinds of codes have a maximum average probability of error greater than or equal to  $\frac{N-1}{2N} - \frac{1}{n} \frac{l_{max}^2}{n(\Lambda - \Lambda_0(P))^2}$ , where  $l_{max}$  is the maximum value of  $l(s)$ . This isn't a good maximum average error probability because it is fairly large,  $\frac{N-1}{2N}$  is close to  $\frac{1}{2}$ , and the other part of the equation will be very small since the  $(\Lambda - \Lambda_0(P))$  value is squared, and  $\Lambda$  is set to be larger than  $\Lambda_0(P)$ . Therefore, it would be very unlikely to receive a codeword without error. This is why the  $\Lambda_0(P)$  condition is present in Theorem 2.

**Theorem 2:** Given a positive  $\Lambda$  and arbitrarily small  $\alpha > 0$ ,  $\beta > 0$ ,  $\delta > 0$ , for any block length  $n \geq n_0$  and for any type  $P$  with conditions  $\Lambda_0(P) \geq \Lambda + \alpha$  and  $\min_{x \in X} P(x) \geq \beta$ , and where  $P_{X_r} = P$ , there exists a code with codewords  $x_1, \dots, x_N$ , each of type  $P$ , that satisfy the following equations:  $\frac{1}{n} \log N > I(P, \Lambda) - \delta$ ,  $\max_{l(s) \leq \Lambda} \bar{e}(s) \leq \exp(-n\gamma)$ , and where positive  $n_0$  and  $\gamma$  depend only on  $\alpha$ ,  $\beta$ ,  $\delta$ , and the given AVC.

*Proof of Theorem 2:* See the paper “The capacity of the arbitrarily varying channel revisited: positivity, constraints,” referenced below for full proof.

### 6.1.3 Capacity of randomized AVCs

The next theorem will be for AVCs with randomized code. For such AVCs the code is a random variable with values from a family of length- $n$  block codes, and these codes are not allowed to depend/rely on the actual value of the codeword. These codes have the same maximum and average error probability value for any channel because of its random nature. These types of codes also

help to make certain properties of the AVC more clear.

Before we go on to Theorem 3, we need to define a couple important terms first:

$W_{\zeta}(y|x) = \sum_{s \in S} W(y|x, s) P_{S_r}(s)$  is very similar

to the  $I(P)$  equation mentioned previously,  $I(P, \zeta) = \min_{Y_r} I(X_r \wedge Y_r)$ , but now the pmf  $P_{S_r}(s)$  is added to the equation, making the minimum of  $I(P, \zeta)$  based a new form of  $P_{X_r S_r Y_r}$ , where  $W_{\zeta}(y|x)$  replaces  $W(y|x, s)$ .

**Theorem 3:** The capacity for randomized codes of the AVC is  $c = \max_P I(P, \zeta)$ .

*Proof of Theorem 3:* See paper “The Capacities of Certain Channel Classes Under Random Coding” referenced below for full proof.

- Lapidith, A. and Narayan, P., “Reliable communication under channel uncertainty,” <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=720535&isnumber=15554>

## 6.2 See also

- Binary symmetric channel
- Binary erasure channel
- Z-channel (information theory)
- Channel model
- Information theory
- Coding theory

## 6.3 References

- Ahlswede, Rudolf and Blinovsky, Vladimir, “Classical Capacity of Classical-Quantum Arbitrarily Varying Channels,” <http://ieeexplore.ieee.org.gate.lib.buffalo.edu/stamp/stamp.jsp?tp=&arnumber=4069128>
- Blackwell, David, Breiman, Leo, and Thomasian, A. J., “The Capacities of Certain Channel Classes Under Random Coding,” <http://www.jstor.org/stable/2237566>
- Csiszar, I. and Narayan, P., “Arbitrarily varying channels with constrained inputs and states,” <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=2598&isnumber=154>
- Csiszar, I. and Narayan, P., “Capacity and Decoding Rules for Classes of Arbitrarily Varying Channels,” <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=32153&isnumber=139>
- Csiszar, I. and Narayan, P., “The capacity of the arbitrarily varying channel revisited: positivity, constraints,” <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=2627&isnumber=155>

# Chapter 7

## Bar product

In information theory, the **bar product** of two linear codes  $C_2 \subseteq C_1$  is defined as

$$C_1 | C_2 = \{(c_1 | c_1 + c_2) : c_1 \in C_1, c_2 \in C_2\},$$

where  $(a | b)$  denotes the concatenation of  $a$  and  $b$ . If the code words in  $C_1$  are of length  $n$ , then the code words in  $C_1 | C_2$  are of length  $2n$ .

The bar product is an especially convenient way of expressing the **Reed–Muller**  $RM(d, r)$  code in terms of the Reed–Muller codes  $RM(d-1, r)$  and  $RM(d-1, r-1)$ .

The bar product is also referred to as the  $|u|u+v|$  construction<sup>[1]</sup> or  $(u|u+v)$  construction.<sup>[2]</sup>

**Proof**

For all  $c_1 \in C_1$ ,

$$(c_1 | c_1 + 0) \in C_1 | C_2$$

which has weight  $2w(c_1)$ . Equally

$$(0 | c_2) \in C_1 | C_2$$

for all  $c_2 \in C_2$  and has weight  $w(c_2)$ . So minimising over  $c_1 \in C_1, c_2 \in C_2$  we have

$$w(C_1 | C_2) \leq \min\{2w(C_1), w(C_2)\}$$

Now let  $c_1 \in C_1$  and  $c_2 \in C_2$ , not both zero. If  $c_2 \neq 0$  then:

$$\begin{aligned} w(c_1 | c_1 + c_2) &= w(c_1) + w(c_1 + c_2) \\ &\geq w(c_1 + c_1 + c_2) \\ &= w(c_2) \\ &\geq w(C_2) \end{aligned}$$

If  $c_2 = 0$  then

$$\begin{aligned} w(c_1 | c_1 + c_2) &= 2w(c_1) \\ &\geq 2w(C_1) \end{aligned}$$

so

$$w(C_1 | C_2) \geq \min\{2w(C_1), w(C_2)\}$$

## 7.1 Properties

### 7.1.1 Rank

The **rank** of the bar product is the sum of the two ranks:

$$\text{rank}(C_1 | C_2) = \text{rank}(C_1) + \text{rank}(C_2)$$

**Proof**

Let  $\{x_1, \dots, x_k\}$  be a basis for  $C_1$  and let  $\{y_1, \dots, y_l\}$  be a basis for  $C_2$ . Then the set

$$\{(x_i | x_i) | 1 \leq i \leq k\} \cup \{(0 | y_j) | 1 \leq j \leq l\}$$

is a basis for the bar product  $C_1 | C_2$ .

### 7.1.2 Hamming weight

The **Hamming weight**  $w$  of the bar product is the lesser of (a) twice the weight of  $C_1$ , and (b) the weight of  $C_2$ :

$$w(C_1 | C_2) = \min\{2w(C_1), w(C_2)\}.$$

## 7.2 See also

- **Reed–Muller code**

## 7.3 References

- [1] F.J. MacWilliams; N.J.A. Sloane (1977). *The Theory of Error-Correcting Codes*. North-Holland. p. 76. ISBN 0-444-85193-3.
- [2] J.H. van Lint (1992). *Introduction to Coding Theory*. GTM **86** (2nd ed.). Springer-Verlag. p. 47. ISBN 3-540-54894-7.

# Chapter 8

## BCH code

In coding theory, the **BCH codes** form a class of cyclic error-correcting codes that are constructed using finite fields. BCH codes were invented in 1959 by French mathematician Alexis Hocquenghem, and independently in 1960 by Raj Bose and D. K. Ray-Chaudhuri.<sup>[1][2][3]</sup> The acronym *BCH* comprises the initials of these inventors' surnames (mistakenly, in the case of Ray-Chaudhuri).

One of the key features of BCH codes is that during code design, there is a precise control over the number of symbol errors correctable by the code. In particular, it is possible to design binary BCH codes that can correct multiple bit errors. Another advantage of BCH codes is the ease with which they can be decoded, namely, via an algebraic method known as syndrome decoding. This simplifies the design of the decoder for these codes, using small low-power electronic hardware.

BCH codes are used in applications such as satellite communications,<sup>[4]</sup> compact disc players, DVDs, disk drives, solid-state drives<sup>[5]</sup> and two-dimensional bar codes.

### 8.1 Definition and illustration

#### 8.1.1 Primitive narrow-sense BCH codes

Given a prime power  $q$  and positive integers  $m$  and  $d$  with  $d \leq q^m - 1$ , a primitive narrow-sense BCH code over the finite field  $GF(q)$  with code length  $n = q^m - 1$  and distance at least  $d$  is constructed by the following method.

Let  $\alpha$  be a primitive element of  $GF(q^m)$ . For any positive integer  $i$ , let  $m_i(x)$  be the minimal polynomial of  $\alpha^i$  over  $GF(q)$ . The generator polynomial of the BCH code is defined as the least common multiple  $g(x) = \text{lcm}(m_1(x), \dots, m_{d-1}(x))$ . It can be seen that  $g(x)$  is a polynomial with coefficients in  $GF(q)$  and divides  $x^n - 1$ . Therefore, the polynomial code defined by  $g(x)$  is a cyclic code.

#### Example

Let  $q=2$  and  $m=4$  (therefore  $n=15$ ). We will consider different values of  $d$ . There is a primitive root  $\alpha$  in  $GF(16)$

satisfying

its minimal polynomial over  $GF(2)$  is

$$m_1(x) = x^4 + x + 1.$$

The minimal polynomials of the first fourteen powers of  $\alpha$  are

$$m_1(x) = m_2(x) = m_4(x) = m_8(x) = x^4 + x + 1,$$

$$m_3(x) = m_6(x) = m_9(x) = m_{12}(x) = x^4 + x^3 + x^2 + x + 1,$$

$$m_5(x) = m_{10}(x) = x^2 + x + 1,$$

$$m_7(x) = m_{11}(x) = m_{13}(x) = m_{14}(x) = x^4 + x^3 + 1.$$

The BCH code with  $d = 2, 3$  has generator polynomial

$$g(x) = m_1(x) = x^4 + x + 1.$$

It has minimal Hamming distance at least 3 and corrects up to one error. Since the generator polynomial is of degree 4, this code has 11 data bits and 4 checksum bits.

The BCH code with  $d = 4, 5$  has generator polynomial

$$g(x) = \text{lcm}(m_1(x), m_3(x)) = (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1) = x^8 + x^7 + x^6 + x^4 + 1.$$

It has minimal Hamming distance at least 5 and corrects up to two errors. Since the generator polynomial is of degree 8, this code has 7 data bits and 8 checksum bits.

The BCH code with  $d = 8$  and higher has generator polynomial

$$\begin{aligned} g(x) &= \text{lcm}(m_1(x), m_3(x), m_5(x), m_7(x)) \\ &= (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^2 + x + 1)(x^4 + x^3 + 1) \\ &= x^{14} + x^{13} + x^{12} + \dots + x^2 + x + 1. \end{aligned}$$

This code has minimal Hamming distance 15 and corrects 7 errors. It has 1 data bit and 14 checksum bits. In fact, this code has only two codewords: 000000000000000 and 111111111111111.



### 8.1.2 General BCH codes

General BCH codes differ from primitive narrow-sense BCH codes in two respects.

First, the requirement that  $\alpha$  be a primitive element of  $GF(q^m)$  can be relaxed. By relaxing this requirement, the code length changes from  $q^m - 1$  to  $\text{ord}(\alpha)$ , the order of the element  $\alpha$ .

Second, the consecutive roots of the generator polynomial may run from  $\alpha^c, \dots, \alpha^{c+d-2}$  instead of  $\alpha, \dots, \alpha^{d-1}$ .

**Definition.** Fix a finite field  $GF(q)$ , where  $q$  is a prime power. Choose positive integers  $m, n, d, c$  such that  $2 \leq d \leq n$ ,  $\gcd(n, q) = 1$ , and  $m$  is the multiplicative order of  $q$  modulo  $n$ .

As before, let  $\alpha$  be a primitive  $n$ th root of unity in  $GF(q^m)$ , and let  $m_i(x)$  be the minimal polynomial over  $GF(q)$  of  $\alpha^i$  for all  $i$ . The generator polynomial of the BCH code is defined as the least common multiple  $g(x) = \text{lcm}(m_c(x), \dots, m_{c+d-2}(x))$ .

**Note:** if  $n = q^m - 1$  as in the simplified definition, then  $\gcd(n, q) = 1$ , and the order of  $q$  modulo  $n$  is  $m$ . Therefore, the simplified definition is indeed a special case of the general one.

### 8.1.3 Special cases

- A BCH code with  $c = 1$  is called a *narrow-sense BCH code*.
- A BCH code with  $n = q^m - 1$  is called *primitive*.

The generator polynomial  $g(x)$  of a BCH code has coefficients from  $GF(q)$ . In general, a cyclic code over  $GF(q^p)$  with  $g(x)$  as the generator polynomial is called a BCH code over  $GF(q^p)$ . The BCH code over  $GF(q^m)$  with  $g(x)$  as the generator polynomial is called a **Reed–Solomon code**. In other words, a Reed–Solomon code is a BCH code where the decoder alphabet is the same as the channel alphabet.<sup>[6]</sup>

## 8.2 Properties

The generator polynomial of a BCH code has degree at most  $(d - 1)m$ . Moreover, if  $q = 2$  and  $c = 1$ , the generator polynomial has degree at most  $dm/2$ .

A BCH code has minimal Hamming distance at least  $d$ .

A BCH code is cyclic.

## 8.3 Encoding

## 8.4 Decoding

There are many algorithms for decoding BCH codes. The most common ones follow this general outline:

1. Calculate the syndromes  $s_j$  for the received vector
2. Determine the number of errors  $t$  and the error locator polynomial  $\Lambda(x)$  from the syndromes
3. Calculate the roots of the error location polynomial to find the error locations  $X_i$
4. Calculate the error values  $Y_i$  at those error locations
5. Correct the errors

During some of these steps, the decoding algorithm may determine that the received vector has too many errors and cannot be corrected. For example, if an appropriate value of  $t$  is not found, then the correction would fail. In a truncated (not primitive) code, an error location may be out of range. If the received vector has more errors than the code can correct, the decoder may unknowingly produce an apparently valid message that is not the one that was sent.

### 8.4.1 Calculate the syndromes

The received vector  $R$  is the sum of the correct codeword  $C$  and an unknown error vector  $E$ . The syndrome values are formed by considering  $R$  as a polynomial and evaluating it at  $\alpha^c, \dots, \alpha^{c+d-2}$ . Thus the syndromes are<sup>[7]</sup>

$$s_j = R(\alpha^j) = C(\alpha^j) + E(\alpha^j)$$

for  $j = c$  to  $c + d - 2$ . Since  $\alpha^j$  are the zeros of  $g(x)$ , of which  $C(x)$  is a multiple,  $C(\alpha^j) = 0$ . Examining the syndrome values thus isolates the error vector so one can begin to solve for it.

If there is no error,  $s_j = 0$  for all  $j$ . If the syndromes are all zero, then the decoding is done.

### 8.4.2 Calculate the error location polynomial

If there are nonzero syndromes, then there are errors. The decoder needs to figure out how many errors and the location of those errors.

If there is a single error, write this as  $E(x) = e x^i$ , where  $i$  is the location of the error and  $e$  is its magnitude. Then the first two syndromes are

$$s_c = e \alpha^{c i}$$

$$s_{c+1} = e \alpha^{(c+1)i} = \alpha^i s_c$$

so together they allow us to calculate  $e$  and provide some information about  $i$  (completely determining it in the case of Reed–Solomon codes).

If there are two or more errors,

$$E(x) = e_1 x^{i_1} + e_2 x^{i_2} + \dots$$

It is not immediately obvious how to begin solving the resulting syndromes for the unknowns  $e_k$  and  $i_k$ . First step is finding locator polynomial

$$\Lambda(x) = \prod_{j=1}^t (x \alpha^{i_j} - 1) \text{ compatible with computed syndromes and with minimal possible } t.$$

Two popular algorithms for this task are:

1. **Peterson–Gorenstein–Zierler algorithm**
2. **Berlekamp–Massey algorithm**

#### Peterson–Gorenstein–Zierler algorithm

Peterson’s algorithm is the step 2 of the generalized BCH decoding procedure. Peterson’s algorithm is used to calculate the error locator polynomial coefficients  $\lambda_1, \lambda_2, \dots, \lambda_v$  of a polynomial

$$\Lambda(x) = 1 + \lambda_1 x + \lambda_2 x^2 + \dots + \lambda_v x^v.$$

Now the procedure of the Peterson–Gorenstein–Zierler algorithm.<sup>[8]</sup> Expect we have at least  $2t$  syndromes  $s_c, \dots, s_{c+2t-1}$ . Let  $v = t$ .

- Start by generating the  $S_{v \times v}$  matrix with elements that are syndrome values

$$S_{v \times v} = \begin{bmatrix} s_c & s_{c+1} & \dots & s_{c+v-1} \\ s_{c+1} & s_{c+2} & \dots & s_{c+v} \\ \vdots & \vdots & \ddots & \vdots \\ s_{c+v-1} & s_{c+v} & \dots & s_{c+2v-2} \end{bmatrix}.$$

- Generate a  $c_{v \times 1}$  vector with elements

$$C_{v \times 1} = \begin{bmatrix} s_{c+v} \\ s_{c+v+1} \\ \vdots \\ s_{c+2v-1} \end{bmatrix}.$$

- Let  $\Lambda$  denote the unknown polynomial coefficients, which are given by

$$\Lambda_{v \times 1} = \begin{bmatrix} \lambda_v \\ \lambda_{v-1} \\ \vdots \\ \lambda_1 \end{bmatrix}.$$

- Form the matrix equation

$$S_{v \times v} \Lambda_{v \times 1} = -C_{v \times 1}.$$

- If the determinant of matrix  $S_{v \times v}$  is nonzero, then we can actually find an inverse of this matrix and solve for the values of unknown  $\Lambda$  values.

- If  $\det(S_{v \times v}) = 0$ , then follow

if  $v = 0$  then declare an empty error locator polynomial stop Peterson procedure. end set  $v \leftarrow v - 1$  continue from the beginning of Peterson’s decoding by making smaller  $S_{v \times v}$

- After you have values of  $\Lambda$ , you have with you the error locator polynomial.
- Stop Peterson procedure.

### 8.4.3 Factor error locator polynomial

Now that you have the  $\Lambda(x)$  polynomial, its roots can be found in the form  $\Lambda(x) = (\alpha^{i_1} x - 1)(\alpha^{i_2} x - 1) \dots (\alpha^{i_v} x - 1)$  by brute force for example using the **Chien search** algorithm. The exponential powers of the primitive element  $\alpha$  will yield the positions where errors occur in the received word; hence the name ‘error locator’ polynomial.

The zeros of  $\Lambda(x)$  are  $\alpha^{-i_1}, \dots, \alpha^{-i_v}$ .

### 8.4.4 Calculate error values

Once the error locations are known, the next step is to determine the error values at those locations. The error values are then used to correct the received values at those locations to recover the original codeword.

For the case of binary BCH, (with all characters readable) this is trivial; just flip the bits for the received word at these positions, and we have the corrected code word. In the more general case, the error weights  $e_j$  can be determined by solving the linear system

$$\begin{aligned}
s_c &= e_1 \alpha^{c i_1} + e_2 \alpha^{c i_2} + \dots \\
s_{c+1} &= e_1 \alpha^{(c+1) i_1} + e_2 \alpha^{(c+1) i_2} + \dots \\
&\vdots
\end{aligned}$$

### Forney algorithm

However, there is a more efficient method known as the Forney algorithm.

Let

$$S(x) = s_c + s_{c+1}x + s_{c+2}x^2 + \dots + s_{c+d-2}x^{d-2}.$$

$$v \leq d-1, \lambda_0 \neq 0 \quad \Lambda(x) = \sum_{i=0}^v \lambda_i x^i = \lambda_0 \prod_{k=0}^v (\alpha^{-i_k} x - 1).$$

And the error evaluator polynomial<sup>[9]</sup>

$$\Omega(x) \equiv S(x)\Lambda(x) \bmod x^{d-1}$$

Finally:

$$\Lambda'(x) = \sum_{i=1}^v i \cdot \lambda_i x^{i-1},$$

where

$$i \cdot x := \sum_{k=1}^i x.$$

Then if syndromes could be explained by an error word, which could be nonzero only on positions  $i_k$ , then error values are

$$e_k = -\frac{\alpha^{i_k} \Omega(\alpha^{-i_k})}{\alpha^{c \cdot i_k} \Lambda'(\alpha^{-i_k})}.$$

For narrow-sense BCH codes,  $c = 1$ , so the expression simplifies to:

$$e_k = -\frac{\Omega(\alpha^{-i_k})}{\Lambda'(\alpha^{-i_k})}.$$

### Explanation of Forney algorithm computation

It is based on Lagrange interpolation and techniques of generating functions.

Consider  $S(x)\Lambda(x)$ , and for the sake of simplicity suppose  $\lambda_k = 0$  for  $k > v$ , and  $s_k = 0$  for  $k > c + d - 2$ . Then

$$S(x)\Lambda(x) = \sum_{j=0}^{\infty} \sum_{i=0}^j s_{j-i+1} \lambda_i x^j.$$

$$\begin{aligned}
S(x)\Lambda(x) &= S(x) \left\{ \lambda_0 \prod_{\ell=1}^v (\alpha^{i_\ell} x - 1) \right\} \\
&= \left\{ \sum_{i=0}^{d-2} \sum_{j=1}^v e_j \alpha^{(c+i) \cdot i_j} x^i \right\} \left\{ \lambda_0 \prod_{\ell=1}^v (\alpha^{i_\ell} x - 1) \right\} \\
&= \left\{ \sum_{j=1}^v e_j \alpha^{c i_j} \sum_{i=0}^{d-2} (\alpha^{i_j})^i x^i \right\} \left\{ \lambda_0 \prod_{\ell=1}^v (\alpha^{i_\ell} x - 1) \right\} \\
&= \left\{ \sum_{j=1}^v e_j \alpha^{c i_j} \frac{(x \alpha^{i_j})^{d-1} - 1}{x \alpha^{i_j} - 1} \right\} \left\{ \lambda_0 \prod_{\ell=1}^v (\alpha^{i_\ell} x - 1) \right\} \\
&= \lambda_0 \sum_{j=1}^v e_j \alpha^{c i_j} \frac{(x \alpha^{i_j})^{d-1} - 1}{x \alpha^{i_j} - 1} \prod_{\ell=1}^v (\alpha^{i_\ell} x - 1) \\
&= \lambda_0 \sum_{j=1}^v e_j \alpha^{c i_j} \left( (x \alpha^{i_j})^{d-1} - 1 \right) \prod_{\ell \in \{1, \dots, v\} \setminus \{j\}} (\alpha^{i_\ell} x - 1)
\end{aligned}$$

We want to compute unknowns  $e_j$ , and we could simplify the context by removing the  $(x \alpha^{i_j})^{d-1}$  terms. This leads to the error evaluator polynomial

$$\Omega(x) \equiv S(x)\Lambda(x) \bmod x^{d-1}.$$

Thanks to  $v \leq d - 1$  we have

$$\Omega(x) = -\lambda_0 \sum_{j=1}^v e_j \alpha^{c i_j} \prod_{\ell \in \{1, \dots, v\} \setminus \{j\}} (\alpha^{i_\ell} x - 1).$$

Thanks to  $\Lambda$  (the Lagrange interpolation trick) the sum degenerates to only one summand for  $x = \alpha^{-i_k}$

$$\Omega(\alpha^{-i_k}) = -\lambda_0 e_k \alpha^{c \cdot i_k} \prod_{\ell \in \{1, \dots, v\} \setminus \{k\}} (\alpha^{i_\ell} \alpha^{-i_k} - 1).$$

To get  $e_k$  we just should get rid of the product. We could compute the product directly from already computed roots  $\alpha^{-i_j}$  of  $\Lambda$ , but we could use simpler form.

As formal derivative

$$\Lambda'(x) = \lambda_0 \sum_{j=1}^v \alpha^{i_j} \prod_{\ell \in \{1, \dots, v\} \setminus \{j\}} (\alpha^{i_\ell} x - 1),$$

we get again only one summand in

$$\Lambda'(\alpha^{-i_k}) = \lambda_0 \alpha^{i_k} \prod_{\ell \in \{1, \dots, v\} \setminus \{k\}} (\alpha^{i_\ell} \alpha^{-i_k} - 1).$$

So finally

$$e_k = -\frac{\alpha^{i_k} \Omega(\alpha^{-i_k})}{\alpha^{c \cdot i_k} \Lambda'(\alpha^{-i_k})}.$$

This formula is advantageous when one computes the formal derivative of  $\Lambda$  form

$$\Lambda(x) = \sum_{i=1}^v \lambda_i x^i$$

yielding:

$$\Lambda'(x) = \sum_{i=1}^v i \cdot \lambda_i x^{i-1},$$

where

$$i \cdot x := \sum_{k=1}^i x.$$

### 8.4.5 Decoding based on extended Euclidean algorithm

An alternate process of finding both the polynomial  $\Lambda$  and the error locator polynomial is based on Yasuo Sugiyama's adaptation of the **Extended Euclidean algorithm**.<sup>[10]</sup> Correction of unreadable characters could be incorporated to the algorithm easily as well.

Let  $k_1, \dots, k_k$  be positions of unreadable characters. One creates polynomial localising these positions  $\Gamma(x) = \prod_{i=1}^k (x\alpha^{k_i} - 1)$ . Set values on unreadable positions to 0 and compute the syndromes.

As we have already defined for the Forney formula let  $S(x) = \sum_{i=0}^{d-2} s_{c+i} x^i$ .

Let us run extended Euclidean algorithm for locating least common divisor of polynomials  $S(x)\Gamma(x)$  and  $x^{d-1}$ . The goal is not to find the least common divisor, but a polynomial  $r(x)$  of degree at most  $\lfloor (d+k-3)/2 \rfloor$  and polynomials  $a(x), b(x)$  such that  $r(x) = a(x)S(x)\Gamma(x) + b(x)x^{d-1}$ . Low degree of  $r(x)$  guarantees, that  $a(x)$  would satisfy extended (by  $\Gamma$ ) defining conditions for  $\Lambda$ .

Defining  $\Xi(x) = a(x)\Gamma(x)$  and using  $\Xi$  on the place of  $\Lambda(x)$  in the Forney formula will give us error values.

The main advantage of the algorithm is that it meanwhile computes  $\Omega(x) = S(x)\Xi(x) \bmod x^{d-1} = r(x)$  required in the Forney formula.

### Explanation of the decoding process

The goal is to find a codeword which differs from the received word minimally as possible on readable positions. When expressing the received word as a sum of nearest codeword and error word, we are trying to find error word with minimal number of non-zeros on readable positions. Syndrome  $s_i$  restricts error word by condition

$$s_i = \sum_{j=0}^{n-1} e_j \alpha^{ij}.$$

We could write these conditions separately or we could create polynomial

$$S(x) = \sum_{i=0}^{d-2} s_{c+i} x^i$$

and compare coefficients near powers 0 to  $d-2$ .

$$S(x)^{\{0, \dots, d-2\}} E(x) = \sum_{i=0}^{d-2} \sum_{j=0}^{n-1} e_j \alpha^{ij} \alpha^{cj} x^i.$$

Suppose there is unreadable letter on position  $k_1$ , we could replace set of syndromes  $\{s_c, \dots, s_{c+d-2}\}$  by set of syndromes  $\{t_c, \dots, t_{c+d-3}\}$  defined by equation  $t_i = \alpha^{k_1} s_i - s_{i+1}$ . Suppose for an error word all restrictions by original set  $\{s_c, \dots, s_{c+d-2}\}$  of syndromes hold, than

$$t_i = \alpha^{k_1} s_i - s_{i+1} = \alpha^{k_1} \sum_{j=0}^{n-1} e_j \alpha^{ij} - \sum_{j=0}^{n-1} e_j \alpha^{j} \alpha^{(i+1)j} = \sum_{j=0}^{n-1} e_j (\alpha^{k_1} - \alpha^j) \alpha^{ij}.$$

New set of syndromes restricts error vector

$$f_j = e_j (\alpha^{k_1} - \alpha^j)$$

the same way the original set of syndromes restricted the error vector  $e_j$ . Note, that except the coordinate  $k_1$ , where we have  $f_{k_1} = 0$ , an  $f_j$  is zero, if  $e_j = 0$ . For the goal of locating error positions we could change the set of syndromes in the similar way to reflect all unreadable characters. This shortens the set of syndromes by  $k$ .

In polynomial formulation, the replacement of syndromes set  $\{s_c, \dots, s_{c+d-2}\}$  by syndromes set  $\{t_c, \dots, t_{c+d-3}\}$  leads to

$$T(x) = \sum_{i=0}^{d-3} t_{c+i} x^i = \alpha^{k_1} \sum_{i=0}^{d-3} s_{c+i} x^i - \sum_{i=1}^{d-2} s_{c+i} x^{i-1}.$$

Therefore

$$xT(x)^{\{1,\dots,d-2\}}(x\alpha^{k_1} - 1)S(x).$$

After replacement of  $S(x)$  by  $S(x)\Gamma(x)$ , one would require equation for coefficients near powers  $k, \dots, d-2$ .

One could consider looking for error positions from the point of view of eliminating influence of given positions similarly as for unreadable characters. If we found  $v$  positions such that eliminating their influence leads to obtaining set of syndromes consisting of all zeros, than there exists error vector with errors only on these coordinates. If  $\Lambda(x)$  denotes the polynomial eliminating the influence of these coordinates, we obtain

$$S(x)\Gamma(x)\Lambda(x)^{\{k+v,\dots,d-2\}}0.$$

In Euclidean algorithm, we try to correct at most  $\frac{1}{2}(d-1-k)$  errors (on readable positions), because with bigger error count there could be more codewords in the same distance from the received word. Therefore, for  $\Lambda(x)$  we are looking for, the equation must hold for coefficients near powers starting from

$$k + \lfloor \frac{1}{2}(d-1-k) \rfloor.$$

In Forney formula,  $\Lambda(x)$  could be multiplied by a scalar giving the same result.

It could happen that the Euclidean algorithm finds  $\Lambda(x)$  of degree higher than  $\frac{1}{2}(d-1-k)$  having number of different roots equal to its degree, where the Forney formula would be able to correct errors in all its roots, anyways correcting such many errors could be risky (especially with no other restrictions on received word). Usually after getting  $\Lambda(x)$  of higher degree, we decide not to correct the errors. Correction could fail in the case  $\Lambda(x)$  has roots with higher multiplicity or the number of roots is smaller than its degree. Fail could be detected as well by Forney formula returning error outside the transmitted alphabet.

#### 8.4.6 Correct the errors

Using the error values and error location, correct the errors and form a corrected code vector by subtracting error values at error locations.

#### 8.4.7 Decoding examples

##### Decoding of binary code without unreadable characters

Consider a BCH code in  $\text{GF}(2^4)$  with  $d = 7$  and  $g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$ . (This is used in QR

codes.) Let the message to be transmitted be  $[1\ 1\ 0\ 1\ 1]$ , or in polynomial notation,  $M(x) = x^4 + x^3 + x + 1$ . The “checksum” symbols are calculated by dividing  $x^{10}M(x)$  by  $g(x)$  and taking the remainder, resulting in  $x^9 + x^4 + x^2$  or  $[1\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0]$ . These are appended to the message, so the transmitted codeword is  $[1\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0]$ .

Now, imagine that there are two bit-errors in the transmission, so the received codeword is  $[1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0]$ . In polynomial notation:

$$R(x) = C(x) + x^{13} + x^5 = x^{14} + x^{11} + x^{10} + x^9 + x^5 + x^4 + x^2$$

In order to correct the errors, first calculate the syndromes. Taking  $\alpha = 0010$ , we have  $s_1 = R(\alpha^1) = 1011$ ,  $s_2 = 1001$ ,  $s_3 = 1011$ ,  $s_4 = 1101$ ,  $s_5 = 0001$ , and  $s_6 = 1001$ . Next, apply the Peterson procedure by row-reducing the following augmented matrix.

$$[S_{3 \times 3} | C_{3 \times 1}] = \begin{bmatrix} s_1 & s_2 & s_3 & s_4 \\ s_2 & s_3 & s_4 & s_5 \\ s_3 & s_4 & s_5 & s_6 \end{bmatrix} = \begin{bmatrix} 1011 & 1001 & 1011 & 1101 \\ 1001 & 1011 & 1101 & 0001 \\ 1011 & 1101 & 0001 & 1001 \end{bmatrix} \Rightarrow$$

Due to the zero row,  $S_{3 \times 3}$  is singular, which is no surprise since only two errors were introduced into the codeword. However, the upper-left corner of the matrix is identical to  $[S_{2 \times 2} | C_{2 \times 1}]$ , which gives rise to the solution  $\lambda_2 = 1000$ ,  $\lambda_1 = 1011$ . The resulting error locator polynomial is  $\Lambda(x) = 1000x^2 + 1011x + 0001$ , which has zeros at  $0100 = \alpha^{-13}$  and  $0111 = \alpha^{-5}$ . The exponents of  $\alpha$  correspond to the error locations. There is no need to calculate the error values in this example, as the only possible value is 1.

##### Decoding with unreadable characters

Suppose the same scenario, but the received word has two unreadable characters  $[1\ 0\ 0\ ?\ 1\ 1\ ?\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0]$ . We replace the unreadable characters by zeros while creating the polynomial reflecting their positions  $\Gamma(x) = (\alpha^8x - 1)(\alpha^{11}x - 1)$ . We compute the syndromes  $s_1 = \alpha^{-7}$ ,  $s_2 = \alpha^1$ ,  $s_3 = \alpha^4$ ,  $s_4 = \alpha^2$ ,  $s_5 = \alpha^5$ , and  $s_6 = \alpha^{-7}$ . (Using log notation which is independent on  $\text{GF}(2^4)$  isomorphisms. For computation checking we can use the same representation for addition as was used in previous example. Hexadecimal description of the powers of  $\alpha$  are consecutively 1,2,4,8,3,6,C,B,5,A,7,E,F,D,9 with the addition based on bitwise xor.)

Let us make syndrome polynomial

$$S(x) = \alpha^{-7} + \alpha^1x + \alpha^4x^2 + \alpha^2x^3 + \alpha^5x^4 + \alpha^{-7}x^5,$$

compute

$$S(x)\Gamma(x) = \alpha^{-7} + \alpha^4x + \alpha^{-1}x^2 + \alpha^6x^3 + \alpha^{-1}x^4 + \alpha^5x^5 + \alpha^7x^6 + \alpha^{-3}x^7.$$

Run the extended Euclidean algorithm:

where  $\alpha^{-i_j}$  are roots of  $\Xi(x)$ .  $\Xi'(x) = \alpha^2 x^2$ . We get

$$\begin{aligned}
 \begin{pmatrix} S(x)\Gamma(x) \\ x^6 \end{pmatrix} &= \begin{pmatrix} \alpha^{-7} + \alpha^4 x + \alpha^{-1} x^2 + \alpha^6 x^3 + \alpha^{-1} x^4 + \alpha^5 x^5 + \alpha^4 x^6 \\ x^6 \end{pmatrix} \begin{pmatrix} \alpha^{-7} + \alpha^4 x + \alpha^{-1} x^2 + \alpha^6 x^3 + \alpha^{-1} x^4 + \alpha^5 x^5 + \alpha^4 x^6 \\ x^6 \end{pmatrix}^{-1} = \frac{\alpha^{-5}}{\alpha^{-5}} = 1 \\
 &= \begin{pmatrix} \alpha^7 + \alpha^{-3} x & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha^{-7} + \alpha^4 x + \alpha^{-1} x^2 + \alpha^6 x^3 + \alpha^{-1} x^4 + \alpha^5 x^5 + \alpha^4 x^6 \\ x^6 \end{pmatrix} \\
 &= \begin{pmatrix} \alpha^7 + \alpha^{-3} x & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha^4 + \alpha^{-5} x & 1 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} \alpha^{-7} + \alpha^4 x + \alpha^{-1} x^2 + \alpha^6 x^3 + \alpha^{-1} x^4 + \alpha^5 x^5 + \alpha^4 x^6 \\ x^6 \end{pmatrix} \\
 &= \begin{pmatrix} (1 + \alpha^{-4}) + (\alpha^1 + \alpha^2)x + \alpha^7 x^2 & \alpha^7 + \alpha^{-3} x \\ \alpha^4 + \alpha^{-5} x & 0 \end{pmatrix} \begin{pmatrix} \alpha^{-7} + \alpha^4 x + \alpha^{-1} x^2 + \alpha^6 x^3 + \alpha^{-1} x^4 + \alpha^5 x^5 + \alpha^4 x^6 \\ x^6 \end{pmatrix} \\
 &= \begin{pmatrix} \alpha^{-3} + \alpha^5 x + \alpha^7 x^2 & \alpha^7 + \alpha^{-3} x \\ \alpha^4 + \alpha^{-5} x & 1 \end{pmatrix} \begin{pmatrix} \alpha^{-5} + \alpha^{-4} x & 1 \\ 0 & 0 \end{pmatrix} \\
 &= \begin{pmatrix} \alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3 & \alpha^{-3} + \alpha^5 x + \alpha^7 x^2 \\ \alpha^3 + \alpha^{-5} x + \alpha^6 x^2 & \alpha^4 + \alpha^{-5} x \end{pmatrix} \begin{pmatrix} \alpha^{-5} + \alpha^{-4} x & 1 \\ 0 & 0 \end{pmatrix} \\
 &= \begin{pmatrix} \alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3 & \alpha^{-3} + \alpha^5 x + \alpha^7 x^2 \\ \alpha^3 + \alpha^{-5} x + \alpha^6 x^2 & \alpha^4 + \alpha^{-5} x \end{pmatrix} \begin{pmatrix} \alpha^{-5} + \alpha^{-4} x & 1 \\ 0 & 0 \end{pmatrix}
 \end{aligned}$$

We have reached polynomial of degree at most 3, and as

Again, replace the unreadable characters by zeros while creating the polynomial reflecting their positions  $\Gamma(x) = (\alpha^8 x - 1)(\alpha^{11} x - 1)$ . Compute the syndromes  $s_1 =$

$$\begin{pmatrix} -(\alpha^4 + \alpha^{-5} x) & \alpha^{-3} + \alpha^5 x + \alpha^7 x^2 \\ \alpha^3 + \alpha^{-5} x + \alpha^6 x^2 & -(\alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3) \end{pmatrix} \begin{pmatrix} \alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3 \\ \alpha^3 + \alpha^{-5} x + \alpha^6 x^2 \end{pmatrix} = \begin{pmatrix} \alpha^0 & \alpha^1 & \alpha^6 \\ 0 & 1 & 0 \end{pmatrix}$$

we get

$$S(x) = \alpha^4 + \alpha^{-7} x + \alpha^1 x^2 + \alpha^1 x^3 + \alpha^0 x^4 + \alpha^2 x^5,$$

$$\begin{pmatrix} -(\alpha^4 + \alpha^{-5} x) & \alpha^{-3} + \alpha^5 x + \alpha^7 x^2 \\ \alpha^3 + \alpha^{-5} x + \alpha^6 x^2 & -(\alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3) \end{pmatrix} \begin{pmatrix} S(x)\Gamma(x) \\ x^6 \end{pmatrix} = \begin{pmatrix} \alpha^4 + \alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3 + \alpha^1 x^4 + \alpha^{-1} x^5 + \alpha^{-1} x^6 \\ \alpha^{-4} + \alpha^4 x + \alpha^2 x^2 + \alpha^{-5} x^3 \end{pmatrix}$$

Therefore

Let us run the extended Euclidean algorithm:

$$\begin{pmatrix} S(x)\Gamma(x) \\ x^6 \end{pmatrix} = \begin{pmatrix} \alpha^4 + \alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3 + \alpha^1 x^4 + \alpha^{-1} x^5 + \alpha^{-1} x^6 \\ x^6 \end{pmatrix}$$

$$S(x)\Gamma(x)(\alpha^3 + \alpha^{-5} x + \alpha^6 x^2) - (\alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3)x^6 = \alpha^{-4} + \alpha^4 x + \alpha^2 x^2$$

Let  $\Lambda(x) = \alpha^3 + \alpha^{-5} x + \alpha^6 x^2$ . Don't worry that  $\lambda_0 \neq 1$ . Find by brute force a root of  $\Lambda$ . The roots are  $\alpha^2$ , and  $\alpha^{10}$  (after finding for example  $\alpha^2$  we can divide  $\Lambda$  by corresponding monom  $(x - \alpha^2)$  and the root of resulting monom could be found easily).

Let

$$\begin{aligned}
 &= \begin{pmatrix} \alpha^{-1} + \alpha^6 x & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha^4 + \alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3 + \alpha^1 x^4 + \alpha^{-1} x^5 + \alpha^{-1} x^6 \\ x^6 \end{pmatrix} \\
 &= \begin{pmatrix} \alpha^{-1} + \alpha^6 x & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha^3 + \alpha^1 x & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha^4 + \alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3 + \alpha^1 x^4 + \alpha^{-1} x^5 + \alpha^{-1} x^6 \\ x^6 \end{pmatrix} \\
 &= \begin{pmatrix} (1 + \alpha^2) + (\alpha^0 + \alpha^{-6})x + \alpha^7 x^2 & \alpha^{-1} + \alpha^6 x \\ \alpha^3 + \alpha^1 x & 1 \end{pmatrix} \begin{pmatrix} \alpha^4 + \alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3 + \alpha^1 x^4 + \alpha^{-1} x^5 + \alpha^{-1} x^6 \\ x^6 \end{pmatrix}
 \end{aligned}$$

We have reached polynomial of degree at most 3, and as

$$\Xi(x) = \Gamma(x)\Lambda(x) = \alpha^3 + \alpha^4 x^2 + \alpha^2 x^3 + \alpha^{-5} x^4$$

$$\begin{pmatrix} -1 & \alpha^{-1} + \alpha^6 x \\ \alpha^3 + \alpha^1 x & -(\alpha^{-7} + \alpha^7 x + \alpha^7 x^2) \end{pmatrix} \begin{pmatrix} \alpha^{-7} + \alpha^7 x + \alpha^7 x^2 & \alpha^{-1} + \alpha^6 x \\ \alpha^3 + \alpha^1 x & 1 \end{pmatrix}$$

$$\Omega(x) = S(x)\Xi(x) \equiv \alpha^{-4} + \alpha^4 x + \alpha^2 x^2 + \alpha^{-5} x^3 \pmod{x^6}$$

we get

Let us look for error values using formula

$$\begin{pmatrix} -1 & \alpha^{-1} + \alpha^6 x \\ \alpha^3 + \alpha^1 x & -(\alpha^{-7} + \alpha^7 x + \alpha^7 x^2) \end{pmatrix} \begin{pmatrix} S(x)\Gamma(x) \\ x^6 \end{pmatrix} = \begin{pmatrix} \alpha^4 + \alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3 + \alpha^1 x^4 + \alpha^{-1} x^5 + \alpha^{-1} x^6 \\ x^6 \end{pmatrix}$$

$$e_j = -\frac{\Omega(\alpha^{-i_j})}{\Xi'(\alpha^{-i_j})},$$

Therefore,

## 8.6 References

$$S(x)\Gamma(x)(\alpha^3 + \alpha^1 x) - (\alpha^{-7} + \alpha^7 x + \alpha^7 x^2)x^6 = \alpha^7 + \alpha^0 x.$$

Let  $\Lambda(x) = \alpha^3 + \alpha^1 x$ . Don't worry that  $\lambda_0 \neq 1$ . The root of  $\Lambda(x)$  is  $\alpha^{3-1}$ .

Let

$$\Xi(x) = \Gamma(x)\Lambda(x) = \alpha^3 + \alpha^{-7}x + \alpha^{-4}x^2 + \alpha^5x^3,$$

$$\Omega(x) = S(x)\Xi(x) \equiv \alpha^7 + \alpha^0 x \pmod{x^6}$$

Let us look for error values using formula  $e_j = -\Omega(\alpha^{-i_j})/\Xi'(\alpha^{-i_j})$ , where  $\alpha^{-i_j}$  are roots of polynomial  $\Xi(x)$ .  $\Xi'(x) = \alpha^{-7} + \alpha^5 x^2$ . We get

$$e_1 = -\frac{\Omega(\alpha^4)}{\Xi'(\alpha^4)} = \frac{\alpha^7 + \alpha^4}{\alpha^{-7} + \alpha^{-2}} = \frac{\alpha^3}{\alpha^3} = 1$$

$$e_2 = -\frac{\Omega(\alpha^7)}{\Xi'(\alpha^7)} = \frac{\alpha^7 + \alpha^7}{\alpha^{-7} + \alpha^4} = 0$$

$$e_3 = -\frac{\Omega(\alpha^2)}{\Xi'(\alpha^2)} = \frac{\alpha^7 + \alpha^2}{\alpha^{-7} + \alpha^{-6}} = \frac{\alpha^{-3}}{\alpha^{-3}} = 1$$

The fact that  $e_3 = 1$  should not be surprising.

Corrected code is therefore [ 1 1 0 1 1 1 0 0 0 0 1 0 1 0 ].

## 8.5 Citations

- [1] Reed & Chen 1999, p. 189
- [2] Hocquenghem 1959
- [3] Bose & Ray-Chaudhuri 1960
- [4] “Phobos Lander Coding System: Software and Analysis” (PDF). Retrieved 25 February 2012.
- [5] “Sandforce SF-2500/2600 Product Brief”. Retrieved 25 February 2012.
- [6] Gill n.d., p. 3
- [7] Lidl & Pilz 1999, p. 229
- [8] Gorenstein, Peterson & Zierler 1960
- [9] Gill n.d., p. 47
- [10] Yasuo Sugiyama, Masao Kasahara, Shigeichi Hirasawa, and Toshihiko Namekawa. A method for solving key equation for decoding Goppa codes. *Information and Control*, 27:87–99, 1975.

### 8.6.1 Primary sources

- Hocquenghem, A. (September 1959), “Codes correcteurs d'erreurs”, *Chiffres* (in French), Paris, **2**: 147–156
- Bose, R. C.; Ray-Chaudhuri, D. K. (March 1960), “On A Class of Error Correcting Binary Group Codes”, *Information and Control*, **3** (1): 68–79, doi:10.1016/s0019-9958(60)90287-4, ISSN 0890-5401

### 8.6.2 Secondary sources

- Gill, John (n.d.), *EE387 Notes #7, Handout #28* (PDF), Stanford University, pp. 42–45, retrieved April 21, 2010 Course notes are apparently being redone for 2012: <http://www.stanford.edu/class/ee387/>
- Gorenstein, Daniel; Peterson, W. Wesley; Zierler, Neal (1960), “Two-Error Correcting Bose-Chaudhuri Codes are Quasi-Perfect”, *Information and Control*, **3** (3): 291–294, doi:10.1016/s0019-9958(60)90877-9
- Lidl, Rudolf; Pilz, Günter (1999), *Applied Abstract Algebra* (2nd ed.), John Wiley
- Reed, Irving S.; Chen, Xuemin (1999), *Error-Control Coding for Data Networks*, Boston, MA: Kluwer Academic Publishers, ISBN 0-7923-8528-4

## 8.7 Further reading

- Blahut, Richard E. (2003), *Algebraic Codes for Data Transmission* (2nd ed.), Cambridge University Press, ISBN 0-521-55374-1
- Gilbert, W. J.; Nicholson, W. K. (2004), *Modern Algebra with Applications* (2nd ed.), John Wiley
- Lin, S.; Costello, D. (2004), *Error Control Coding: Fundamentals and Applications*, Englewood Cliffs, NJ: Prentice-Hall
- MacWilliams, F. J.; Sloane, N. J. A. (1977), *The Theory of Error-Correcting Codes*, New York, NY: North-Holland Publishing Company
- Rudra, Atri, *CSE 545, Error Correcting Codes: Combinatorics, Algorithms and Applications*, University at Buffalo, retrieved April 21, 2010



## Chapter 9

# Belief propagation

**Belief propagation**, also known as **sum-product message passing**, is a message passing algorithm for performing inference on graphical models, such as Bayesian networks and Markov random fields. It calculates the marginal distribution for each unobserved node, conditional on any observed nodes. Belief propagation is commonly used in artificial intelligence and information theory and has demonstrated empirical success in numerous applications including low-density parity-check codes, turbo codes, free energy approximation, and satisfiability.<sup>[1]</sup>

The algorithm was first proposed by Judea Pearl in 1982,<sup>[2]</sup> who formulated this algorithm on trees, and was later extended to polytrees.<sup>[3]</sup> It has since been shown to be a useful approximate algorithm on general graphs.<sup>[4]</sup>

If  $X = \{X_i\}$  is a set of discrete random variables with a joint mass function  $p$ , the marginal distribution of a single  $X_i$  is simply the summation of  $p$  over all other variables:

$$p_{X_i}(x_i) = \sum_{\mathbf{x}': x'_i < > x_i} p(\mathbf{x}').$$

However, this quickly becomes computationally prohibitive: if there are 100 binary variables, then one needs to sum over  $2^{99} \approx 6.338 \times 10^{29}$  possible values. By exploiting the polytree structure, belief propagation allows the marginals to be computed much more efficiently.

### 9.1 Description of the sum-product algorithm

Variants of the belief propagation algorithm exist for several types of graphical models (Bayesian networks and Markov random fields,<sup>[5]</sup> in particular). We describe here the variant that operates on a factor graph. A factor graph is a bipartite graph containing nodes corresponding to variables  $V$  and factors  $F$ , with edges between variables and the factors in which they appear. We can write the joint mass function:

$$p(\mathbf{x}) = \prod_{a \in F} f_a(\mathbf{x}_a)$$

where  $\mathbf{x}_a$  is the vector of neighbouring variable nodes to the factor node  $a$ . Any Bayesian network or Markov random field can be represented as a factor graph.

The algorithm works by passing real valued functions called *messages* along with the edges between the hidden nodes. More precisely, if  $v$  is a variable node and  $a$  is a factor node connected to  $v$  in the factor graph, the messages from  $v$  to  $a$ , (denoted by  $\mu_{v \rightarrow a}$ ) and from  $a$  to  $v$  ( $\mu_{a \rightarrow v}$ ), are real-valued functions whose domain is  $\text{Dom}(v)$ , the set of values that can be taken by the random variable associated with  $v$ . These messages contain the “influence” that one variable exerts on another. The messages are computed differently depending on whether the node receiving the message is a variable node or a factor node. Keeping the same notation:

- A message from a variable node  $v$  to a factor node  $a$  is the product of the messages from all other neighbouring factor nodes (except the recipient; alternatively one can say the recipient sends as message the constant function equal to “1”):

$$\forall x_v \in \text{Dom}(v), \mu_{v \rightarrow a}(x_v) = \prod_{a^* \in N(v) \setminus \{a\}} \mu_{a^* \rightarrow v}(x_v).$$

where  $N(v)$  is the set of neighbouring (factor) nodes to  $v$ . If  $N(v) \setminus \{a\}$  is empty, then  $\mu_{v \rightarrow a}(x_v)$  is set to the uniform distribution.

- A message from a factor node  $a$  to a variable node  $v$  is the product of the factor with messages from all other nodes, marginalised over all variables except the one associated with  $v$ :

$$\forall x_v \in \text{Dom}(v), \mu_{a \rightarrow v}(x_v) = \sum_{\mathbf{x}'_a: x'_v = x_v} f_a(\mathbf{x}'_a) \prod_{v^* \in N(a) \setminus \{v\}} \mu_{v^* \rightarrow a}(x'_{v^*}).$$



where  $N(a)$  is the set of neighbouring (variable) nodes to  $a$ . If  $N(a) \setminus \{v\}$  is empty then  $\mu_{a \rightarrow v}(x_v) = f_a(x_v)$ , since in this case  $x_v = x_a$ .

As shown by the previous formula: the complete marginalisation is reduced to a sum of products of simpler terms than the ones appearing in the full joint distribution. This is the reason why it is called the sum-product algorithm.

In a typical run, each message will be updated iteratively from the previous value of the neighbouring messages. Different scheduling can be used for updating the messages. In the case where the graphical model is a tree, an optimal scheduling allows to reach convergence after computing each messages only once (see next subsection). When the factor graph has cycles, such an optimal scheduling does not exist, and a typical choice is to update all messages simultaneously at each iteration.

Upon convergence (if convergence happened), the estimated marginal distribution of each node is proportional to the product of all messages from adjoining factors (missing the normalization constant):

$$p_{X_v}(x_v) \propto \prod_{a \in N(v)} \mu_{a \rightarrow v}(x_v).$$

Likewise, the estimated joint marginal distribution of the set of variables belonging to one factor is proportional to the product of the factor and the messages from the variables:

$$p_{X_a}(\mathbf{x}_a) \propto f_a(\mathbf{x}_a) \prod_{v \in N(a)} \mu_{v \rightarrow a}(x_v).$$

In the case where the factor graph is acyclic (i.e. is a tree or a forest), these estimated marginal actually converge to the true marginals in a finite number of iterations. This can be shown by **mathematical induction**.

## 9.2 Exact algorithm for trees

In the case when the **factor graph** is a **tree**, the belief propagation algorithm will compute the exact marginals. Furthermore, with proper scheduling of the message updates, it will terminate after 2 steps. This optimal scheduling can be described as follows:

Before starting, the graph is oriented by designating one node as the *root*; any non-root node which is connected to only one other node is called a *leaf*.

In the first step, messages are passed inwards: starting at the leaves, each node passes a message along the (unique) edge towards the root node. The tree structure guarantees that it is possible to obtain messages from all other

adjoining nodes before passing the message on. This continues until the root has obtained messages from all of its adjoining nodes.

The second step involves passing the messages back out: starting at the root, messages are passed in the reverse direction. The algorithm is completed when all leaves have received their messages.

## 9.3 Approximate algorithm for general graphs

Curiously, although it was originally designed for acyclic graphical models, it was found that the Belief Propagation algorithm can be used in general **graphs**. The algorithm is then sometimes called “loopy” belief propagation, because graphs typically contain **cycles**, or loops. The initialization and scheduling of message updates must be adjusted slightly (compared with the previously described schedule for acyclic graphs) because graphs might not contain any leaves. Instead, one initializes all variable messages to 1 and uses the same message definitions above, updating all messages at every iteration (although messages coming from known leaves or tree-structured subgraphs may no longer need updating after sufficient iterations). It is easy to show that in a tree, the message definitions of this modified procedure will converge to the set of message definitions given above within a number of iterations equal to the diameter of the tree.

The precise conditions under which loopy belief propagation will converge are still not well understood; it is known that on graphs containing a single loop it converges in most cases, but the probabilities obtained might be incorrect.<sup>[6]</sup> Several sufficient (but not necessary) conditions for convergence of loopy belief propagation to a unique fixed point exist.<sup>[7]</sup> There exist graphs which will fail to converge, or which will oscillate between multiple states over repeated iterations. Techniques like **EXIT charts** can provide an approximate visualisation of the progress of belief propagation and an approximate test for convergence.

There are other approximate methods for marginalization including **variational methods** and **Monte Carlo methods**.

One method of exact marginalization in general graphs is called the **junction tree algorithm**, which is simply belief propagation on a modified graph guaranteed to be a tree. The basic premise is to eliminate cycles by clustering them into single nodes.

## 9.4 Related algorithm and complexity issues

A similar algorithm is commonly referred to as the **Viterbi algorithm**, but also known as a special case of the

max-product or min-sum algorithm, which solves the related problem of maximization, or most probable explanation. Instead of attempting to solve the marginal, the goal here is to find the values  $\mathbf{x}$  that maximises the global function (i.e. most probable values in a probabilistic setting), and it can be defined using the **arg max**:

$$* \arg \max_{\mathbf{x}} g(\mathbf{x}).$$

An algorithm that solves this problem is nearly identical to belief propagation, with the sums replaced by maxima in the definitions.<sup>[8]</sup>

It is worth noting that **inference** problems like marginalization and maximization are **NP-hard** to solve exactly and approximately (at least for **relative error**) in a graphical model. More precisely, the marginalization problem defined above is **#P-complete** and maximization is **NP-complete**.

The memory usage of belief propagation can be reduced through the use of the **Island algorithm** (at a small cost in time complexity).

## 9.5 Relation to free energy

The sum-product algorithm is related to the calculation of **free energy in thermodynamics**. Let  $Z$  be the **partition function**. A probability distribution

$$P(\mathbf{X}) = \frac{1}{Z} \prod_{f_j} f_j(x_j)$$

(as per the factor graph representation) can be viewed as a measure of the **internal energy** present in a system, computed as

$$E(\mathbf{X}) = \log \prod_{f_j} f_j(x_j).$$

The free energy of the system is then

$$F = U - H = \sum_{\mathbf{X}} P(\mathbf{X}) E(\mathbf{X}) + \sum_{\mathbf{X}} P(\mathbf{X}) \log P(\mathbf{X}).$$

It can then be shown that the points of convergence of the sum-product algorithm represent the points where the free energy in such a system is minimized. Similarly, it can be shown that a fixed point of the iterative belief propagation algorithm in graphs with cycles is a stationary point of a free energy approximation.<sup>[9]</sup>

## 9.6 Generalized belief propagation (GBP)

Belief propagation algorithms are normally presented as message update equations on a factor graph, involving messages between variable nodes and their neighboring factor nodes and vice versa. Considering messages between *regions* in a graph is one way of generalizing the belief propagation algorithm.<sup>[9]</sup> There are several ways of defining the set of regions in a graph that can exchange messages. One method uses ideas introduced by Kikuchi in the physics literature, and is known as Kikuchi's **cluster variation method**.

Improvements in the performance of belief propagation algorithms are also achievable by breaking the replicas symmetry in the distributions of the fields (messages). This generalization leads to a new kind of algorithm called **survey propagation** (SP), which have proved to be very efficient in **NP-complete** problems like **satisfiability**<sup>[11]</sup> and **graph coloring**.

The cluster variational method and the survey propagation algorithms are two different improvements to belief propagation. The name **generalized survey propagation** (GSP) is waiting to be assigned to the algorithm that merges both generalizations.

## 9.7 Gaussian belief propagation (GaBP)

Gaussian belief propagation is a variant of the belief propagation algorithm when the underlying **distributions are Gaussian**. The first work analyzing this special model was the seminal work of Weiss and Freeman<sup>[10]</sup>

The GaBP algorithm solves the following marginalization problem:

$$P(x_i) = \frac{1}{Z} \int_{j \neq i} \exp(-1/2x^T A x + b^T x) dx_j$$

where  $Z$  is a normalization constant,  $A$  is a symmetric **positive definite matrix** (inverse covariance matrix a.k.a. precision matrix) and  $b$  is the shift vector.

Equivalently, it can be shown that using the Gaussian model, the solution of the marginalization problem is equivalent to the **MAP** assignment problem:

$$\arg \max_x P(x) = \frac{1}{Z} \exp(-1/2x^T A x + b^T x).$$

This problem is also equivalent to the following minimization problem of the quadratic form:

$$\min_x 1/2x^T A x - b^T x.$$

Which is also equivalent to the linear system of equations

$$Ax = b.$$

Convergence of the GaBP algorithm is easier to analyze (relatively to the general BP case) and there are two known sufficient convergence conditions. The first one was formulated by Weiss et al. in the year 2000, when the information matrix  $A$  is **diagonally dominant**. The second convergence condition was formulated by Johnson et al.<sup>[11]</sup> in 2006, when the **spectral radius** of the matrix

$$\rho(I - |D^{-1/2}AD^{-1/2}|) < 1$$

where  $D = \text{diag}(A)$ . Later, Su and Wu established the necessary and sufficient convergence conditions for synchronous GaBP and damped GaBP, as well as another sufficient convergence condition for asynchronous GaBP. For each case, the convergence condition involves verifying 1) a set (determined by  $A$ ) being non-empty, 2) the spectral radius of a certain matrix being smaller than one, and 3) the singularity issue (when converting BP message into belief) does not occur.<sup>[12]</sup>

The GaBP algorithm was linked to the linear algebra domain,<sup>[13]</sup> and it was shown that the GaBP algorithm can be viewed as an iterative algorithm for solving the linear system of equations  $Ax = b$  where  $A$  is the information matrix and  $b$  is the shift vector. Empirically, the GaBP algorithm is shown to converge faster than classical iterative methods like the Jacobi method, the **Gauss–Seidel method**, **successive over-relaxation**, and others.<sup>[14]</sup> Additionally, the GaBP algorithm is shown to be immune to numerical problems of the preconditioned conjugate gradient method<sup>[15]</sup>

## 9.8 References

- [1] Braunstein, A.; Mézard, M.; Zecchina, R. (2005). “Survey propagation: An algorithm for satisfiability”. *Random Structures & Algorithms*. **27** (2): 201–226. doi:10.1002/rsa.20057.
- [2] Pearl, Judea (1982). “Reverend Bayes on inference engines: A distributed hierarchical approach” (PDF). *Proceedings of the Second National Conference on Artificial Intelligence*. AAAI-82: Pittsburgh, PA. Menlo Park, California: AAAI Press. pp. 133–136. Retrieved 2009-03-28.
- [3] Kim, Jin H.; Pearl, Judea (1983). “A computational model for combined causal and diagnostic reasoning in inference systems” (PDF). *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*. IJCAI-83: Karlsruhe, Germany. **1**. pp. 190–193. Retrieved 2016-03-20.
- [4] Pearl, Judea (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* (2nd ed.). San Francisco, CA: Morgan Kaufmann. ISBN 1-55860-479-0.
- [5] Yedidia, J.S.; Freeman, W.T.; Y. (January 2003). “Understanding Belief Propagation and Its Generalizations”. In Lakemeyer, Gerhard; Nebel, Bernhard. *Exploring Artificial Intelligence in the New Millennium*. Morgan Kaufmann. pp. 239–236. ISBN 1-55860-811-7. Retrieved 2009-03-30.
- [6] Weiss, Yair (2000). “Correctness of Local Probability Propagation in Graphical Models with Loops”. *Neural Computation*. **12** (1): 1–41. doi:10.1162/089976600300015880.
- [7] Mooij, J.; Kappen, H (2007). “Sufficient Conditions for Convergence of the Sum–Product Algorithm”. *IEEE Transactions on Information Theory*. **53** (12): 4422–4437. doi:10.1109/TIT.2007.909166.
- [8] Löfliger, Hans-Andrea (2004). “An Introduction to Factor Graphs”. *IEEE Signal Processing Magazine*. **21**: 28–41. doi:10.1109/msp.2004.1267047.
- [9] Yedidia, J.S.; Freeman, W.T.; Weiss, Y.; Y. (July 2005). “Constructing free-energy approximations and generalized belief propagation algorithms”. *IEEE Transactions on Information Theory*. **51** (7): 2282–2312. doi:10.1109/TIT.2005.850085. Retrieved 2009-03-28.
- [10] Weiss, Yair; Freeman, William T. (October 2001). “Correctness of Belief Propagation in Gaussian Graphical Models of Arbitrary Topology”. *Neural Computation*. **13** (10): 2173–2200. doi:10.1162/089976601750541769. PMID 11570995.
- [11] Malioutov, Dmitry M.; Johnson, Jason K.; Willsky, Alan S. (October 2006). “Walk-sums and belief propagation in Gaussian graphical models”. *Journal of Machine Learning Research*. **7**: 2031–2064. Retrieved 2009-03-28.
- [12] Su, Qinliang; Wu, Yik-Chung (March 2015). “On convergence conditions of Gaussian belief propagation”. *IEEE Trans. Signal Process.* **63** (5): 1144–1155. doi:10.1109/TSP.2015.2389755.
- [13] Gaussian belief propagation solver for systems of linear equations. By O. Shental, D. Bickson, P. H. Siegel, J. K. Wolf, and D. Dolev, IEEE Int. Symp. on Inform. Theory (ISIT), Toronto, Canada, July 2008. <http://www.cs.huji.ac.il/labs/danss/p2p/gabp/>
- [14] Linear Detection via Belief Propagation. Danny Bickson, Danny Dolev, Ori Shental, Paul H. Siegel and Jack K. Wolf. In the 45th Annual Allerton Conference on Communication, Control, and Computing, Allerton House, Illinois, 7 Sept.. <http://www.cs.huji.ac.il/labs/danss/p2p/gabp/>
- [15] Distributed large scale network utility maximization. D. Bickson, Y. Tock, A. Zymnis, S. Boyd and D. Dolev. In the International symposium on information theory (ISIT), July 2009. <http://www.cs.huji.ac.il/labs/danss/p2p/gabp/>

## 9.9 Further reading

- Bickson, Danny. (2009). *Gaussian Belief Propagation Resource Page* —Webpage containing recent publications as well as Matlab source code.
- Bishop, Christopher M. (2006). “Chapter 8: Graphical models” (PDF). *Pattern Recognition and Machine Learning*. Springer. pp. 359–418. ISBN 0-387-31073-8. Retrieved 2014-03-20.
- Coughlan, James. (2009). *A Tutorial Introduction to Belief Propagation*.
- Koch, Volker M. (2007). *A Factor Graph Approach to Model-Based Signal Separation* —A tutorial-style dissertation
- Lölliger, Hans-Andrea (2004). “An Introduction to Factor Graphs”. *IEEE Signal Proc. Mag.* **21**: 28–41.
- Mackenzie, Dana (2005). “Communication Speed Nears Terminal Velocity”, *New Scientist*. 9 July 2005. Issue 2507 (Registration required)
- Wymeersch, Henk (2007). *Iterative Receiver Design*. Cambridge University Press. ISBN 0-521-87315-0.
- Yedidia, J.S.; Freeman, W.T.; Weiss, Y. (January 2003). “Understanding Belief Propagation and Its Generalizations”. In Lakemeyer, Gerhard; Nebel, Bernhard. *Exploring Artificial Intelligence in the New Millennium*. Morgan Kaufmann. pp. 239–236. ISBN 1-55860-811-7. Retrieved 2009-03-30.
- Yedidia, J.S.; Freeman, W.T.; Weiss, Y. (July 2005). “Constructing free-energy approximations and generalized belief propagation algorithms”. *IEEE Transactions on Information Theory*. **51** (7): 2282–2312. doi:10.1109/TIT.2005.850085. Retrieved 2009-03-28.

# Chapter 10

## Berger code

In telecommunication, a **Berger code** is a unidirectional error detecting code, named after its inventor, J. M. Berger. Berger codes can detect all unidirectional errors. Unidirectional errors are errors that only flip ones into zeroes or only zeroes into ones, such as in asymmetric channels. The check bits of Berger codes are computed by summing all the zeroes in the information word, and expressing that sum in natural binary. If the information word consists of  $n$  bits, then the Berger code needs  $k = \lceil \log_2(n + 1) \rceil$  “check bits”, giving a Berger code of length  $k+n$ . (In other words, the  $k$  check bits are enough to check up to  $n = 2^k - 1$  information bits). Berger codes can detect any number of one-to-zero bit-flip errors, as long as no zero-to-one errors occurred in the same code word. Similarly, Berger codes can detect any number of zero-to-one bit-flip errors, as long as no one-to-zero bit-flip errors occur in the same code word. Berger codes cannot correct any error.

Like all unidirectional error detecting codes, Berger codes can also be used in **delay-insensitive** circuits.

### 10.1 Unidirectional error detection

As stated above, Berger codes detect *any* number of unidirectional errors. For a *given code word*, if the only errors that have occurred are that some (or all) bits with value 1 have changed to value 0, then this transformation will be detected by the Berger code implementation. To understand why, consider that there are three such cases:

1. Some 1s bit in the information part of the code word have changed to 0s.
2. Some 1s bits in the check (or *redundant*) portion of the code word have changed to 0s.
3. Some 1s bits in both the information and check portions have changed to 0s.

For case 1, the number of 0-valued bits in the information section will, by definition of the error, increase. Therefore, our berger check code will be lower than the actual 0-bit-count for the data, and so the check will fail.

For case 2, the number of 0-valued bits in the information section have stayed the same, but the value of the check data has changed. Since we know some 1s turned into 0s, but no 0s have turned into 1s (that’s how we defined the error model in this case), the encoded binary value of the check data will go down (e.g., from binary 1011 to 1010, or to 1001, or 0011). Since the information data has stayed the same, it has the same number of zeros it did before, and that will no longer match the mutated check value.

For case 3, where bits have changed in both the information and the check sections, notice that the number of zeros in the information section has *gone up*, as described for case 1, and the binary value stored in the check portion has *gone down*, as described for case 2. Therefore, there is no chance that the two will end up mutating in such a way as to become a different valid code word.

A similar analysis can be performed, and is perfectly valid, in the case where the only errors that occur are that some 0-valued bits change to 1. Therefore, if all the errors that occur on a specific codeword all occur in the same direction, these errors will be detected. For the next code word being transmitted (for instance), the errors can go in the opposite direction, and they will still be detected, as long as they all go in the same direction as each other.

Unidirectional errors are common in certain situations. For instance, in **flash memory**, bits can more easily be programmed to a 0 than can be reset to a 1.

### 10.2 References

- J. M. Berger, “A note on an error detection code for asymmetric channels”, *Information and Control*, vol 4, pp. 68–73, March 1961.
- Subhasish Mitra and Edward J. McCluskey, “Which concurrent error detection scheme to choose?”, Center for Reliable Computing, Stanford University, 2000.
- “**Delay-Insensitive Codes -- An Overview**” by Tom Verhoeff



# Chapter 11

## Berlekamp–Welch algorithm

The **Berlekamp–Welch algorithm**, also known as the **Welch–Berlekamp algorithm**, is named for Elwyn R. Berlekamp and Lloyd R. Welch. The algorithm efficiently corrects errors in **BCH codes** and **Reed–Solomon codes** (which are a subset of BCH codes). Unlike many other decoding algorithms, and in correspondence with the code-domain **Berlekamp–Massey algorithm** that uses **syndrome decoding** and the dual of the codes, the Berlekamp–Welch decoding algorithm provides a method for decoding Reed–Solomon codes using just the generator matrix and not syndromes.

### 11.1 History on decoding Reed–Solomon codes

- In 1960, Peterson developed an algorithm for decoding **BCH codes**.<sup>[1][2]</sup> His algorithm solves the important second stage of the generalized BCH decoding procedure and is used to calculate the error locator polynomial coefficients that in turn provide the error locator polynomial. This is crucial to the decoding of BCH codes.
- In 1963, Gorenstein–Zierler saw that BCH codes and **Reed–Solomon codes** have a common generalization and that the decoding algorithm extends to more general situation.
- In 1968 / 69, Elwyn Berlekamp invented an algorithm for decoding BCH codes. James Massey recognized its application to linear feedback shift registers and simplified the algorithm.<sup>[3][4]</sup> Massey termed the algorithm the LFSR Synthesis Algorithm (Berlekamp Iterative Algorithm) but it is now known as the **Berlekamp–Massey algorithm**.
- In 1975, Sugiyama *et al* developed a decoder based on the **extended Euclidean algorithm**.<sup>[5]</sup> **Reed–Solomon\_error\_correction#Euclidean\_decoder**
- In 1986, The Welch–Berlekamp algorithm was developed to solve the decoding equation of **Reed–Solomon codes**, using a fast method to solve a certain polynomial equation. The Berlekamp–Welch

algorithm has a running time complexity of  $\mathcal{O}(N^3)$ . The following sections look at the Gemmel and Sudan’s exposition of the Berlekamp–Welch algorithm.<sup>[6]</sup>

### 11.2 Error locator polynomial of Reed–Solomon codes

In the problem of decoding Reed–Solomon codes, the inputs are pair wise distinct evaluation points  $\alpha_1, \dots, \alpha_n$  where  $\alpha_i \in \mathbb{F}$  with **dimension**  $k$  and **distance**  $d = n - k + 1$  and a codeword  $y = (y_1, \dots, y_n) \in \mathbb{F}^n$ . Our goal is to describe an algorithm that can correct  $e < \frac{1}{2}(n - k + 1)$  many errors in polynomial time. To do so we have to find  $P \in \mathbb{F}[X]$  such that  $\deg(P) < k - 1$  and the number of indices for which  $P(\alpha_i) \neq y_i$  is less than or equal to  $e$ . We can assume that there exists a polynomial  $P$  such that

$$\Delta(y, (P(\alpha_i))_{i=1}^N) \leq e \leq \frac{d}{2} = \frac{1}{2}(n - k + 1).$$

Note that the coefficients of  $P$  are the encoded information. To solve this, we use an indicator for those indices where an error may have occurred. Thus we define an error locator polynomial,  $E \in \mathbb{F}[X]$ , by:

$$E(X) = \prod_{\substack{1 \leq i \leq n \\ y_i \neq P(\alpha_i)}} (X - \alpha_i)$$

Note that  $\deg(E) \leq \frac{1}{2}(n - k)$ . We can also claim that  $y_i E(\alpha_i) = P(\alpha_i) E(\alpha_i)$  holds for all  $1 \leq i \leq n$ . This fact holds true because in the event of  $y_i \neq P(\alpha_i)$ , both sides of the above equation vanish because  $E(\alpha_i) = 0$ .

However, since  $E$  and  $P$  are both unknown, the main task of the decoding algorithm would be to find  $P$ . To do this we use a seemingly useless yet very powerful method and define another polynomial  $Q = PE$ . This is because the  $n$  equations with  $e + k$  we need to solve are quadratic in nature. Thus by defining a product of two variables that gives rise to a quadratic term as one unknown variable, we increase the number of unknowns but make the equations

linear in nature. This method is called linearization<sup>[7]</sup> and is a very powerful tool.

Thus  $Q \in \mathbb{F}[X]$  having the properties:

1.  $\deg(Q) \leq \frac{1}{2}(n - k) + k - 1$
2.  $Q(\alpha_i) = E(\alpha_i)y_i, \quad 1 \leq i \leq n$

This helps because if we now manage to find  $Q$  and  $E$ , we can easily find  $P$  using  $P = \frac{Q}{E}$ . The main purpose of the Berlekamp Welch algorithm is to find out  $P$  using degree bounded polynomials  $Q$  and  $E$  and the properties of  $E$  and  $N$ .

Computing  $E$  is as hard as finding the end solution  $P$ . Once  $E$  is computed, using erasure decoding for Reed–Solomon codes, we can easily recover  $P$ . However, in a few cases, even the polynomial  $Q$  is as hard to find as  $E$ . As an example, given  $Q$  and  $y$  (such that  $y_i \neq 0$  for  $1 \leq i \leq n$ ), by checking positions where  $Q(i) = 0$ , we can find the error locations. Thus the algorithm works on the principle that while each of the polynomials  $E$  and  $Q$  are hard to find individually; computing them together is much easier.

### 11.3 The Berlekamp–Welch decoder and algorithm

The Welch–Berlekamp decoder for Reed–Solomon codes consists of the Welch–Berlekamp algorithm augmented by some additional steps that prepare the received word for the algorithm and interpret the result of the algorithm.

The inputs given to the Berlekamp Welch decoder are the integers denoting Block Length  $n$ , the number of errors  $e$  such that  $e < \frac{1}{2}(n - k + 1)$ , and the received word  $(y_i, \alpha_i)_{i=1}^n$  satisfying the condition that there exists at most one  $P$  with  $\deg(P) \leq k - 1$  with  $\Delta(y, P(\alpha_i)_i) \leq e$ .

The output of the decoder is either the polynomial  $P$ , or in some cases, a failure. This decoder functions in two steps as follows:

1. This step is called the interpolation step in which the decoder computes a non zero polynomial  $E$  of degree  $e$  (This implies that the coefficient of  $X^e$  must be 1<sup>[8]</sup>) and another polynomial  $Q$  with  $\deg(Q) \leq e + k - 1$ . These polynomials are created such that the condition  $y_i E(\alpha_i) = Q(\alpha_i)$  for all  $1 \leq i \leq n$ . In the case that polynomials satisfying the above condition cannot be computed, the output of the decoder would be a failure.
2. If  $E \mid Q$  then a  $P$  is defined which equals  $\frac{Q}{E}$ . If  $\Delta(y, (P(\alpha_i)_i) \leq e$ , then the decoder outputs  $P$ . If the above condition is not satisfied, i.e. if  $E \nmid Q$  then a failure is returned by the decoder.

According to the algorithm, in the cases where it does not output a failure, it outputs a  $P$  that is the correct and desired polynomial. To prove that, the algorithm always outputs the desired polynomial, we need to prove a few claims we have made while describing the algorithm. Let us go ahead and do so now.

**Claim 1.** There exist a pair of polynomials,  $(E, Q)$ , that satisfy Step 1 of the BW algorithm and  $\frac{Q}{E} = P$ .

Let  $E$  be the error-locating polynomial for  $P$ :

$$E(X) = X^{e - \Delta(y, P(\alpha_i)_i)} \prod_{\substack{1 \leq i \leq n \\ y_i \neq P(\alpha_i)}} (X - \alpha_i)$$

Notice that  $E$  has the following properties by definition:

$$\deg(E) = e, \quad E(\alpha_i) = 0 \Leftrightarrow y_i \neq P(\alpha_i).$$

Now define  $Q = PE$  and note that:

$$\deg(Q) \leq \deg(P) + \deg(E) \leq e + k - 1.$$

We can now claim that  $y_i E(\alpha_i) = Q(\alpha_i)$  from the first step of the BW algorithm holds. If  $E(\alpha_i) = 0$ , then  $Q(\alpha_i) = P(\alpha_i)E(\alpha_i) = y_i E(\alpha_i) = 0$ . For  $E(\alpha_i) \neq 0$  we have  $P(\alpha_i) = y_i$  and therefore  $Q(\alpha_i) = P(\alpha_i)E(\alpha_i) = y_i E(\alpha_i)$  just as we claimed.

This above claim however just reiterates and proves the fact that there exists a pair of polynomials  $E$  and  $Q$  such that  $P = \frac{Q}{E}$ . It however does not necessarily guarantee the fact that the algorithm we discussed above would indeed output such a pair of polynomials. We therefore move on to look at another claim that helps establish this fact using the above claim and thereby proving the correctness of the algorithm.

**Claim 2.** If  $(E_1, Q_1), (E_2, Q_2)$  are two distinct solutions that satisfy the first step of the Berlekamp Welch algorithm, then we have  $\frac{Q_1}{E_1} = \frac{Q_2}{E_2}$ .

First note that

$$\deg(Q_1 E_2), \deg(Q_2 E_1) \leq 2e + k - 1.$$

Then we define:

$$R := Q_1 E_2 - Q_2 E_1$$

Note that  $\deg(R) \leq 2e + k - 1$ . From step 1 of the Berlekamp Welch algorithm we also know that

$y_i E_1(\alpha_i) = Q_1(\alpha_i)$  and  $y_i E_2(\alpha_i) = Q_2(\alpha_i)$ . Now for all  $i \in \{1, \dots, n\}$  we calculate:

$$\begin{aligned} R(\alpha_i) &= Q_1(\alpha_i)E_2(\alpha_i) - Q_2(\alpha_i)E_1(\alpha_i) \\ &= y_i E_1(\alpha_i)E_2(\alpha_i) - y_i E_2(\alpha_i)E_1(\alpha_i) \\ &= 0 \end{aligned}$$

Thus  $R$  has  $n$  roots, on the other hand

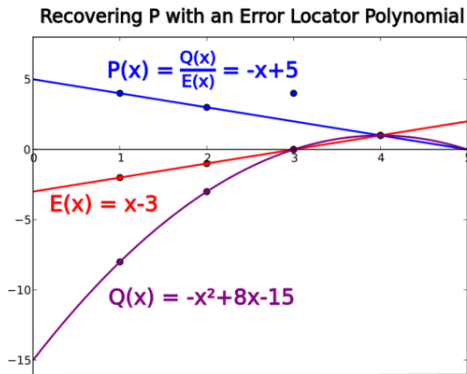
$$\deg(R(X)) \leq 2e + k - 1 < 2\frac{1}{2}(n - k + 1) + k - 1 = n.$$

Therefore,  $R$  is the zero polynomial which means that  $Q_1 E_2$  and  $Q_2 E_1$  are identical. Since  $E_1, E_2$  are non-zero we can write:  $\frac{Q_1}{E_1} = \frac{Q_2}{E_2}$  as per our initial claim.

Thus based on the above claims, we can safely state that the output of the Berlekamp Welch algorithm, when outputting the polynomial  $P(X)$  is correct.

We can now claim that the algorithm can be implemented such that it has a running time of  $O(n^3)$ . This can be proved as follows: In Step 1 of the algorithm, the polynomials  $Q$  and  $E$  have  $e + k$  and  $e + 1$  unknown values respectively and the constraints  $y_i E(\alpha_i) = Q(\alpha_i)$  for all  $1 \leq i \leq n$  acts as a linear equation with these unknowns. We therefore get a system of  $n$  linear equations in  $2e + k + 1 < n + 2$  unknowns. Using our first claim, this system of equations has a solution since  $\deg(E) = e$ . This can be solved in  $O(n^3)$  time, by say Gaussian elimination. Finally, we can note that Step 2 of the algorithm can also be implemented in time  $O(n^3)$  by “long division” method. Hence we can state that the Berlekamp Welch algorithm can be used to uniquely decode any  $[n, k]_q$  Reed–Solomon code in  $O(n^3)$  time for a maximum of  $\frac{1}{2}(n - k + 1)$  errors.

## 11.4 Example



The error locator polynomial serves to “neutralize” errors in  $P$  by making  $Q$  zero at those points, so that the system of linear equations is not affected by the inaccuracy in the input.

Consider a simple example where a redundant set of points are used to represent the line  $y = 5 - x$ , and one of the points is incorrect. The points that the algorithm gets as an input are  $(1, 4), (2, 3), (3, 4), (4, 1)$ , where  $(3, 4)$  is the defective point. The algorithm must solve the following system of equations:

$$\begin{aligned} Q(1) &= 4 * E(1) \\ Q(2) &= 3 * E(2) \\ Q(3) &= 4 * E(3) \\ Q(4) &= 1 * E(4) \end{aligned}$$

Given a solution pair  $(Q, E)$  to this system of equations, it is evident that at any of the points  $x = 1, 2, 3, 4$  one of the following must be true:

$$Q(\alpha_i) = E(\alpha_i) = 0, \quad \text{or} \quad P(\alpha_i) = \frac{Q(\alpha_i)}{E(\alpha_i)} = y_i.$$

Since  $E$  is defined as only having a degree of one, the former can only be true in one point. Therefore,  $P(\alpha_i) = y_i$  at the three other points.

Letting  $E(x) = x + e_0$  and  $Q = q_0 + q_1x + q_2x^2$  we can rewrite the system:

$$\begin{cases} q_0 + q_1 + q_2 - 4e_0 - 4 = 0 \\ q_0 + 2q_1 + 4q_2 - 3e_0 - 6 = 0 \\ q_0 + 3q_1 + 9q_2 - 4e_0 - 12 = 0 \\ q_0 + 4q_1 + 16q_2 - e_0 - 4 = 0 \end{cases}$$

This system can be solved through Gaussian elimination, and gives the values:

$$q_0 = -15, q_1 = 8, q_2 = -1, e_0 = -3$$

Thus:

$$Q = -x^2 + 8x - 15, E = x - 3, \quad \text{and} \quad \frac{Q}{E} = P = 5 - x.$$


$5 - x$  fits three of the four points given, so it is the most likely to be the original polynomial.

## 11.5 See also

- BCH code
- Berlekamp–Massey algorithm
- Reed–Solomon error correction



## 11.6 References

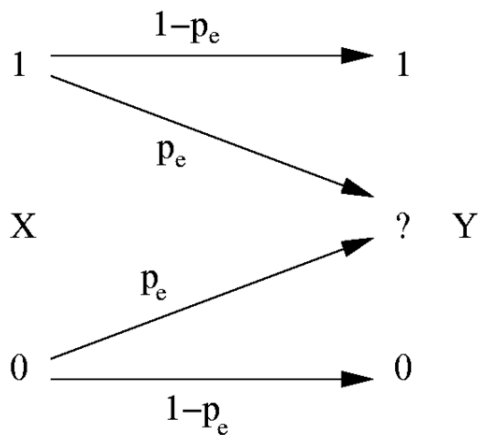
- [1] Berlekamp, Elwyn R. (1967), *Nonbinary BCH decoding*, International Symposium on Information Theory, San Remo, Italy
- [2] Berlekamp, Elwyn R. (1984) [1968], *Algebraic Coding Theory* (Revised ed.), Laguna Hills, CA: Aegean Park Press, ISBN 0-89412-063-8. Previous publisher McGraw-Hill, New York, NY.
- [3] Massey, J. L. (1969), “Shift-register synthesis and BCH decoding”, *IEEE Trans. Information Theory*, IT-15 (1): 122–127
- [4] Ben Atti, Nadia; Diaz-Toca, Gema M.; Lombardi, Henri, *The Berlekamp–Massey Algorithm revisited*, CiteSeerX 10.1.1.96.2743 
- [5] Sugiyama, Yasuo; Kasahara, Masao; Hirasawa, Shige-ichi; Namekawa, Toshihiko (1975), “A method for solving key equation for decoding Goppa codes”, *Information and Control*, **27** (1): 87–99, doi:10.1016/S0019-9958(75)90090-X
- [6] Highly resilient correctors for polynomials – Peter Gemmel and Madhu Sudan’s Exposition.
- [7] A provable example of the linearization method – Dick Lipton
- [8] Atri, Rudra. “Error Correcting Codes: Combinatorics, Algorithms and Applications” (PDF). *CSE\_BUFFALO*. Retrieved 2014-12-12.

## 11.7 External links

- MIT Lecture Notes on Essential Coding Theory – Dr. Madhu Sudan
- University at Buffalo Lecture Notes on Coding Theory – Dr. Atri Rudra
- Algebraic Codes on Lines, Planes and Curves, An Engineering Approach – Richard E. Blahut
- Welch Berlekamp Decoding of Reed–Solomon Codes – L. R. Welch
- US 4,633,470, Welch, Lloyd R. & Elwyn R. Berlekamp, “Error Correction for Algebraic Block Codes”, published September 27, 1983, issued December 30, 1986 – The patent by Lloyd R. Welch and Elwyn R. Berlekamp

## Chapter 12

# Binary erasure channel



*The channel model for the binary erasure channel showing a mapping from channel input  $X$  to channel output  $Y$  (with known erasure symbol ?). The probability of erasure is  $p_e$ .*

A **binary erasure channel** (or BEC) is a common communications channel model used in coding theory and information theory. In this model, a transmitter sends a bit (a zero or a one), and the receiver either receives the bit or it receives a message that the bit was not received ("erased"). This channel is used frequently in information theory because it is one of the simplest channels to analyze. The BEC was introduced by Peter Elias of MIT in 1955 as a toy example.

Closely related to the binary erasure channel is the **packet erasure channel** which shares many similar theoretical results with the binary erasure channel.

### 12.1 Description

The BEC is a *binary channel*; that is, it can transmit only one of two symbols (usually called 0 and 1). (A non-binary channel would be capable of transmitting more than two symbols, possibly even an infinite number of choices.) The channel is not perfect and sometimes the bit gets "erased"; that is, the bit gets scrambled so the receiver has no idea what the bit was.

The BEC is, in a sense, error-free. Unlike the **binary symmetric channel**, when the receiver gets a bit, it is 100% certain that the bit is correct. The only confusion arises when the bit is erased.

This channel is often used by theorists because it is one of the simplest **noisy channels** to analyze. Many problems in **communication theory** can be **reduced** to a BEC.

### 12.2 Definition

A **binary erasure channel with erasure probability  $P_e$**  is a channel with binary input, ternary output, and probability of erasure  $P_e$ . That is, let  $X$  be the transmitted random variable with alphabet  $\{0, 1\}$ . Let  $Y$  be the received variable with alphabet  $\{0, 1, e\}$ , where  $e$  is the erasure symbol. Then, the channel is characterized by the **conditional probabilities**

#### 12.2.1 Capacity of the BEC

The capacity of a BEC is  $1 - P_e$ .

Intuitively  $1 - P_e$  can be seen to be an upper bound on the channel capacity. Suppose there is an omniscient "genie" that tells the source whenever a transmitted bit gets erased. There is nothing the source can do to avoid erasure, but it can fix them when they happen. For example, the source could repeatedly transmit a bit until it gets through. There is no need for  $X$  to code, as  $Y$  will simply ignore erasures, knowing that the next successfully received bit is the one that  $X$  intended to send. Therefore, having a genie allows us to achieve a rate of  $1 - P_e$  on average. This additional information is not available normally and hence  $1 - P_e$  is an upper bound.

### 12.3 Deletion channel

The binary erasure channel should not be confused with a **deletion channel** where bits from the transmitter are either transmitted to the receiver (with probability  $p$ ) or dropped *without notifying the receiver* (with probability

$1 - p$ ). Determining the entropy of the deletion channel is an open problem.

## 12.4 See also

- Erasure code

## 12.5 References

- David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms* Cambridge: Cambridge University Press, 2003. ISBN 0-521-64298-1
- Thomas M. Cover, Joy A. Thomas. *Elements of information theory*, 1st Edition. New York: Wiley-Interscience, 1991. ISBN 0-471-06259-6.

## Chapter 13

# Binary Goppa code

In mathematics and computer science, the **binary Goppa code** is an error-correcting code that belongs to the class of general Goppa codes originally described by Valerii Denisovich Goppa, but the binary structure gives it several mathematical advantages over non-binary variants, also providing a better fit for common usage in computers and telecommunication. Binary Goppa codes have interesting properties suitable for cryptography in McEliece-like cryptosystems and similar setups.

alternant codes, thus alternant decoders can be used on this form. Such decoders usually provide only limited error-correcting capability (in most cases  $t/2$ ).

For practical purposes, parity-check matrix of a binary Goppa code is usually converted to a more computer-friendly binary form by a trace construction, that converts the  $t$ -by- $n$  matrix over  $GF(2^m)$  to a  $mt$ -by- $n$  binary matrix by writing polynomial coefficients of  $GF(2^m)$  elements on  $m$  successive rows.

### 13.1 Construction and properties

A binary Goppa code is defined by a polynomial  $g(x)$  of degree  $t$  over a finite field  $GF(2^m)$  without multiple zeros, and a sequence  $L$  of  $n$  distinct elements from  $GF(2^m)$  that aren't roots of the polynomial:

$$\forall i, j \in \{0, \dots, n-1\} : L_i \in GF(2^m) \wedge L_i \neq L_j \wedge g(L_i) \neq 0$$

Codewords belong to the kernel of syndrome function, forming a subspace of  $\{0, 1\}^n$  :

$$\Gamma(g, L) = \left\{ c \in \{0, 1\}^n \mid \sum_{i=0}^{n-1} \frac{c_i}{x - L_i} \equiv 0 \pmod{g(x)} \right\}$$

Code defined by a tuple  $(g, L)$  has minimum distance  $2t + 1$ , thus it can correct  $t = \left\lfloor \frac{(2t+1)-1}{2} \right\rfloor$  errors in a word of size  $n - mt$  using codewords of size  $n$ . It also possesses a convenient parity-check matrix  $H$  in form

$$H = VD = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ L_0^1 & L_1^1 & L_2^1 & \cdots & L_{n-1}^1 \\ L_0^2 & L_1^2 & L_2^2 & \cdots & L_{n-1}^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ L_0^t & L_1^t & L_2^t & \cdots & L_{n-1}^t \end{pmatrix} \begin{pmatrix} \frac{1}{g(L_0)} \\ \frac{1}{g(L_1)} \\ \frac{1}{g(L_2)} \\ \vdots \\ \frac{1}{g(L_{n-1})} \end{pmatrix}$$

Note that this form of the parity-check matrix, being composed of a Vandermonde matrix  $V$  and diagonal matrix  $D$ , shares the form with check matrices of

### 13.2 Decoding

Decoding of binary Goppa codes is traditionally done by Patterson algorithm, which gives good error-correcting capability (it corrects all  $t$  design errors), and is also fairly simple to implement.

Patterson algorithm converts a syndrome to a vector of errors. The syndrome of a word  $c = (c_0, \dots, c_{n-1})$  is expected to take a form of

$$s(x) \equiv \sum_{c_i=1} \frac{1}{x - L_i} \pmod{g(x)}$$

Alternative form of a parity-check matrix based on formula for  $s(x)$  can be used to produce such syndrome with a simple matrix multiplication.

The algorithm then computes  $v(x) \equiv \sqrt{s(x)^{-1} - x} \pmod{g(x)}$ . That fails when  $s(x) \equiv 0$ , but that is the case when the input word is a codeword, so no error correction is necessary.

$v(x)$  is reduced to polynomials  $a(x)$  and  $b(x)$  using the extended euclidean algorithm, so that  $a(x) \equiv b(x) \cdot v(x) \pmod{g(x)}$ , while  $\deg(a) \leq \lfloor t/2 \rfloor$  and  $\deg(b) \leq \lfloor (t-1)/2 \rfloor$ .

Finally, the error locator polynomial is computed as  $\sigma(x) = a(x)^2 + x \cdot b(x)^2$ . Note that in binary case, locating the errors is sufficient to correct them, as there's only one other value possible. Note that in all non-binary cases, separate error correction polynomial has to be computed as well.

If the original codeword was decodable and the  $e = (e_0, e_1, \dots, e_{n-1})$  was the error vector, then

$$\sigma(x) = \prod_{e_i=1} (x - L_i)$$

Factoring or evaluating all roots of  $\sigma(x)$  therefore gives enough information to recover the error vector and fix the errors.

### 13.3 Properties and usage

Binary Goppa codes viewed as a special case of Goppa codes have the interesting property that they correct full  $\deg(g)$  errors, while only  $\deg(g)/2$  errors in ternary and all other cases. Asymptotically, this error correcting capability meets the famous **Gilbert–Varshamov bound**.

Because of the high error correction capacity compared to code rate and form of parity-check matrix (which is usually hardly distinguishable from a random binary matrix of full rank), the binary Goppa codes are used in several **post-quantum cryptosystems**, notably **McEliece cryptosystem** and **Niederreiter cryptosystem**.

### 13.4 References

- Elwyn R. Berlekamp, Goppa Codes, IEEE Transactions on information theory, Vol. IT-19, No. 5, September 1973, [http://infosec.seu.edu.cn/space/kangwei/senior\\_thesis/Goppa.pdf](http://infosec.seu.edu.cn/space/kangwei/senior_thesis/Goppa.pdf)
- Daniela Engelbert, Raphael Overbeck, Arthur Schmidt. “A summary of McEliece-type cryptosystems and their security.” Journal of Mathematical Cryptology 1, 151–199. MR 2008h:94056. Previous version: <http://eprint.iacr.org/2006/162/>
- Daniel J. Bernstein. “List decoding for binary Goppa codes.” <http://cr.yp.to/codes/goppalist-20110303.pdf>

### 13.5 See also

- **BCH codes**
- **Code rate**
- **Reed–Solomon error correction**

## Chapter 14

# Binary symmetric channel

A **binary symmetric channel** (or BSC) is a common communications channel model used in coding theory and information theory. In this model, a transmitter wishes to send a bit (a zero or a one), and the receiver receives a bit. It is assumed that the bit is *usually* transmitted correctly, but that it will be “flipped” with a small probability (the “crossover probability”). This channel is used frequently in information theory because it is one of the simplest channels to analyze.

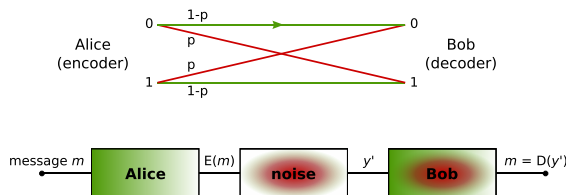
$$\Pr(Y = 0 | X = 1) = p$$

$$\Pr(Y = 1 | X = 0) = p$$

$$\Pr(Y = 1 | X = 1) = 1 - p$$

It is assumed that  $0 \leq p \leq 1/2$ . If  $p > 1/2$ , then the receiver can swap the output (interpret 1 when it sees 0, and vice versa) and obtain an equivalent channel with crossover probability  $1 - p \leq 1/2$ .

### 14.1 Description



The BSC is a *binary channel*; that is, it can transmit only one of two symbols (usually called 0 and 1). (A non-binary channel would be capable of transmitting more than 2 symbols, possibly even an infinite number of choices.) The transmission is not perfect, and occasionally the receiver gets the wrong bit.

This channel is often used by theorists because it is one of the simplest *noisy* channels to analyze. Many problems in *communication theory* can be *reduced* to a BSC. Conversely, being able to transmit effectively over the BSC can give rise to solutions for more complicated channels.

### 14.2 Definition

A **binary symmetric channel with crossover probability  $p$**  denoted by  $BSC_p$ , is a channel with binary input and binary output and probability of error  $p$ ; that is, if  $X$  is the transmitted *random variable* and  $Y$  the received variable, then the channel is characterized by the *conditional probabilities*

$$\Pr(Y = 0 | X = 0) = 1 - p$$

#### 14.2.1 Capacity of BSC $p$

The *capacity* of the channel is  $1 - H(p)$ , where  $H(p)$  is the *binary entropy function*.

The converse can be shown by a *sphere packing* argument. Given a codeword, there are roughly  $2^{nH(p)}$  typical output sequences. There are  $2^n$  total possible outputs, and the input chooses from a *codebook* of size  $2^{nR}$ . Therefore, the receiver would choose to *partition* the space into “spheres” with  $2^n / 2^{nR} = 2^{n(1-R)}$  potential outputs each. If  $R > 1 - H(p)$ , then the spheres will be packed too tightly *asymptotically* and the receiver will not be able to identify the correct codeword with vanishing probability.

### 14.3 Shannon’s channel capacity theorem for BSC $p$

Shannon’s noisy coding theorem is general for all kinds of channels. We consider a special case of this theorem for a binary symmetric channel with an error probability  $p$ .

#### 14.3.1 Noisy coding theorem for BSC $p$

The noise  $e$  that characterizes  $BSC_p$  is a *random variable* consisting of  $n$  independent random bits ( $n$  is defined below) where each random bit is a 1 with probability  $p$  and a 0 with probability  $1 - p$ . We indicate this by writing “ $e \in BSC_p$ ”.

**Theorem 1.** For all  $p < \frac{1}{2}$ , all  $0 < \epsilon < \frac{1}{2} - p$ , all sufficiently large  $n$  (depending on  $p$  and  $\epsilon$

, and all  $k \leq \lfloor (1 - H(p + \epsilon))n \rfloor$ , there exists a pair of **encoding** and **decoding** functions  $E : \{0, 1\}^k \rightarrow \{0, 1\}^n$  and  $D : \{0, 1\}^n \rightarrow \{0, 1\}^k$  respectively, such that every message  $m \in \{0, 1\}^k$  has the following property:

$$\Pr_{e \in BSC_p} [D(E(m) + e) \neq m] \leq 2^{-\delta n}$$

What this theorem actually implies is, a message when picked from  $\{0, 1\}^k$ , encoded with a random encoding function  $E$ , and sent across a noisy  $BSC_p$ , there is a very high probability of recovering the original message by decoding, if  $k$  or in effect the rate of the channel is bounded by the quantity stated in the theorem. The decoding error probability is exponentially small.

**Proof of Theorem 1.** First we describe the encoding function and decoding functions used in the theorem. We will use the **probabilistic method** to prove this theorem. Shannon's theorem was one of the earliest applications of this method.

**Encoding function:** Consider an encoding function  $E : \{0, 1\}^k \rightarrow \{0, 1\}^n$  that is selected at random. This means that for each message  $m \in \{0, 1\}^k$ , the value  $E(m) \in \{0, 1\}^n$  is selected at random (with equal probabilities).

**Decoding function:** For a given encoding function  $E$ , the decoding function  $D : \{0, 1\}^n \rightarrow \{0, 1\}^k$  is specified as follows: given any received codeword  $y \in \{0, 1\}^n$ , we find the message  $m \in \{0, 1\}^k$  such that the **Hamming distance**  $\Delta(y, E(m))$  is as small as possible (with ties broken arbitrarily). This kind of a decoding function is called a **maximum likelihood decoding (MLD)** function.

Ultimately, we will show (by integrating the probabilities) that at least one such choice  $(E, D)$  satisfies the conclusion of theorem; that is what is meant by the probabilistic method.

The proof runs as follows. Suppose  $p$  and  $\epsilon$  are fixed. First we show, for a fixed  $m \in \{0, 1\}^k$  and  $E$  chosen randomly, the probability of failure over  $BSC_p$  noise is exponentially small in  $n$ . At this point, the proof works for a fixed message  $m$ . Next we extend this result to work for *all*  $m$ . We achieve this by eliminating half of the codewords from the code with the argument that the proof for the decoding error probability holds for at least half of the codewords. The latter method is called **expurgation**. This gives the total process the name *random coding with expurgation*.

**A high level proof:** Fix  $p$  and  $\epsilon$ . Given a fixed message  $m \in \{0, 1\}^k$ , we need to estimate the **expected value** of the **probability** of the received codeword along with the noise does not give back  $m$  on decoding. That is to say, we need to estimate:

$$\mathbb{E}_E \left[ \Pr_{e \in BSC_p} [D(E(m) + e) \neq m] \right].$$

Let  $y$  be the received codeword. In order for the decoded codeword  $D(y)$  not to be equal to the message  $m$ , one of the following events must occur:

- $y$  does not lie within the Hamming ball of radius  $(p + \epsilon)n$  centered at  $E(m)$ . This condition is mainly used to make the calculations easier.
- There is another message  $m' \in \{0, 1\}^k$  such that  $\Delta(y, E(m')) \leq \Delta(y, E(m))$ . In other words the errors due to noise take the transmitted codeword closer to another encoded message.

We can apply **Chernoff bound** to ensure the non occurrence of the first event. By applying **Chernoff bound** we have,

$$\Pr_{e \in BSC_p} [\Delta(y, E(m)) > (p + \epsilon)n] \leq 2^{-\epsilon^2 n}.$$

This is exponentially small for large  $n$  (recall that  $\epsilon$  is fixed).

As for the second event, we note that the probability that  $E(m') \in B(y, (p + \epsilon)n)$  is  $\text{Vol}(B(y, (p + \epsilon)n)) / 2^n$  where  $B(x, r)$  is the Hamming ball of radius  $r$  centered at vector  $x$  and  $\text{Vol}(B(x, r))$  is its volume. Using approximation to estimate the number of codewords in the Hamming ball, we have  $\text{Vol}(B(y, (p + \epsilon)n)) \approx 2^{H(p)n}$ . Hence the above probability amounts to  $2^{H(p)n} / 2^n = 2^{H(p)n - n}$ . Now using **union bound**, we can upper bound the existence of such an  $m' \in \{0, 1\}^k$  by  $\leq 2^{k + H(p)n - n}$  which is  $2^{-\Omega(n)}$ , as desired by the choice of  $k$ .

**A detailed proof:** From the above analysis, we calculate the probability of the event that the decoded codeword plus the channel noise is not the same as the original message sent. We shall introduce some symbols here. Let  $p(y|E(m))$  denote the probability of receiving codeword  $y$  given that codeword  $E(m)$  was sent. Let  $B_0$  denote  $B(E(m), (p + \epsilon)n)$ .

$$\begin{aligned} \Pr_{e \in BSC_p} [D(E(m) + e) \neq m] &= \sum_{y \in \{0, 1\}^n} p(y|E(m)) \cdot 1_{D(y) \neq m} \\ &\leq \sum_{y \notin B_0} p(y|E(m)) \cdot 1_{D(y) \neq m} + \sum_{y \in B_0} p(y|E(m)) \cdot 1_{D(y) \neq m} \\ &\leq 2^{-\epsilon^2 n} + \sum_{y \in B_0} p(y|E(m)) \cdot 1_{D(y) \neq m} \end{aligned}$$

We get the last inequality by our analysis using the **Chernoff bound** above. Now taking expectation on both sides we have,



$$\begin{aligned}
\mathbb{E}_E \left[ \Pr_{e \in BSC_p} [D(E(m) + e) \neq m] \right] &\leq 2^{-\epsilon^2 n} + \sum_{y \in B_0} p(y|E(m)) \mathbb{E}[1_{D(y) \neq m}] \\
&\leq 2^{-\epsilon^2 n} + \sum_{y \in B_0} \mathbb{E}[1_{D(y) \neq m}] \\
&\leq 2^{-\epsilon^2 n} + 2^{k+H(p+\epsilon)n} \Pr_{e \in BSC_p} [D(E(m) + e) \neq m] \\
&\leq 2^{-\delta n}
\end{aligned}$$

by appropriately choosing the value of  $\delta$ . Since the above bound holds for **each** message, we have

$$\mathbb{E}_m \left[ \mathbb{E}_E \left[ \Pr_{e \in BSC_p} [D(E(m) + e)] \neq m \right] \right] \leq 2^{-\delta n}.$$

Now we can change the order of summation in the expectation with respect to the message and the choice of the encoding function  $E$ . Hence:

$$\mathbb{E}_E \left[ \mathbb{E}_m \left[ \Pr_{e \in BSC_p} [D(E(m) + e)] \neq m \right] \right] \leq 2^{-\delta n}.$$

Hence in conclusion, by probabilistic method, we have some encoding function  $E^*$  and a corresponding decoding function  $D^*$  such that

$$\mathbb{E}_m \left[ \Pr_{e \in BSC_p} [D^*(E^*(m) + e) \neq m] \right] \leq 2^{-\delta n}.$$

At this point, the proof works for a fixed message  $m$ . But we need to make sure that the above bound holds for **all** the messages  $m$  **simultaneously**. For that, let us sort the  $2^k$  messages by their decoding error probabilities. Now by applying **Markov's inequality**, we can show the decoding error probability for the first  $2^{k-1}$  messages to be at most  $2 \cdot 2^{-\delta n}$ . Thus in order to confirm that the above bound to hold for **every** message  $m$ , we could just trim off the last  $2^{k-1}$  messages from the sorted order. This essentially gives us another encoding function  $E'$  with a corresponding decoding function  $D'$  with a decoding error probability of at most  $2^{-\delta n+1}$  with the same rate. Taking  $\delta'$  to be equal to  $\delta - \frac{1}{n}$  we bound the decoding error probability to  $2^{-\delta' n}$ . This expurgation process completes the proof of Theorem 1.

## 14.4 Converse of Shannon's capacity theorem

The converse of the capacity theorem essentially states that  $1 - H(p)$  is the best rate one can achieve over a binary symmetric channel. Formally the theorem states:

**Theorem 2** If  $k \geq \lceil (1 - H(p + \epsilon))n \rceil$  then the following is true for every **encoding** and **decoding** function  $E$ :

$\{0, 1\}^k \rightarrow \{0, 1\}^n$  and  $D : \{0, 1\}^n \rightarrow \{0, 1\}^k$  respectively:  $\Pr_{e \in BSC_p} [D(E(m) + e) \neq m] \geq \frac{1}{2}$ .

For a detailed proof of this theorem, the reader is asked to refer to the bibliography. The intuition behind the proof is, however showing the **Number of errors** to grow rapidly as the rate grows beyond the channel capacity. The idea is, the sender generates messages of dimension  $k$ , while the channel  $BSC_p$  introduces transmission errors. When the capacity of the channel is  $H(p)$ , the number of errors is typically  $2^{H(p+\epsilon)n}$  for a code of block length  $n$ . The maximum number of messages is  $2^k$ . The output of the channel on the other hand has  $2^n$  possible values. If there is any confusion between any two messages, it is likely that  $2^k 2^{H(p+\epsilon)n} \geq 2^n$ . Hence we would have  $k \geq \lceil (1 - H(p + \epsilon))n \rceil$ , a case we would like to avoid to keep the decoding error probability exponentially small. (see above)

## 14.5 Codes for BSC<sub>p</sub>

Very recently, a lot of work has been done and is also being done to design explicit error-correcting codes to achieve the capacities of several standard communication channels. The motivation behind designing such codes is to relate the rate of the code with the fraction of errors which it can correct.

The approach behind the design of codes which meet the channel capacities of  $BSC$ ,  $BEC$  have been to correct a lesser number of errors with a high probability, and to achieve the highest possible rate. Shannon's theorem gives us the best rate which could be achieved over a  $BSC_p$ , but it does not give us an idea of any explicit codes which achieve that rate. In fact such codes are typically constructed to correct only a small fraction of errors with a high probability, but achieve a very good rate. The first such code was due to George D. Forney in 1966. The code is a concatenated code by concatenating two different kinds of codes. We shall discuss the construction Forney's code for the Binary Symmetric Channel and analyze its rate and decoding error probability briefly here. Various explicit codes for achieving the capacity of the binary erasure channel have also come up recently.

## 14.6 Forney's code for BSC<sub>p</sub>

Forney constructed a **concatenated** code  $C^* = C_{\text{out}} \circ C_{\text{in}}$  to achieve the capacity of Theorem 1 for  $BSC_p$ . In his code,

- The outer code  $C_{\text{out}}$  is a code of block length  $N$  and rate  $1 - \frac{\epsilon}{2}$  over the field  $F_{2^k}$ , and  $k = O(\log N)$ . Additionally, we have a **decoding** algorithm  $D_{\text{out}}$  for  $C_{\text{out}}$  which can correct up to  $\gamma$  fraction of worst case errors and runs in  $t_{\text{out}}(N)$  time.



- The inner code  $C_{\text{in}}$  is a code of block length  $n$ , dimension  $k$ , and a rate of  $1 - H(p) - \frac{\epsilon}{2}$ . Additionally, we have a decoding algorithm  $D_{\text{in}}$  for  $C_{\text{in}}$  with a **decoding** error probability of at most  $\frac{\gamma}{2}$  over  $BSC_p$  and runs in  $t_{\text{in}}(N)$  time.

For the outer code  $C_{\text{out}}$ , a Reed-Solomon code would have been the first code to have come in mind. However, we would see that the construction of such a code cannot be done in polynomial time. This is why a **binary linear code** is used for  $C_{\text{out}}$ .

For the inner code  $C_{\text{in}}$  we find a **linear code** by exhaustively searching from the **linear code** of block length  $n$  and dimension  $k$ , whose rate meets the capacity of  $BSC_p$ , by Theorem 1.

The rate  $R(C^*) = R(C_{\text{in}}) \times R(C_{\text{out}}) = (1 - \frac{\epsilon}{2})(1 - H(p) - \frac{\epsilon}{2}) \geq 1 - H(p) - \epsilon$  which almost meets the  $BSC_p$  capacity. We further note that the encoding and decoding of  $C^*$  can be done in polynomial time with respect to  $N$ . As a matter of fact, encoding  $C^*$  takes time  $O(N^2) + O(Nk^2) = O(N^2)$ . Further, the decoding algorithm described takes time  $Nt_{\text{in}}(k) + t_{\text{out}}(N) = N^{O(1)}$  as long as  $t_{\text{out}}(N) = N^{O(1)}$ ; and  $t_{\text{in}}(k) = 2^{O(k)}$ .

### 14.6.1 Decoding error probability for $C^*$

A natural decoding algorithm for  $C^*$  is to:

- Assume  $y'_i = D_{\text{in}}(y_i)$ ,  $i \in (0, N)$
- Execute  $D_{\text{out}}$  on  $y' = (y'_1 \dots y'_N)$

Note that each block of code for  $C_{\text{in}}$  is considered a symbol for  $C_{\text{out}}$ . Now since the probability of error at any index  $i$  for  $D_{\text{in}}$  is at most  $\frac{\gamma}{2}$  and the errors in  $BSC_p$  are independent, the expected number of errors for  $D_{\text{in}}$  is at most  $\frac{\gamma N}{2}$  by linearity of expectation. Now applying **Chernoff** bound, we have bound error probability of more than  $\gamma N$  errors occurring to be  $e^{-\frac{\gamma N}{6}}$ . Since the outer code  $C_{\text{out}}$  can correct at most  $\gamma N$  errors, this is the **decoding** error probability of  $C^*$ . This when expressed in asymptotic terms, gives us an error probability of  $2^{-\Omega(\gamma N)}$ . Thus the achieved decoding error probability of  $C^*$  is exponentially small as Theorem 1.

We have given a general technique to construct  $C^*$ . For more detailed descriptions on  $C_{\text{in}}$  and  $C_{\text{out}}$  please read the following references. Recently a few other codes have also been constructed for achieving the capacities. LDPC codes have been considered for this purpose for their faster decoding time.<sup>[1]</sup>

## 14.7 See also

- Z channel

## 14.8 Notes

- [1] Richardson and Urbanke

## 14.9 References

- David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms* Cambridge: Cambridge University Press, 2003. ISBN 0-521-64298-1
- Thomas M. Cover, Joy A. Thomas. *Elements of information theory*, 1st Edition. New York: Wiley-Interscience, 1991. ISBN 0-471-06259-6.
- Atri Rudra's course on Error Correcting Codes: Combinatorics, Algorithms, and Applications (Fall 2007), Lectures 9, 10, 29, and 30.
- Madhu Sudan's course on Algorithmic Introduction to Coding Theory (Fall 2001), Lecture 1 and 2.
- G. David Forney. *Concatenated Codes*. MIT Press, Cambridge, MA, 1966.
- Venkat Guruswamy's course on Error-Correcting Codes: Constructions and Algorithms, Autumn 2006.
- A mathematical theory of communication C. E Shannon, ACM SIGMOBILE Mobile Computing and Communications Review.
- *Modern Coding Theory* by Tom Richardson and Rudiger Urbanke., Cambridge University Press

## 14.10 External links

- A Java applet implementing Binary Symmetric Channel

# Chapter 15

## Blackwell channel

The **Blackwell channel** is a **deterministic broadcast channel** model used in **coding theory** and **information theory**. It was first proposed by mathematician **David Blackwell**.<sup>[1]</sup> In this model, a transmitter transmits one of three symbols to two receivers. For two of the symbols, both receivers receive exactly what was sent; the third symbol, however, is received differently at each of the receivers. This is one of the simplest examples of a non-trivial capacity result for a non-stochastic channel.

### 15.1 Definition

The Blackwell channel is composed of one input (transmitter) and two outputs (receivers). The channel input is ternary (three symbols) and is selected from  $\{0, 1, 2\}$ . This symbol is **broadcast** to the receivers; that is, the transmitter sends one symbol simultaneously to both receivers. Each of the channel outputs is **binary** (two symbols), labeled  $\{0, 1\}$ .

Whenever a **0** is sent, both outputs receive a **0**. Whenever a **1** is sent, both outputs receive a **1**. When a **2** is sent, however, the first output is **0** and the second output is **1**. Therefore, the symbol **2** is confused by each of the receivers in a different way.

The operation of the channel is **memoryless** and **completely deterministic**.

#### 15.1.1 Capacity of the Blackwell channel

The **capacity** of the channel was found by **S. I. Gel'fand**.<sup>[2][3]</sup> It is defined by the region:

1.  $R_1 = 1, 0 \leq R_2 \leq \frac{1}{2}$
2.  $R_1 = H(a), R_2 = 1 - a$ , for  $\frac{1}{3} \leq a \leq \frac{1}{2}$
3.  $R_1 + R_2 = \log_2 3, \log_2 3 - \frac{2}{3} \leq R_1 \leq \frac{2}{3}$
4.  $R_1 = 1 - a, R_2 = H(a)$ , for  $\frac{1}{3} \leq a \leq \frac{1}{2}$
5.  $0 \leq R_1 \leq \frac{1}{2}, R_2 = 1$

A solution was also found by Pinkser et al. (1995).<sup>[4]</sup>

### 15.2 References

- [1] L Breiman; D Blackwell; A J Thomasian (1958). "Proof of shannon's transmission theorem for finite-state indecomposable channels". *The Annals of Mathematical Statistics*. United States: Institute of Mathematical Statistics. **29**: 1209–2220. doi:10.1214/aoms/1177706452.
- [2] S I Gel'fand (1977). "Capacity of one broadcast channel". *Problemy Peredachi Informatsii*. Moscow, Russia: Russian Academy of Sciences, Branch of Informatics, Computer Equipment and Automatization. **13** (3): 106–108.
- [3] E van der Meulen (1977). "A survey of multi-way channels in information theory: 1961-1976" (PDF). *IEEE Transactions on Information Theory*. New York City, New York, United States: Institute of Electrical and Electronics Engineers. **23** (1): 1–37. doi:10.1109/tit.1977.1055652.
- [4] M Pinsker; S. Prelov; S. Verdú (November 1995). "Sensitivity of Channel Capacity". *IEEE Transactions on Information Theory*. New York City, New York, United States: Institute of Electrical and Electronics Engineers. **41** (6): 1877–1888. doi:10.1109/18.476313.

# Chapter 16

## Blahut–Arimoto algorithm

The **Blahut–Arimoto algorithm**, is often used to refer to a class of algorithms for computing numerically either the information theoretic capacity of a channel, or the rate-distortion function of a source. They are iterative algorithms that eventually converge to the optimal solution of the convex optimization problem that is associated with these information theoretic concepts.

For the case of **channel capacity**, the algorithm was independently invented by Arimoto and Blahut. In the case of lossy compression, the corresponding algorithm was invented by **Richard Blahut**. The algorithm is most applicable to the case of arbitrary finite alphabet sources. Much work has been done to extend it to more general problem instances.<sup>[1][2]</sup>

- [2] Iddo Naiss; Haim Permuter (2010). “Extension of the Blahut-Arimoto algorithm for maximizing directed information”. [arXiv:1012.5071v2](https://arxiv.org/abs/1012.5071v2) [cs.IT].

### 16.1 Algorithm

Suppose we have a source  $X$  with probability  $p(x)$  of any given symbol. We wish to find an encoding  $p(\hat{x}|x)$  that generates a compressed signal  $\hat{X}$  from the original signal while minimizing the expected distortion  $\langle d(x, \hat{x}) \rangle$ , where the expectation is taken over the joint probability of  $X$  and  $\hat{X}$ . We can find an encoding that minimizes the rate-distortion functional locally by repeating the following iteration until convergence:

$$p_{t+1}(\hat{x}) = \sum_x p(x) p_t(\hat{x}|x)$$

$$p_{t+1}(\hat{x}|x) = \frac{p_t(\hat{x}) \exp(-\beta d(x, \hat{x}))}{\sum_{\hat{x}} p_t(\hat{x}) \exp(-\beta d(x, \hat{x}))}$$

where  $\beta$  is a parameter related to the slope in the rate-distortion curve that we are targeting and thus is related to how much we favor compression versus distortion (higher  $\beta$  means less compression).

### 16.2 References

- [1] Pascal O. Vontobel (2002). “A Generalized Blahut–Arimoto Algorithm”. [CiteSeerX 10.1.1.1.2567](https://arxiv.org/abs/10.1.1.1.2567).

# Chapter 17

## Block code

In **coding theory**, a **block code** is any member of the large and important family of **error-correcting codes** that encode data in blocks. There is a vast number of examples for block codes, many of which have a wide range of practical applications. Block codes are conceptually useful because they allow coding theorists, **mathematicians**, and **computer scientists** to study the limitations of *all* block codes in a unified way. Such limitations often take the form of *bounds* that relate different parameters of the block code to each other, such as its rate and its ability to detect and correct errors.

Examples of block codes are Reed–Solomon codes, Hamming codes, Hadamard codes, Expander codes, Golay codes, and Reed–Muller codes. These examples also belong to the class of linear codes, and hence they are called **linear block codes**. More particularly, these codes are known as algebraic block codes, or cyclic block codes, because they can be generated using boolean polynomials.

Algebraic block codes are typically hard-decoded using algebraic decoders.

The term *block code* may also refer to any error-correcting code that acts on a block of  $k$  bits of input data to produce  $n$  bits of output data  $(n,k)$ . Consequently, the block coder is a *memoryless* device. Under this definition codes such as turbo codes, terminated convolutional codes and other iteratively decodable codes (turbo-like codes) would also be considered block codes. A non-terminated convolutional encoder would be an example of a non-block (unframed) code, which has *memory* and is instead classified as a *tree code*.

This article deals with “algebraic block codes”.

### 17.1 The block code and its parameters

Error-correcting codes are used to reliably transmit digital data over unreliable **communication channels** subject to **channel noise**. When a sender wants to transmit a possibly very long data stream using a block code, the sender breaks the stream up into pieces of some fixed size. Each such piece is called *message* and the procedure

given by the block code encodes each message individually into a codeword, also called a *block* in the context of block codes. The sender then transmits all blocks to the receiver, who can in turn use some decoding mechanism to (hopefully) recover the original messages from the possibly corrupted received blocks. The performance and success of the overall transmission depends on the parameters of the channel and the block code.

Formally, a block code is an **injective** mapping

$$C : \Sigma^k \rightarrow \Sigma^n$$

Here,  $\Sigma$  is a finite and nonempty set and  $k$  and  $n$  are integers. The meaning and significance of these three parameters and other parameters related to the code are described below.

#### 17.1.1 The alphabet $\Sigma$

The data stream to be encoded is modeled as a **string** over some **alphabet**  $\Sigma$ . The size  $|\Sigma|$  of the alphabet is often written as  $q$ . If  $q = 2$ , then the block code is called a **binary** block code. In many applications it is useful to consider  $q$  to be a **prime power**, and to identify  $\Sigma$  with the **finite field**  $\mathbb{F}_q$ .

#### 17.1.2 The message length $k$

Messages are elements  $m$  of  $\Sigma^k$ , that is, strings of length  $k$ . Hence the number  $k$  is called the **message length** or **dimension** of a block code.

#### 17.1.3 The block length $n$

The **block length**  $n$  of a block code is the number of symbols in a block. Hence, the elements  $c$  of  $\Sigma^n$  are strings of length  $n$  and correspond to blocks that may be received by the receiver. Hence they are also called received words. If  $c = C(m)$  for some message  $m$ , then  $c$  is called the codeword of  $m$ .

### 17.1.4 The rate $R$

The **rate** of a block code is defined as the ratio between its message length and its block length:

$$R = k/n$$

A large rate means that the amount of actual message per transmitted block is high. In this sense, the rate measures the transmission speed and the quantity  $1 - R$  measures the overhead that occurs due to the encoding with the block code. It is a simple **information theoretical** fact that the rate cannot exceed 1 since data cannot in general be losslessly compressed. Formally, this follows from the fact that the code  $C$  is an injective map.

### 17.1.5 The distance $d$

The **distance** or **minimum distance**  $d$  of a block code is the minimum number of positions in which any two distinct codewords differ, and the **relative distance**  $\delta$  is the fraction  $d/n$ . Formally, for received words  $c_1, c_2 \in \Sigma^n$ , let  $\Delta(c_1, c_2)$  denote the **Hamming distance** between  $c_1$  and  $c_2$ , that is, the number of positions in which  $c_1$  and  $c_2$  differ. Then the minimum distance  $d$  of the code  $C$  is defined as

$$d := \min_{m_1, m_2 \in \Sigma^k; m_1 \neq m_2} \Delta[C(m_1), C(m_2)]$$

Since any code has to be **injective**, any two codewords will disagree in at least one position, so the distance of any code is at least 1. Besides, the **distance** equals the **minimum weight** for linear block codes because:

$$\min_{m_1, m_2 \in \Sigma^k; m_1 \neq m_2} \Delta[C(m_1), C(m_2)] = \min_{m \in \Sigma^k; m \neq 0} \Delta[C(m), C(0)]$$

A larger distance allows for more error correction and detection. For example, if we only consider errors that may change symbols of the sent codeword but never erase or add them, then the number of errors is the number of positions in which the sent codeword and the received word differ. A code with distance  $d$  allows the receiver to detect up to  $d - 1$  transmission errors since changing  $d - 1$  positions of a codeword can never accidentally yield another codeword. Furthermore, if no more than  $(d - 1)/2$  transmission errors occur, the receiver can uniquely decode the received word to a codeword. This is because every received word has at most one codeword at distance  $(d - 1)/2$ . If more than  $(d - 1)/2$  transmission errors occur, the receiver cannot uniquely decode the received word in general as there might be several possible codewords. One way for the receiver to cope with this situation is to use **list-decoding**, in which the decoder outputs a list of all codewords in a certain radius.

### 17.1.6 Popular notation

The notation  $(n, k, d)_q$  describes a block code over an alphabet  $\Sigma$  of size  $q$ , with a block length  $n$ , message length  $k$ , and distance  $d$ . If the block code is a linear block code, then the square brackets in the notation  $[n, k, d]_q$  are used to represent that fact. For binary codes with  $q = 2$ , the index is sometimes dropped. For **maximum distance separable codes**, the distance is always  $d = n - k + 1$ , but sometimes the precise distance is not known, non-trivial to prove or state, or not needed. In such cases, the  $d$ -component may be missing.

Sometimes, especially for non-block codes, the notation  $(n, M, d)_q$  is used for codes that contain  $M$  codewords of length  $n$ . For block codes with messages of length  $k$  over an alphabet of size  $q$ , this number would be  $M = q^k$ .

## 17.2 Examples

As mentioned above, there are a vast number of error-correcting codes that are actually block codes. The first error-correcting code was the **Hamming(7,4)-code**, developed by **Richard W. Hamming** in 1950. This code transforms a message consisting of 4 bits into a codeword of 7 bits by adding 3 parity bits. Hence this code is a block code. It turns out that it is also a linear code and that it has distance 3. In the shorthand notation above, this means that the Hamming(7,4)-code is a  $[7, 4, 3]_2$ -code.

**Reed–Solomon codes** are a family of  $[n, k, d]_q$ -codes with  $d = n - k + 1$  and  $q$  being a **prime power**. **Rank codes** are family of  $[n, k, d]_q$ -codes with  $d \leq n - k + 1$ . **Hadamard codes** are a family of  $[n, k, d]_2$ -codes with  $n = 2^{k-1}$  and  $d = 2^{k-2}$ .

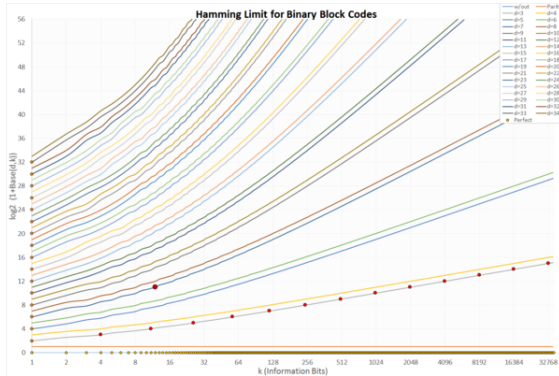
## 17.3 Error detection and correction properties

A codeword  $c \in \Sigma^n$  could be considered as a point in the  $n$ -dimension space  $\Sigma^n$  and the code  $C$  is the subset of  $\Sigma^n$ . A code  $C$  has distance  $d$  means that  $\forall c \in C$ , there is no other codeword in the **Hamming ball** centered at  $c$  with radius  $d - 1$ , which is defined as the collection of  $n$ -dimension words whose **Hamming distance** to  $c$  is no more than  $d - 1$ . Similarly,  $C$  with (minimum) distance  $d$  has the following properties:

- $C$  can detect  $d - 1$  errors : Because a codeword  $c$  is the only codeword in the Hamming ball centered at itself with radius  $d - 1$ , no error pattern of  $d - 1$  or fewer errors could change one codeword to another. When the receiver detects that the received vector is not a codeword of  $C$ , the errors are detected (but no guarantee to correct).

- $\mathcal{C}$  can correct  $\lfloor \frac{d-1}{2} \rfloor$  errors. Because a codeword  $c$  is the only codeword in the Hamming ball centered at itself with radius  $d-1$ , the two Hamming balls centered at two different codewords respectively with both radius  $\lfloor \frac{d-1}{2} \rfloor$  do not overlap with each other. Therefore, if we consider the error correction as finding the codeword closest to the received word  $y$ , as long as the number of errors is no more than  $\lfloor \frac{d-1}{2} \rfloor$ , there is only one codeword in the hamming ball centered at  $y$  with radius  $\lfloor \frac{d-1}{2} \rfloor$ , therefore all errors could be corrected.
- In order to decode in the presence of more than  $(d-1)/2$  errors, **list-decoding** or **maximum likelihood decoding** can be used.
- $\mathcal{C}$  can correct  $d-1$  erasures. By *erasure* it means that the position of the erased symbol is known. Correcting could be achieved by  $q$ -passing decoding: In  $i^{th}$  passing the erased position is filled with the  $i^{th}$  symbol and error correcting is carried out. There must be one passing that the number of errors is no more than  $\lfloor \frac{d-1}{2} \rfloor$  and therefore the erasures could be corrected.

## 17.4 Lower and upper bounds of block codes



Hamming Limit

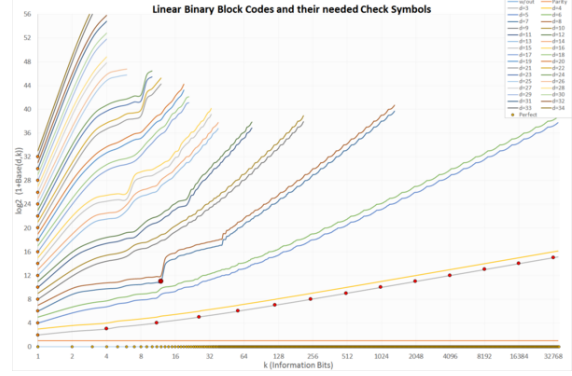
### 17.4.1 Family of codes

$\mathcal{C} = \{C_i\}_{i \geq 1}$  is called *family of codes*, where  $C_i$  is an  $(n_i, k_i, d_i)_q$  code with monotonic increasing  $n_i$ .

**Rate** of family of codes  $\mathcal{C}$  is defined as  $R(\mathcal{C}) = \lim_{i \rightarrow \infty} \frac{k_i}{n_i}$

**Relative distance** of family of codes  $\mathcal{C}$  is defined as  $\delta(\mathcal{C}) = \lim_{i \rightarrow \infty} \frac{d_i}{n_i}$

To explore the relationship between  $R(\mathcal{C})$  and  $\delta(\mathcal{C})$ , a set of lower and upper bounds of block codes are known.



There theoretical limits (such as the hamming limit), but another question is which codes can actually constructed. It is like *packing spheres in a box in many dimensions ...* This diagram shows the constructable codes, which are linear and binary. The x-axis shows the number of protected symbols  $k$ , the y-axis the number of needed check symbols  $n-k$ . Plotted are the limits for different Hamming distances from 1 (unprotected) to 34. Marked with dots are perfect codes:

- light orange on x-axis: trivial unprotected codes
- orange on y-axis: trivial repeat codes
- dark orange on data set  $d=3$ : classic perfect hamming codes
- dark red and larger: the only perfect binary Golay code

### 17.4.2 Hamming bound

$$R \leq 1 - \frac{1}{n} \cdot \log_q \left[ \sum_{i=0}^{\lfloor \frac{\delta \cdot n - 1}{2} \rfloor} \binom{n}{i} (q-1)^i \right]$$

### 17.4.3 Singleton bound

The Singleton bound is that the sum of the rate and the relative distance of a block code cannot be much larger than 1:

$$R + \delta \leq 1 + \frac{1}{n}$$

In other words, every block code satisfies the inequality  $k + d \leq n + 1$ . **Reed–Solomon codes** are non-trivial examples of codes that satisfy the singleton bound with equality.

### 17.4.4 Plotkin bound

For  $q = 2$ ,  $R + 2\delta \leq 1$ . In other words,  $k + 2d \leq n$ .

For the general case, the following Plotkin bounds holds for any  $\mathcal{C} \subseteq \mathbb{F}_q^n$  with distance  $d$ :

1. If  $d = (1 - \frac{1}{q})n$ ,  $|\mathcal{C}| \leq 2qn$



2. If  $d > (1 - \frac{1}{q})n$ ,  $|C| \leq \frac{qd}{qd - (q-1)n}$

For any  $q$ -ary code with distance  $\delta$ ,  $R \leq 1 - (\frac{q}{q-1})\delta + o(1)$

### 17.4.5 Gilbert–Varshamov bound

$R \geq 1 - H_q(\delta) - \epsilon$ , where  $0 \leq \delta \leq 1 - \frac{1}{q}$ ,  $0 \leq \epsilon \leq 1 - H_q(\delta)$ ,  $H_q(x) \equiv_{def} -x \cdot \log_q \frac{x}{q-1} - (1-x) \cdot \log_q (1-x)$  is the  $q$ -ary entropy function.

### 17.4.6 Johnson bound

Define  $J_q(\delta) \equiv_{def} (1 - \frac{1}{q})(1 - \sqrt{1 - \frac{q\delta}{q-1}})$ .

Let  $J_q(n, d, e)$  be the maximum number of codewords in a Hamming ball of radius  $e$  for any code  $C \subseteq \mathbb{F}_q^n$  of distance  $d$ .

Then we have the *Johnson Bound*:  $J_q(n, d, e) \leq qnd$ , if  $\frac{e}{n} \leq \frac{q-1}{q} \left(1 - \sqrt{1 - \frac{q}{q-1} \cdot \frac{d}{n}}\right) = J_q(\frac{d}{n})$

### 17.4.7 Elias–Bassalygo bound

$$R = \frac{\log_q |C|}{n} \leq 1 - H_q(J_q(\delta)) + o(1)$$

## 17.5 Sphere packings and lattices

Block codes are tied to the **sphere packing problem** which has received some attention over the years. In two dimensions, it is easy to visualize. Take a bunch of pennies flat on the table and push them together. The result is a hexagon pattern like a bee's nest. But block codes rely on more dimensions which cannot easily be visualized. The powerful **Golay code** used in deep space communications uses 24 dimensions. If used as a binary code (which it usually is), the dimensions refer to the length of the codeword as defined above.

The theory of coding uses the  $N$ -dimensional sphere model. For example, how many pennies can be packed into a circle on a tabletop or in 3 dimensions, how many marbles can be packed into a globe. Other considerations enter the choice of a code. For example, hexagon packing into the constraint of a rectangular box will leave empty space at the corners. As the dimensions get larger, the percentage of empty space grows smaller. But at certain dimensions, the packing uses all the space and these codes are the so-called perfect codes. There are very few of these codes.

Another property is the number of neighbors a single codeword may have.<sup>[1]</sup> Again, consider pennies as an example. First we pack the pennies in a rectangular grid. Each penny will have 4 near neighbors (and 4 at the corners which are farther away). In a hexagon, each penny

will have 6 near neighbors. Respectively, in three and four dimensions, the maximum packing is given by the **12-face** and **24-cell** with 12 and 24 neighbors, respectively. When we increase the dimensions, the number of near neighbors increases very rapidly. In general, the value is given by the **kissing numbers**.

The result is that the number of ways for noise to make the receiver choose a neighbor (hence an error) grows as well. This is a fundamental limitation of block codes, and indeed all codes. It may be harder to cause an error to a single neighbor, but the number of neighbors can be large enough so the total error probability actually suffers.<sup>[1]</sup>

## 17.6 See also

- Channel Capacity
- Shannon–Hartley theorem
- Noisy channel
- List decoding
- Sphere packing

## 17.7 References

- [1] Christian Schlegel and Lance Pérez (2004). *Trellis and turbo coding*. Wiley-IEEE. p. 73. ISBN 978-0-471-22755-7.
- J.H. van Lint (1992). *Introduction to Coding Theory*. GTM. **86** (2nd ed.). Springer-Verlag. p. 31. ISBN 3-540-54894-7.
  - F.J. MacWilliams; N.J.A. Sloane (1977). *The Theory of Error-Correcting Codes*. North-Holland. p. 35. ISBN 0-444-85193-3.
  - W. Huffman; V.Pless (2003). *Fundamentals of error-correcting codes*. Cambridge University Press. ISBN 978-0-521-78280-7.
  - S. Lin; D. J. Jr. Costello (1983). *Error Control Coding: Fundamentals and Applications*. Prentice-Hall. ISBN 0-13-283796-X.

## 17.8 External links

- Charan Langton (2001) **Coding Concepts and Block Coding**

# Chapter 18

## Burst error-correcting code

In coding theory, **burst error-correcting codes** employ methods of correcting **burst errors**, which are errors that occur in many consecutive bits rather than occurring in bits independently of each other.

Many codes have been designed to correct **random errors**. Sometimes, however, channels may introduce errors which are localized in a short interval. Such errors occur in a burst (called **burst errors**) because they occur in many consecutive bits. Examples of burst errors can be found extensively in storage mediums. These errors may be due to physical damage such as scratch on a disc or a stroke of lightning in case of wireless channels. They are not independent; they tend to be spatially concentrated. If one bit has an error, it is likely that the adjacent bits could also be corrupted. The methods used to correct random errors are inefficient to correct burst errors.

### 18.1 Definitions

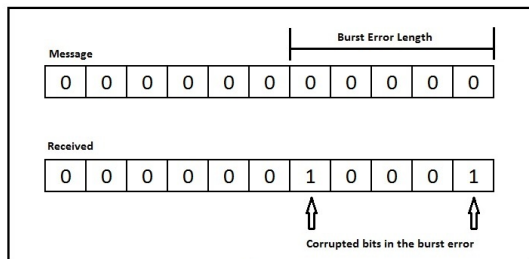


Figure 1

A burst of length 5

#### A burst of length $\ell$ <sup>[1]</sup>

Say a codeword  $C$  is transmitted, and it is received as  $Y = C + E$ . Then, the error vector  $E$  is called a burst of length  $\ell$  if the nonzero components of  $E$  are confined to  $\ell$  consecutive components. For example,  $E = (010000110)$  is a burst of length  $\ell = 7$ .

Although this definition is sufficient to describe what a burst error is, the majority of the tools developed for burst error correction rely on cyclic codes. This motivates our next definition.

#### A cyclic burst of length $\ell$ <sup>[1]</sup>

An error vector  $E$  is called a cyclic burst error of length  $\ell$  if its nonzero components are confined to  $\ell$  cyclically consecutive components. For example, the previously considered error vector  $E = (010000110)$ , is a cyclic burst of length  $\ell = 5$ , since we consider the error starting at position 6 and ending at position 1. Notice the indices are 0-based, that is, the first element is at position 0.

For the remainder of this article, we will use the term burst to refer to a cyclic burst, unless noted otherwise.

#### 18.1.1 Burst description

It is often useful to have a compact definition of a burst error, that encompasses not only its length, but also the pattern, and location of such error. We define a burst description to be a tuple  $(P, L)$  where  $P$  is the pattern of the error (that is the string of symbols beginning with the first nonzero entry in the error pattern, and ending with the last nonzero symbol), and  $L$  is the location, on the codeword, where the burst can be found. <sup>[1]</sup>

For example, the burst description of the error pattern  $E = (010000110)$  is  $D = (1000011, 1)$ . Notice that such description is not unique, because  $D' = (11001, 6)$  describes the same burst error. In general, if the number of nonzero components in  $E$  is  $w$ , then  $E$  will have  $w$  different burst descriptions each starting at a different nonzero entry of  $E$ . To remedy the issues that arise by the ambiguity of burst descriptions with the theorem below, however before doing so we need a definition first.

**Definition.** The number of symbols in a given error pattern  $y$ , is denoted by  $\text{length}(y)$ .

**Theorem (Uniqueness of burst descriptions).** Suppose  $E$  is an error vector of length  $n$  with two burst descriptions  $(P_1, L_1)$  and  $(P_2, L_2)$ . If  $\text{length}(P_1) + \text{length}(P_2) \leq n + 1$ , then the two descriptions are identical that is, their components are equivalent. <sup>[2]</sup>

**Proof.** Let  $w$  be the **hamming weight** (or the number of nonzero entries) of  $E$ . Then  $E$  has exactly  $w$  error descriptions. For  $w = 0, 1$ , there is nothing to prove. So we assume that



$w \geq 2$  and that the descriptions are not identical. We notice that each nonzero entry of  $E$  will appear in the pattern, and so, the components of  $E$  not included in the pattern will form a cyclic run of zeros, beginning after the last nonzero entry, and continuing just before the first nonzero entry of the pattern. We call the set of indices corresponding to this run as the zero run. We immediately observe that each burst description has a zero run associated with it and that each zero run is disjoint. Since we have  $w$  zero runs, and each is disjoint, we have a total of  $n - w$  distinct elements in all the zero runs. On the other hand we have:

$$\begin{aligned} n - w &= \text{in zeros of number } E = (n - \text{length}(P_1)) + (n - \text{length}(P_2)) \\ &= 2n - (\text{length}(P_1) + \text{length}(P_2)) \\ &\geq 2n - (n + 1) \\ &= n - 1 \end{aligned}$$

This contradicts  $w \geq 2$ . Thus, the burst error descriptions are identical.

A corollary of the above theorem is that we cannot have two distinct burst descriptions for bursts of length  $\frac{1}{2}(n + 1)$ .

## 18.2 Cyclic codes for burst error correction

**Cyclic codes** are defined as follows: think of the  $q$  symbols as elements in  $\mathbb{F}_q$ . Now, we can think of words as polynomials over  $\mathbb{F}_q$ , where the individual symbols of a word correspond to the different coefficients of the polynomial. To define a cyclic code, we pick a fixed polynomial, called **generator polynomial**. The codewords of this cyclic code are all the polynomials that are divisible by this generator polynomial.

Codewords are polynomials of degree  $\leq n - 1$ . Suppose that the generator polynomial  $g(x)$  has degree  $r$ . Polynomials of degree  $\leq n - 1$  that are divisible by  $g(x)$  result from multiplying  $g(x)$  by polynomials of degree  $\leq n - 1 - r$ . We have  $q^{n-r}$  such polynomials. Each one of them corresponds to a codeword. Therefore,  $k = n - r$  for cyclic codes.

Cyclic codes can detect all bursts of length up to  $\ell = n - k = r$ . We will see later that the burst error detection ability of any  $(n, k)$  code is bounded from above by  $\ell \leq n - k$ . Cyclic codes are considered optimal for burst error detection since they meet this upper bound:

**Theorem (Cyclic burst correction capability).** Every cyclic code with generator polynomial of degree  $r$  can detect all bursts of length  $\leq r$ .

**Proof.** We need to prove that if you add a burst of length  $\leq r$  to a codeword (i.e. to a polynomial that is divisible by  $g(x)$ ), then the result is not going to be a codeword (i.e. the corresponding polynomial is not divisible by  $g(x)$ ). It suffices to show that no burst of length  $\leq r$  is divisible by  $g(x)$ . Such a burst has the form  $x^i b(x)$ , where  $\deg(b(x)) < r$ . Therefore,  $b(x)$  is not divisible by  $g(x)$  (because the latter has degree  $r$ ).  $g(x)$  is not divisible by  $x$  (Otherwise, all codewords would start with 0). Therefore,  $x^i$  is not divisible by  $g(x)$  as well.

The above proof suggests a simple algorithm for burst error detection/correction in cyclic codes: given a transmitted word (i.e. a polynomial of degree  $\leq n - 1$ ), compute the remainder of this word when divided by  $g(x)$ . If the remainder is zero (if the word is divisible by  $g(x)$ ), then it is a valid codeword. Otherwise, report an error. To correct this error, subtract this remainder from the transmitted word. The subtraction result is going to be divisible by  $g(x)$  (i.e. it is going to be a valid codeword).

By the upper bound on burst error detection ( $\ell \leq n - k = r$ ), we know that a cyclic code can not detect *all* bursts of length  $\ell > r$ . However cyclic codes can indeed detect *most* bursts of length  $> r$ . The reason is that detection fails only when the burst is divisible by  $g(x)$ . Over binary alphabets, there exist  $2^{\ell-2}$  bursts of length  $\ell$ . Out of those, only  $2^{\ell-2-r}$  are divisible by  $g(x)$ . Therefore, the detection failure probability is very small ( $2^{-r}$ ) assuming a uniform distribution over all bursts of length  $\ell$ .

We now consider a fundamental theorem about cyclic codes that will aid in designing efficient burst-error correcting codes, by categorizing bursts into different cosets.

**Theorem (Distinct Cosets).** A linear code  $C$  is an  $\ell$ -burst-error-correcting code if all the burst errors of length  $\leq \ell$  lie in distinct cosets of  $C$ .

**Proof.** Let  $\mathbf{e}_1, \mathbf{e}_2$  be distinct burst errors of length  $\leq \ell$  which lie in same coset of code  $C$ . Then  $\mathbf{c} = \mathbf{e}_1 - \mathbf{e}_2$  is a codeword. Hence, if we receive  $\mathbf{e}_1$ , we can decode it either to  $\mathbf{0}$  or  $\mathbf{c}$ . In contrast, if all the burst errors  $\mathbf{e}_1$  and  $\mathbf{e}_2$  do not lie in same coset, then each burst error is determined by its syndrome. The error can then be corrected through its syndrome. Thus, a linear code  $C$  is an  $\ell$ -burst-error-correcting code if and only if all the burst errors of length  $\leq \ell$  lie in distinct cosets of  $C$ .

**Theorem (Burst error codeword classification).** Let  $C$  be a linear  $\ell$ -burst-error-correcting code. Then no nonzero burst of length  $\leq 2\ell$  can be a codeword.

**Proof.** Let  $c$  be a codeword with a burst of length  $\leq 2\ell$ . Thus it has the pattern  $(0, 1, u, v, 1, 0)$ , where  $u$  and  $v$  are words of length  $\leq \ell - 1$ . Hence, the words  $w = (0, 1, u, 0, 0, 0)$  and  $c - w = (0, 0, 0, v, 1, 0)$  are two bursts of length  $\leq \ell$ . For binary linear codes, they belong to the same coset. This contradicts the Distinct Cosets Theorem, therefore no nonzero burst of length  $\leq 2\ell$  can be a codeword.

## 18.3 Burst error correction bounds

### 18.3.1 Upper bounds on burst error detection and correction

By upper bound, we mean a limit on our error detection ability that we can never go beyond. Suppose that we want to design an  $(n, k)$  code that can detect all burst errors of length  $\leq \ell$ . A natural question to ask is: given  $n$  and  $k$ , what is the maximum  $\ell$  that we can never achieve beyond? In other words, what is the upper bound on the length  $\ell$  of bursts that we can detect using any  $(n, k)$  code? The following theorem provides an answer to this question.

**Theorem (Burst error detection ability).**

The burst error detection ability of any  $(n, k)$  code is  $\ell \leq n - k$ .

**Proof.** First we observe that a code can detect all bursts of length  $\leq \ell$  if and only if no two codewords differ by a burst of length  $\leq \ell$ . Suppose that we have two code words  $c_1$  and  $c_2$  that differ by a burst  $b$  of length  $\leq \ell$ . Upon receiving  $c_1$ , we can not tell whether the transmitted word is indeed  $c_1$  with no transmission errors, or whether it is  $c_2$  with a burst error  $b$  that occurred during transmission. Now, suppose that every two codewords differ by more than a burst of length  $\ell$ . Even if the transmitted codeword  $c_1$  is hit by a burst  $b$  of length  $\ell$ , it is not going to change into another valid codeword. Upon receiving it, we can tell that this is  $c_1$  with a burst  $b$ . By the above observation, we know that no two codewords can share the first  $n - \ell$  symbols. The reason is that even if they differ in all the other  $\ell$  symbols, they are still going to be different by a burst of length  $\ell$ . Therefore, the number of codewords  $q^k$  satisfies  $q^k \leq q^{n-\ell}$ . Applying  $\log_q$  to both sides and rearranging, we can see that  $\ell \leq n - k$ .

Now, we repeat the same question but for error correction: given  $n$  and  $k$ , what is the upper bound on the length  $\ell$  of bursts that we can correct using any  $(n, k)$  code? The following theorem provides a preliminary answer to this question:

**Theorem (Burst error correction ability).**

The burst error correction ability of any  $(n, k)$  code satisfies  $\ell \leq n - k - \log_q(n - \ell) + 2$

**Proof.** First we observe that a code can correct all bursts of length  $\leq \ell$  if and only if no two codewords differ by the sum of two bursts of length  $\leq \ell$ . Suppose that two codewords  $c_1$  and  $c_2$  differ by bursts  $b_1$  and  $b_2$  of length  $\leq \ell$  each. Upon receiving  $c_1$  hit by a burst  $b_1$ , we could interpret that as if it was  $c_2$  hit by a burst  $-b_2$ . We can not tell whether the transmitted word is  $c_1$  or  $c_2$ . Now, suppose that every two codewords differ by more than two bursts of length  $\ell$ . Even if the transmitted codeword  $c_1$  is hit by a burst of length  $\ell$ , it is not going to look like another codeword that has been hit by another burst. For each codeword  $c$ , let  $B(c)$  denote the set of all words that differ from  $c$  by a burst of length  $\leq \ell$ . Notice that  $B(c)$  includes  $c$  itself. By the above observation, we know that for two different codewords  $c_i$  and  $c_j$ ,  $B(c_i)$  and  $B(c_j)$  are disjoint. We have  $q^k$  codewords. Therefore, we can say that  $q^k |B(c)| \leq q^n$ . Moreover, we have  $(n - \ell)q^{\ell-2} \leq |B(c)|$ . By plugging the latter inequality into the former, then taking the base  $q$  logarithm and rearranging, we get the above theorem.

A stronger result is given by the Rieger bound:

**Theorem (Rieger bound).** If  $\ell$  is the burst error correcting ability of an  $(n, k)$  linear block code, then  $2\ell \leq n - k$ .

**Proof.** Any linear code that can correct any burst pattern of length  $\leq \ell$  cannot have a burst of length  $\leq 2\ell$  as a codeword. If it had a burst of length  $\leq 2\ell$  as a codeword, then a burst of length  $\ell$  could change the codeword to a burst pattern of length  $\ell$ , which also could be obtained by making a burst error of length  $\ell$  in all zero codeword. If vectors are non-zero in first  $2\ell$  symbols, then the vectors should be from different subsets of an array so that their difference is not a codeword of bursts of length  $2\ell$ . Ensuring this condition, the number of such subsets is at least equal to number of vectors. Thus, the number of subsets would be at least  $q^{2\ell}$ . Hence, we have at least  $2\ell$  distinct symbols, otherwise, the difference of two such polynomials would be a codeword that is a sum of two bursts of length  $\leq \ell$ . Thus, this proves the Rieger Bound.

**Definition.** A linear burst-error-correcting code achieving the above Rieger bound is called an optimal burst-error-correcting code.

### 18.3.2 Further bounds on burst error correction

There is more than one upper bound on the achievable code rate of linear block codes for multiple phased-burst correction (MPBC). One such bound is constrained to a maximum correctable cyclic burst length within every subblock, or equivalently a constraint on the minimum error free length or gap within every phased-burst. This bound, when reduced to the special case of a bound for single burst correction, is the Abramson bound (a corollary of the Hamming bound for burst-error correction) when the cyclic burst length is less than half the block length.<sup>[3]</sup>

**Theorem (number of bursts).** For  $1 \leq \ell \leq \frac{1}{2}(n+1)$ , over a binary alphabet, there are  $n2^{\ell-1} + 1$  vectors of length  $n$  which are bursts of length  $\leq \ell$ .<sup>[1]</sup>

**Proof.** Since the burst length is  $\leq \frac{1}{2}(n+1)$ , there is a unique burst description associated with the burst. The burst can begin at any of the  $n$  positions of the pattern. Each pattern begins with 1 and contain a length of  $\ell$ . We can think of it as the set of all strings that begin with 1 and have length  $\ell$ . Thus, there are a total of  $2^{\ell-1}$  possible such patterns, and a total of  $n2^{\ell-1}$  bursts of length  $\leq \ell$ . If we include the all-zero burst, we have  $n2^{\ell-1} + 1$  vectors representing bursts of length  $\leq \ell$ .

**Theorem (Bound on the number of codewords).** If  $1 \leq \ell \leq \frac{1}{2}(n+1)$ , a binary  $\ell$ -burst error correcting code has at most  $2^n / (n2^{\ell-1} + 1)$  codewords.

**Proof.** Since  $\ell \leq \frac{1}{2}(n+1)$ , we know that there are  $n2^{\ell-1} + 1$  bursts of length  $\leq \ell$ . Say the code has  $M$  codewords, then there are  $Mn2^{\ell-1}$  codewords that differ from a codeword by a burst of length  $\leq \ell$ . Each of the  $M$  words must be distinct, otherwise the code would have distance  $< 1$ . Therefore,  $M(n2^{\ell-1} + 1) \leq 2^n$  implies  $M \leq 2^n / (n2^{\ell-1} + 1)$ .

**Theorem (Abramson's bounds).** If  $1 \leq \ell \leq \frac{1}{2}(n+1)$  is a binary linear  $(n, k)$ ,  $\ell$ -burst error correcting code, its block-length must satisfy:

$$n \leq 2^{n-k-\ell+1} - 1.$$

**Proof:** For a linear  $(n, k)$  code, there are  $2^k$  codewords. By our previous result, we know that

$$2^k \leq \frac{2^n}{n2^{\ell-1} + 1}.$$

Isolating  $n$ , we get  $n \leq 2^{n-k-\ell+1} - 2^{-\ell+1}$ . Since  $\ell \geq 1$  and  $n$  must be an integer, we have  $n \leq 2^{n-k-\ell+1} - 1$ .

*Remark.*  $r = n - k$  is called the redundancy of the code and in an alternative formulation for the Abramson's bounds is  $r \geq \lceil \log_2(n+1) \rceil + \ell - 1$ .

## 18.4 Fire codes<sup>[3][4][5]</sup>

While **cyclic codes** in general are powerful tools for detecting burst errors, we now consider a family of binary cyclic codes named Fire Codes, which possess good single burst error correction capabilities. By single burst, say of length  $\ell$ , we mean that all errors that a received codeword possess lie within a fixed span of  $\ell$  digits.

Let  $p(x)$  be an **irreducible polynomial** of degree  $m$  over  $\mathbb{F}_2$ , and let  $p$  be the period of  $p(x)$ . The period of  $p(x)$ , and indeed of any polynomial, is defined to be the least positive integer  $r$  such that  $p(x)|x^r - 1$ . Let  $\ell$  be a positive integer satisfying  $\ell \leq m$  and  $2\ell - 1$  not divisible by  $p$ , where  $p$  is the period of  $p(x)$ . Define the Fire Code  $G$  by the following **generator polynomial**:

$$g(x) = (x^{2\ell-1} + 1)p(x).$$

We will show that  $G$  is an  $\ell$ -burst-error correcting code.

**Lemma 1.**  $\gcd(p(x), x^{2\ell-1} + 1) = 1$ .

**Proof.** Let  $d(x)$  be the greatest common divisor of the two polynomials. Since  $p(x)$  is irreducible,  $\deg(d(x)) = 0$  or  $\deg(p(x))$ . Assume  $\deg(d(x)) \neq 0$ , then  $p(x) = cd(x)$  for some constant  $c$ . But,  $(1/c)p(x)$  is a divisor of  $x^{2\ell-1} + 1$  since  $d(x)$  is a divisor of  $x^{2\ell-1} + 1$ . But this contradicts our assumption that  $p(x)$  does not divide  $x^{2\ell-1} + 1$ . Thus,  $\deg(d(x)) = 0$ , proving the lemma.

**Lemma 2.** If  $p(x)$  is a polynomial of period  $p$ , then  $p(x)|x^k - 1$  if and only if  $p|k$ .

**Proof.** If  $p|k$ , then  $x^k - 1 = (x^p - 1)(1 + x^p + x^{2p} + \dots + x^{k/p})$ . Thus,  $p(x)|x^k - 1$ .

Now suppose  $p(x)|x^k - 1$ . Then,  $k \geq p$ . We show that  $k$  is divisible by  $p$  by induction on  $k$ . The base case  $k = p$  follows. Therefore, assume  $k > p$ . We know that  $p(x)$  divides both (since it has period  $p$ )

$$x^p - 1 = (x-1)(1 + x + \dots + x^{p-1}) \quad \text{and} \quad x^k - 1 = (x-1)(1 + x + \dots + x^{k-1})$$

But  $p(x)$  is irreducible, therefore it must divide both  $(1 + x + \dots + x^{p-1})$  and  $(1 + x + \dots + x^{k-1})$ ; thus, it also divides the difference of the last two polynomials,  $x^p(1 + x + \dots + x^{p-k-1})$ . Then, it follows that  $p(x)$  divides  $(1 + x + \dots + x^{p-k-1})$ . Finally, it also divides:  $x^{k-p} - 1 = (x - 1)(1 + x + \dots + x^{p-k-1})$ . By the induction hypothesis,  $p|k - p$ , then  $p|k$ .

A corollary to Lemma 2 is that since  $p(x) = x^p - 1$  has period  $p$ , then  $p(x)$  divides  $x^k - 1$  if and only if  $p|k$ .

**Theorem.** The Fire Code is  $\ell$ -burst error correcting<sup>[4][5]</sup>

If we can show that all bursts of length  $\ell$  or less occur in different cosets, we can use them as coset leaders that form correctable error patterns. The reason is simple: we know that each coset has a unique syndrome decoding associated with it, and if all bursts of different lengths occur in different cosets, then all have unique syndromes, facilitating error correction.

### 18.4.1 Proof of Theorem

Let  $x^i a(x)$  and  $x^j b(x)$  be polynomials with degrees  $\ell_1 - 1$  and  $\ell_2 - 1$ , representing bursts of length  $\ell_1$  and  $\ell_2$  respectively with  $\ell_1, \ell_2 \leq \ell$ . The integers  $i, j$  represent the starting positions of the bursts, and are less than the block length of the code. For contradiction sake, assume that  $x^i a(x)$  and  $x^j b(x)$  are in the same coset. Then,  $v(x) = x^i a(x) + x^j b(x)$  is a valid codeword (since both terms are in the same coset). Without loss of generality, pick  $i \leq j$ . By the division theorem we can write:  $j - i = g(2\ell - 1) + r$ , for integers  $g$  and  $r$ ,  $0 \leq r < 2\ell - 1$ . We rewrite the polynomial  $v(x)$  as follows:

$$v(x) = x^i a(x) + x^{i+g(2\ell-1)+r} b(x) = x^i a(x) + x^{i+g(2\ell-1)+r} b(x) + x^{i+g(2\ell-1)+r} (x^{-(2\ell-1)} - 1) b(x)$$

Notice that at the second manipulation, we introduced the term  $2x^{i+r}b(x)$ . We are allowed to do so, since Fire Codes operate on  $\mathbb{F}_2$ . By our assumption,  $v(x)$  is a valid codeword, and thus, must be a multiple of  $g(x)$ . As mentioned earlier, since the factors of  $g(x)$  are relatively prime,  $v(x)$  has to be divisible by  $x^{2\ell-1} + 1$ . Looking closely at the last expression derived for  $v(x)$  we notice that  $x^{g(2\ell-1)} + 1$  is divisible by  $x^{2\ell-1} + 1$  (by the corollary of Lemma 2). Therefore,  $a(x) + x^b b(x)$  is either divisible by  $x^{2\ell-1} + 1$  or is 0. Applying the division theorem again, we see that there exists a polynomial  $d(x)$  with degree  $\delta$  such that:

$$a(x) + x^b b(x) = d(x)(x^{2\ell-1} + 1)$$

Then we may write:

$$\begin{aligned} \delta + 2\ell - 1 &= \deg(d(x)(x^{2\ell-1} + 1)) \\ &= \deg(a(x) + x^b b(x)) \\ &= \deg(x^b b(x)) & \deg(a(x)) = \ell_1 - 1 < 2\ell - 1 \\ &= b + \ell_2 - 1 \end{aligned}$$

Equating the degree of both sides, gives us  $b = 2\ell - \ell_2 + \delta$ . Since  $\ell_1, \ell_2 \leq \ell$  we can conclude  $b \geq \ell + \delta$ , which implies  $b > \ell - 1$  and  $b > \delta$ . Notice that in the expansion:

$$a(x) + x^b b(x) = 1 + a_1 x + a_2 x^2 + \dots + x^{\ell_1-1} + x^b (1 + b_1 x + b_2 x^2 + \dots +$$

the term  $x^b$  appears, but since  $\delta < b < 2\ell - 1$ , the resulting expression  $d(x)(x^{2\ell-1} + 1)$  does not contain  $x^b$ , therefore  $d(x) = 0$  and subsequently  $a(x) + x^b b(x) = 0$ . This requires that  $b = 0$ , and  $a(x) = b(x)$ . We can further revise our division of  $j - i$  by  $g(2\ell - 1)$  to reflect  $b = 0$ , that is  $j - i = g(2\ell - 1)$ . Substituting back into  $v(x)$  gives us,

$$v(x) = x^i b(x) (x^{j-1} + 1).$$

Since  $\deg(b(x)) = \ell_2 - 1 < \ell$ , we have  $\deg(b(x)) < \deg(p(x)) = m$ . But  $p(x)$  is irreducible, therefore  $b(x)$  and  $p(x)$  must be relatively prime. Since  $v(x)$  is a codeword,  $x^{j-1} + 1$  must be divisible by  $p(x)$ , as it cannot be divisible by  $x^{2\ell-1} + 1$ . Therefore,  $j - i$  must be a multiple of  $p$ . But it must also be a multiple of  $2\ell - 1$ , which implies it must be a multiple of  $n = \text{lcm}(2\ell - 1, p)$  but that is precisely the block-length of the code. Therefore,  $j - i$  cannot be a multiple of  $n$  since they are both less than  $n$ . Thus, our assumption of  $v(x)$  being a codeword is incorrect, and therefore  $x^i a(x)$  and  $x^j b(x)$  are in different cosets, with unique syndromes, and therefore correctable.

### 18.4.2 Example: 5-burst error correcting fire code

With the theory presented in the above section, let us consider the construction of a 5-burst error correcting Fire Code. Remember that to construct a Fire Code, we need an irreducible polynomial  $p(x)$ , an integer  $\ell$ , representing the burst error correction capability of our code, and we need to satisfy the property that  $2\ell - 1$  is not divisible by the period of  $p(x)$ . With these requirements in mind, consider the irreducible polynomial  $p(x) = 1 + x^2 + x^5$ , and let  $\ell = 5$ . Since  $p(x)$  is a primitive polynomial, its period is  $2^5 - 1 = 31$ . We confirm that  $2\ell - 1 = 9$  is not divisible by 31. Thus,

$$g(x) = (x^9 + 1)(1 + x^2 + x^5) = 1 + x^2 + x^5 + x^9 + x^{11} + x^{14}$$

is a Fire Code generator. We can calculate the block-length of the code by evaluating the least common multiple of  $p$  and  $2\ell - 1$ . In other words,  $n = \text{lcm}(9, 31) = 279$ . Thus, the Fire Code above is a cyclic code capable of correcting any burst of length 5 or less.

## 18.5 Binary Reed–Solomon codes

Certain families of codes, such as **Reed–Solomon**, operate on alphabet sizes larger than binary. This property awards such codes powerful burst error correction capabilities. Consider a code operating on  $\mathbb{F}_{2^m}$ . Each symbol of the alphabet can be represented by  $m$  bits. If  $C$  is an  $(n, k)$  Reed–Solomon code over  $\mathbb{F}_{2^m}$ , we can think of  $C$  as an  $[mn, mk]_2$  code over  $\mathbb{F}_2$ .

The reason such codes are powerful for burst error correction is that each symbol is represented by  $m$  bits, and in general, it is irrelevant how many of those  $m$  bits are erroneous; whether a single bit, or all of the  $m$  bits contain errors, from a decoding perspective it is still a single symbol error. In other words, since burst errors tend to occur in clusters, there is a strong possibility of several binary errors contributing to a single symbol error.

Notice that a burst of  $(m + 1)$  errors can affect at most 2 symbols, and a burst of  $2m + 1$  can affect at most 3 symbols. Then, a burst of  $tm + 1$  can affect at most  $t + 1$  symbols; this implies that a  $t$ -symbols-error correcting code can correct a burst of length at most  $(t - 1)m + 1$ .

In general, a  $t$ -error correcting Reed–Solomon code over  $\mathbb{F}_{2^m}$  can correct any combination of

$$\frac{t}{1 + \lfloor (l + m - 2)/m \rfloor}$$

or fewer bursts of length  $l$ , on top of being able to correct  $t$ -random worst case errors.

### 18.5.1 An example of a binary RS code

Let  $G$  be a  $[255, 223, 33]$  RS code over  $\mathbb{F}_{2^8}$ . This code was employed by **NASA** in their **Cassini-Huygens** spacecraft.<sup>[6]</sup> It is capable of correcting  $\lfloor 33/2 \rfloor = 16$  symbol errors. We now construct a Binary RS Code  $G'$  from  $G$ . Each symbol will be written using  $\lceil \log_2(255) \rceil = 8$  bits. Therefore, the Binary RS code will have  $[2040, 1784, 33]_2$  as its parameters. It is capable of correcting any single burst of length  $l = 121$ .

## 18.6 Interleaved codes

Interleaving is used to convert convolutional codes from random error correctors to burst error correctors. The basic idea behind the use of interleaved codes is to jumble

symbols at the receiver. This leads to randomization of bursts of received errors which are closely located and we can then apply the analysis for random channel. Thus, the main function performed by the interleaver at transmitter is to alter the input symbol sequence. At the receiver, the deinterleaver will alter the received sequence to get back the original unaltered sequence at the transmitter.

### 18.6.1 Burst error correcting capacity of interleaver

**Theorem.** If the burst error correcting ability of some code is  $\ell$ , then the burst error correcting ability of its  $\lambda$ -way interleave is  $\lambda\ell$ .

**Proof:** Suppose that we have an  $(n, k)$  code that can correct all bursts of length  $\leq \ell$ . Interleaving can provide us with a  $(\lambda n, \lambda k)$  code that can correct all bursts of length  $\leq \lambda\ell$ , for any given  $\lambda$ . If we want to encode a message of an arbitrary length using interleaving, first we divide it into blocks of length  $\lambda k$ . We write the  $\lambda k$  entries of each block into a  $\lambda \times k$  matrix using row-major order. Then, we encode each row using the  $(n, k)$  code. What we will get is a  $\lambda \times n$  matrix. Now, this matrix is read out and transmitted in column-major order. The trick is that if there occurs a burst of length  $h$  in the transmitted word, then each row will contain approximately  $\frac{h}{\lambda}$  consecutive errors (More specifically, each row will contain a burst of length at least  $\lfloor \frac{h}{\lambda} \rfloor$  and at most  $\lceil \frac{h}{\lambda} \rceil$ ). If  $h \leq \lambda\ell$ , then  $\frac{h}{\lambda} \leq \ell$  and the  $(n, k)$  code can correct each row. Therefore, the interleaved  $(\lambda n, \lambda k)$  code can correct the burst of length  $h$ . Conversely, if  $h > \lambda\ell$ , then at least one row will contain more than  $\frac{h}{\lambda}$  consecutive errors, and the  $(n, k)$  code might fail to correct them. Therefore, the error correcting ability of the interleaved  $(\lambda n, \lambda k)$  code is exactly  $\lambda\ell$ . The BEC efficiency of the interleaved code remains the same as the original  $(n, k)$  code. This is true because:

$$\frac{2\lambda\ell}{\lambda n - \lambda k} = \frac{2\ell}{n - k}$$

### 18.6.2 Block interleaver

The figure below shows a 4 by 3 interleaver.

The above interleaver is called as a **block interleaver**. Here, the input symbols are written sequentially in the rows and the output symbols are obtained by reading the columns sequentially. Thus, this is in the form of  $M \times N$  array. Generally,  $N$  is length of the codeword.

**Capacity of block interleaver:** For an  $M \times N$  block interleaver and burst of length  $\ell$ , the upper limit on number



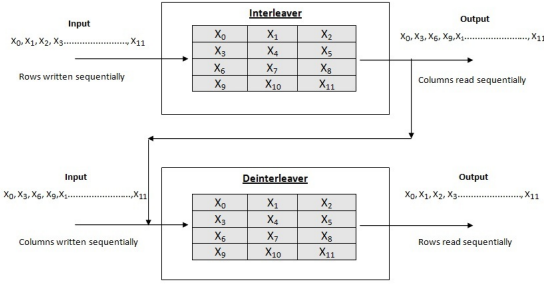


Fig. A 4 X 3 Interleaver and Deinterleaver

An example of a block interleaver

of errors is  $\frac{\ell}{M}$ . This is obvious from the fact that we are reading the output column wise and the number of rows is  $M$ . By the theorem above for error correction capacity up to  $t$ , the maximum burst length allowed is  $Mt$ . For burst length of  $Mt + 1$ , the decoder may fail.

**Efficiency of block interleaver ( $\gamma$ ):** It is found by taking ratio of burst length where decoder may fail to the interleaver memory. Thus, we can formulate  $\gamma$  as

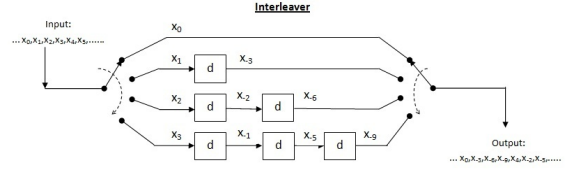
$$\gamma = \frac{Mt + 1}{MN} \approx \frac{t}{N}.$$

**Drawbacks of block interleaver :** As it is clear from the figure, the columns are read sequentially, the receiver can interpret single row only after it receives complete message and not before that. Also, the receiver requires a considerable amount of memory in order to store the received symbols and has to store the complete message. Thus, these factors give rise to two drawbacks, one is the latency and other is the storage (fairly large amount of memory). These drawbacks can be avoided by using the convolutional interleaver described below.

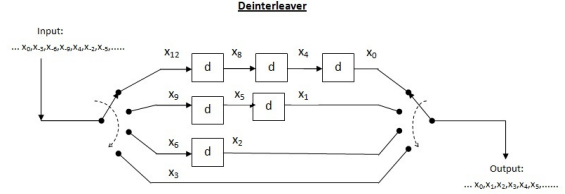
### 18.6.3 Convolutional interleaver

Cross interleaver is a kind of multiplexer-demultiplexer system. In this system, delay lines are used to progressively increase length. Delay line is basically an electronic circuit used to delay the signal by certain time duration. Let  $n$  be the number of delay lines and  $d$  be the number of symbols introduced by each delay line. Thus, the separation between consecutive inputs =  $nd$  symbols. Let the length of codeword  $\leq n$ . Thus, each symbol in the input codeword will be on distinct delay line. Let a burst error of length  $\ell$  occur. Since the separation between consecutive symbols is  $nd$ , the number of errors that the deinterleaved output may contain is  $\frac{\ell}{nd+1}$ . By the theorem above, for error correction capacity up to  $t$ , maximum burst length allowed is  $(nd + 1)(t - 1)$ . For burst length of  $(nd + 1)(t - 1) + 1$ , decoder may fail.

**Efficiency of cross interleaver ( $\gamma$ ):** It is found by taking the ratio of burst length where decoder may fail to



An example of a convolutional interleaver



An example of a deinterleaver

the interleaver memory. In this case, the memory of interleaver can be calculated as

$$(0 + 1 + 2 + 3 + \dots + (n - 1))d = \frac{n(n - 1)}{2}d.$$

Thus, we can formulate  $\gamma$  as follows:

$$\gamma = \frac{(nd + 1)(t - 1) + 1}{\frac{n(n - 1)}{2}d}.$$

**Performance of cross interleaver :** As shown in the above interleaver figure, the output is nothing but the diagonal symbols generated at the end of each delay line. In this case, when the input multiplexer switch completes around half switching, we can read first row at the receiver. Thus, we need to store maximum of around half message at receiver in order to read first row. This drastically brings down the storage requirement by half. Since just half message is now required to read first row, the latency is also reduced by half which is good improvement over the block interleaver. Thus, the total interleaver memory is split between transmitter and receiver.

## 18.7 Applications

### 18.7.1 Compact disc

Without error correcting codes, digital audio would not be technically feasible.<sup>[7]</sup> The Reed–Solomon codes can correct a corrupted symbol with a single bit error just as easily as it can correct a symbol with all bits wrong. This makes the RS codes particularly suitable for correcting burst errors.<sup>[5]</sup> By far, the most common application of RS codes is in compact discs. In addition to basic error correction provided by RS codes, protection against burst

errors due to scratches on the disc is provided by a cross interleaver.<sup>[3]</sup>

Current compact disc digital audio system was developed by N. V. Philips of The Netherlands and Sony Corporation of Japan (agreement signed in 1979).

A compact disc comprises a 120 mm aluminized disc coated with a clear plastic coating, with spiral track, approximately 5 km in length, which is optically scanned by a laser of wavelength  $\sim 0.8 \mu\text{m}$ , at a constant speed of  $\sim 1.25 \text{ m/s}$ . For achieving this constant speed, rotation of the disc is varied from  $\sim 8 \text{ rev/s}$  while scanning at the inner portion of the track to  $\sim 3.5 \text{ rev/s}$  at the outer portion. Pits and lands are the depressions ( $0.12 \mu\text{m}$  deep) and flat segments constituting the binary data along the track ( $0.6 \mu\text{m}$  width).<sup>[8]</sup>

The CD process can be abstracted as a sequence of the following sub-processes:  $\rightarrow$  Channel encoding of source of signals  $\rightarrow$  Mechanical sub-processes of preparing a master disc, producing user discs and sensing the signals embedded on user discs while playing – the channel  $\rightarrow$  Decoding the signals sensed from user discs

The process is subject to both burst errors and random errors.<sup>[7]</sup> Burst errors include those due to disc material (defects of aluminum reflecting film, poor reflective index of transparent disc material), disc production (faults during disc forming and disc cutting etc.), disc handling (scratches – generally thin, radial and orthogonal to direction of recording) and variations in play-back mechanism. Random errors include those due to jitter of reconstructed signal wave and interference in signal. CIRC (Cross-Interleaved Reed–Solomon code) is the basis for error detection and correction in the CD process. It corrects error bursts up to 3,500 bits in sequence (2.4 mm in length as seen on CD surface) and compensates for error bursts up to 12,000 bits (8.5 mm) that may be caused by minor scratches.

**Encoding:** Sound-waves are sampled and converted to digital form by an A/D converter. The sound wave is sampled for amplitude (at 44.1 kHz or 44,100 pairs, one each for the left and right channels of the stereo sound). The amplitude at an instance is assigned a binary string of length 16. Thus, each sample produces two binary vectors from  $\mathbb{F}_2^{16}$  or 4  $\mathbb{F}_2^8$  bytes of data. Every second of sound recorded results in  $44,100 \times 32 = 1,411,200$  bits (176,400 bytes) of data.<sup>[5]</sup> The 1.41 Mbit/s sampled data stream passes through the error correction system eventually getting converted to a stream of 1.88 Mbit/s.

Input for the encoder consists of input frames each of 24 8-bit symbols (12 16-bit samples from the A/D converter, 6 each from left and right data (sound) sources). A frame can be represented by  $L_1 R_1 L_2 R_2 \dots L_6 R_6$  where  $L_i$  and  $R_i$  are bytes from the left and right channels from the  $i^{\text{th}}$  sample of the frame.

Initially, the bytes are permuted to form new frames represented by  $L_1 L_3 L_5 R_1 R_3 R_5 L_2 L_4 L_6 R_2 R_4 R_6$  where

$L_i, R_i$  represent  $i^{\text{th}}$  left and right samples from the frame after 2 intervening frames.

Next, these 24 message symbols are encoded using C2 (28,24,5) Reed–Solomon code which is a shortened RS code over  $\mathbb{F}_{256}$ . This is two-error-correcting, being of minimum distance 5. This adds 4 bytes of redundancy,  $P_1 P_2$  forming a new frame:  $L_1 L_3 L_5 R_1 R_3 R_5 P_1 P_2 L_2 L_4 L_6 R_2 R_4 R_6$ . The resulting 28-symbol codeword is passed through a (28,4) cross interleaver leading to 28 interleaved symbols. These are then passed through C1 (32,28,5) RS code, resulting in codewords of 32 coded output symbols. Further regrouping of odd numbered symbols of a codeword with even numbered symbols of the next codeword is done to break up any short bursts that may still be present after the above 4-frame delay interleaving. Thus, for every 24 input symbols there will be 32 output symbols giving  $R = 24/32$ . Finally one byte of control and display information is added.<sup>[5]</sup> Each of the 33 bytes is then converted to 17 bits through EFM (eight to fourteen modulation) and addition of 3 merge bits. Therefore, the frame of six samples results in 33 bytes  $\times$  17 bits (561 bits) to which are added 24 synchronization bits and 3 merging bits yielding a total of 588 bits.

**Decoding:** The CD player (CIRC decoder) receives the 32 output symbol data stream. This stream passes through the decoder D1 first. It is up to individual designers of CD systems to decide on decoding methods and optimize their product performance. Being of minimum distance 5 The D1, D2 decoders can each correct a combination of  $e$  errors and  $f$  erasures such that  $2e + f < 5$ .<sup>[5]</sup> In most decoding solutions, D1 is designed to correct single error. And in case of more than 1 error, this decoder outputs 28 erasures. The deinterleaver at the succeeding stage distributes these erasures across 28 D2 codewords. Again in most solutions, D2 is set to deal with erasures only (a simpler and less expensive solution). If more than 4 erasures were to be encountered, 24 erasures are output by D2. Thereafter, an error concealment system attempts to interpolate (from neighboring symbols) in case of uncorrectable symbols, failing which sounds corresponding to such erroneous symbols get muted.

**Performance of CIRC:**<sup>[7]</sup> CIRC conceals long burst errors by simple linear interpolation. 2.5 mm of track length (4000 bits) is the maximum completely correctable burst length. 7.7 mm track length (12,300 bits) is the maximum burst length that can be interpolated. Sample interpolation rate is one every 10 hours at Bit Error Rate (BER) =  $10^{-4}$  and 1000 samples per minute at BER =  $10^{-3}$  Undetectable error samples (clicks): less than one every 750 hours at BER =  $10^{-3}$  and negligible at BER =  $10^{-4}$ .

## 18.8 See also

- Error detection and correction

- Error-correcting codes with feedback
- Code rate
- Reed–Solomon error correction

## 18.9 References

- [1] Coding Bounds for Multiple Phased-Burst Correction and Single Burst Correction Codes
- [2] *The Theory of Information and Coding: Student Edition*, by R. J. McEliece
- [3] Ling, San, and Chaoping Xing. Coding Theory: A First Course. Cambridge, UK: Cambridge UP, 2004. Print
- [4] Moon, Todd K. Error Correction Coding: Mathematical Methods and Algorithms. Hoboken, NJ: Wiley-Interscience, 2005. Print
- [5] Lin, Shu, and Daniel J. Costello. Error Control Coding: Fundamentals and Applications. Upper Saddle River, NJ: Pearson-Prentice Hall, 2004. Print
- [6] [http://webcache.googleusercontent.com/search?q=cache:http://quest.arc.nasa.gov/saturn/qa/cassini/Error\\_correction.txt\[\]](http://webcache.googleusercontent.com/search?q=cache:http://quest.arc.nasa.gov/saturn/qa/cassini/Error_correction.txt[])
- [7] Algebraic Error Control Codes (Autumn 2012) – Handouts from Stanford University
- [8] McEliece, Robert J. The Theory of Information and Coding: A Mathematical Framework for Communication. Reading, MA: Addison-Wesley Pub., Advanced Book Program, 1977. Print



## Chapter 19

# Canonical Huffman code

A **canonical Huffman code** is a particular type of **Huffman code** with unique properties which allow it to be described in a very compact manner.

**Data compressors** generally work in one of two ways. Either the decompressor can infer what **codebook** the compressor has used from previous context, or the compressor must tell the decompressor what the codebook is. Since a canonical Huffman codebook can be stored especially efficiently, most compressors start by generating a “normal” Huffman codebook, and then convert it to canonical Huffman before using it.

In order for a **symbol code** scheme such as the **Huffman code** to be decompressed, the same model that the encoding algorithm used to compress the source data must be provided to the decoding algorithm so that it can use it to decompress the encoded data. In standard Huffman coding this model takes the form of a tree of variable-length codes, with the most frequent symbols located at the top of the structure and being represented by the fewest number of bits.

However, this code tree introduces two critical inefficiencies into an implementation of the coding scheme. Firstly, each node of the tree must store either references to its child nodes or the symbol that it represents. This is expensive in memory usage and if there is a high proportion of unique symbols in the source data then the size of the code tree can account for a significant amount of the overall encoded data. Secondly, traversing the tree is computationally costly, since it requires the algorithm to jump randomly through the structure in memory as each bit in the encoded data is read in.

Canonical Huffman codes address these two issues by generating the codes in a clear standardized format; all the codes for a given length are assigned their values sequentially. This means that instead of storing the structure of the code tree for decompression only the lengths of the codes are required, reducing the size of the encoded data. Additionally, because the codes are sequential, the decoding algorithm can be dramatically simplified so that it is computationally efficient.

### 19.1 Algorithm

The normal Huffman coding **algorithm** assigns a variable length code to every symbol in the alphabet. More frequently used symbols will be assigned a shorter code. For example, suppose we have the following *non-canonical* codebook:

A = 11 B = 0 C = 101 D = 100

Here the letter A has been assigned 2 bits, B has 1 bit, and C and D both have 3 bits. To make the code a *canonical* Huffman code, the codes are renumbered. The bit lengths stay the same with the code book being sorted *first* by codeword length and *secondly* by **alphabetical value**:

B = 0 A = 11 C = 101 D = 100

Each of the existing codes are replaced with a new one of the same length, using the following algorithm:

- The *first* symbol in the list gets assigned a codeword which is the same length as the symbol's original codeword but all zeros. This will often be a single zero ('0').
- Each subsequent symbol is assigned the next **binary** number in sequence, ensuring that following codes are always higher in value.
- When you reach a longer codeword, then *after* incrementing, append zeros until the length of the new codeword is equal to the length of the old codeword. This can be thought of as a **left shift**.

By following these three rules, the *canonical* version of the code book produced will be:

B = 0 A = 10 C = 110 D = 111

#### 19.1.1 As a fractional binary number

Another perspective on the canonical codewords is that they are the digits past the **radix point** (binary decimal point) in a binary representation of a certain series. Specifically, suppose the lengths of the codewords are  $l_1 \dots l_n$ . Then the canonical codeword for symbol  $i$  is the

first  $l_i$  binary digits past the radix point in the binary representation of

$$\sum_{j=1}^{i-1} 2^{-l_j}.$$

This perspective is particularly useful in light of **Kraft's inequality**, which says that the sum above will always be less than or equal to 1 (since the lengths come from a prefix free code). This shows that adding one in the algorithm above never overflows and creates a codeword that is longer than intended.

## 19.2 Encoding the codebook

The whole advantage of a canonical Huffman tree is that one can encode the description (the codebook) in fewer bits than a fully described tree.

Let us take our original Huffman codebook:

A = 11 B = 0 C = 101 D = 100

There are several ways we could encode this Huffman tree. For example, we could write each **symbol** followed by the **number of bits** and **code**:

('A',2,11), ('B',1,0), ('C',3,101), ('D',3,100)

Since we are listing the symbols in sequential alphabetical order, we can omit the symbols themselves, listing just the **number of bits** and **code**:

(2,11), (1,0), (3,101), (3,100)

With our *canonical* version we have the knowledge that the symbols are in sequential alphabetical order *and* that a later code will always be higher in value than an earlier one. The only parts left to transmit are the **bit-lengths** (**number of bits**) for each symbol. Note that our canonical Huffman tree always has higher values for longer bit lengths and that any symbols of the same bit length (C and D) have higher code values for higher symbols:

A = 10 (code value: 2 decimal, bits: 2) B = 0 (code value: 0 decimal, bits: 1) C = 110 (code value: 6 decimal, bits: 3) D = 111 (code value: 7 decimal, bits: 3)

Since two-thirds of the constraints are known, only the **number of bits** for each symbol need be transmitted:

2, 1, 3, 3

With knowledge of the canonical Huffman algorithm, it is then possible to recreate the entire table (symbol and code values) from just the bit-lengths. Unused symbols are normally transmitted as having zero bit length.

Another efficient way representing the codebook is to list all symbols in increasing order by their bit-lengths, and record the number of symbols for each bit-length. For the example mentioned above, the encoding becomes:

(1,1,2), ('B','A','C','D')

This means that the first symbol B is of length 1, then the A of length 2, and remains of 3. Since the symbols

are sorted by bit-length, we can efficiently reconstruct the codebook. A **pseudo code** describing the reconstruction is introduced on the next section.

This type of encoding is advantageous when only a few symbols in the alphabet are being compressed. For example, suppose the codebook contains only 4 letters C, O, D and E, each of length 2. To represent the letter O using the previous method, we need to either add a lot of zeros:

0, 0, 2, 2, 2, 0, ... , 2, ...

or record which 4 letters we have used. Each way makes the description longer than:

(0,4), ('C','O','D','E')

The **JPEG File Interchange Format** uses this method of encoding, because at most only 162 symbols out of the 8-bit alphabet, which has size 256, will be in the codebook.

## 19.3 Pseudo code

Given a list of symbols sorted by bit-length, the following **pseudo code** will print a canonical Huffman code book:

code = 0 while more symbols: print symbol, code code = (code + 1) << ((bit length of the next symbol) - (current bit length))

## 19.4 Algorithm

The algorithm described in: "A Method for the Construction of Minimum-Redundancy Codes" David A. Huffman, Proceedings of the I.R.E. is:

compute huffman code: input: message ensemble (set of (message, probability)). base D. output: code ensemble (set of (message, code)). algorithm: 1- sort the message ensemble by decreasing probability. 2- N is the cardinal of the message ensemble (number of different messages). 3- compute the integer  $n_0$  such as  $2 \leq n_0 \leq D$  and  $(N - n_0)/(D - 1)$  is integer. 4- select the  $n_0$  least probable messages, and assign them each a digit code. 5- substitute the selected messages by a composite message summing their probability, and re-order it. 6- while there remains more than one message, do steps thru 8. 7- select D least probable messages, and assign them each a digit code. 8- substitute the selected messages by a composite message summing their probability, and re-order it. 9- the code of each message is given by the concatenation of the code digits of the aggregate they've been put in.

References: 1. **Managing Gigabytes**: book with an implementation of canonical huffman codes for word dictionaries.

# Chapter 20

## Coding gain

In coding theory and related engineering problems, **coding gain** is the measure in the difference between the signal-to-noise ratio (SNR) levels between the uncoded system and coded system required to reach the same bit error rate (BER) levels when used with the error correcting code (ECC).

### 20.1 Example

If the uncoded BPSK system in AWGN environment has a bit error rate (BER) of  $10^{-2}$  at the SNR level 4 dB, and the corresponding coded (e.g., BCH) system has the same BER at an SNR of 2.5 dB, then we say the *coding gain* = 4 dB – 2.5 dB = 1.5 dB, due to the code used (in this case BCH).

### 20.2 Power-limited regime

In the *power-limited regime* (where the nominal spectral efficiency  $\rho \leq 2$  [b/2D or b/s/Hz], i.e. the domain of binary signaling), the effective coding gain  $\gamma_{\text{eff}}(A)$  of a signal set  $A$  at a given target error probability per bit  $P_b(E)$  is defined as the difference in dB between the  $E_b/N_0$  required to achieve the target  $P_b(E)$  with  $A$  and the  $E_b/N_0$  required to achieve the target  $P_b(E)$  with 2-PAM or (2×2)-QAM (i.e. no coding). The nominal coding gain  $\gamma_c(A)$  is defined as

$$\gamma_c(A) = \frac{d_{\min}^2(A)}{4E_b}.$$

This definition is normalized so that  $\gamma_c(A) = 1$  for 2-PAM or (2×2)-QAM. If the average number of nearest neighbors per transmitted bit  $K_b(A)$  is equal to one, the effective coding gain  $\gamma_{\text{eff}}(A)$  is approximately equal to the nominal coding gain  $\gamma_c(A)$ . However, if  $K_b(A) > 1$ , the effective coding gain  $\gamma_{\text{eff}}(A)$  is less than the nominal coding gain  $\gamma_c(A)$  by an amount which depends on the steepness of the  $P_b(E)$  vs.  $E_b/N_0$  curve at the target  $P_b(E)$ . This curve can be plotted using the union bound estimate (UBE)

$$P_b(E) \approx K_b(A)Q\sqrt{\frac{2\gamma_c(A)E_b}{N_0}},$$

where  $Q$  is the Gaussian probability-of-error function.

For the special case of a binary linear block code  $C$  with parameters  $(n, k, d)$ , the nominal spectral efficiency is  $\rho = 2k/n$  and the nominal coding gain is  $kd/n$ .

### 20.3 Example

The table below lists the nominal spectral efficiency, nominal coding gain and effective coding gain at  $P_b(E) \approx 10^{-5}$  for Reed–Muller codes of length  $n \leq 64$ :

### 20.4 Bandwidth-limited regime

In the *bandwidth-limited regime* ( $\rho > 2b/2D$ , i.e. the domain of non-binary signaling), the effective coding gain  $\gamma_{\text{eff}}(A)$  of a signal set  $A$  at a given target error rate  $P_s(E)$  is defined as the difference in dB between the  $SNR_{\text{norm}}$  required to achieve the target  $P_s(E)$  with  $A$  and the  $SNR_{\text{norm}}$  required to achieve the target  $P_s(E)$  with M-PAM or (M×M)-QAM (i.e. no coding). The nominal coding gain  $\gamma_c(A)$  is defined as

$$\gamma_c(A) = \frac{(2^\rho - 1)d_{\min}^2(A)}{6E_s}.$$

This definition is normalized so that  $\gamma_c(A) = 1$  for M-PAM or (M×M)-QAM. The UBE becomes

$$P_s(E) \approx K_s(A)Q\sqrt{3\gamma_c(A)SNR_{\text{norm}}},$$

where  $K_s(A)$  is the average number of nearest neighbors per two dimensions.

## 20.5 See also

- Channel capacity
- Eb/N0

## 20.6 References

MIT OpenCourseWare, 6.451 Principles of Digital Communication II, Lecture Notes sections 5.3, 5.5, 6.3, 6.4

# Chapter 21

## Comma code

A **comma code** is a type of **prefix-free code** in which a **comma**, a particular symbol or sequence of symbols, occurs at the end of a code word and never occurs otherwise.<sup>[1]</sup>

For example, **Fibonacci coding** is a comma code in which the comma is 11. 11 and 1011 are valid Fibonacci code words, but 101, 0111, and 11011 are not.

### 21.1 Examples

- **Unary coding**, in which the comma is 0.
- **Fibonacci coding**, in which the comma is 11.

### 21.2 See also

- Self-synchronizing code

### 21.3 References

- [1] Wade, Graham (8 September 1994). *Signal Coding and Processing*. Cambridge University Press. p. 56. ISBN 978-0-521-42336-6.

## Chapter 22

# Comma-free code

A **comma-free code** is **block code** in which no concatenation of two code words contains a valid code word that overlaps both.<sup>[1]</sup>

Comma-free codes are also known as **self-synchronizing block codes**<sup>[2]</sup> because no synchronization is required to find the beginning of a code word.

### 22.1 References

- [1] S. W. Golomb; Gordon, Basil; L. R. Welch (1958). “Comma-free Codes”. *Canadian Journal of Mathematics*. Canadian Mathematical Society: 202–209. doi:10.4153/CJM-1958-023-9.
- [2] Donald Knuth (11 December 2015). *Universal Commafree Codes*. Stanford University. Retrieved 6 February 2016.

### 22.2 External links

- Donald Knuth’s 21st Annual Christmas Lecture: Universal Commafree Codes on YouTube

## Chapter 23

# Computationally bounded adversary

In information theory, the **computationally bounded adversary** problem is a different way of looking at the problem of sending data over a noisy channel. In previous models the best that could be done was ensuring correct decoding for up to  $d/2$  errors, where  $d$  was the Hamming distance of the code. The problem with doing it this way is that it does not take into consideration the actual amount of computing power available to the adversary. Rather, it only concerns itself with how many bits of a given code word can change and still have the message decode properly. In the computationally bounded adversary model the channel – the **adversary** – is restricted to only being able to perform a reasonable amount of computation to decide which bits of the code word need to change. In other words, this model does not need to consider how many errors can possibly be handled, but only how many errors could possibly be introduced given a reasonable amount of computing power on the part of the adversary. Once the channel has been given this restriction it becomes possible to construct codes that are both faster to encode and decode compared to previous methods that can also handle a large number of errors.

## 23.1 Comparison to other models

### 23.1.1 Worst-case model

At first glance, the **worst-case model** seems intuitively ideal. The guarantee that an algorithm will succeed no matter what is, of course, highly alluring. However, it demands too much. A real-life adversary cannot spend an indefinite amount of time examining a message in order to find the one error pattern which an algorithm would struggle with.

As a comparison, consider the **Quicksort** algorithm. In the worst-case scenario, Quicksort makes  $O(n^2)$  comparisons; however, such an occurrence is rare. Quicksort almost invariably makes  $O(n \log n)$  comparisons instead, and even outperforms other algorithms which can guarantee  $O(n \log n)$  behavior. Let us suppose an adversary wishes to force the Quicksort algorithm to make  $O(n^2)$  comparisons. Then he would have to search all of the  $n!$  permutations of the input string and test the algorithm

on each until he found the one for which the algorithm runs significantly slower. But since this would take  $O(n!)$  time, it is clearly infeasible for an adversary to do this. Similarly, it is unreasonable to assume an adversary for an encoding and decoding system would be able to test every single error pattern in order to find the most effective one.

### 23.1.2 Stochastic noise model

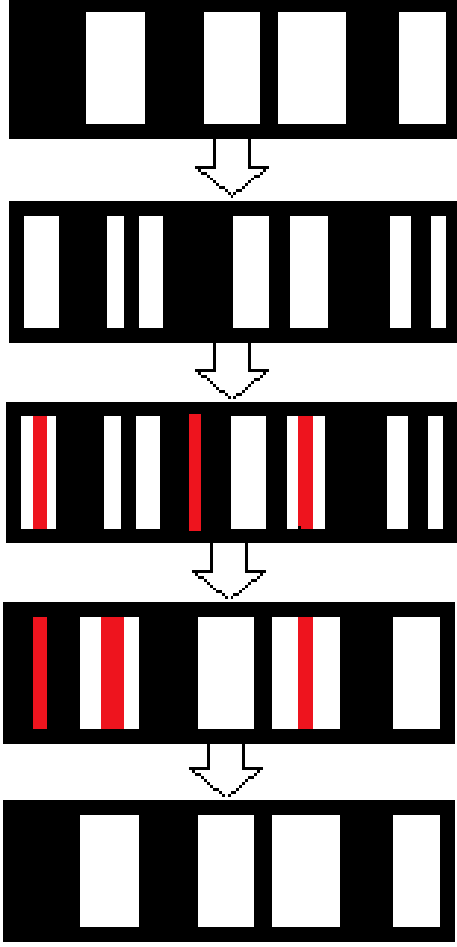
The stochastic noise model can be described as a kind of “dumb” noise model. That is to say that it does not have the adaptability to deal with “intelligent” threats. Even if the attacker is bounded it is still possible that they might be able to overcome the stochastic model with a bit of cleverness. The stochastic model has no real way to fight against this sort of attack and as such is unsuited to dealing with the kind of “intelligent” threats that would be preferable to have defenses against.

Therefore, a computationally bounded adversarial model has been proposed as a compromise between the two.<sup>[1]</sup> This forces one to consider that messages may be perverted in conscious, even malicious ways, but without forcing an algorithm designer to worry about rare cases which likely will never occur.

## 23.2 Applications

### 23.2.1 Comparison to stochastic noise channel

Since any computationally bounded adversary could in  $O(n)$  time flip a coin for each bit, it is intuitively clear that any encoding and decoding system which can work against this adversary must also work in the stochastic noise model. The converse is less simple; however, it can be shown that any system which works in the stochastic noise model can also efficiently encode and decode against a computationally bounded adversary, and only at an additional cost which is polynomial in  $n$ .<sup>[1]</sup> The following method to accomplish this was designed by Dick Lipton, and is taken from:<sup>[1]</sup>



An illustration of the method. The first row gives the initial encoded message; the second, after random permutation and random  $R$ ; the third, after the adversary adds  $N$ ; the fourth, after unpermuting; the fifth, the encoded message with the adversary's error removed.

Let  $E()$  be an encoder for the stochastic noise model and  $D()$  be a simple decoder for the same, each of which runs in polynomial time. Furthermore, let both the sender and receiver share some random permutation function  $\pi$  and a random pattern  $R$ .

For encoding: 1. Let  $X = E(M)$ .

2. Let  $Y = \pi(X) \oplus R$ .

3. Transmit  $Y$

Then for decoding: 1. Receive  $Y'$ . Compute  $Z = \pi^{-1}(Y' \oplus R)$ .

2. Calculate  $M = D(Z)$ .

Similarly to the Quicksort comparison above, if the channel wants to do something smart, it must first test all the permutations. However, this is infeasible for a computationally bounded adversary, so the most it can do is make a random error pattern  $N$ . But then:

$$Z = \pi^{-1}(Y' \oplus R) = \pi^{-1}(Y \oplus N \oplus R),$$

since  $Y' = Y \oplus N$  by definition.

$$= \pi^{-1}(Y \oplus R) \oplus N', \text{ where } N' = \pi^{-1}(N),$$

since any permutation is linear with respect to XOR,

$$= X \oplus N',$$

as per the definition of  $Y$  above.

Since  $\pi$  is random,  $N'$  is just random noise and we can use the simple decoder to decode the received message and get back  $M$ .

### 23.3 Specific applications

By assuming a computationally bounded adversary, it is possible to design a **locally decodable code** which is both efficient and near-optimal, with a negligible error probability. These codes are used in complexity theory for things like self-correcting computations, probabilistically checkable proof systems, and worst-case to average-case hardness reductions in the constructions of pseudo-random generators. They are useful in cryptography as a result of their connection with **private information retrieval** protocols. They are also in a number of database applications like **fault-tolerant** data storage.<sup>[2]</sup>

Furthermore, it is possible to construct codes which surpass known bounds for worst-case codes—specifically, unique decoding with a  $1 - R$  error rate.<sup>[3]</sup> This can be done by concatenating timestamped digital signatures onto messages. A computationally bounded channel cannot forge a signature; and while it may have valid past signatures, the receiver can use **list decoding** and select a message only if its signature has the correct timestamp.

### 23.4 References

- [1] Lipton (6 May 2009). “Worst Case Complexity”. Gödel’s Lost Letter and P=NP. Retrieved 2013-04-01.
- [2] Ostrovsky, Pandey, Sahai. “Private Locally Decodable Codes” (PDF). Retrieved 2013-04-01.
- [3] Micali, Peikert; Sudan, A. Wilson. “Optimal Error Correction for Computationally Bounded Noise.” (PDF). Retrieved 2013-04-01.



## Chapter 24

# Concatenated error correction code

In coding theory, **concatenated codes** form a class of error-correcting codes that are derived by combining an **inner code** and an **outer code**. They were conceived in 1966 by Dave Forney as a solution to the problem of finding a code that has both exponentially decreasing error probability with increasing block length and polynomial-time decoding complexity.<sup>[1]</sup> Concatenated codes became widely used in space communications in the 1970s.

### 24.1 Background

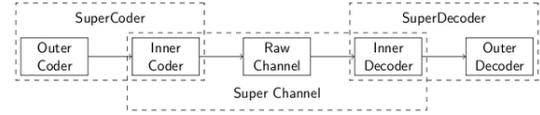
The field of **channel coding** is concerned with sending a stream of data at the highest possible rate over a given **communications channel**, and then decoding the original data reliably at the receiver, using encoding and decoding algorithms that are feasible to implement in a given technology.

Shannon's **channel coding theorem** shows that over many common channels there exist channel coding schemes that are able to transmit data reliably at all rates  $R$  less than a certain threshold  $C$ , called the **channel capacity** of the given channel. In fact, the probability of decoding error can be made to decrease exponentially as the block length  $N$  of the coding scheme goes to infinity. However, the complexity of a naive optimum decoding scheme that simply computes the likelihood of every possible transmitted codeword increases exponentially with  $N$ , so such an optimum decoder rapidly becomes infeasible.

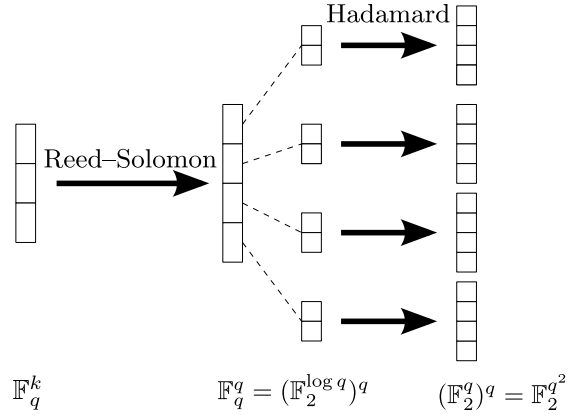
In his **doctoral thesis**, Dave Forney showed that concatenated codes could be used to achieve exponentially decreasing error probabilities at all data rates less than capacity, with decoding complexity that increases only polynomially with the code block length.

### 24.2 Description

Let  $C_{in}$  be a  $[n, k, d]$  code, that is, a **block code** of length  $n$ , **dimension**  $k$ , minimum **Hamming distance**  $d$ , and rate  $r = k/n$ , over an alphabet  $A$ :



Schematic depiction of a concatenated code built upon an inner code and an outer code.



This is a pictorial representation of a code concatenation, and, in particular, the **Reed-Solomon code** with  $n=q=4$  and  $k=2$  is used as the outer code and the **Hadamard code** with  $n=q$  and  $k=\log q$  is used as the inner code. Overall, the concatenated code is a  $[q^2, k \log q]$ -code.

$$C_{in} : A^k \rightarrow A^n$$

Let  $C_{out}$  be a  $[N, K, D]$  code over an alphabet  $B$  with  $|B| = |A|^k$  symbols:

$$C_{out} : B^K \rightarrow B^N$$

The inner code  $C_{in}$  takes one of  $|A|^k = |B|$  possible inputs, encodes into an  $n$ -tuple over  $A$ , transmits, and decodes into one of  $|B|$  possible outputs. We regard this as a (super) channel which can transmit one symbol from the alphabet  $B$ . We use this channel  $N$  times to transmit each of the  $N$  symbols in a codeword of  $C_{out}$ . The **concatenation** of  $C_{out}$  (as outer code) with  $C_{in}$  (as inner code),

denoted  $C_{out} \circ C_{in}$ , is thus a code of length  $Nn$  over the alphabet  $A$ .<sup>[1]</sup>

$$C_{out} \circ C_{in} : A^{kK} \rightarrow A^{nN}$$

It maps each input message  $m = (m_1, m_2, \dots, m_K)$  to a codeword  $(C_{in}(m'_1), C_{in}(m'_2), \dots, C_{in}(m'_N))$ , where  $(m'_1, m'_2, \dots, m'_N) = C_{out}(m_1, m_2, \dots, m_K)$ .

The *key insight* in this approach is that if  $C_{in}$  is decoded using a **maximum-likelihood approach** (thus showing an exponentially decreasing error probability with increasing length), and  $C_{out}$  is a code with length  $N = 2^{nr}$  that can be decoded in polynomial time of  $N$ , then the concatenated code can be decoded in polynomial time of its combined length  $n2^{nr} = O(N \cdot \log(N))$  and shows an exponentially decreasing error probability, even if  $C_{in}$  has exponential decoding complexity.<sup>[1]</sup> This is discussed in more detail in section **Decoding concatenated codes**.

In a generalization of above concatenation, there are  $N$  possible inner codes  $C_{in,i}$  and the  $i$ -th symbol in a codeword of  $C_{out}$  is transmitted across the inner channel using the  $i$ -th inner code. The **Justesen codes** are examples of generalized concatenated codes, where the outer code is a **Reed-Solomon code**.

## 24.3 Properties

1. The distance of the concatenated code  $C_{out} \circ C_{in}$  is at least  $dD$ , that is, it is a  $[nN, kK, D']$  code with  $D' \geq dD$ .

*Proof:* Consider two different messages  $m^1 \neq m^2 \in B^K$ . Let  $\Delta$  denote the distance between two codewords. Then

$$\Delta(C_{out}(m^1), C_{out}(m^2)) \geq D.$$

Thus, there are at least  $D$  positions in which the sequence of  $N$  symbols of the codewords  $C_{out}(m^1)$  and  $C_{out}(m^2)$  differ. For these positions, denoted  $i$ , we have

$$\Delta(C_{in}(C_{out}(m^1)_i), C_{in}(C_{out}(m^2)_i)) \geq d.$$

Consequently, there are at least  $d \cdot D$  positions in the sequence of  $n \cdot N$  symbols taken from the alphabet  $A$  in which the two codewords differ, and hence

$$\Delta(C_{in}(C_{out}(m^1)), C_{in}(C_{out}(m^2))) \geq dD.$$

2. If  $C_{out}$  and  $C_{in}$  are **linear block codes**, then  $C_{out} \circ C_{in}$  is also a linear block code.

This property can be easily shown based on the idea of defining a **generator matrix** for the concatenated code in terms of the generator matrices of  $C_{out}$  and  $C_{in}$ .

## 24.4 Decoding concatenated codes

A natural concept for a decoding algorithm for concatenated codes is to first decode the inner code and then the outer code. For the algorithm to be practical it must be **polynomial-time** in the final block length. Consider that there is a polynomial-time unique decoding algorithm for the outer code. Now we have to find a polynomial-time decoding algorithm for the inner code. It is understood that polynomial running time here means that running time is polynomial in the final block length. The main idea is that if the inner block length is selected to be logarithmic in the size of the outer code then the decoding algorithm for the inner code may run in **exponential time** of the inner block length, and we can thus use an exponential-time but optimal **maximum likelihood decoder (MLD)** for the inner code.

In detail, let the input to the decoder be the vector  $y = (y_1, \dots, y_N) \in (A^n)^N$ . Then the decoding algorithm is a two-step process:

1. Use the MLD of the inner code  $C_{in}$  to reconstruct a set of inner code words  $y' = (y'_1, \dots, y'_N)$ , with  $y'_i = \text{MLDC}_{in}(y_i)$ ,  $1 \leq i \leq N$ .
2. Run the unique decoding algorithm for  $C_{out}$  on  $y'$ .

Now, the time complexity of the first step is  $O(N \cdot \exp(n))$ , where  $n = O(\log(N))$  is the inner block length. In other words, it is  $N^{O(1)}$  (i.e., polynomial-time) in terms of the outer block length  $N$ . As the outer decoding algorithm in step two is assumed to run in polynomial time the complexity of the overall decoding algorithm is polynomial-time as well.

### 24.4.1 Remarks

The decoding algorithm described above can be used to correct all errors up to less than  $dD/4$  in number. Using **minimum distance decoding**, the outer decoder can correct all inputs  $y'$  with less than  $D/2$  symbols  $y'_i$  in error. Similarly, the inner code can reliably correct an input  $y_i$  if less than  $d/2$  inner symbols are erroneous. Thus, for an outer symbol  $y'_i$  to be incorrect after inner decoding at least  $d/2$  inner symbols must have been in error, and for the outer code to fail this must have happened for at least  $D/2$  outer symbols. Consequently, the total number of inner symbols that must be received incorrectly for the concatenated code to fail must be at least  $d/2 \cdot D/2 = dD/4$ .

The algorithm also works if the inner codes are different, e.g., for Justesen codes. The **generalized minimum distance algorithm**, developed by Forney, can be used to correct up to  $dD/2$  errors.<sup>[2]</sup> It uses erasure information from the inner code to improve performance of the outer code, and was the first example of an algorithm using **soft-decision decoding**.<sup>[3][4]</sup>

## 24.5 Applications

Although a simple concatenation scheme was implemented already for the 1971 *Mariner* Mars orbiter mission,<sup>[5]</sup> concatenated codes were starting to be regularly used for deep space communication with the *Voyager* program, which launched two space probes in 1977.<sup>[6]</sup> Since then, concatenated codes became the workhorse for efficient error correction coding, and stayed so at least until the invention of turbo codes and LDPC codes.<sup>[5][6]</sup>

Typically, the inner code is not a block code but a soft-decision convolutional Viterbi-decoded code with a short constraint length.<sup>[7]</sup> For the outer code, a longer hard-decision block code, frequently a Reed-Solomon code with eight-bit symbols, is used.<sup>[1][5]</sup> The larger symbol size makes the outer code more robust to error bursts that can occur due to channel impairments, and also because erroneous output of the convolutional code itself is bursty.<sup>[1][5]</sup> An interleaving layer is usually added between the two codes to spread error bursts across a wider range.<sup>[5]</sup>

The combination of an inner Viterbi convolutional code with an outer Reed-Solomon code (known as an RSV code) was first used in *Voyager 2*,<sup>[5][8]</sup> and it became a popular construction both within and outside of the space sector. It is still notably used today for satellite communications, such as the DVB-S digital television broadcast standard.<sup>[9]</sup>

In a looser sense, any (serial) combination of two or more codes may be referred to as a concatenated code. For example, within the DVB-S2 standard, a highly efficient LDPC code is combined with an algebraic outer code in order to remove any resilient errors left over from the inner LDPC code due to its inherent error floor.<sup>[10]</sup>

A simple concatenation scheme is also used on the compact disc (CD), where an interleaving layer between two Reed-Solomon codes of different sizes spreads errors across various blocks.

## 24.6 Turbo codes: A parallel concatenation approach

The description above is given for what is now called a serially concatenated code. Turbo codes, as described first in 1993, implemented a parallel concatenation of two convolutional codes, with an interleaver between the two codes and an iterative decoder that passes information forth and back between the codes.<sup>[6]</sup> This design has a better performance than any previously conceived concatenated codes.

However, a key aspect of turbo codes is their iterated decoding approach. Iterated decoding is now also applied to serial concatenations in order to achieve higher coding

gains, such as within serially concatenated convolutional codes (SCCCs). An early form of iterated decoding was implemented with two to five iterations in the “Galileo code” of the Galileo space probe.<sup>[5]</sup>

## 24.7 See also

- Justesen code
- Zyablov bound
- Singleton bound
- Gilbert–Varshamov bound

## 24.8 References

- [1] G. D. Forney (1967). “Concatenated codes”. Cambridge, Massachusetts: MIT Press.
- [2] Forney, G. David (April 1966). “Generalized Minimum Distance Decoding”. *Transactions on Information Theory*. IEEE. **12** (2): 125–131.
- [3] Yu, Christopher C.H.; Costello, Daniel J. (March 1980). “Generalized Minimum Distance Decoding for Qary Output Channels”. *Transactions on Information Theory*. IEEE. **26** (2): 238–243.
- [4] Wu, Yingquan; Hadjicostis, Christoforos (January 2007). “Soft-Decision Decoding of Linear Block Codes Using Preprocessing and Diversification”. *Transactions on Information Theory*. IEEE. **53** (1): 387–393.
- [5] Robert J. McEliece; Laif Swanson (20 August 1993). “Reed-Solomon Codes and the Exploration of the Solar System”. JPL.
- [6] K. Andrews et al., *The Development of Turbo and LDPC Codes for Deep-Space Applications*, Proceedings of the IEEE, Vol. 95, No. 11, Nov. 2007.
- [7] J. P. Odenwalder (1970). “Optimal decoding of convolutional codes”. U.C.L.A., Systems Science Dept. (*dissertation*).
- [8] R. Ludwig, J. Taylor, *Voyager Telecommunications Manual*, JPL DESCANSO (*Design and Performance Summary Series*), March 2002.
- [9] Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for 11/12 GHz satellite services, ETSI EN 300 421, V1.1.2, August 1997.
- [10] Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2), ETSI EN 302 307, V1.2.1, April 2009.

## 24.9 Further reading

- Shu Lin; Daniel J. Costello, Jr. (1983). *Error Control Coding: Fundamentals and Applications*. Prentice Hall. pp. 278–280. ISBN 0-13-283796-X.
- F.J. MacWilliams; N.J.A. Sloane (1977). *The Theory of Error-Correcting Codes*. North-Holland. pp. 307–316. ISBN 0-444-85193-3.

## 24.10 External links

- Dave Forney (ed.). “Concatenated codes”. *Scholarpedia*.
- University at Buffalo Lecture Notes on Coding Theory – Dr. Atri Rudra

# Chapter 25

## Coset leader

In coding theory, a **coset leader** is a word of minimum weight in any particular coset - that is, a word with the lowest amount of non-zero entries. Sometimes there are several words of equal minimum weight in a coset, and in that case, any one of those words may be chosen to be the coset leader.

Coset leaders are used in the construction of a standard array for a linear code, which can then be used to decode received vectors. For a received vector  $y$ , the decoded message is  $y - e$ , where  $e$  is the coset leader of  $y$ . Coset leaders can also be used to construct a fast decoding strategy. For each coset leader  $u$  we calculate the syndrome  $uH'$ . When we receive  $v$  we evaluate  $vH'$  and find the matching syndrome. The corresponding coset leader is the most likely error pattern and we assume that  $v+u$  was the codeword sent.

### 25.1 References

- Hill, Raymond (1986). *A First Course in Coding Theory*. Oxford Applied Mathematics and Computing Science series. Oxford University Press. ISBN 978-0-19-853803-5.

# Chapter 26

## Covering code

In coding theory, a **covering code** is a set of elements (called *codewords*) in a space, with the property that every element of the space is within a fixed distance of some codeword.

### 26.1 Definition

Let  $q \geq 2$ ,  $n \geq 1$ ,  $R \geq 0$  be integers. A code  $C \subseteq Q^n$  over an alphabet  $Q$  of size  $|Q| = q$  is called  $q$ -ary  $R$ -covering code of length  $n$  if for every word  $y \in Q^n$  there is a codeword  $x \in C$  such that the Hamming distance  $d_H(x, y) \leq R$ . In other words, the spheres (or balls or rook-domains) of radius  $R$  with respect to the Hamming metric around the codewords of  $C$  have to exhaust the finite metric space  $Q^n$ . The covering radius of a code  $C$  is the smallest  $R$  such that  $C$  is  $R$ -covering. Every perfect code is a covering code of minimal size.

### 26.2 Example

$C = \{0134, 0223, 1402, 1431, 1444, 2123, 2234, 3002, 3310, 4010, 4341\}$  is a 5-ary 2-covering code of length 4.<sup>[1]</sup>

### 26.3 Covering problem

The determination of the minimal size  $K_q(n, R)$  of a  $q$ -ary  $R$ -covering code of length  $n$  is a very hard problem. In many cases, only upper and lower bounds are known with a large gap between them. Every construction of a covering code gives an upper bound on  $K_q(n, R)$ . Lower bounds include the sphere covering bound and Rodemich's bounds  $K_q(n, 1) \geq q^{n-1}/(n-1)$  and  $K_q(n, n-2) \geq q^2/(n-1)$ .<sup>[2]</sup> The covering problem is closely related to the packing problem in  $Q^n$ , i.e. the determination of the maximal size of a  $q$ -ary  $e$ -error correcting code of length  $n$ .

### 26.4 Football pools problem

A particular case is the **football pools problem**, based on football pool betting, where the aim is to predict the results of  $n$  football matches as a home win, draw or away win, or to at least predict  $n-1$  of them with multiple bets. Thus a ternary covering,  $K_3(n, 1)$ , is sought.

If  $n = \frac{1}{2}(3^k - 1)$  then  $3^{n-k}$  are needed, so for  $n = 4$ ,  $k = 2$ , 9 are needed; for  $n = 13$ ,  $k = 3$ , 59049 are needed.<sup>[3]</sup> The best bounds known as of 2011<sup>[4]</sup> are

### 26.5 Applications

The standard work<sup>[5]</sup> on covering codes lists the following applications.

- Compression with distortion
- Data compression
- Decoding errors and erasures
- Broadcasting in interconnection networks
- Football pools<sup>[6]</sup>
- Write-once memories
- Berlekamp-Gale game
- Speech coding
- Cellular telecommunications
- Subset sums and Cayley graphs

### 26.6 References

- [1] P.R.J. Östergård, Upper bounds for  $q$ -ary covering codes, *IEEE Transactions on Information Theory*, 37 (1991), 660-664
- [2] E.R. Rodemich, Covering by rook-domains, *Journal of Combinatorial Theory*, 9 (1970), 117-128

- [3] <http://alexandria.tue.nl/repository/freearticles/593454.pdf>
- [4] [http://www.sztaki.hu/~{ }keri/codes/3\\_tables.pdf](http://www.sztaki.hu/~{ }keri/codes/3_tables.pdf)
- [5] G. Cohen, I. Honkala, S. Litsyn, A. Lobstein, *Covering Codes*, Elsevier (1997) ISBN 0-444-82511-8
- [6] H. Härmäläinen, I. Honkala, S. Litsyn, P.R.J. Östergård, Football pools - a game for mathematicians, *American Mathematical Monthly*, 102 (1995), 579-588

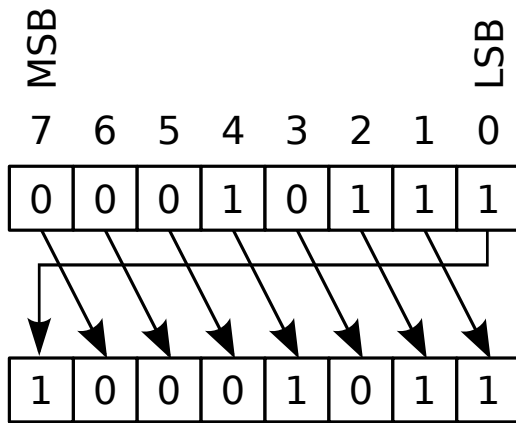
## 26.7 External links

- Literature on covering codes
- Bounds on  $K_q(n, R)$

# Chapter 27

## Cyclic code

In coding theory, a **cyclic code** is a block code, where the circular shifts of each codeword gives another word that belongs to the code. They are **error-correcting codes** that have algebraic properties that are convenient for efficient error detection and correction.



If 00010111 is a valid codeword, applying a right circular shift gives the string 10001011. If the code is cyclic, then 10001011 is again a valid codeword. In general, applying a right circular shift moves the least significant bit (LSB) to the leftmost position, so that it becomes the most significant bit (MSB); the other positions are shifted by 1 to the right.

### 27.1 Definition

Let  $C$  be a linear code over a finite field  $GF(q)$  of block length  $n$ .  $C$  is called a **cyclic code** if, for every codeword  $c = (c_1, \dots, c_n)$  from  $C$ , the word  $(c_n, c_1, \dots, c_{n-1})$  in  $GF(q)^n$  obtained by a **cyclic right shift** of components is again a codeword. Because one cyclic right shift is equal to  $n - 1$  cyclic left shifts, a cyclic code may also be defined via cyclic left shifts. Therefore the linear code  $C$  is cyclic precisely when it is invariant under all cyclic shifts.

Cyclic Codes have some additional structural constraint on the codes. They are based on **Galois fields** and because of their structural properties they are very useful for error controls. Their structure is strongly related to Galois fields because of which the encoding and decoding algorithms

for cyclic codes are computationally efficient.

### 27.2 Algebraic structure

Cyclic codes can be linked to ideals in certain rings. Let  $R = A[x]/(x^n - 1)$  be a **polynomial ring** over the finite field  $A = GF(q)$ . Identify the elements of the cyclic code  $C$  with polynomials in  $R$  such that  $(c_0, \dots, c_{n-1})$  maps to the polynomial  $c_0 + c_1x + \dots + c_{n-1}x^{n-1}$ ; thus multiplication by  $x$  corresponds to a cyclic shift. Then  $C$  is an **ideal** in  $R$ , and hence **principal**, since  $R$  is a **principal ideal ring**. The ideal is generated by the unique monic element in  $C$  of minimum degree, the *generator polynomial*  $g$ .<sup>[1]</sup> This must be a divisor of  $x^n - 1$ . It follows that every cyclic code is a **polynomial code**. If the generator polynomial  $g$  has degree  $d$  then the rank of the code  $C$  is  $n - d$ .

The **idempotent** of  $C$  is a codeword  $e$  such that  $e^2 = e$  (that is,  $e$  is an **idempotent element** of  $C$ ) and  $e$  is an identity for the code, that is  $ec = c$  for every codeword  $c$ . If  $n$  and  $q$  are **coprime** such a word always exists and is unique;<sup>[2]</sup> it is a generator of the code.

An **irreducible code** is a cyclic code in which the code, as an ideal is irreducible, i.e. is minimal in  $R$ , so that its check polynomial is an **irreducible polynomial**.

### 27.3 Examples

For example, if  $A = \mathbb{F}_2$  and  $n=3$ , the set of codewords contained in cyclic code generated by  $(1,1,0)$  is precisely

$$((0, 0, 0), (1, 1, 0), (0, 1, 1), (1, 0, 1))$$

It corresponds to the ideal in  $\mathbb{F}_2[x]/(x^3 - 1)$  generated by  $(1 + x)$ .

Note that  $(1 + x)$  is an irreducible polynomial in the polynomial ring, and hence the code is an irreducible code.

The idempotent of this code is the polynomial  $x + x^2$ , corresponding to the codeword  $(1,1,0)$ .



### 27.3.1 Trivial examples

Trivial examples of cyclic codes are  $A^n$  itself and the code containing only the zero codeword. These correspond to generators 1 and  $x^n - 1$  respectively: these two polynomials must always be factors of  $x^n - 1$ .

Over  $GF(2)$  the **parity bit** code, consisting of all words of even weight, corresponds to generator  $x + 1$ . Again over  $GF(2)$  this must always be a factor of  $x^n - 1$ .

## 27.4 Quasi-cyclic codes and shortened codes

Before delving into the details of cyclic codes first we will discuss quasi-cyclic and shortened codes which are closely related to the cyclic codes and they all can be converted into each other.

### 27.4.1 Definition

Quasi-cyclic codes:

An  $(n, k)$  *quasi-cyclic code* is a linear block code such that, for some  $b$  which is coprime to  $n$ , the polynomial  $x^b c(x) \pmod{x^n - 1}$  is a *codeword polynomial* whenever  $c(x)$  is a codeword polynomial.

Here, *codeword polynomial* is an element of a linear code whose **code words** are polynomials that are divisible by a polynomial of shorter length called the *generator polynomial*. Every codeword polynomial can be expressed in the form  $c(x) = a(x)g(x)$ , where  $g(x)$  is the generator polynomial. Any codeword  $(c_0, \dots, c_{n-1})$  of a cyclic code  $C$  can be associated with a codeword polynomial, namely,  $\sum_{i=0}^{n-1} c_i * x^i$ . A quasi-cyclic code with  $b$  equal to 1 is a cyclic code.

### 27.4.2 Definition

Shortened codes:

An  $[n, k]$  linear code is called a *proper shortened cyclic code* if it can be obtained by deleting  $b$  positions from an  $(n + b, k + b)$  cyclic code.

In shortened codes information symbols are deleted to obtain a desired blocklength smaller than the design blocklength. The missing information symbols are usually imagined to be at the beginning of the codeword and are considered to be 0. Therefore,  $n - k$  is fixed, and then  $k$  is decreased which eventually decreases  $n$ . Note that it is not necessary to delete the starting symbols. Depending on the application sometimes consecutive positions are considered as 0 and are deleted.

All the symbols which are dropped need not be transmitted and at the receiving end can be reinserted. To convert

$(n, k)$  cyclic code to  $(n - b, k - b)$  shortened code, set  $b$  symbols to zero and drop them from each codeword. Any cyclic code can be converted to quasi-cyclic codes by dropping every  $b$ th symbol where  $b$  is a factor of  $n$ . If the dropped symbols are not check symbols then this cyclic code is also a shortened code.

## 27.5 Cyclic codes for correcting errors

Now, we will begin the discussion of cyclic codes explicitly with **error detection and correction**. Cyclic codes can be used to correct errors, like **Hamming codes** as a cyclic codes can be used for correcting single error. Likewise, they are also used to correct double errors and burst errors. All types of error corrections are covered briefly in the further subsections.

The (7,4) Hamming code has a **generator polynomial**  $g(x) = x^3 + x + 1$ . This polynomial has a zero in **Galois extension field**  $GF(8)$  at the primitive element  $\alpha$ , and all codewords satisfy  $C(\alpha) = 0$ . Cyclic codes can also be used to correct double errors over the field  $GF(2)$ . Blocklength will be  $n$  equal to  $2^m - 1$  and primitive elements  $\alpha$  and  $\alpha^3$  as zeros in the  $GF(2^m)$  because we are considering the case of two errors here, so each will represent one error.

The received word is a polynomial of degree  $n - 1$  given as  $v(x) = a(x)g(x) + e(x)$

where  $e(x)$  can have at most two nonzero coefficients corresponding to 2 errors.

We define the **Syndrome Polynomial**,  $S(x)$  as the remainder of polynomial  $v(x)$  when divided by the generator polynomial  $g(x)$  i.e.

$$S(x) = v(x) \pmod{g(x)} = (a(x)g(x) + e(x)) \pmod{g(x)} = e(x) \pmod{g(x)} \text{ as } (a(x)g(x)) \pmod{g(x)} \text{ is zero.}$$

### 27.5.1 For correcting two errors

Let the field elements  $X_1$  and  $X_2$  be the two error location numbers. If only one error occurs then  $X_2$  is equal to zero and if none occurs both are zero.

Let  $S_1 = v(\alpha)$  and  $S_3 = v(\alpha^3)$ .

These field elements are called “syndromes”. Now because  $g(x)$  is zero at primitive elements  $\alpha$  and  $\alpha^3$ , so we can write  $S_1 = e(\alpha)$  and  $S_3 = e(\alpha^3)$ . If say two errors occur, then

$$S_1 = \alpha^i + \alpha^{i'} \text{ and } S_3 = \alpha^{3i} + \alpha^{3i'}.$$

And these two can be considered as two pair of equations in  $GF(2^m)$  with two unknowns and hence we can write

$$S_1 = X_1 + X_2 \text{ and } S_3 = (X_1)^3 + (X_2)^3.$$

Hence if the two pair of nonlinear equations can be solved cyclic codes can be used to correct two errors.

code is a single error correcting code over  $GF(2)$  with  $n = 2^m - 1$  and  $k = n - m$ .

## 27.6 Hamming code

The **Hamming(7,4)** code may be written as a cyclic code over  $GF(2)$  with generator  $1 + x + x^3$ . In fact, any binary **Hamming code** of the form  $Ham(r, 2)$  is equivalent to a cyclic code,<sup>[3]</sup> and any Hamming code of the form  $Ham(r, q)$  with  $r$  and  $q-1$  relatively prime is also equivalent to a cyclic code.<sup>[4]</sup> Given a Hamming code of the form  $Ham(r, 2)$  with  $r \geq 3$ , the set of even codewords forms a cyclic  $[2^r - 1, 2^r - r - 2, 4]$ -code.<sup>[5]</sup>

### 27.6.1 Hamming code for correcting single errors

A code whose minimum distance is at least 3, have a check matrix all of whose columns are distinct and non zero. If a check matrix for a binary code has  $m$  rows, then each column is an  $m$ -bit binary number. There are  $2^m - 1$  possible columns. Therefore if a check matrix of a binary code with  $d_{min}$  at least 3 has  $m$  rows, then it can only have  $2^m - 1$  columns, not more than that. This defines a  $(2^m - 1, 2^m - 1 - m)$  code, called Hamming code.

It is easy to define Hamming codes for large alphabets of size  $q$ . We need to define one  $H$  matrix with linearly independent columns. For any word of size  $q$  there will be columns who are multiples of each other. So, to get linear independence all non zero  $m$ -tuples with one as a top most non zero element will be chosen as columns. Then two columns will never be linearly dependent because three columns could be linearly dependent with the minimum distance of the code as 3.

So, there are  $(q^m - 1)/(q - 1)$  nonzero columns with one as top most non zero element. Therefore, Hamming code is a  $[(q^m - 1)/(q - 1), (q^m - 1)/(q - 1) - m]$  code.

Now, for cyclic codes, Let  $\alpha$  be primitive element in  $GF(q^m)$ , and let  $\beta = \alpha^{q-1}$ . Then  $\beta^{(q^m-1)/(q-1)} = 1$  and thus  $\beta$  is a zero of the polynomial  $x^{(q^m-1)/(q-1)} - 1$  and is a generator polynomial for the cyclic code of block length  $n = (q^m - 1)/(q - 1)$ .

But for  $q = 2$ ,  $\alpha = \beta$ . And the received word is a polynomial of degree  $n - 1$  given as

$$v(x) = a(x)g(x) + e(x)$$

where,  $e(x) = 0$  or  $x^i$  where  $i$  represents the error locations.

But we can also use  $\alpha^i$  as an element of  $GF(2^m)$  to index error location. Because  $g(\alpha) = 0$ , we have  $v(\alpha) = \alpha^i$  and all powers of  $\alpha$  from 0 to  $2^m - 2$  are distinct. Therefore we can easily determine error location  $i$  from  $\alpha^i$  unless  $v(\alpha) = 0$  which represents no error. So, hamming

## 27.7 Cyclic codes for correcting burst errors

From **Hamming distance** concept, a code with minimum distance  $2t+1$  can correct any  $t$  errors. But in many channels error pattern is not very arbitrary, it occurs within very short segment of the message. Such kind of errors are called **burst errors**. So, for correcting such errors we will get a more efficient code of higher rate because of the less constraints. Cyclic codes are used for correcting burst error. In fact, cyclic codes can also correct cyclic burst errors along with burst errors. Cyclic burst errors are defined as

A cyclic burst of length  $t$  is a vector whose nonzero components are among  $t$  (cyclically) consecutive components, the first and the last of which are nonzero.

In polynomial form cyclic burst of length  $t$  can be described as  $e(x) = x^i b(x) \mod (x^n - 1)$  with  $b(x)$  as a polynomial of degree  $t - 1$  with nonzero coefficient  $b_0$ . Here  $b(x)$  defines the pattern and  $x^i$  defines the starting point of error. Length of the pattern is given by  $\deg b(x) + 1$ . Syndrome polynomial is unique for each pattern and is given by

$$s(x) = e(x) \mod g(x)$$

A linear block code that corrects all burst errors of length  $t$  or less must have at least  $2t$  check symbols. Proof: Because any linear code that can correct burst pattern of length  $t$  or less cannot have a burst of length  $2t$  or less as a codeword because if it did then a burst of length  $t$  could change the codeword to burst pattern of length  $t$ , which also could be obtained by making a burst error of length  $t$  in all zero codeword. Now, any two vectors that are non zero in the first  $2t$  components must be from different co-sets of an array to avoid their difference being a codeword of bursts of length  $2t$ . Therefore number of such co-sets are equal to number of such vectors which are  $q^{2t}$ . Hence at least  $q^{2t}$  co-sets and hence at least  $2t$  check symbol.

This property is also known as Rieger bound and it is similar to the **singleton bound** for random error correcting.

### 27.7.1 Fire codes as cyclic bounds

In 1959, Philip Fire<sup>[6]</sup> presented a construction of cyclic codes generated by a product of a binomial and a primitive polynomial. The binomial has the form  $x^c + 1$  for some positive odd integer  $c$ .<sup>[7]</sup> **Fire code** is a cyclic burst error correcting code over  $GF(q)$  with the generator polynomial

$$g(x) = (x^{2t-1} - 1)p(x)$$

where  $p(x)$  is a prime polynomial with degree  $m$  not smaller than  $t$  and  $p(x)$  does not divide  $x^{2t-1} - 1$ . Block length of the fire code is the smallest integer  $n$  such that  $g(x)$  divides  $x^n - 1$ .

A fire code can correct all burst errors of length  $t$  or less if no two bursts  $b(x)$  and  $x^j b'(x)$  appear in the same co-set. This can be proved by contradiction. Suppose there are two distinct nonzero bursts  $b(x)$  and  $x^j b'(x)$  of length  $t$  or less and are in the same co-set of the code. So, their difference is a codeword. As the difference is a multiple of  $g(x)$  it is also a multiple of  $x^{2t-1} - 1$ . Therefore,

$$b(x) = x^j b'(x) \pmod{x^{2t-1} - 1}.$$

This shows that  $j$  is a multiple of  $2t - 1$ , So

$$b(x) = x^{l(2t-1)} b'(x)$$

for some  $l$ . Now, as  $l(2t - 1)$  is less than  $t$  and  $l$  is less than  $q^m - 1$  so  $(x^{l(2t-1)} - 1)b(x)$  is a codeword. Therefore,

$$(x^{l(2t-1)} - 1)b(x) = a(x)(x^{2t-1} - 1)p(x).$$

Since  $b(x)$  degree is less than degree of  $p(x)$ ,  $p(x)$  cannot divide  $b(x)$ . If  $l$  is not zero, then  $p(x)$  also cannot divide  $x^{l(2t-1)} - 1$  as  $l$  is less than  $q^m - 1$  and by definition of  $m$ ,  $p(x)$  divides  $x^{l(2t-1)} - 1$  for no  $l$  smaller than  $q^m - 1$ . Therefore  $l$  and  $j$  equals to zero. That means both that both the bursts are same, contrary to assumption.

Fire codes are the best single burst correcting codes with high rate and they are constructed analytically. They are of very high rate and when  $m$  and  $t$  are equal, redundancy is least and is equal to  $3t - 1$ . By using multiple fire codes longer burst errors can also be corrected.

For error detection cyclic codes are widely used and are called  $t - 1$  **cyclic redundancy codes**.

## 27.8 Cyclic codes on Fourier transform

Applications of **Fourier transform** are widespread in signal processing. But their applications are not limited to the complex fields only; Fourier transforms also exist in the Galois field  $GF(q)$ . Cyclic codes using Fourier transform can be described in a setting closer to the signal processing.

### 27.8.1 Fourier transform over finite fields

#### Fourier transform over finite fields

The discrete Fourier transform of vector  $v = v_0, v_1, \dots, v_{n-1}$  is given by a vector  $V = V_0, V_1, \dots, V_{n-1}$  where,

$$V_k = \sum_{i=0}^{n-1} e^{-j2\pi n^{-1}ik} v_i \text{ where,}$$

$$k = 0, \dots, n - 1$$

where  $\exp(-j2\pi/n)$  is an  $n$ th root of unity. Similarly in the finite field  $n$ th root of unity is element  $\omega$  of order  $n$ . Therefore

If  $v = (v_0, v_1, \dots, v_{n-1})$  is a vector over  $GF(q)$ , and  $\omega$  be an element of  $GF(q)$  of order  $n$ , then Fourier transform of the vector  $v$  is the vector  $V = (V_0, V_1, \dots, V_{n-1})$  and components are given by

$$V_j = \sum_{i=0}^{n-1} \omega^{ij} v_i \text{ where,}$$

$$k = 0, \dots, n - 1$$

Here  $i$  is time index,  $j$  is frequency and  $V$  is the spectrum. One important difference between Fourier transform in complex field and Galois field is that complex field  $\omega$  exists for every value of  $n$  while in Galois field  $\omega$  exists only if  $n$  divides  $q - 1$ . In case of extension fields, there will be a Fourier transform in the extension field  $GF(q^m)$  if  $n$  divides  $q^m - 1$  for some  $m$ . In Galois field time domain vector  $v$  is over the field  $GF(q)$  but the spectrum  $V$  may be over the extension field  $GF(q^m)$ .

### 27.8.2 Spectral description of cyclic codes

Any codeword of cyclic code of blocklength  $n$  can be represented by a polynomial  $c(x)$  of degree at most  $n - 1$ . Its encoder can be written as  $c(x) = a(x)g(x)$ . Therefore in frequency domain encoder can be written as  $C_j = A_j G_j$ . Here *codeword spectrum*  $C_j$  has a value in  $GF(q^m)$  but all the components in the time domain are from  $GF(q)$ . As the data spectrum  $A_j$  is arbitrary, the role of  $G_j$  is to specify those  $j$  where  $C_j$  will be zero.

Thus, cyclic codes can also be defined as

Given a set of spectral indices,  $A = (j_1, \dots, j_{n-k})$ , whose elements are called *check frequencies*, the cyclic code  $C$  is the set of words over  $GF(q)$  whose spectrum is zero in the components indexed by  $j_1, \dots, j_{n-k}$ . Any such spectrum  $C$  will have components of the form  $A_j G_j$ .

So, cyclic codes are vectors in the field  $GF(q)$  and the spectrum given by its inverse fourier transform is over the field  $GF(q^m)$  and are constrained to be zero at certain components. But note that every spectrum in the field  $GF(q^m)$  and zero at certain components may not have inverse transforms with components in the field  $GF(q)$ . Such spectrum can not be used as cyclic codes.

Following are the few bounds on the spectrum of cyclic codes.

#### BCH bound

If  $n$  be a factor of  $(q^m - 1)$  for some  $m$ . The only vector in  $GF(q)^n$  of weight  $d - 1$  or less that has  $d - 1$  consecutive components of its spectrum equal to zero is all-zero vector.

### Hartmann-Tzeng bound

If  $n$  be a factor of  $(q^m - 1)$  for some  $m$ , and  $b$  an integer that is coprime with  $n$ . The only vector  $v$  in  $GF(q)^n$  of weight  $d - 1$  or less whose spectral components  $V_j$  equal zero for  $j = \ell_1 + \ell_2 b \pmod{n}$ , where  $\ell_1 = 0, \dots, d - s - 1$  and  $\ell_2 = 0, \dots, s - 1$ , is the all zero vector.

### Roos bound

If  $n$  be a factor of  $q^m - 1$  for some  $m$  and  $GCD(n, b) = 1$ . The only vector in  $GF(q)^n$  of weight  $d - 1$  or less whose spectral components  $V_j$  equal to zero for  $j = l_1 + l_2 b \pmod{n}$ , where  $l_1 = 0, \dots, d - s - 2$  and  $l_2$  takes at least  $s + 1$  values in the range  $0, \dots, d - 2$ , is the all-zero vector.

### 27.8.3 Quadratic residue codes

When the prime  $l$  is a quadratic residue modulo the prime  $p$  there is a **quadratic residue code** which is a cyclic code of length  $p$ , dimension  $(p + 1)/2$  and minimum weight at least  $\sqrt{p}$  over  $GF(l)$ .

## 27.9 Generalizations

A **constacyclic code** is a linear code with the property that for some constant  $\lambda$  if  $(c_1, c_2, \dots, c_n)$  is a codeword then so is  $(\lambda c_n, c_1, \dots, c_{n-1})$ . A **negacyclic code** is a constacyclic code with  $\lambda = -1$ .<sup>[8]</sup> A **quasi-cyclic code** has the property that for some  $s$ , any cyclic shift of a codeword by  $s$  places is again a codeword.<sup>[9]</sup> A **double circulant code** is a quasi-cyclic code of even length with  $s=2$ .<sup>[9]</sup>

### 27.10 See also

- Cyclic redundancy check
- Polynomial code
- BCH code
- Reed–Muller code
- Binary Golay code
- Ternary Golay code
- Eugene Prange

### 27.11 Notes

- [1] Van Lint 1998, p. 76  
[2] Van Lint 1998, p. 80

- [3] Hill 1988, pp. 159-160  
[4] Balahut 1983, Theorem 5.5.1  
[5] Hill 1988, pp. 162-163  
[6] P. Fire, E. P. (1959). *A class of multiple-error-correcting binary codes for non-independent errors*. Sylvania Reconnaissance Systems Laboratory, Mountain View, CA, Rept. RSL-E-2, 1959.  
[7] Wei Zhou, Shu Lin, Khaled Abdel-Ghaffar. *Burst or random error correction based on Fire and BCH codes*. ITA 2014: 1-5 2013.  
[8] Van Lint 1998, p. 75  
[9] MacWilliams & Sloane 1977, p. 506

## 27.12 References

- Blahut, Richard E. (2003), *Algebraic Codes for Data Transmission* (2nd ed.), Cambridge University Press, ISBN 0-521-55374-1
- Hill, Raymond (1988), *A First Course In Coding Theory*, Oxford University Press, ISBN 0-19-853803-0
- MacWilliams, F. J.; Sloane, N. J. A. (1977), *The Theory of Error-Correcting Codes*, New York: North-Holland Publishing, ISBN 0-444-85011-2
- Van Lint, J. H. (1998), *Introduction to Coding Theory*, Graduate Texts in Mathematics **86** (3rd ed.), Springer Verlag, ISBN 3-540-64133-5

## 27.13 Further reading

- Ranjan Bose, *Information theory, coding and cryptography*, ISBN 0-07-048297-7
- Irving S. Reed and Xuemin Chen, *Error-Control Coding for Data Networks*, Boston: Kluwer Academic Publishers, 1999, ISBN 0-7923-8528-4.
- Scott A. Vanstone, Paul C. Van Oorschot, *An introduction to error correcting codes with applications*, ISBN 0-7923-9017-2

## 27.14 External links

- John Gill's (Stanford) class notes – Notes #3, October 8, Handout #9, EE 387.
- Jonathan Hall's (MSU) class notes – Chapter 8. Cyclic codes - pp. 100 - 123
- David Terr. "Cyclic Code". *MathWorld*.

*This article incorporates material from [cyclic code on PlanetMath](#), which is licensed under the [Creative Commons Attribution/Share-Alike License](#).*

# Chapter 28

## Decoding methods

In coding theory, **decoding** is the process of translating received messages into **codewords** of a given code. There have been many common methods of mapping messages to codewords. These are often used to recover messages sent over a noisy channel, such as a **binary symmetric channel**.

### 28.1 Notation

$C \subset \mathbb{F}_2^n$  is considered a **binary code** with the length  $n$ ;  $x, y$  shall be elements of  $\mathbb{F}_2^n$ ; and  $d(x, y)$  is the distance between those elements.

### 28.2 Ideal observer decoding

One may be given the message  $x \in \mathbb{F}_2^n$ , then **ideal observer decoding** generates the codeword  $y \in C$ . The process results in this solution:

$$\mathbb{P}(y \text{ sent} \mid x \text{ received})$$

For example, a person can choose the codeword  $y$  that is most likely to be received as the message  $x$  after transmission.

#### 28.2.1 Decoding conventions

Each codeword does not have an expected possibility: there may be more than one codeword with an equal likelihood of mutating into the received message. In such a case, the sender and receiver(s) must agree ahead of time on a decoding convention. Popular conventions include:

1. Request that the codeword be resent -- **automatic repeat-request**
2. Choose any random codeword from the set of most likely codewords which is nearer to that.

3. If another code follows, mark the ambiguous bits of the codeword as erasures and hope that the outer code disambiguates them

### 28.3 Maximum likelihood decoding

Further information: **Maximum likelihood**

Given a received codeword  $x \in \mathbb{F}_2^n$  **maximum likelihood decoding** picks a codeword  $y \in C$  that maximizes

$$\mathbb{P}(x \text{ received} \mid y \text{ sent})$$

that is, the codeword  $y$  that maximizes the probability that  $x$  was received, **given that**  $y$  was sent. If all codewords are equally likely to be sent then this scheme is equivalent to ideal observer decoding. In fact, by **Bayes Theorem**,

$$\begin{aligned} \mathbb{P}(x \text{ received} \mid y \text{ sent}) &= \frac{\mathbb{P}(x \text{ received}, y \text{ sent})}{\mathbb{P}(y \text{ sent})} \\ &= \mathbb{P}(y \text{ sent} \mid x \text{ received}) \cdot \frac{\mathbb{P}(x \text{ received})}{\mathbb{P}(y \text{ sent})}. \end{aligned}$$

Upon fixing  $\mathbb{P}(x \text{ received})$ ,  $x$  is restructured and  $\mathbb{P}(y \text{ sent})$  is constant as all codewords are equally likely to be sent. Therefore  $\mathbb{P}(x \text{ received} \mid y \text{ sent})$  is maximised as a function of the variable  $y$  precisely when  $\mathbb{P}(y \text{ sent} \mid x \text{ received})$  is maximised, and the claim follows.

As with ideal observer decoding, a convention must be agreed to for non-unique decoding.

The maximum likelihood decoding problem can also be modeled as an **integer programming problem**.<sup>[1]</sup>

The maximum likelihood decoding algorithm is an instance of the “marginalize a product function” problem which is solved by applying the **generalized distributive law**.<sup>[2]</sup>



## 28.4 Minimum distance decoding

Given a received codeword  $x \in \mathbb{F}_2^n$ , **minimum distance decoding** picks a codeword  $y \in C$  to minimise the **Hamming distance** :

$$d(x, y) = \#\{i : x_i \neq y_i\}$$

i.e. choose the codeword  $y$  that is as close as possible to  $x$ .

Note that if the probability of error on a **discrete memoryless channel**  $p$  is strictly less than one half, then *minimum distance decoding* is equivalent to *maximum likelihood decoding*, since if

$$d(x, y) = d,$$

then:

$$\begin{aligned} \mathbb{P}(y \text{ received} \mid x \text{ sent}) &= (1-p)^{n-d} \cdot p^d \\ &= (1-p)^n \cdot \left(\frac{p}{1-p}\right)^d \end{aligned}$$

which (since  $p$  is less than one half) is maximised by minimising  $d$ .

Minimum distance decoding is also known as *nearest neighbour decoding*. It can be assisted or automated by using a **standard array**. Minimum distance decoding is a reasonable decoding method when the following conditions are met:

1. The probability  $p$  that an error occurs is independent of the position of the symbol
2. Errors are independent events - an error at one position in the message does not affect other positions

These assumptions may be reasonable for transmissions over a **binary symmetric channel**. They may be unreasonable for other media, such as a DVD, where a single scratch on the disk can cause an error in many neighbouring symbols or codewords.

As with other decoding methods, a convention must be agreed to for non-unique decoding.

## 28.5 Syndrome decoding

**Syndrome decoding** is a highly efficient method of decoding a **linear code** over a **noisy channel**, i.e. one on which errors are made. In essence, syndrome decoding is *minimum distance decoding* using a reduced lookup table. This is allowed by the linearity of the code.<sup>[3]</sup>

Suppose that  $C \subset \mathbb{F}_2^n$  is a linear code of length  $n$  and minimum distance  $d$  with **parity-check matrix**  $H$ . Then clearly  $C$  is capable of correcting up to

$$t = \left\lfloor \frac{d-1}{2} \right\rfloor$$

errors made by the channel (since if no more than  $t$  errors are made then minimum distance decoding will still correctly decode the incorrectly transmitted codeword).

Now suppose that a codeword  $x \in \mathbb{F}_2^n$  is sent over the channel and the error pattern  $e \in \mathbb{F}_2^n$  occurs. Then  $z = x + e$  is received. Ordinary minimum distance decoding would lookup the vector  $z$  in a table of size  $|C|$  for the nearest match - i.e. an element (not necessarily unique)  $c \in C$  with

$$d(c, z) \leq d(y, z)$$

for all  $y \in C$ . Syndrome decoding takes advantage of the property of the parity matrix that:

$$Hx = 0$$

for all  $x \in C$ . The *syndrome* of the received  $z = x + e$  is defined to be:

$$Hz = H(x + e) = Hx + He = 0 + He = He$$

To perform ML decoding in a **Binary symmetric channel**, one has to look-up a precomputed table of size  $2^{n-k}$ , mapping  $He$  to  $e$ .

Note that this is already of significantly less complexity than that of a **Standard array decoding**.

However, under the assumption that no more than  $t$  errors were made during transmission, the receiver can look up the value  $He$  in a further reduced table of size

$$\sum_{i=0}^t \binom{n}{i} < |C|$$

only (for a binary code). The table is against pre-computed values of  $He$  for all possible error patterns  $e \in \mathbb{F}_2^n$ .

Knowing what  $e$  is, it is then trivial to decode  $x$  as:

$$x = z - e$$

## 28.6 Partial response maximum likelihood

Main article: **PRML**



Partial response maximum likelihood (PRML) is a method for converting the weak analog signal from the head of a magnetic disk or tape drive into a digital signal.

## 28.7 Viterbi decoder

Main article: [Viterbi decoder](#)

A Viterbi decoder uses the Viterbi algorithm for decoding a bitstream that has been encoded using [forward error correction](#) based on a convolutional code. The [Hamming distance](#) is used as a metric for hard decision Viterbi decoders. The *squared Euclidean distance* is used as a metric for soft decision decoders.

## 28.8 See also

- [Error detection and correction](#)

## 28.9 Sources

- Hill, Raymond (1986). *A first course in coding theory*. Oxford Applied Mathematics and Computing Science Series. Oxford University Press. ISBN 0-19-853803-0.
- Pless, Vera (1982). *Introduction to the theory of error-correcting codes*. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons. ISBN 0-471-08684-3.
- J.H. van Lint (1992). *Introduction to Coding Theory*. GTM. **86** (2nd ed.). Springer-Verlag. ISBN 3-540-54894-7.

## 28.10 References

- [1] Feldman, Jon; Wainwright, Martin J.; Karger, David R. (March 2005). "Using Linear Programming to Decode Binary Linear Codes". *IEEE Transactions on Information Theory*. **51** (3). pp. 954–972. doi:10.1109/TIT.2004.842696.
- [2] Aji, Srinivas M.; McEliece, Robert J. (March 2000). "The Generalized Distributive Law". *IEEE Transactions on Information Theory*. **46** (2): 325–343. doi:10.1109/18.825794.
- [3] Albrecht Beutelspacher & Ute Rosenbaum (1998) *Projective Geometry*, page 190, Cambridge University Press ISBN 0-521-48277-1

# Chapter 29

## Deletion channel

A **deletion channel** is a **communications channel** model used in **coding theory** and **information theory**. In this model, a transmitter sends a **bit** (a zero or a one), and the receiver either receives the bit (with probability  $p$ ) or does not receive anything without being notified that the bit was dropped (with probability  $1-p$ ). Determining the **capacity** of the deletion channel is an open problem.<sup>[1][2]</sup>

The deletion channel should not be confused with the **binary erasure channel** which is much simpler to analyze.

with small deletion probability”, *IEEE Transactions on Information Theory*, **59** (10): 6192–6219, doi:10.1109/TIT.2013.2262020, MR 3106824.

### 29.1 Formal description

Let  $p$  be the deletion probability,  $0 < p < 1$ . The **iid** binary deletion channel is defined as follows:

Given an input sequence of  $n$  bits  $(X_i)$  as input, each bit in  $X_n$  can be deleted with probability  $p$ . The deletion positions are unknown to the sender and the receiver. The output sequence  $(Y_i)$  is the sequence of the  $(X_i)$  which were not deleted, in the correct order and with no errors.

### 29.2 Capacity

The **capacity** of the binary deletion channel (as an **analytical expression** of the deletion rate  $p$ ) is unknown. It has a **mathematical expression**. Several upper and lower bounds are known.

### 29.3 External links

- [Implementation of correction for deletion channel](#)

### 29.4 References

- [1] [Mitzenmacher, Michael](#) (2009), “A survey of results for deletion channels and related synchronization channels”, *Probability Surveys*, **6**: 1–33, doi:10.1214/08-PS141, MR 2525669.
- [2] [Kanoria, Yashodhan; Montanari, Andrea](#) (2013), “Optimal coding for the binary deletion channel

## Chapter 30

# Distributed source coding

**Distributed source coding (DSC)** is an important problem in **information theory** and **communication**. DSC problems regard the compression of multiple correlated information sources that do not communicate with each other.<sup>[1]</sup> By modeling the correlation between multiple sources at the decoder side together with **channel codes**, DSC is able to shift the computational complexity from encoder side to decoder side, therefore provide appropriate frameworks for applications with complexity-constrained sender, such as **sensor networks** and video/multimedia compression (see **distributed video coding**<sup>[2]</sup>). One of the main properties of distributed source coding is that the computational burden in encoders is shifted to the joint decoder.

### 30.1 History

In 1973, **David Slepian** and **Jack Keil Wolf** proposed the information theoretical lossless compression bound on distributed compression of two correlated i.i.d. sources  $X$  and  $Y$ .<sup>[3]</sup> After that, this bound was extended to cases with more than two sources by **Thomas M. Cover** in 1975,<sup>[4]</sup> while the theoretical results in the lossy compression case are presented by **Aaron D. Wyner** and **Jacob Ziv** in 1976.<sup>[5]</sup>

Although the theorems on DSC were proposed on 1970s, it was after about 30 years that attempts were started for practical techniques, based on the idea that DSC is closely related to channel coding proposed in 1974 by **Aaron D. Wyner**.<sup>[6]</sup> The asymmetric DSC problem was addressed by **S. S. Pradhan** and **K. Ramchandran** in 1999, which focused on statistically dependent binary and Gaussian sources and used scalar and trellis **coset** constructions to solve the problem.<sup>[7]</sup> They further extended the work into the symmetric DSC case.<sup>[8]</sup>

**Syndrome decoding** technology was first used in distributed source coding by the **DISCUS** system of **SS Pradhan** and **K Ramachandran** (Distributed Source Coding Using Syndromes).<sup>[7]</sup> They compress binary block data from one source into syndromes and transmit data from the other source uncompressed as **side information**. This kind of DSC scheme achieves asymmetric compression

rates per source and results in *asymmetric* DSC. This asymmetric DSC scheme can be easily extended to the case of more than two correlated information sources. There are also some DSC schemes that use **parity bits** rather than syndrome bits.

The correlation between two sources in DSC has been modeled as a **virtual channel** which is usually referred as a **binary symmetric channel**.<sup>[9][10]</sup>

Starting from **DISCUS**, DSC has attracted significant research activity and more sophisticated channel coding techniques have been adopted into DSC frameworks, such as **Turbo Code**, **LDPC Code**, and so on.

Similar to the previous lossless coding framework based on Slepian–Wolf theorem, efforts have been taken on lossy cases based on the Wyner–Ziv theorem. Theoretical results on quantizer designs was provided by **R. Zamir** and **S. Shamai**,<sup>[11]</sup> while different frameworks have been proposed based on this result, including a nested lattice quantizer and a trellis-coded quantizer.

Moreover, DSC has been used in video compression for applications which require low complexity video encoding, such as sensor networks, multiview video camcorders, and so on.<sup>[12]</sup>

With deterministic and probabilistic discussions of correlation model of two correlated information sources, DSC schemes with more general compressed rates have been developed.<sup>[13][14][15]</sup> In these *non-asymmetric* schemes, both of two correlated sources are compressed.

Under a certain deterministic assumption of correlation between information sources, a DSC framework in which any number of information sources can be compressed in a distributed way has been demonstrated by **X. Cao** and **M. Kuijper**.<sup>[16]</sup> This method performs non-asymmetric compression with flexible rates for each source, achieving the same overall compression rate as repeatedly applying asymmetric DSC for more than two sources. Then, by investigating the unique connection between syndromes and complementary codewords of linear codes, they have translated the major steps of DSC joint decoding into syndrome decoding followed by channel encoding via a linear block code and also via its complement code,<sup>[17]</sup> which theoretically illustrated a method of assembling a

DSC joint decoder from linear code encoders and decoders.

## 30.2 Theoretical bounds

The information theoretical lossless compression bound on DSC (the Slepian–Wolf bound) was first purposed by David Slepian and Jack Keil Wolf in terms of entropies of correlated information sources in 1973.<sup>[3]</sup> They also showed that two isolated sources can compress data as efficiently as if they were communicating with each other. This bound has been extended to the case of more than two correlated sources by Thomas M. Cover in 1975.<sup>[4]</sup>

Similar results were obtained in 1976 by Aaron D. Wyner and Jacob Ziv with regard to lossy coding of joint Gaussian sources.<sup>[5]</sup>

### 30.2.1 Slepian–Wolf bound

Distributed Coding is the coding of two or more dependent sources with separate encoders and joint decoder. Given two statistically dependent i.i.d. finite-alphabet random sequences  $X$  and  $Y$ , Slepian–Wolf theorem includes theoretical bound for the lossless coding rate for distributed coding of the two sources as below:<sup>[3]</sup>

$$R_X \geq H(X|Y),$$

$$R_Y \geq H(Y|X),$$

$$R_X + R_Y \geq H(X, Y).$$

If both the encoder and decoder of the two sources are independent, the lowest rate we can achieve for lossless compression is  $H(X)$  and  $H(Y)$  for  $X$  and  $Y$  respectively, where  $H(X)$  and  $H(Y)$  are the entropies of  $X$  and  $Y$ . However, with joint decoding, if vanishing error probability for long sequences is accepted, the Slepian–Wolf theorem shows that much better compression rate can be achieved. As long as the total rate of  $X$  and  $Y$  is larger than their joint entropy  $H(X, Y)$  and none of the sources is encoded with a rate larger than its entropy, distributed coding can achieve arbitrarily small error probability for long sequences.

A special case of distributed coding is compression with decoder side information, where source  $Y$  is available at the decoder side but not accessible at the encoder side. This can be treated as the condition that  $R_Y = H(Y)$  has already been used to encode  $Y$ , while we intend to use  $H(X|Y)$  to encode  $X$ . The whole system is operating in an asymmetric way (compression rate for the two sources are asymmetric).

### 30.2.2 Wyner–Ziv bound

Shortly after Slepian–Wolf theorem on lossless distributed compression was published, the extension to lossy compression with decoder side information was proposed as Wyner–Ziv theorem.<sup>[5]</sup> Similarly to lossless case, two statistically dependent i.i.d. sources  $X$  and  $Y$  are given, where  $Y$  is available at the decoder side but not accessible at the encoder side. Instead of lossless compression in Slepian–Wolf theorem, Wyner–Ziv theorem looked into the lossy compression case.

Wyner–Ziv theorem presents the achievable lower bound for the bit rate of  $X$  at given distortion  $D$ . It was found that for Gaussian memoryless sources and mean-squared error distortion, the lower bound for the bit rate of  $X$  remain the same no matter whether side information is available at the encoder or not.

## 30.3 Virtual channel

**Deterministic model**

**Probabilistic model**

## 30.4 Asymmetric DSC vs. symmetric DSC

Asymmetric DSC means that, different bitrates are used in coding the input sources, while same bitrate is used in symmetric DSC. Taking a DSC design with two sources for example, in this example  $X$  and  $Y$  are two discrete, memoryless, uniformly distributed sources which generate set of variables  $\mathbf{x}$  and  $\mathbf{y}$  of length 7 bits and the Hamming distance between  $\mathbf{x}$  and  $\mathbf{y}$  is at most one. The Slepian–Wolf bound for them is:

$$R_X + R_Y \geq 10$$

$$R_X \geq 5$$

$$R_Y \geq 5$$

This means, the theoretical bound is  $R_X + R_Y = 10$  and symmetric DSC means 5 bits for each source. Other pairs with  $R_X + R_Y = 10$  are asymmetric cases with different bit rate distributions between  $X$  and  $Y$ , where  $R_X = 3$ ,  $R_Y = 7$  and  $R_X = 7$ ,  $R_Y = 3$  represent two extreme cases called decoding with side information.

## 30.5 Practical distributed source coding

### 30.5.1 Slepian–Wolf coding – lossless distributed coding

It was understood that Slepian–Wolf coding is closely related to channel coding in 1974,<sup>[6]</sup> and after about 30 years, practical DSC started to be implemented by different channel codes. The motivation behind the use of channel codes is from two sources case, the correlation between input sources can be modeled as a virtual channel which has input as source  $X$  and output as source  $Y$ . The DISCUS system proposed by S. S. Pradhan and K. Ramchandran in 1999 implemented DSC with syndrome decoding, which worked for asymmetric case and was further extended to symmetric case.<sup>[7][8]</sup>

The basic framework of syndrome based DSC is that, for each source, its input space is partitioned into several cosets according to the particular channel coding method used. Every input of each source gets an output indicating which coset the input belongs to, and the joint decoder can decode all inputs by received coset indices and dependence between sources. The design of channel codes should consider the correlation between input sources.

A group of codes can be used to generate coset partitions,<sup>[18]</sup> such as trellis codes and lattice codes. Pradhan and Ramchandran designed rules for construction of sub-codes for each source, and presented result of trellis-based coset constructions in DSC, which is based on convolution code and set-partitioning rules as in Trellis modulation, as well as lattice code based DSC.<sup>[7][8]</sup> After this, embedded trellis code was proposed for asymmetric coding as an improvement over their results.<sup>[19]</sup>

After DISCUS system was proposed, more sophisticated channel codes have been adapted to the DSC system, such as Turbo Code, LDPC Code and Iterative Channel Code. The encoders of these codes are usually simple and easy to implement, while the decoders have much higher computational complexity and are able to get good performance by utilizing source statistics. With sophisticated channel codes which have performance approaching the capacity of the correlation channel, corresponding DSC system can approach the Slepian–Wolf bound.

Although most research focused on DSC with two dependent sources, Slepian–Wolf coding has been extended to more than two input sources case, and sub-codes generation methods from one channel code was proposed by V. Stankovic, A. D. Liveris, etc. given particular correlation models.<sup>[20]</sup>

#### General theorem of Slepian–Wolf coding with syndromes for two sources

**Theorem:** Any pair of correlated uniformly distributed sources,  $X, Y \in \{0, 1\}^n$ , with  $d_H(X, Y) \leq t$ , can be compressed separately at a rate pair  $(R_1, R_2)$  such that  $R_1, R_2 \geq n - k$ ,  $R_1 + R_2 \geq 2n - k$ , where  $R_1$  and  $R_2$  are integers, and  $k \leq n - \log(\sum_{i=0}^t \binom{n}{i})$ . This can be

achieved using an  $(n, k, 2t + 1)$  binary linear code.

*Proof:* The Hamming bound for an  $(n, k, 2t + 1)$  binary linear code is  $k \leq n - \log(\sum_{i=0}^t \binom{n}{i})$ , and we have Hamming code achieving this bound, therefore we have such a binary linear code  $\mathbf{C}$  with  $k \times n$  generator matrix  $\mathbf{G}$ . Next we will show how to construct syndrome encoding based on this linear code.

Let  $R_1 + R_2 = 2n - k$  and  $\mathbf{G}_1$  be formed by taking first  $(n - R_1)$  rows from  $\mathbf{G}$ , while  $\mathbf{G}_2$  is formed using the remaining  $(n - R_2)$  rows of  $\mathbf{G}$ .  $\mathbf{C}_1$  and  $\mathbf{C}_2$  are the subcodes of the Hamming code generated by  $\mathbf{G}_1$  and  $\mathbf{G}_2$  respectively, with  $\mathbf{H}_1$  and  $\mathbf{H}_2$  as their parity check matrices.

For a pair of input  $(\mathbf{x}, \mathbf{y})$ , the encoder is given by  $\mathbf{s}_1 = \mathbf{H}_1 \mathbf{x}$  and  $\mathbf{s}_2 = \mathbf{H}_2 \mathbf{y}$ . That means, we can represent  $\mathbf{x}$  and  $\mathbf{y}$  as  $\mathbf{x} = \mathbf{u}_1 \mathbf{G}_1 + \mathbf{c}_{s1}$ ,  $\mathbf{y} = \mathbf{u}_2 \mathbf{G}_2 + \mathbf{c}_{s2}$ , where  $\mathbf{c}_{s1}, \mathbf{c}_{s2}$  are the representatives of the cosets of  $\mathbf{s}_1, \mathbf{s}_2$  with regard to  $\mathbf{C}_1, \mathbf{C}_2$  respectively. Since we have  $\mathbf{y} = \mathbf{x} + \mathbf{e}$  with  $w(\mathbf{e}) \leq t$ . We can get  $\mathbf{x} + \mathbf{y} = \mathbf{u} \mathbf{G} + \mathbf{c}_s = \mathbf{e}$ , where  $\mathbf{u} = [\mathbf{u}_1, \mathbf{u}_2]$ ,  $\mathbf{c}_s = \mathbf{c}_{s1} + \mathbf{c}_{s2}$ .

Suppose there are two different input pairs with the same syndromes, that means there are two different strings  $\mathbf{u}^1, \mathbf{u}^2 \in \{0, 1\}^k$ , such that  $\mathbf{u}^1 \mathbf{G} + \mathbf{c}_s = \mathbf{e}$  and  $\mathbf{u}^2 \mathbf{G} + \mathbf{c}_s = \mathbf{e}$ . Thus we will have  $(\mathbf{u}^1 - \mathbf{u}^2) \mathbf{G} = \mathbf{0}$ . Because minimum Hamming weight of the code  $\mathbf{C}$  is  $2t + 1$ , the distance between  $\mathbf{u}_1 \mathbf{G}$  and  $\mathbf{u}_2 \mathbf{G}$  is  $\geq 2t + 1$ . On the other hand, according to  $w(\mathbf{e}) \leq t$  together with  $\mathbf{u}^1 \mathbf{G} + \mathbf{c}_s = \mathbf{e}$  and  $\mathbf{u}^2 \mathbf{G} + \mathbf{c}_s = \mathbf{e}$ , we will have  $d_H(\mathbf{u}^1 \mathbf{G}, \mathbf{c}_s) \leq t$  and  $d_H(\mathbf{u}^2 \mathbf{G}, \mathbf{c}_s) \leq t$ , which contradict with  $d_H(\mathbf{u}^1 \mathbf{G}, \mathbf{u}^2 \mathbf{G}) \geq 2t + 1$ . Therefore, we cannot have more than one input pairs with the same syndromes.

Therefore, we can successfully compress the two dependent sources with constructed subcodes from an  $(n, k, 2t + 1)$  binary linear code, with rate pair  $(R_1, R_2)$  such that  $R_1, R_2 \geq n - k$ ,  $R_1 + R_2 \geq 2n - k$ , where  $R_1$  and  $R_2$  are integers, and  $k \leq n - \log(\sum_{i=0}^t \binom{n}{i})$ .  $\log$  indicates  $\log_2$ .

#### Slepian–Wolf coding example

Take the same example as in the previous Asymmetric DSC vs. Symmetric DSC part, this part presents the corresponding DSC schemes with coset codes and syndromes including asymmetric case and symmetric case. The Slepian–Wolf bound for DSC design is shown in the previous part.

**Asymmetric case** ( $R_X = 3, R_Y = 7$ ) In this case, the length of an input variable  $\mathbf{y}$  from source  $Y$  is 7 bits, therefore it can be sent lossless with 7 bits independent of any other bits. Based on the knowledge that  $\mathbf{x}$  and  $\mathbf{y}$  have Hamming distance at most one, for input  $\mathbf{x}$  from source  $X$ , since the receiver already has  $\mathbf{y}$ , the only possible  $\mathbf{x}$  are those with at most 1 distance from  $\mathbf{y}$ . If we model the correlation between two sources as a virtual channel,



which has input  $\mathbf{x}$  and output  $\mathbf{y}$ , as long as we get  $\mathbf{y}$ , all we need to successfully “decode”  $\mathbf{x}$  is “parity bits” with particular error correction ability, taking the difference between  $\mathbf{x}$  and  $\mathbf{y}$  as channel error. We can also model the problem with cosets partition. That is, we want to find a channel code, which is able to partition the space of input  $X$  into several cosets, where each coset has a unique syndrome associated with it. With a given coset and  $\mathbf{y}$ , there is only one  $\mathbf{x}$  that is possible to be the input given the correlation between two sources.

In this example, we can use the (7, 4, 3) binary **Hamming Code C**, with parity check matrix  $\mathbf{H}$ . For an input  $\mathbf{x}$  from source  $X$ , only the syndrome given by  $\mathbf{s} = \mathbf{H}\mathbf{x}$  is transmitted, which is 3 bits. With received  $\mathbf{y}$  and  $\mathbf{s}$ , suppose there are two inputs  $\mathbf{x}_1$  and  $\mathbf{x}_2$  with same syndrome  $\mathbf{s}$ . That means  $\mathbf{H}\mathbf{x}_1 = \mathbf{H}\mathbf{x}_2$ , which is  $\mathbf{H}(\mathbf{x}_1 - \mathbf{x}_2) = \mathbf{0}$ . Since the minimum Hamming weight of (7, 4, 3) Hamming Code is 3,  $d_H(\mathbf{x}_1, \mathbf{x}_2) \geq 3$ . Therefore the input  $\mathbf{x}$  can be recovered since  $d_H(\mathbf{x}, \mathbf{y}) \leq 1$ .

Similarly, the bits distribution with  $R_X = 7$ ,  $R_Y = 3$  can be achieved by reversing the roles of  $X$  and  $Y$ .

**Symmetric case** In symmetric case, what we want is equal bitrate for the two sources: 5 bits each with separate encoder and joint decoder. We still use linear codes for this system, as we used for asymmetric case. The basic idea is similar, but in this case, we need to do coset partition for both sources, while for a pair of received syndromes (corresponds to one coset), only one pair of input variables are possible given the correlation between two sources.

Suppose we have a pair of **linear code C<sub>1</sub>** and **C<sub>2</sub>** and an encoder-decoder pair based on linear codes which can achieve symmetric coding. The encoder output is given by:  $\mathbf{s}_1 = \mathbf{H}_1\mathbf{x}$  and  $\mathbf{s}_2 = \mathbf{H}_2\mathbf{y}$ . If there exists two pair of valid inputs  $\mathbf{x}_1, \mathbf{y}_1$  and  $\mathbf{x}_2, \mathbf{y}_2$  generating the same syndromes, i.e.  $\mathbf{H}_1\mathbf{x}_1 = \mathbf{H}_1\mathbf{x}_2$  and  $\mathbf{H}_2\mathbf{y}_1 = \mathbf{H}_2\mathbf{y}_2$ , we can get following(  $w()$  represents Hamming weight):

$$\mathbf{y}_1 = \mathbf{x}_1 + \mathbf{e}_1, \text{ where } w(\mathbf{e}_1) \leq 1$$

$$\mathbf{y}_2 = \mathbf{x}_2 + \mathbf{e}_2, \text{ where } w(\mathbf{e}_2) \leq 1$$

$$\text{Thus: } \mathbf{x}_1 + \mathbf{x}_2 \in \mathbf{C}_1$$

$$\mathbf{y}_1 + \mathbf{y}_2 = \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{e}_3 \in \mathbf{C}_2$$

where  $\mathbf{e}_3 = \mathbf{e}_2 + \mathbf{e}_1$  and  $w(\mathbf{e}_3) \leq 2$ . That means, as long as we have the minimum distance between the two codes larger than 3, we can achieve error-free decoding.

The two codes  $\mathbf{C}_1$  and  $\mathbf{C}_2$  can be constructed as subcodes of the (7, 4, 3) Hamming code and thus has minimum distance of 3. Given the **generator matrix G** of the original Hamming code, the generator matrix  $\mathbf{G}_1$  for  $\mathbf{C}_1$  is constructed by taking any two rows from  $\mathbf{G}$ , and  $\mathbf{G}_2$  is constructed by the remaining two rows of  $\mathbf{G}$ . The corresponding (5 × 7) **parity-check matrix** for each sub-code can be generated according to the generator matrix and used to generate syndrome bits.

### 30.5.2 Wyner–Ziv coding – lossy distributed coding

In general, a Wyner–Ziv coding scheme is obtained by adding a quantizer and a de-quantizer to the Slepian–Wolf coding scheme. Therefore, a Wyner–Ziv coder design could focus on the quantizer and corresponding reconstruction method design. Several quantizer designs have been proposed, such as a nested lattice quantizer,<sup>[21]</sup> trellis code quantizer<sup>[22]</sup> and Lloyd quantization method.<sup>[23]</sup>

### 30.5.3 Large scale distributed quantization

Unfortunately, the above approaches do not scale (in design or operational complexity requirements) to sensor networks of large sizes, a scenario where distributed compression is most helpful. If there are  $N$  sources transmitting at  $R$  bits each (with some distributed coding scheme), the number of possible reconstructions scales  $2^{NR}$ . Even for moderate values of  $N$  and  $R$  (say  $N=10$ ,  $R=2$ ), prior design schemes become impractical. Recently, an approach,<sup>[24]</sup> using ideas borrowed from Fusion Coding of Correlated Sources, has been proposed where design and operational complexity are traded against decoder performance. This has allowed distributed quantizer design for network sizes reaching 60 sources, with substantial gains over traditional approaches.

The central idea is the presence of a bit-subset selector which maintains a certain subset of the received ( $NR$  bits, in the above example) bits for each source. Let  $\mathcal{B}$  be the set of all subsets of the  $NR$  bits i.e.

$$\mathcal{B} = 2^{\{1, \dots, NR\}}$$

Then, we define the bit-subset selector mapping to be

$$\mathcal{S} : \{1, \dots, N\} \rightarrow \mathcal{B}$$

Note that each choice of the bit-subset selector imposes a storage requirement ( $C$ ) that is exponential in the cardinality of the set of chosen bits.

$$C = \sum_{n=1}^N 2^{|\mathcal{S}(n)|}$$

This allows a judicious choice of bits that minimize the distortion, given the constraints on decoder storage. Additional limitations on the set of allowable subsets are still needed. The effective cost function that needs to be minimized is a weighted sum of distortion and decoder

storage

$$J = D + \lambda C$$

The system design is performed by iteratively (and incrementally) optimizing the encoders, decoder and bit-subset selector till convergence.

### 30.6 Non-asymmetric DSC

### 30.7 Non-asymmetric DSC for more than two sources

The syndrome approach can still be used for more than two sources. Let us consider  $a$  binary sources of length- $n$   $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_a \in \{0, 1\}^n$ . Let  $\mathbf{H}_1, \mathbf{H}_2, \dots, \mathbf{H}_s$  be the corresponding coding matrices of sizes  $m_1 \times n, m_2 \times n, \dots, m_a \times n$ . Then the input binary sources are compressed into  $\mathbf{s}_1 = \mathbf{H}_1 \mathbf{x}_1, \mathbf{s}_2 = \mathbf{H}_2 \mathbf{x}_2, \dots, \mathbf{s}_a = \mathbf{H}_a \mathbf{x}_a$  of total  $m = m_1 + m_2 + \dots + m_a$  bits. Apparently, two source tuples cannot be recovered at the same time if they share the same syndrome. In other words, if all source tuples of interest have different syndromes, then one can recover them losslessly.

General theoretical result does not seem to exist. However, for a restricted kind of source so-called Hamming source<sup>[25]</sup> that only has at most one source different from the rest and at most one bit location not all identical, practical lossless DSC is shown to exist in some cases. For the case when there are more than two sources, the number of source tuple in a Hamming source is  $2^n(an+1)$ . Therefore, a packing bound that  $2^m \geq 2^n(an+1)$  obviously has to satisfy. When the packing bound is satisfied with equality, we may call such code to be perfect (an analogous of perfect code in error correcting code).<sup>[25]</sup>

A simplest set of  $a, n, m$  to satisfy the packing bound with equality is  $a = 3, n = 5, m = 9$ . However, it turns out that such syndrome code does not exist.<sup>[26]</sup> The simplest (perfect) syndrome code with more than two sources have  $n = 21$  and  $m = 27$ . Let

$$\mathbf{Q}_1 = \begin{pmatrix} 100000100001110110000 \\ 010000110000100000111 \\ 001000011000011101011 \\ 000100001100010011110 \\ 000010000110101101111 \\ 000001000011001001101 \\ 000101101111010001111 \\ 100010110111101111000 \\ 010001111011100000101 \\ 101000111101011100111 \\ 010100111110001011011 \\ 001010011111110101110 \end{pmatrix},$$

$$\mathbf{Q}_2 = \begin{pmatrix} 000101101111010001111 \\ 100010110111101111000 \\ 010001111011100000101 \\ 101000111101011100111 \\ 010100111110001011011 \\ 001010011111110101110 \end{pmatrix},$$

$$\mathbf{Q}_3 = \begin{pmatrix} 100101001110100111111 \\ 110010000111001111111 \\ 011001100011111101110 \\ 101100110001001111001 \\ 010110111000100110100 \\ 001011011100111100011 \end{pmatrix},$$

$\mathbf{G} = [\mathbf{0} | \mathbf{I}_9]$ , and  $\mathbf{G} = \begin{pmatrix} \mathbf{G}_1 \\ \mathbf{G}_2 \\ \mathbf{G}_3 \end{pmatrix}$  such that  $\mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3$  are any partition of  $\mathbf{G}$ .

$\mathbf{H}_1 = \begin{pmatrix} \mathbf{G}_1 \\ \mathbf{Q}_1 \end{pmatrix}, \mathbf{H}_2 = \begin{pmatrix} \mathbf{G}_2 \\ \mathbf{Q}_2 \end{pmatrix}, \mathbf{H}_3 = \begin{pmatrix} \mathbf{G}_3 \\ \mathbf{Q}_3 \end{pmatrix}$  can compress a Hamming source (i.e., sources that have no more than one bit different will all have different syndromes).<sup>[25]</sup> For example, for the symmetric case, a possible set of coding matrices are

$$\mathbf{H}_1 = \begin{pmatrix} 0000000000000000000100 \\ 0000000000000000000010 \\ 0000000000000000000001 \\ 100000100001110110000 \\ 010000110000100000111 \\ 001000011000011101011 \\ 000100001100010011110 \\ 000010000110101101111 \\ 000001000011001001101 \end{pmatrix},$$

$$\mathbf{H}_2 = \begin{pmatrix} 0000000000000000100000 \\ 0000000000000000010000 \\ 0000000000000000001000 \\ 000101101111010001111 \\ 100010110111101111000 \\ 010001111011100000101 \\ 101000111101011100111 \\ 010100111110001011011 \\ 001010011111110101110 \end{pmatrix},$$

$$\mathbf{H}_3 = \begin{pmatrix} 0000000000000100000000 \\ 0000000000000010000000 \\ 0000000000000001000000 \\ 100101001110100111111 \\ 110010000111001111111 \\ 011001100011111101110 \\ 101100110001001111001 \\ 010110111000100110100 \\ 001011011100111100011 \end{pmatrix}.$$

### 30.8 See also

- Linear code
- Syndrome decoding
- Low-density parity-check code
- Turbo Code

### 30.9 References

- [1] "Distributed source coding for sensor networks" by Z.



- Xiong, A.D. Liveris, and S. Cheng
- [2] “Distributed video coding in wireless sensor networks” by Puri, R. Majumdar, A. Ishwar, P. Ramchandran, K.
  - [3] “Noiseless coding of correlated information sources” by D. Slepian and J. Wolf
  - [4] “A proof of the data compression theorem of Slepian and Wolf for ergodic sources” by T. Cover
  - [5] “The rate-distortion function for source coding with side information at the decoder” by A. Wyner and J. Ziv
  - [6] “Recent results in Shannon theory” by A. D. Wyner
  - [7] “Distributed source coding using syndromes (DISCUS): design and construction” by S. S. Pradhan and K. Ramchandran
  - [8] “Distributed source coding: symmetric rates and applications to sensor networks” by S. S. Pradhan and K. Ramchandran
  - [9] “Distributed code constructions for the entire Slepian–Wolf rate region for arbitrarily correlated sources” by Schonberg, D. Ramchandran, K. Pradhan, S.S.
  - [10] “Generalized coset codes for distributed binning” by Pradhan, S.S. Ramchandran, K.
  - [11] “Nested linear/lattice codes for Wyner–Ziv encoding” by R. Zamir and S. Shamai
  - [12] “Distributed Video Coding” by B. Girod, etc.
  - [13] “On code design for the Slepian–Wolf problem and lossless multiterminal networks” by Stankovic, V. Liveris, A.D. Zixiang Xiong Georgiades, C.N.
  - [14] “A general and optimal framework to achieve the entire rate region for Slepian–Wolf coding” by P. Tan and J. Li
  - [15] “Distributed source coding using short to moderate length rate-compatible LDPC codes: the entire Slepian–Wolf rate region” by Sartipi, M. Fekri, F.
  - [16] “A distributed source coding framework for multiple sources” by Xiaomin Cao and Kuijper, M.
  - [17] “Distributed Source Coding via Linear Block Codes: A General Framework for Multiple Sources” by Xiaomin Cao and Kuijper, M.
  - [18] “Coset codes. I. Introduction and geometrical classification” by G. D. Forney
  - [19] “Design of trellis codes for source coding with side information at the decoder” by X. Wang and M. Orchard
  - [20] “Design of Slepian–Wolf codes by channel code partitioning” by V. Stankovic, A. D. Liveris, Z. Xiong and C. N. Georgiades
  - [21] “Nested quantization and Slepian–Wolf coding: a Wyner–Ziv coding paradigm for i.i.d. sources” by Z. Xiong, A. D. Liveris, S. Cheng and Z. Liu
  - [22] “Wyner–Ziv coding based on TCQ and LDPC codes” by Y. Yang, S. Cheng, Z. Xiong and W. Zhao
  - [23] “Design of optimal quantizers for distributed source coding” by D. Rebollo-Monedero, R. Zhang and B. Girod
  - [24] “Towards large scale distributed source coding” by S. Ramaswamy, K. Viswanatha, A. Saxena and K. Rose
  - [25] “Hamming Codes for Multiple Sources” by R. Ma and S. Cheng
  - [26] “The Non-existence of Length-5 Slepian–Wolf Codes of Three Sources” by S. Cheng and R. Ma Archived April 25, 2012, at the Wayback Machine.

# Chapter 31

## Dual code

For players of both rugby codes, see [List of dual-code rugby internationals](#).

In coding theory, the **dual code** of a linear code

$$C \subset \mathbb{F}_q^n$$

is the linear code defined by

$$C^\perp = \{x \in \mathbb{F}_q^n \mid \langle x, c \rangle = 0 \ \forall c \in C\}$$

where

$$\langle x, c \rangle = \sum_{i=1}^n x_i c_i$$

is a scalar product. In [linear algebra](#) terms, the dual code is the [annihilator](#) of  $C$  with respect to the [bilinear form](#)  $\langle, \rangle$ . The [dimension](#) of  $C$  and its dual always add up to the length  $n$ :

$$\dim C + \dim C^\perp = n.$$

A [generator matrix](#) for the dual code is a [parity-check matrix](#) for the original code and vice versa. The dual of the dual code is always the original code.

### 31.1 Self-dual codes

A **self-dual code** is one which is its own dual. This implies that  $n$  is even and  $\dim C = n/2$ . If a self-dual code is such that each codeword's weight is a multiple of some constant  $c > 1$ , then it is of one of the following four types:<sup>[1]</sup>

- **Type I** codes are binary self-dual codes which are not doubly even. Type I codes are always even (every codeword has even [Hamming weight](#)).
- **Type II** codes are binary self-dual codes which are doubly even.

- **Type III** codes are ternary self-dual codes. Every codeword in a Type III code has Hamming weight divisible by 3.

- **Type IV** codes are self-dual codes over  $\mathbf{F}_4$ . These are again even.

Codes of types I, II, III, or IV exist only if the length  $n$  is a multiple of 2, 8, 4, or 2 respectively.

If a self-dual code has a generator matrix of the form  $G = [I_k | A]$ , then the dual code  $C^\perp$  has [generator matrix](#)  $[-\bar{A}^T | I_k]$ , where  $I_k$  is the  $(n/2) \times (n/2)$  identity matrix and  $\bar{a} = a^q \in \mathbb{F}_q$ .

### 31.2 References

- [1] Conway, J.H.; Sloane, N.J.A. (1988). *Sphere packings, lattices and groups*. Grundlehren der mathematischen Wissenschaften. **290**. Springer-Verlag. p. 77. ISBN 0-387-96617-X.
- Hill, Raymond (1986). *A first course in coding theory*. Oxford Applied Mathematics and Computing Science Series. Oxford University Press. p. 67. ISBN 0-19-853803-0.
- Pless, Vera (1982). *Introduction to the theory of error-correcting codes*. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons. p. 8. ISBN 0-471-08684-3.
- J.H. van Lint (1992). *Introduction to Coding Theory*. GTM. **86** (2nd ed.). Springer-Verlag. p. 34. ISBN 3-540-54894-7.

### 31.3 External links

- [MATH32031: Coding Theory - Dual Code](#) - pdf with some examples and explanations

## Chapter 32

# Elias Bassalygo bound

The **Elias-Bassalygo bound** is a mathematical limit used in coding theory for error correction during data transmission or communications. The properties of the Elias-Bassalygo bound are defined, below, using mathematical expressions.

### 32.1 Definition

Let  $C$  be a  $q$ -ary code of length  $n$ , i.e. a subset of  $[q]^n$ .<sup>[1]</sup> Let  $R$  be the rate of  $C$ ,  $\delta$  the relative distance and

$$B_q(y, \rho n) = \{x \in [q]^n : \Delta(x, y) \leq \rho n\}$$

be the *Hamming ball* of radius  $\rho n$  centered at  $y$ . Let  $\text{Vol}_q(y, \rho n) = |B_q(y, \rho n)|$  be the *volume* of the Hamming ball of radius  $\rho n$ . It is obvious that the volume of a Hamming Ball is translation-invariant, i.e. indifferent to  $y$ . In particular,  $|B_q(y, \rho n)| = |B_q(0, \rho n)|$ .

With large enough  $n$ , the rate  $R$  and the relative distance  $\delta$  satisfy the **Elias-Bassalygo bound**:

$$R \leq 1 - H_q(J_q(\delta)) + o(1),$$

where

$$H_q(x) \equiv_{\text{def}} -x \log_q \left( \frac{x}{q-1} \right) - (1-x) \log_q (1-x)$$

is the  $q$ -ary entropy function and

$$J_q(\delta) \equiv_{\text{def}} \left(1 - \frac{1}{q}\right) \left(1 - \sqrt{1 - \frac{q\delta}{q-1}}\right)$$

is a function related with **Johnson bound**.

### 32.2 Proof

To prove the Elias-Bassalygo bound, start with the following Lemma:

**Lemma.** For  $C \subseteq [q]^n$  and  $0 \leq e \leq n$ , there exists a Hamming ball of radius  $e$  with at least

$$\frac{|C| \text{Vol}_q(0, e)}{q^n}$$

codewords in it.

**Proof of Lemma.** Randomly pick a received word  $y \in [q]^n$  and let  $B_q(y, 0)$  be the Hamming ball centered at  $y$  with radius  $e$ . Since  $y$  is (uniform) randomly selected the expected size of overlapped region  $|B_q(y, e) \cap C|$  is

$$\frac{|C| \text{Vol}_q(y, e)}{q^n}$$

Since this is the expected value of the size, there must exist at least one  $y$  such that

$$|B_q(y, e) \cap C| \geq \frac{|C| \text{Vol}_q(y, e)}{q^n} = \frac{|C| \text{Vol}_q(0, e)}{q^n},$$

otherwise the expectation must be smaller than this value.

Now we prove the Elias-Bassalygo bound. Define  $e = nJ_q(\delta) - 1$ . By Lemma, there exists a Hamming ball with  $B$  codewords such that:

$$B \geq \frac{|C| \text{Vol}_q(0, e)}{q^n}$$

By the **Johnson bound**, we have  $B \leq qdn$ . Thus,

$$|C| \leq qnd \cdot \frac{q^n}{\text{Vol}_q(0, e)} \leq q^{n(1-H_q(J_q(\delta))+o(1))}$$

The second inequality follows from lower bound on the volume of a Hamming ball:

$$\text{Vol}_q \left( 0, \left\lfloor \frac{d-1}{2} \right\rfloor \right) \leq q^{H_q(\frac{\delta}{2})n - o(n)}.$$

Putting in  $d = 2e + 1$  and  $\delta = \frac{d}{n}$  gives the second inequality.

Therefore we have

$$R = \frac{\log_q |C|}{n} \leq 1 - H_q(J_q(\delta)) + o(1)$$

### 32.3 See also

- Singleton bound
- Hamming bound
- Plotkin bound
- Gilbert–Varshamov bound
- Johnson bound

### 32.4 References

- [1] Each  $q$ -ary block code of length  $n$  is a subset of the strings of  $\mathcal{A}_q^n$ , where the alphabet set  $\mathcal{A}_q$  has  $q$  elements.

Bassalygo, L. A. (1965), “New upper boundes for error-correcting codes”, *Problems of Information Transmission*, **1** (1): 32–35

Claude E. Shannon, Robert G. Gallager; Berlekamp, Elwyn R. (1967), “Lower bounds to error probability for coding on discrete memoryless channels. Part I.”, *Information and Control*, **10**: 65–103, doi:10.1016/s0019-9958(67)90052-6

Claude E. Shannon, Robert G. Gallager; Berlekamp, Elwyn R. (1967), “Lower bounds to error probability for coding on discrete memoryless channels. Part II.”, *Information and Control*, **10**: 522–552, doi:10.1016/s0019-9958(67)91200-4

## Chapter 33

# Enumerator polynomial

In coding theory, the **weight enumerator polynomial** of a binary linear code specifies the number of words of each possible Hamming weight.

Let  $C \subset \mathbb{F}_2^n$  be a binary linear code length  $n$ . The **weight distribution** is the sequence of numbers

$$A_t = \#\{c \in C \mid w(c) = t\}$$

giving the number of **codewords**  $c$  in  $C$  having weight  $t$  as  $t$  ranges from 0 to  $n$ . The **weight enumerator** is the bivariate **polynomial**

$$W(C; x, y) = \sum_{w=0}^n A_w x^w y^{n-w}.$$

### 33.1 Basic properties

1.  $W(C; 0, 1) = A_0 = 1$
2.  $W(C; 1, 1) = \sum_{w=0}^n A_w = |C|$
3.  $W(C; 1, 0) = A_n = 1$  if  $(1, \dots, 1) \in C$  and 0 otherwise
4.  $W(C; 1, -1) = \sum_{w=0}^n A_w (-1)^{n-w} = A_n + (-1)^1 A_{n-1} + \dots + (-1)^{n-1} A_1 + (-1)^n A_0$

### 33.2 MacWilliams identity

Denote the **dual code** of  $C \subset \mathbb{F}_2^n$  by

$$C^\perp = \{x \in \mathbb{F}_2^n \mid \langle x, c \rangle = 0 \forall c \in C\}$$

(where  $\langle, \rangle$  denotes the vector **dot product** and which is taken over  $\mathbb{F}_2$ ).

The **MacWilliams identity** states that

$$W(C^\perp; x, y) = \frac{1}{|C|} W(C; y - x, y + x).$$

The identity is named after **Jessie MacWilliams**.

### 33.3 Distance enumerator

The **distance distribution** or **inner distribution** of a code  $C$  of size  $M$  and length  $n$  is the sequence of numbers

$$A_i = \frac{1}{M} \#\{(c_1, c_2) \in C \times C \mid d(c_1, c_2) = i\}$$

where  $i$  ranges from 0 to  $n$ . The **distance enumerator polynomial** is

$$A(C; x, y) = \sum_{i=0}^n A_i x^i y^{n-i}$$

and when  $C$  is linear this is equal to the weight enumerator.

The **outer distribution** of  $C$  is the  $2^n$ -by- $n+1$  matrix  $B$  with rows indexed by elements of  $\text{GF}(2)^n$  and columns indexed by integers  $0 \dots n$ , and entries

$$B_{x,i} = \#\{c \in C \mid d(c, x) = i\}.$$

The sum of the rows of  $B$  is  $M$  times the inner distribution vector  $(A_0, \dots, A_n)$ .

A code  $C$  is **regular** if the rows of  $B$  corresponding to the codewords of  $C$  are all equal.

### 33.4 References

- Hill, Raymond (1986). *A first course in coding theory*. Oxford Applied Mathematics and Computing Science Series. Oxford University Press. pp. 165–173. ISBN 0-19-853803-0.
- Pless, Vera (1982). *Introduction to the theory of error-correcting codes*. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons. pp. 103–119. ISBN 0-471-08684-3.

- J.H. van Lint (1992). *Introduction to Coding Theory*. GTM. **86** (2nd ed.). Springer-Verlag. ISBN 3-540-54894-7. Chapters 3.5 and 4.3.

# Chapter 34

## Erasur code

In information theory, an **erasure code** is a forward error correction (FEC) code for the binary erasure channel, which transforms a message of  $k$  symbols into a longer message (code word) with  $n$  symbols such that the original message can be recovered from a subset of the  $n$  symbols. The fraction  $r = k/n$  is called the **code rate**. The fraction  $k'/k$ , where  $k'$  denotes the number of symbols required for recovery, is called **reception efficiency**.

### 34.1 Optimal erasure codes

Optimal erasure codes have the property that any  $k$  out of the  $n$  code word symbols are sufficient to recover the original message (i.e., they have optimal reception efficiency). Optimal erasure codes are **maximum distance separable codes** (MDS codes).

Optimal codes are often costly (in terms of memory usage, CPU time, or both) when  $n$  is large. Except for very simple schemes, practical solutions usually have quadratic encoding and decoding **complexity**. In 2014, Lin et al. gave an approach with  $O(n \log n)$  operations.<sup>[1]</sup>

#### 34.1.1 Parity check

See also: **Parity bit**

Parity check is the special case where  $n = k + 1$ . From a set of  $k$  values  $\{v_i\}_{1 \leq i \leq k}$ , a checksum is computed and appended to the  $k$  source values:

$$v_{k+1} = - \sum_{i=1}^k v_i.$$

The set of  $k + 1$  values  $\{v_i\}_{1 \leq i \leq k+1}$  is now consistent with regard to the checksum. If one of these values,  $v_e$ , is erased, it can be easily recovered by summing the remaining variables:

$$v_e = - \sum_{i=1, i \neq e}^{k+1} v_i.$$

#### 34.1.2 Polynomial oversampling

##### Example: Err-mail ( $k = 2$ )

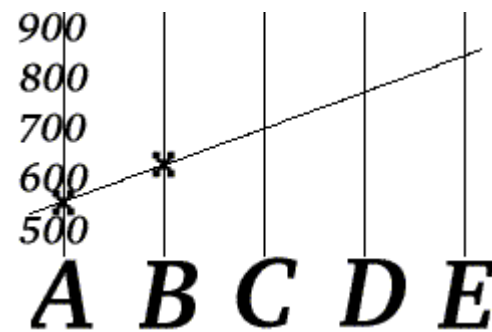
In the simple case where  $k = 2$ , redundancy symbols may be created by sampling different points along the line between the two original symbols. This is pictured with a simple example, called err-mail:

Alice wants to send her telephone number (555629) to Bob using err-mail. Err-mail works just like e-mail, except

1. About half of all the mail gets lost.<sup>[2]</sup>
2. Messages longer than 5 characters are illegal.
3. It is very expensive (similar to air-mail).

Instead of asking Bob to acknowledge the messages she sends, Alice devises the following scheme.

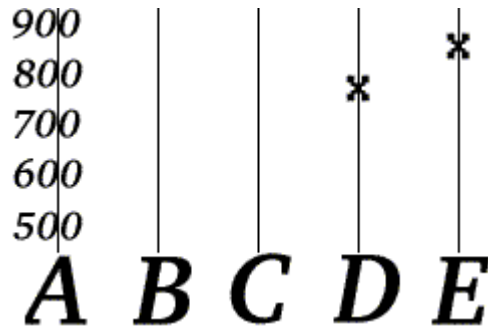
1. She breaks her telephone number up into two parts  $a = 555, b = 629$ , and sends 2 messages – “A=555” and “B=629” – to Bob.
2. She constructs a linear function,  $f(i) = a + (b - a)(i - 1)$ , in this case  $f(i) = 555 + 74(i - 1)$ , such that  $f(1) = 555$  and  $f(2) = 629$ .



1. She computes the values  $f(3)$ ,  $f(4)$ , and  $f(5)$ , and then transmits three redundant messages: “C=703”, “D=777” and “E=851”.



Bob knows that the form of  $f(k)$  is  $f(i) = a + (b - a)(i - 1)$ , where  $a$  and  $b$  are the two parts of the telephone number. Now suppose Bob receives “D=777” and “E=851”.



Bob can reconstruct Alice's phone number by computing the values of  $a$  and  $b$  from the values ( $f(4)$  and  $f(5)$ ) he has received. Bob can perform this procedure using any two err-mails, so the erasure code in this example has a rate of 40%.

Note that Alice cannot encode her telephone number in just one err-mail, because it contains six characters, and that the maximum length of one err-mail message is five characters. If she sent her phone number in pieces, asking Bob to acknowledge receipt of each piece, at least four messages would have to be sent anyway (two from Alice, and two acknowledgments from Bob). So the erasure code in this example, which requires five messages, is quite economical.

This example is a little bit contrived. For truly generic erasure codes that work over any data set, we would need something other than the  $f(i)$  given.

### General case

The linear construction above can be generalized to **polynomial interpolation**. Additionally, points are now computed over a **finite field**.

First we choose a finite field  $F$  with order of at least  $n$ , but usually a power of 2. The sender numbers the data symbols from 0 to  $k - 1$  and sends them. He then constructs a (**Lagrange**) **polynomial**  $p(x)$  of order  $k$  such that  $p(i)$  is equal to data symbol  $i$ . He then sends  $p(k)$ , ...,  $p(n - 1)$ . The receiver can now also use polynomial interpolation to recover the lost packets, provided he receives  $k$  symbols successfully. If the order of  $F$  is less than  $2^b$ , where  $b$  is the number of bits in a symbol, then multiple polynomials can be used.

The sender can construct symbols  $k$  to  $n - 1$  'on the fly', i.e., distribute the workload evenly between transmission of the symbols. If the receiver wants to do his calculations 'on the fly', he can construct a new polynomial  $q$ , such that  $q(i) = p(i)$  if symbol  $i < k$  was received successfully and  $q(i) = 0$  when symbol  $i < k$  was not received. Now let  $r(i) = p(i) - q(i)$ . Firstly we know that  $r(i) = 0$  if symbol  $i < k$  has been received successfully. Secondly, if symbol  $i \geq k$  has

been received successfully, then  $r(i) = p(i) - q(i)$  can be calculated. So we have enough data points to construct  $r$  and evaluate it to find the lost packets. So both the sender and the receiver require  $O(n(n - k))$  operations and only  $O(n - k)$  space for operating 'on the fly'.

### Real world implementation

This process is implemented by **Reed-Solomon codes**, with code words constructed over a **finite field** using a **Vandermonde matrix**.

## 34.2 Near-optimal erasure codes

**Near-optimal erasure codes** require  $(1 + \epsilon)k$  symbols to recover the message (where  $\epsilon > 0$ ). Reducing  $\epsilon$  can be done at the cost of CPU time. **Near-optimal erasure codes** trade correction capabilities for computational complexity: practical algorithms can encode and decode with linear time complexity.

**Fountain codes** (also known as **rateless erasure codes**) are notable examples of **near-optimal erasure codes**. They can transform a  $k$  symbol message into a practically infinite encoded form, i.e., they can generate an arbitrary amount of redundancy symbols that can all be used for error correction. Receivers can start decoding after they have received slightly more than  $k$  encoded symbols.

**Regenerating codes** address the issue of rebuilding (also called repairing) lost encoded fragments from existing encoded fragments. This issue arises in distributed storage systems where communication to maintain encoded redundancy is a problem.

## 34.3 Examples

Here are some examples of implementations of the various codes:

### 34.3.1 Near optimal erasure codes

- **Tornado codes**
- **Low-density parity-check codes**

### 34.3.2 Near optimal fountain (rateless erasure) codes

- **Fountain code**
- **Online codes**
- **LT codes**
- **Raptor codes**

### 34.3.3 Optimal erasure codes

- Parity: used in RAID storage systems.
- [Parchive](#)
- Tahoe-LAFS includes [zfec](#)
- Reed–Solomon codes
- Erasure Resilient Systematic Code, an MDS code outperforming Reed–Solomon in the maximal number of redundant packets, see RS(4,2) with 2 bits or RS(9,2) with 3 bits
- Regenerating Codes<sup>[3]</sup> see also [Storage Wiki](#).
- any other MDS code (a type of “Maximum distance separable code”)

### 34.3.4 Other

- [Spelling alphabet](#)

## 34.4 See also

- Forward error correction codes.
- Secret sharing (differs in that the original secret is encrypted and obscured until the decode quorum is reached)

## 34.5 References

- [1] Sian-Jheng Lin, Wei-Ho Chung, and Yunghsiang S. Han, “Novel polynomial basis and its application to Reed–Solomon erasure codes”, The 55th Annual Symposium on Foundations of Computer Science (FOCS 2014). [arXiv:1404.3458](#)
- [2] Some versions of this story refer to the err-mail daemon.
- [3] Dimakis, Alexandros G.; Godfrey, P. Brighten; Wu, Yunnan; Wainwright, Martin J.; Ramchandran, Kannan (September 2010). “Network Coding for Distributed Storage Systems”. *IEEE Transactions on Information Theory*. **56** (9): 4539–4551. doi:10.1109/TIT.2010.2054295.

## 34.6 External links

- [Jerasure](#) is a Free Software library implementing Reed–Solomon and Cauchy erasure code techniques with SIMD optimisations.
- [Software FEC in computer communications](#) by Luigi Rizzo describes optimal erasure correction codes

- [Feclib](#) is a near optimal extension to Luigi Rizzo’s work that uses band matrices. Many parameters can be set, like the size of the width of the band and size of the finite field. It also successfully exploits the large register size of modern CPUs. How it compares to the near optimal codes mentioned above is unknown.
- [Coding for Distributed Storage wiki](#) for regenerating codes and rebuilding erasure codes.
- ECIP “Erasure Code Internet Protocol” Developed in 1996, was the first use of FEC “Forward Error correction” on the Internet. It was first used commercially to stream live video of Sir Arthur C. Clarke in Sri Lanka to UIUC in Indiana.

## Chapter 35

# Even code

A **binary code** is called an **even code** if the **Hamming weight** of each of its codewords is even. An even code should have a generator polynomial that include  $(1+x)$  minimal polynomial as a product. Furthermore, a binary code is called **doubly even** if the Hamming weight of all its codewords is **divisible by 4**. An even code which is not doubly even is said to be strictly even.

Examples of doubly even codes are the extended binary Hamming code of block length 8 and the extended binary Golay code of block length 24. These two codes are, in addition, self-dual.

*This article incorporates material from [even code](#) on PlanetMath, which is licensed under the [Creative Commons Attribution/Share-Alike License](#).*

## Chapter 36

# Expander code

In coding theory, **expander codes** form a class of error-correcting codes that are constructed from bipartite expander graphs. Along with Justesen codes, expander codes are of particular interest since they have a constant positive rate, a constant positive relative distance, and a constant alphabet size. In fact, the alphabet contains only two elements, so expander codes belong to the class of binary codes. Furthermore, expander codes can be both encoded and decoded in time proportional to the block length of the code. Expander codes are the only known asymptotically good codes which can be both encoded and decoded from a constant fraction of errors in polynomial time.

### 36.1 Expander codes

In coding theory, an expander code is a  $[n, n-m]_2$  linear block code whose parity check matrix is the adjacency matrix of a bipartite expander graph. These codes have good relative distance  $2(1-\varepsilon)\gamma$ , where  $\varepsilon$  and  $\gamma$  are properties of the expander graph as defined later), rate  $(1-\frac{m}{n})$ , and decodability (algorithms of running time  $O(n)$  exist).

### 36.2 Definition

Consider a bipartite graph  $G(L, R, E)$ , where  $L$  and  $R$  are the vertex sets and  $E$  is the set of edges connecting vertices in  $L$  to vertices of  $R$ . Suppose every vertex in  $L$  has degree  $d$  (the graph is  $d$ -regular),  $|L| = n$ , and  $|R| = m$ ,  $m < n$ . Then  $G$  is a  $(N, M, d, \gamma, \alpha)$  expander graph if every small enough subset  $S \subset L$ ,  $|S| \leq \gamma n$  has the property that  $S$  has at least  $d\alpha|S|$  distinct neighbors in  $R$ . Note that this holds trivially for  $\gamma \leq \frac{1}{n}$ . When  $\frac{1}{n} < \gamma \leq 1$  and  $\alpha = 1-\varepsilon$  for a constant  $\varepsilon$ , we say that  $G$  is a lossless expander.

Since  $G$  is a bipartite graph, we may consider its  $n \times m$  adjacency matrix. Then the linear code  $C$  generated by viewing the transpose of this matrix as a parity check matrix is an expander code.

It has been shown that nontrivial lossless expander graphs

exist. Moreover, we can explicitly construct them.<sup>[1]</sup>

### 36.3 Rate

The rate of  $C$  is its dimension divided by its block length. In this case, the parity check matrix has size  $m \times n$ , and hence  $C$  has dimension at least  $(n-m)/n = 1 - \frac{m}{n}$ .

### 36.4 Distance

Suppose  $\varepsilon < \frac{1}{2}$ . Then the distance of a  $(n, m, d, \gamma, 1-\varepsilon)$  expander code  $C$  is at least  $2(1-\varepsilon)\gamma n$ .

#### 36.4.1 Proof

Note that we can consider every codeword  $c$  in  $C$  as a subset of vertices  $S \subset L$ , by saying that vertex  $v_i \in S$  if and only if the  $i$ th index of the codeword is a 1. Then  $c$  is a codeword iff every vertex  $v \in R$  is adjacent to an even number of vertices in  $S$ . (In order to be a codeword,  $cP = 0$ , where  $P$  is the parity check matrix. Then, each vertex in  $R$  corresponds to each column of  $P$ . Matrix multiplication over  $\text{GF}(2) = \{0, 1\}$  then gives the desired result.) So, if a vertex  $v \in R$  is adjacent to a single vertex in  $S$ , we know immediately that  $c$  is not a codeword. Let  $N(S)$  denote the neighbors in  $R$  of  $S$ , and  $U(S)$  denote those neighbors of  $S$  which are unique, i.e., adjacent to a single vertex of  $S$ .

#### Lemma 1

For every  $S \subset L$  of size  $|S| \leq \gamma n$ ,  $d|S| \geq |N(S)| \geq |U(S)| \geq d(1-2\varepsilon)|S|$ .

#### Proof

Trivially,  $|N(S)| \geq |U(S)|$ , since  $v \in U(S)$  implies  $v \in N(S)$ .  $|N(S)| \leq d|S|$  follows since the degree of every vertex in  $S$  is  $d$ . By the expansion property of the graph, there must be a set of  $d(1-\varepsilon)|S|$  edges

which go to distinct vertices. The remaining  $d\varepsilon|S|$  edges make at most  $d\varepsilon|S|$  neighbors not unique, so  $|U(S)| \geq d(1 - \varepsilon)|S| - d\varepsilon|S| = d(1 - 2\varepsilon)|S|$ .

### Corollary

Every sufficiently small  $S$  has a unique neighbor. This follows since  $\varepsilon < \frac{1}{2}$ .

### Lemma 2

Every subset  $T \subset L$  with  $|T| < 2(1 - \varepsilon)\gamma n$  has a unique neighbor.

### Proof

Lemma 1 proves the case  $|T| \leq \gamma n$ , so suppose  $2(1 - \varepsilon)\gamma n > |T| > \gamma n$ . Let  $S \subset T$  such that  $|S| = \gamma n$ . By Lemma 1, we know that  $|U(S)| \geq d(1 - 2\varepsilon)|S|$ . Then a vertex  $v \in U(S)$  is in  $U(T)$  iff  $v \notin N(T \setminus S)$ , and we know that  $|T \setminus S| \leq 2(1 - \varepsilon)\gamma n - \gamma n = (1 - 2\varepsilon)\gamma n$ , so by the first part of Lemma 1, we know  $|N(T \setminus S)| \leq d(1 - 2\varepsilon)\gamma n$ . Since  $\varepsilon < \frac{1}{2}$ ,  $|U(T)| \geq |U(S) \setminus N(T \setminus S)| \geq |U(S)| - |N(T \setminus S)| > 0$ , and hence  $U(T)$  is not empty.

### Corollary

Note that if a  $T \subset L$  has at least 1 unique neighbor, i.e.  $|U(T)| > 0$ , then the corresponding word  $c$  corresponding to  $T$  cannot be a codeword, as it will not multiply to the all zeros vector by the parity check matrix. By the previous argument,  $c \in C \implies wt(c) \geq 2(1 - \varepsilon)\gamma n$ . Since  $C$  is linear, we conclude that  $C$  has distance at least  $2(1 - \varepsilon)\gamma n$ .

## 36.5 Encoding

The encoding time for an expander code is upper bounded by that of a general linear code -  $O(n^2)$  by matrix multiplication. A result due to Spielman shows that encoding is possible in  $O(n)$  time.<sup>[2]</sup>

## 36.6 Decoding

Decoding of expander codes is possible in  $O(n)$  time when  $\varepsilon < \frac{1}{4}$  using the following algorithm.

Let  $v_i$  be the vertex of  $L$  that corresponds to the  $i$ th index in the codewords of  $C$ . Let  $y \in \{0, 1\}^n$  be a received word, and  $V(y) = \{v_i \mid \text{the } i^{\text{th}} \text{ of position } y \text{ is } 1\}$ . Let  $e(i)$  be  $|\{v \in$

$R \mid N(v) \cap V(y) \text{ is even}\}|$ , and  $o(i)$  be  $|\{v \in R \mid N(v) \cap V(y) \text{ is odd}\}|$ . Then consider the greedy algorithm:

**Input:** received codeword  $y$ .

initialize  $y'$  to  $y$  while there is a  $v$  in  $R$  adjacent to an odd number of vertices in  $V(y')$  if there is an  $i$  such that  $o(i) > e(i)$  flip entry  $i$  in  $y'$  else fail

**Output:** fail, or modified codeword  $y'$ .

### 36.6.1 Proof

We show first the correctness of the algorithm, and then examine its running time.

#### Correctness

We must show that the algorithm terminates with the correct codeword when the received codeword is within half the code's distance of the original codeword. Let the set of corrupt variables be  $S$ ,  $s = |S|$ , and the set of unsatisfied (adjacent to an odd number of vertices) vertices in  $R$  be  $c$ . The following lemma will prove useful.

**Lemma 3** If  $0 < s < \gamma n$ , then there is a  $v_i$  with  $o(i) > e(i)$ .

**Proof** By Lemma 1, we know that  $|U(S)| \geq d(1 - 2\varepsilon)s$ . So an average vertex has at least  $d(1 - 2\varepsilon) > d/2$  unique neighbors (recall unique neighbors are unsatisfied and hence contribute to  $o(i)$ ), since  $\varepsilon < \frac{1}{4}$ , and thus there is a vertex  $v_i$  with  $o(i) > e(i)$ .

So, if we have not yet reached a codeword, then there will always be some vertex to flip. Next, we show that the number of errors can never increase beyond  $\gamma n$ .

**Lemma 4** If we start with  $s < \gamma(1 - 2\varepsilon)n$ , then we never reach  $s = \gamma n$  at any point in the algorithm.

**Proof** When we flip a vertex  $v_i$ ,  $o(i)$  and  $e(i)$  are interchanged, and since we had  $o(i) > e(i)$ , this means the number of unsatisfied vertices on the right decreases by at least one after each flip. Since  $s < \gamma(1 - 2\varepsilon)n$ , the initial number of unsatisfied vertices is at most  $d\gamma(1 - 2\varepsilon)n$ , by the graph's  $d$ -regularity. If we reached a string with  $\gamma n$  errors, then by Lemma 1, there would be at least  $d\gamma(1 - 2\varepsilon)n$  unique neighbors, which means there would be at least  $d\gamma(1 - 2\varepsilon)n$  unsatisfied vertices, a contradiction.

Lemmas 3 and 4 show us that if we start with  $s < \gamma(1 - 2\varepsilon)n$  (half the distance of  $C$ ), then we will always find a vertex  $v_i$  to flip. Each flip reduces the number of unsatisfied vertices in  $R$  by at least 1, and hence

the algorithm terminates in at most  $m$  steps, and it terminates at some codeword, by Lemma 3. (Were it not at a codeword, there would be some vertex to flip). Lemma 4 shows us that we can never be farther than  $\gamma n$  away from the correct codeword. Since the code has distance  $2(1-\varepsilon)\gamma n > \gamma n$  (since  $\varepsilon < \frac{1}{2}$ ), the codeword it terminates on must be the correct codeword, since the number of bit flips is less than half the distance (so we couldn't have traveled far enough to reach any other codeword).

### Complexity

We now show that the algorithm can achieve linear time decoding. Let  $\frac{n}{m}$  be constant, and  $r$  be the maximum degree of any vertex in  $R$ . Note that  $r$  is also constant for known constructions.

1. Pre-processing: It takes  $O(mr)$  time to compute whether each vertex in  $R$  has an odd or even number of neighbors.
2. Pre-processing 2: We take  $O(dn) = O(dmr)$  time to compute a list of vertices  $v_i$  in  $L$  which have  $o(i) > e(i)$ .
3. Each Iteration: We simply remove the first list element. To update the list of odd / even vertices in  $R$ , we need only update  $O(d)$  entries, inserting / removing as necessary. We then update  $O(dr)$  entries in the list of vertices in  $L$  with more odd than even neighbors, inserting / removing as necessary. Thus each iteration takes  $O(dr)$  time.
4. As argued above, the total number of iterations is at most  $m$ .

This gives a total runtime of  $O(mdr) = O(n)$  time, where  $d$  and  $r$  are constants.

## 36.7 See also

- Expander graph
- Low-density parity-check code
- Linear time encoding and decoding of error-correcting codes
- ABNNR and AEL codes

## 36.8 Notes

This article is based on Dr. Venkatesan Guruswami's course notes.<sup>[3]</sup>

## 36.9 References

- [1] Capalbo, M.; Reingold, O.; Vadhan, S.; Wigderson, A. (2002). "Randomness conductors and constant-degree lossless expanders". *STOC '02 Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*. ACM. pp. 659–668. doi:10.1145/509907.510003. ISBN 1-58113-495-9.
- [2] Spielman, D. (1996). "Linear-time encodable and decodable error-correcting codes". *IEEE Transactions on Information Theory*. **42** (6): 1723–31. doi:10.1109/18.556668.
- [3] Guruswami, V. (15 November 2006). "Lecture 13: Expander Codes" (PDF). *CSE 533: Error-Correcting*. University of Washington.  
Guruswami, V. (March 2010). "Notes 8: Expander Codes and their decoding" (PDF). *Introduction to Coding Theory*. Carnegie Mellon University.  
Guruswami, V. (September 2004). "Guest column: error-correcting codes and expander graphs". *ACM SIGACT News*. **35** (3): 25–41. doi:10.1145/1027914.1027924.

## Chapter 37

# Factorization of polynomials over finite fields

In mathematics and computer algebra the factorization of a polynomial consists of decomposing it into a product of irreducible factors. This decomposition is theoretically possible and is unique for polynomials with coefficients in any field, but rather strong restrictions on the field of the coefficients are needed to allow the computation of the factorization by means of an algorithm. In practice, algorithms have been designed only for polynomials with coefficients in a finite field, in the field of rationals or in a finitely generated field extension of one of them.

The case of the **factorization of univariate polynomials over a finite field**, which is the subject of this article, is especially important, because all the algorithms (including the case of multivariate polynomials over the rational numbers), which are sufficiently efficient to be implemented, reduce the problem to this case (see **Polynomial factorization**). It is also interesting for various applications of finite fields, such as **coding theory** (cyclic redundancy codes and **BCH codes**), **cryptography** (public key cryptography by the means of **elliptic curves**), and **computational number theory**.

As the reduction of the factorization of **multivariate polynomials** to that of univariate polynomials does not have any specificity in the case of coefficients in a finite field, only polynomials with one variable are considered in this article.

## 37.1 Background

### 37.1.1 Finite field

Main article: **Finite field**

The theory of finite fields, whose origins can be traced back to the works of **Gauss** and **Galois**, has played a part in various branches of mathematics. Due to the applicability of the concept in other topics of mathematics and sciences like computer science there has been a resurgence of interest in finite fields and this is partly due to important applications in **coding theory**

and **cryptography**. Applications of finite fields introduce some of these developments in **cryptography**, **computer algebra** and **coding theory**.

A finite field or **Galois field** is a field with a finite order (number of elements). The order of a finite field is always a prime or a power of prime. For each prime power  $q = p^r$ , there exists exactly one finite field with  $q$  elements, up to isomorphism. This field is denoted  $GF(q)$  or  $\mathbf{F}_q$ . If  $p$  is prime,  $GF(p)$  is the prime field of order  $p$ ; it is the field of residue classes modulo  $p$ , and its  $p$  elements are denoted  $0, 1, \dots, p-1$ . Thus  $a = b$  in  $GF(p)$  means the same as  $a \equiv b \pmod{p}$ .

### 37.1.2 Irreducible polynomials

Let  $F$  be a finite field. As for general fields, a non-constant polynomial  $f$  in  $F[x]$  is said to be **irreducible** over  $F$  if it is not the product of two polynomials of positive degree. A polynomial of positive degree that is not irreducible over  $F$  is called **reducible** over  $F$ .

Irreducible polynomials allow us to construct the finite fields of non prime order. In fact, for a prime power  $q$ , let  $\mathbf{F}_q$  be the finite field with  $q$  elements, unique up to an isomorphism. A polynomial  $f$  of degree  $n$  greater than one, which is irreducible over  $\mathbf{F}_q$ , defines a field extension of degree  $n$  which is isomorphic to the field with  $q^n$  elements: the elements of this extension are the polynomials of degree lower than  $n$ ; addition, subtraction and multiplication by an element of  $\mathbf{F}_q$  are those of the polynomials; the product of two elements is the remainder of the division by  $f$  of their product as polynomials; the inverse of an element may be computed by the extended GCD algorithm (see **Arithmetic of algebraic extensions**).

It follows that, to compute in a finite field of non prime order, one needs to generate an irreducible polynomial. For this, the common method is to take a polynomial at random and test it for irreducibility. For sake of efficiency of the multiplication in the field, it is usual to search for polynomials of the shape  $x^n + ax + b$ .

Irreducible polynomials over finite fields are also use-



ful for Pseudorandom number generators using feedback shift registers and discrete logarithm over  $\mathbf{F}_2^n$ .

### Example

The polynomial  $P = x^4 + 1$  is irreducible over  $\mathbf{Q}$  but not over any finite field.

- On any field extension of  $\mathbf{F}_2$ ,  $P = (x+1)^4$ .
  - On every other finite field, at least one of  $-1$ ,  $2$  and  $-2$  is a square, because the product of two non squares is a square and so we have
1. If  $-1 = a^2$ , then  $P = (x^2 + a)(x^2 - a)$ .
  2. If  $2 = b^2$ , then  $P = (x^2 + bx + 1)(x^2 - bx + 1)$ .
  3. If  $-2 = c^2$ , then  $P = (x^2 + cx - 1)(x^2 - cx - 1)$ .

### 37.1.3 Complexity

Polynomial factoring algorithms use basic polynomial operations such as products, divisions, gcd, powers of one polynomial modulo another, etc. A multiplication of two polynomials of degree at most  $n$  can be done in  $O(n^2)$  operations in  $\mathbf{F}q$  using “classical” arithmetic, or in  $O(n \log(n) \log(\log(n)))$  operations in  $\mathbf{F}q$  using “fast” arithmetic. A Euclidean division (division with remainder) can be performed within the same time bounds. The cost of a polynomial greatest common divisor between two polynomials of degree at most  $n$  can be taken as  $O(n^2)$  operations in  $\mathbf{F}q$  using classical methods, or as  $O(n \log^2(n) \log(\log(n)))$  operations in  $\mathbf{F}q$  using fast methods. For polynomials  $h, g$  of degree at most  $n$ , the exponentiation  $h^q \bmod g$  can be done with  $O(\log(q))$  polynomial products, using exponentiation by squaring method, that is  $O(n^2 \log(q))$  operations in  $\mathbf{F}q$  using classical methods, or  $O(n \log(q) \log(n) \log(\log(n)))$  operations in  $\mathbf{F}q$  using fast methods.

In the algorithms that follow, the complexities are expressed in terms of number of arithmetic operations in  $\mathbf{F}q$ , using classical algorithms for the arithmetic of polynomials.

## 37.2 Factoring algorithms

Many algorithms for factoring polynomials over finite fields include the following three stages:

1. Square-free factorization
2. Distinct-degree factorization
3. Equal-degree factorization

An important exception is Berlekamp’s algorithm, which combines stages 2 and 3.

### 37.2.1 Berlekamp’s algorithm

Main article: Berlekamp’s algorithm

The Berlekamp’s algorithm is historically important as being the first factorization algorithm, which works well in practice. However, it contains a loop on the elements of the ground field, which implies that it is practicable only over small finite fields. For a fixed ground field, its time complexity is polynomial, but, for general ground fields, the complexity is exponential in the size of the ground field.

### 37.2.2 Square-free factorization

The algorithm determines a square-free factorization for polynomials whose coefficients come from the finite field  $\mathbf{F}q$  of order  $q = p^m$  with  $p$  a prime. This algorithm firstly determines the derivative and then computes the gcd of the polynomial and its derivative. If it is not one then the gcd is again divided into the original polynomial, provided that the derivative is not zero (a case that exists for non-constant polynomials defined over finite fields).

This algorithm uses the fact that, if the derivative of a polynomial is zero, then it is a polynomial in  $x^p$ , which is, if the coefficients belong to  $\mathbf{F}p$ , the  $p$ th power of the polynomial obtained by substituting  $x$  by  $x^{1/p}$ . If the coefficients do not belong to  $\mathbf{F}p$ , the  $p$ -th root of a polynomial with zero derivative is obtained by the same substitution on  $x$ , completed by applying the inverse of the Frobenius automorphism to the coefficients.

This algorithm works also over a field of characteristic zero, with the only difference that it never enters in the blocks of instructions where  $p$ th roots are computed. However, in this case, Yun’s algorithm is much more efficient because it computes the greatest common divisors of polynomials of lower degrees. A consequence is that, when factoring a polynomial over the integers, the algorithm which follows is not used: one compute first the square-free factorization over the integers, and to factor the resulting polynomials, one chooses a  $p$  such that they remain square-free modulo  $p$ .

**Algorithm: SFF** (Square-Free Factorization) **Input:** A monic polynomial  $f$  in  $\mathbf{F}q[x]$  **Output:** Square-free factorization of  $f$   $i \leftarrow 1$ ;  $R \leftarrow 1$ ;  $g \leftarrow f'$ ; **if**  $g \neq 0$  **then**  $c \leftarrow \gcd(f, g)$   $w \leftarrow f/c$  **while**  $w \neq 1$  **do**  $y \leftarrow \gcd(w, c)$ ;  $z \leftarrow w/y$   $R \leftarrow R \cdot z^i$ ;  $i \leftarrow i+1$   $w \leftarrow y$ ;  $c \leftarrow c/y$  **end while** **if**  $c \neq 1$  **then**  $c \leftarrow c^{1/p}$  **Output**( $R \cdot \text{SFF}(c^p)$ ) **else** **Output**( $R$ ) **end if** **else**  $f \leftarrow f^{1/p}$  **Output**( $\text{SFF}(f^p)$ ) **end if**

### Example of a square-free factorization

Let

$$f = x^{11} + 2x^9 + 2x^8 + x^6 + x^5 + 2x^3 + 2x^2 + 1 \in \mathbf{F}_3[x],$$

to be factored over the field with three elements.

The algorithm computes first

$$c = \gcd(f, f') = x^9 + 2x^6 + x^3 + 2.$$

Since the derivative is non-zero we have  $w = f/c = x^2 + 2$  and we enter the while loop. After one loop we have  $y = x + 2, z = x + 1$  and  $R = x + 1$  with updates  $i = 2, w = x + 2$  and  $c = x^8 + x^7 + x^6 + x^2 + x + 1$ . The second time through the loop gives  $y = x + 2, z = 1, R = x + 1$ , with updates  $i = 3, w = x + 2$  and  $c = x^7 + 2x^6 + x + 2$ . The third time through the loop also does not change  $R$ . For the fourth time through the loop we get  $y = 1, z = x + 2, R = (x + 1)(x + 2)^4$ , with updates  $i = 5, w = 1$  and  $c = x^6 + 1$ . Since  $w = 1$ , we exit the while loop. Since  $c \neq 1$ , it must be a perfect cube. The cube root of  $c$ , obtained by replacing  $x^3$  by  $x$  is  $x^2 + 1$ , and calling the square-free procedure recursively determines that it is square-free. Therefore, cubing it and combining it with the value of  $R$  to that point gives the square-free decomposition

$$f = (x + 1)(x^2 + 1)^3(x + 2)^4.$$

### 37.2.3 Distinct-degree factorization

This algorithm splits a square-free polynomial into a product of polynomials whose irreducible factors all have the same degree. Let  $f \in \mathbf{F}_q[x]$  of degree  $n$  be the polynomial to be factored.

**Algorithm** Distinct-degree factorization(DDF) **Input:** A monic square-free polynomial  $f \in \mathbf{F}_q[x]$  **Output:** The set of all pairs  $(g, d)$ , such that  $f$  has an irreducible factor of degree  $d$  and  $g$  is the product of all monic irreducible factors of  $f$  of degree  $d$ . **Begin**  $i := 1$ ;  $S := \emptyset$ ,  $f^* := f$ ; **while**  $\deg f^* \geq 2i$  **do**  $g = \gcd(f^*, x^{q^i} - x)$  **if**  $g \neq 1$ , **then**  $S := S \cup (g, i)$ ;  $f^* := f^*/g$ ; **end if**  $i := i + 1$ ; **end while**; **if**  $f^* \neq 1$ , **then**  $S := S \cup (f^*, \deg f^*)$ ; **if**  $S = \emptyset$  **then return**  $\{(f, 1)\}$  **else return**  $S$  **End**

The correctness of the algorithm is based on the following:

**Lemma.** For  $i \geq 1$  the polynomial

$$x^{q^i} - x \in \mathbf{F}_q[x]$$

is the product of all monic irreducible polynomials in  $\mathbf{F}_q[x]$  whose degree divides  $i$ .

At first glance, this is not efficient since it involves computing the GCD of polynomials of a degree which is exponential in the degree of the input polynomial. However

$$g = \gcd(f^*, x^{q^i} - x)$$

may be replaced by

$$g = \gcd(f^*, (x^{q^i} - x \bmod f^*)).$$

Therefore, we have to compute:

$$x^{q^i} - x \bmod f^*,$$

there are two methods:

**Method I.** Start from the value of

$$x^{q^{i-1}} \bmod f^*$$

computed at the preceding step and to compute its  $q$ -th power modulo the new  $f^*$ , using **exponentiation by squaring** method. This needs

$$O(\log(q) \deg(f)^2)$$

arithmetic operations in  $\mathbf{F}_q$  at each step, and thus

$$O(\log(q) \deg(f)^3)$$

arithmetic operations for the whole algorithm.

**Method II.** Using the fact that the  $q$ -th power is a linear map over  $\mathbf{F}_q$  we may compute its matrix with

$$O(\deg(f)^2(\log(q) + \deg(f)))$$

operations. Then at each iteration of the loop, compute the product of a matrix by a vector (with  $O(\deg(f)^2)$  operations). This induces a total number of operations in  $\mathbf{F}_q$  which is

$$O(\deg(f)^2(\log(q) + \deg(f))).$$

Thus this second method is more efficient and is usually preferred. Moreover, the matrix that is computed in this method is used, by most algorithms, for equal-degree factorization (see below); thus using it for the distinct-degree factorization saves further computing time.

### 37.2.4 Equal-degree factorization

#### Cantor–Zassenhaus algorithm

Main article: [Cantor–Zassenhaus algorithm](#)

In this section, we consider the factorization of a monic squarefree univariate polynomial  $f$ , of degree  $n$ , over a finite field  $\mathbf{F}_q$ , which has  $r \geq 2$  pairwise distinct irreducible factors  $f_1, \dots, f_r$  each of degree  $d$ .

We first describe an algorithm by Cantor and Zassenhaus (1981) and then a variant that has a slightly better complexity. Both are probabilistic algorithms whose running time depends on random choices ([Las Vegas algorithms](#)), and have a good average running time. In next section we describe an algorithm by Shoup (1990), which is also an equal-degree factorization algorithm, but is deterministic. All these algorithms require an odd order  $q$  for the field of coefficients. For more factorization algorithms see e.g. Knuth's book [The Art of Computer Programming](#) volume 2.

**Algorithm Cantor–Zassenhaus algorithm.** Input: A finite field  $\mathbf{F}_q$  of odd order  $q$ . A monic square free polynomial  $f$  in  $\mathbf{F}_q[x]$  of degree  $n = rd$ , which has  $r \geq 2$  irreducible factors each of degree  $d$ . Output: The set of monic irreducible factors of  $f$ . Factors := { $f$ }; while Size(Factors)  $< r$  do, Choose  $h$  in  $\mathbf{F}_q[x]$  with  $\deg(h) < n$  at random;  $g := h^{\frac{q^d-1}{2}} - 1 \pmod{f}$  for each  $u$  in Factors with  $\deg(u) > d$  do if  $\gcd(g, u) \neq 1$  and  $\gcd(g, u) \neq u$ , then Factors := Factors  $\setminus \{u\} \cup \{\gcd(g, u), u/\gcd(g, u)\}$ ; endif; endwhile return Factors.

The correctness of this algorithm relies on the fact that the ring  $\mathbf{F}_q[x]/f$  is a direct product of the fields  $\mathbf{F}_q[x]/f_i$  where  $f_i$  runs on the irreducible factors of  $f$ . As all these fields have  $q^d$  elements, the component of  $g$  in any of these fields is zero with probability

$$\frac{q^d - 1}{2q^d} \sim \frac{1}{2}.$$

This implies that the polynomial  $\gcd(g, u)$  is the product of the factors of  $g$  for which the component of  $g$  is zero.

It has been shown that the average number of iterations of the while loop of the algorithm is less than  $2.5 \log_2 r$ , giving an average number of arithmetic operations in  $\mathbf{F}_q$  which is  $O(dn^2 \log(r) \log(q))$ .<sup>[1]</sup>

In the typical case where  $d \log(q) > n$ , this complexity may be reduced to

$$O(n^2(\log(r) \log(q) + n))$$

by choosing  $h$  in the kernel of the linear map

$$v \rightarrow v^q - v \pmod{f}$$

and replacing the instruction

$$g := h^{\frac{q^d-1}{2}} - 1 \pmod{f}$$

by

$$g := h^{\frac{q-1}{2}} - 1 \pmod{f}.$$

The proof of validity is the same as above, replacing the direct product of the fields  $\mathbf{F}_q[x]/f_i$  by the direct product of their subfields with  $q$  elements. The complexity is decomposed in  $O(n^2 \log(r) \log(q))$  for the algorithm itself,  $O(n^2(\log(q) + n))$  for the computation of the matrix of the linear map (which may be already computed in the square-free factorization) and  $O(n^3)$  for computing its kernel. It may be noted that this algorithm works also if the factors have not the same degree (in this case the number  $r$  of factors, needed for stopping the while loop, is found as the dimension of the kernel). Nevertheless, the complexity is slightly better if square-free factorization is done before using this algorithm (as  $n$  may decrease with square-free factorization, this reduces the complexity of the critical steps).

#### Victor Shoup's algorithm

Like the algorithms of the preceding section, [Victor Shoup's algorithm](#) is an equal-degree factorization algorithm.<sup>[2]</sup> Unlike them, it is a deterministic algorithm. However, it is less efficient, in practice, than the algorithms of preceding section. For Shoup's algorithm, the input is restricted to polynomials over prime fields  $\mathbf{F}_p$ .

The worst case time complexity of Shoup's algorithm has a factor  $\sqrt{p}$ . Although exponential, this complexity is much better than previous deterministic algorithms (Berlekamp's algorithm) which have  $p$  as a factor. However, there are very few polynomials for which the computing time is exponential, and the average time complexity of the algorithm is polynomial in  $d \log(p)$ , where  $d$  is the degree of the polynomial, and  $p$  is the number of elements of the ground field.

Let  $g = g_1 \dots g_k$  be the desired factorization, where the  $g_i$  are distinct monic irreducible polynomials of degree  $d$ . Let  $n = \deg(g) = kd$ . We consider the ring  $R = \mathbf{F}_q[x]/g$  and denote also by  $x$  the image of  $x$  in  $R$ . The ring  $R$  is the direct product of the fields  $R_i = \mathbf{F}_q[x]/g_i$ , and we denote by  $p_i$  the natural homomorphism from the  $R$  onto  $R_i$ . The Galois group of  $R_i$  over  $\mathbf{F}_q$  is cyclic of order  $d$ , generated by the field automorphism  $u \rightarrow u^q$ . It follows that the roots of  $g_i$  in  $R_i$  are

$$p_i(x), p_i(x^q), p_i(x^{q^2}), \dots, p_i(x^{q^{d-1}}).$$

Like in the preceding algorithm, this algorithm uses the same subalgebra  $B$  of  $R$  as the [Berlekamp's algorithm](#),

sometimes called the “Berlekamp subalgebra” and defined as

$$\begin{aligned} B &= \{\alpha \in R : p_1(\alpha), \dots, p_k(\alpha) \in \mathbf{F}_q\} \\ &= \{u \in R : u^q = u\} \end{aligned}$$

A subset  $S$  of  $B$  is said a **separating set** if, for every  $1 \leq i < j \leq k$  there exists  $s \in S$  such that  $p_i(s) \neq p_j(s)$ . In the preceding algorithm, a separating set is constructed by choosing at random the elements of  $S$ . In Shoup’s algorithm, the separating set is constructed in the following way. Let  $s$  in  $R[Y]$  be such that

$$\begin{aligned} s &= (Y - x)(Y - x^q) \cdots (Y - x^{q^{d-1}}) \\ &= s_0 + \cdots + s_{d-1}Y^{d-1} + Y^d \end{aligned}$$

Then  $\{s_0, \dots, s_{d-1}\}$  is a separating set because  $p_i(s) = g_i$  for  $i = 1, \dots, k$  (the two monic polynomials have the same roots). As the  $g_i$  are pairwise distinct, for every pair of distinct indexes  $(i, j)$ , at least one of the coefficients  $s_h$  will satisfy  $p_i(s_h) \neq p_j(s_h)$ .

Having a separating set, Shoup’s algorithm proceeds as the last algorithm of the preceding section, simply by replacing the instruction “choose at random  $h$  in the kernel of the linear map  $v \rightarrow v^q - v \pmod{f}$ ” by “choose  $h + i$  with  $h$  in  $S$  and  $i$  in  $\{1, \dots, k-1\}$ ”.

### 37.3 Time complexity

As described in previous sections, for the factorization over finite fields, there are **randomized algorithms** of polynomial **time complexity** (for example Cantor–Zassenhaus algorithm). There are also deterministic algorithms with a polynomial average complexity (for example Shoup’s algorithm).

The existence of a deterministic algorithm with a polynomial worst-case complexity is still an open problem.

### 37.4 Rabin’s test of irreducibility

Like distinct-degree factorization algorithm, Rabin’s algorithm<sup>[3]</sup> is based on the Lemma stated above. Distinct-degree factorization algorithm tests every  $d$  not greater than half the degree of the input polynomial. Rabin’s algorithm takes advantage that the factors are not needed for considering fewer  $d$ . Otherwise, it is similar to distinct-degree factorization algorithm. It is based on the following fact.

Let  $p_1, \dots, p_k$ , be all the prime divisors of  $n$ , and denote  $n/p_i = n_i$ , for  $1 \leq i \leq k$  polynomial  $f$  in  $\mathbf{F}_q[x]$  of degree  $n$  is irreducible in  $\mathbf{F}_q[x]$  if and only if

$\gcd(f, x^{q^{n_i}} - x) = 1$ , for  $1 \leq i \leq k$ , and  $f$  divides  $x^{q^n} - x$ . In fact, if  $f$  has a factor of degree not dividing  $n$ , then  $f$  does not divide  $x^{q^n} - x$ ; if  $f$  has a factor of degree dividing  $n$ , then this factor divides at least one of the  $x^{q^{n_i}} - x$ .

**Algorithm Rabin Irreducibility Test** **Input:** A monic polynomial  $f$  in  $\mathbf{F}_q[x]$  of degree  $n$ ,  $p_1, \dots, p_k$  all distinct prime divisors of  $n$ . **Output:** Either “ $f$  is irreducible” or “ $f$  is reducible”. **Begin** for  $j = 1$  to  $k$  **do**  $n_j = n/p_j$ ; **for**  $i = 1$  to  $k$  **do**  $h := x^{q^{n_i}} - x \pmod{f}$ ;  $g := \gcd(f, h)$ ; **if**  $g \neq 1$ , **then return** ‘ $f$  is reducible’ **and STOP; end for;**  $g := x^{q^n} - x \pmod{f}$ ; **if**  $g = 0$ , **then return** “ $f$  is irreducible”, **else return** “ $f$  is reducible” **end.**

The basic idea of this algorithm is to compute  $x^{q^{n_i}} \pmod{f}$  starting from the smallest  $n_1, \dots, n_k$  by repeated squaring or using the **Frobenius automorphism**, and then to take the correspondent gcd. Using the elementary polynomial arithmetic, the computation of the matrix of the Frobenius automorphism needs  $O(n^2(n + \log q))$  operations in  $\mathbf{F}_q$ , the computation of

$$x^{q^{n_i}} - x \pmod{f}$$

needs  $O(n^3)$  further operations, and the algorithm itself needs  $O(kn^2)$  operations, giving a total of  $O(n^2(n + \log q))$  operations in  $\mathbf{F}_q$ . Using fast arithmetic (complexity  $O(n \log n)$  for multiplication and division, and  $O(n(\log n)^2)$  for GCD computation), the computation of the  $x^{q^{n_i}} - x \pmod{f}$  by repeated squaring is  $O(n^2 \log n \log q)$ , and the algorithm itself is  $O(kn(\log n)^2)$ , giving a total of  $O(n^2 \log n \log q)$  operations in  $\mathbf{F}_q$ .

### 37.5 See also

- Berlekamp’s algorithm
- Cantor–Zassenhaus algorithm
- Polynomial factorization

### 37.6 References

- KEMPFERT, H (1969) On the *Factorization of Polynomials* Department of Mathematics, The Ohio State University, Columbus, Ohio 43210
- Shoup, Victor (1996) *Smoothness and Factoring Polynomials over Finite Fields* Computer Science Department University of Toronto
- Von Zur Gathen, J.; Panario, D. (2001). Factoring Polynomials Over Finite Fields: A Survey. *Journal of Symbolic Computation*, Volume 31, Issues 1-2, January 2001, 3–17.

- Gao Shuhong, Panario Daniel, *Test and Construction of Irreducible Polynomials over Finite Fields* Department of mathematical Sciences, Clemson University, South Carolina, 29634-1907, USA. and Department of computer science University of Toronto, Canada M5S-1A4
- Shoup, Victor (1989) *New Algorithms for Finding Irreducible Polynomials over Finite Fields* Computer Science Department University of Wisconsin-Madison
- Geddes, Keith O.; Czapor, Stephen R.; Labahn, George (1992). *Algorithms for computer algebra*. Boston, MA: Kluwer Academic Publishers. pp. xxii+585. ISBN 0-7923-9259-0.

### 37.7 External links

- Some irreducible polynomials <http://www.math.umn.edu/~jgarrett/m/algebra/notes/07.pdf>
- Field and Galois Theory :<http://www.jmilne.org/math/CourseNotes/FT.pdf>
- Galois Field:<http://designtheory.org/library/encyc/topics/gf.pdf>
- Factoring polynomials over finite fields: <http://www.science.unitn.it/~degtraaf/compalg/polfact.pdf>

### 37.8 Notes

- [1] Flajolet, Philippe; Steyaert, Jean-Marc (1982), *A branching process arising in dynamic hashing, trie searching and polynomial factorization*, *Lecture Notes in Comput. Sci.*, **140**, Springer, pp. 239–251
- [2] Victor Shoup, *On the deterministic complexity of factoring polynomials over finite fields*, *Information Processing Letters* 33:261-267, 1990
- [3] Rabin, Michael (1980). “Probabilistic algorithms in finite fields”. *SIAM Journal on Computing*. **9** (2): 273–280. doi:10.1137/0209024.

# Chapter 38

## Second Johnson bound

In applied mathematics, the **Johnson bound** (named after **Selmer Martin Johnson**) is a limit on the size of **error-correcting codes**, as used in **coding theory** for **data transmission** or **communications**.

$$A_q(n, d) \leq \frac{q^n}{\sum_{i=0}^t \binom{n}{i} (q-1)^i + \frac{\binom{n}{t+1} (q-1)^{t+1}}{A_q(n, d, t+1)}}.$$

**Theorem 2 (Johnson bound for  $A_q(n, d, w)$ ):**

(i) If  $d > 2w$ ,

$$A_q(n, d, w) = 1.$$

(ii) If  $d \leq 2w$ , then define the variable  $e$  as follows. If  $d$  is even, then define  $e$  through the relation  $d = 2e$ ; if  $d$  is odd, define  $e$  through the relation  $d = 2e - 1$ . Let  $q^* = q - 1$ . Then,

$$A_q(n, d, w) \leq \left\lfloor \frac{nq^*}{w} \left\lfloor \frac{(n-1)q^*}{w-1} \left\lfloor \dots \left\lfloor \frac{(n-w+e)q^*}{e} \right\rfloor \dots \right\rfloor \right\rfloor \right\rfloor$$

where  $\lfloor \cdot \rfloor$  is the **floor function**.

**Remark:** Plugging the bound of Theorem 2 into the bound of Theorem 1 produces a numerical upper bound on  $A_q(n, d)$ .

### 38.1 Definition

Let  $C$  be a  $q$ -ary **code** of length  $n$ , i.e. a subset of  $\mathbb{F}_q^n$ . Let  $d$  be the minimum distance of  $C$ , i.e.

$$d = \min_{x, y \in C, x \neq y} d(x, y),$$

where  $d(x, y)$  is the **Hamming distance** between  $x$  and  $y$ .

Let  $C_q(n, d)$  be the set of all  $q$ -ary codes with length  $n$  and minimum distance  $d$  and let  $C_q(n, d, w)$  denote the set of codes in  $C_q(n, d)$  such that every element has exactly  $w$  nonzero entries.

Denote by  $|C|$  the number of elements in  $C$ . Then, we define  $A_q(n, d)$  to be the largest size of a code with length  $n$  and minimum distance  $d$ :

$$A_q(n, d) = \max_{C \in C_q(n, d)} |C|.$$

Similarly, we define  $A_q(n, d, w)$  to be the largest size of a code in  $C_q(n, d, w)$ :

$$A_q(n, d, w) = \max_{C \in C_q(n, d, w)} |C|.$$

**Theorem 1 (Johnson bound for  $A_q(n, d)$ ):**

If  $d = 2t + 1$ ,

$$A_q(n, d) \leq \frac{q^n}{\sum_{i=0}^t \binom{n}{i} (q-1)^i + \frac{\binom{n}{t+1} (q-1)^{t+1} - \binom{d}{t+1} A_q(n, d, t+1)}{A_q(n, d, t+1)}}.$$

If  $d = 2t$ ,

### 38.2 See also

- Singleton bound
- Hamming bound
- Plotkin bound
- Elias Bassalygo bound
- Gilbert–Varshamov bound
- Griesmer bound

### 38.3 References

- Johnson, Selmer Martin (April 1962). "A new upper bound for error-correcting codes". *IRE Transactions on Information Theory*: 203–207.

- Huffman, William Cary; Pless, Vera S. (2003). *Fundamentals of Error-Correcting Codes*. Cambridge University Press. ISBN 978-0-521-78280-7.



## Chapter 39

# Folded Reed–Solomon code

In coding theory, **folded Reed–Solomon codes** are like **Reed–Solomon codes**, which are obtained by mapping  $m$  Reed–Solomon codewords over a larger alphabet by careful bundling of codeword symbols.

Folded Reed–Solomon codes are also a special case of **Parvaresh–Vardy codes**.

Using optimal parameters one can decode with a **rate** of  $R$ , and achieve a decoding radius of  $1 - R$ .

The term “folded Reed–Solomon codes” was coined in a paper by V.Y. Krachkovsky with an algorithm that presented Reed–Solomon codes with many random “phased burst” errors . The list-decoding algorithm for folded RS codes corrects beyond the  $1 - \sqrt{R}$  bound for Reed–Solomon codes achieved by the **Guruswami–Sudan** algorithm for such phased burst errors.

### 39.1 History

One of the ongoing challenges in Coding Theory is to have error correcting codes achieve an optimal trade-off between (Coding) Rate and Error-Correction Radius. Though this may not be possible to achieve practically (due to Noisy Channel Coding Theory issues), quasi optimal tradeoffs can be achieved theoretically.

Prior to Folded Reed–Solomon codes being devised, the best Error-Correction Radius achieved was  $1 - \sqrt{R}$ , by **Reed–Solomon codes** for all rates  $R$ .

An improvement upon this  $1 - \sqrt{R}$  bound was achieved by Parvaresh and Vardy for rates  $R < \frac{1}{16}$ .

For  $R \rightarrow 0$  the Parvaresh–Vardy algorithm can decode a fraction  $1 - O(R \log(1/R))$  of errors.

Folded Reed–Solomon Codes improve on these previous constructions, and can be list decoded in polynomial time for a fraction  $(1 - R - \varepsilon)$  of errors for any constant  $\varepsilon > 0$ .

### 39.2 Definition

$$f(X) \mapsto \begin{bmatrix} f(1) \\ f(\gamma) \\ \vdots \\ f(\gamma^{m-1}) \end{bmatrix}, \begin{bmatrix} f(\gamma^m) \\ f(\gamma^{m+1}) \\ \vdots \\ f(\gamma^{2m-1}) \end{bmatrix}, \dots, \begin{bmatrix} f(\gamma^{n-m}) \\ f(\gamma^{n-m+1}) \\ \vdots \\ f(\gamma^{n-1}) \end{bmatrix}$$

Consider a Reed–Solomon  $[n = q - 1, k]_q$  code of length  $n$  and **dimension**  $k$  and a folding parameter  $m \geq 1$ . Assume that  $m$  divides  $n$ .

Mapping for Reed–Solomon codes like this:

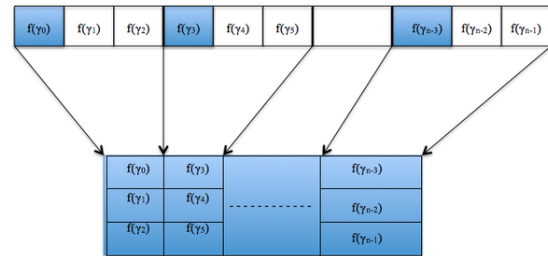
$$f \mapsto \langle f(1), f(\gamma^1), f(\gamma^2), \dots, f(\gamma^{n-1}) \rangle$$

where  $\gamma \in \mathbb{F}_q$  is a **primitive element** in

$$\mathbb{F}_q = \{0, 1, \gamma, \gamma^2, \dots, \gamma^{n-1}\}$$

The  $m$  folded version of Reed Solomon code  $C$ , denoted  $FRS_{\mathbb{F}, \gamma, m, k}$  is a code of block length  $N = n/m$  over  $\mathbb{F}^m$ .  $FRS_{\mathbb{F}, \gamma, m, k}$  are just  $[q - 1, k]$  Reed Solomon codes with  $m$  consecutive symbols from RS codewords grouped together.

#### 39.2.1 Graphic description



Folding of Reed–Solomon code with folding parameter  $m=3$

The above definition is made more clear by means of the diagram with  $m = 3$ , where  $m$  is the folding parameter.

The message is denoted by  $f(X)$ , which when encoded using Reed–Solomon encoding, consists of values of  $f$  at  $x_0, x_1, x_2, \dots, x_{n-1}$ , where  $x_i = \gamma^i$ .

Then bundling is performed in groups of 3 elements, to give a codeword of length  $n/3$  over the alphabet  $\mathbb{F}_q^3$ .

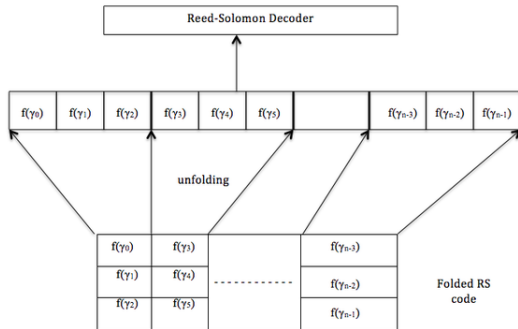
Something to be observed here is that the folding operation demonstrated does not change the rate  $R$  of the original Reed–Solomon code.

To prove this, consider a linear  $[n, k, d]_q$  code, of length  $n$ , dimension  $k$  and distance  $d$ . The  $m$  folding operation will make it a  $[\frac{n}{m}, \frac{k}{m}, \frac{d}{m}]_{q^m}$  code. By this, the rate  $R = \frac{n}{k}$  will be the same.

### 39.3 Folded Reed–Solomon codes and the singleton bound

According to the asymptotic version of the **singleton bound**, it is known that the **relative distance**  $\delta$ , of a code must satisfy  $R \leq 1 - \delta + o(1)$  where  $R$  is the rate of the code. As proved earlier, since the rate  $R$  is maintained, the relative distance  $\delta \geq 1 - R$  also meets the Singleton bound.

#### 39.3.1 Why folding might help?



Decoding a folded Reed–Solomon code

Folded Reed–Solomon codes are basically the same as Reed Solomon codes, just viewed over a larger alphabet. To show how this might help, consider a folded Reed–Solomon code with  $m = 3$ . Decoding a Reed–Solomon code and folded Reed–Solomon code from the same fraction of errors  $\rho$  are tasks of almost of the same computational intensity: one can **unfold** the received word of the folded Reed–Solomon code, treat it as an received word of the original Reed–Solomon code, and run the Reed–Solomon list decoding algorithm on it. Obviously, this list will contain all the folded Reed–Solomon codewords within distance  $\rho$  of the received word, along with some extras, which we can expurgate.

Also, decoding a folded Reed–Solomon code is an easier task. Suppose we want to correct a third of errors. The decoding algorithm chosen must correct an error pattern that corrects every third symbol in the Reed–Solomon encoding. But after folding, this error pattern will corrupt

all symbols over  $\mathbb{F}_q^3$  and will eliminate the need for error correction. This propagation of errors is indicated by the blue color in the graphical description. This proves that the for a fixed fraction of errors  $\rho$ , the folding operation reduces the channel's flexibility to distribute errors, which in turn leads to a reduction in the number of error patterns that need to be corrected.

### 39.4 How folded Reed–Solomon (FRS) codes and Parvaresh Vardy (PV) codes are related

We can relate Folded Reed Solomon codes with **Parvaresh Vardy** codes which encodes a polynomial  $f$  of degree  $k$  with polynomials  $f_0 = f, f_1, \dots, f_{s-1} (s \geq 2)$  where  $f_i(X) = f_{i-1}(X)^d \mod E(X)$  where  $E(X)$  is an **irreducible polynomial**. While choosing irreducible polynomial  $E(X) = X^q - \gamma$  and parameter  $d$  we should check if every polynomial  $f$  of degree at most  $k$  satisfies  $f(\gamma X) = f(X)^d \mod E(X)$  since  $f(\gamma X)$  is just the shifted counterpart of  $f(X)$  where  $\gamma$  is the **primitive element** in  $\mathbb{F}_q$ . Thus folded RS code with bundling together code symbols is PV code of order  $s = m$  for the set of evaluation points

$$\{1, \gamma, \gamma^{2m}, \dots, \gamma^{(\frac{n}{m}-1)m}\}$$

If we compare the folded RS code to a PV code of order 2 for the set of evaluation points

$$\{1, \gamma, \dots, \gamma^{m-2}, \gamma^m, \gamma^{m+1}, \dots, \gamma^{2m-2}, \dots, \gamma^{n-m}, \gamma^{n-m+1}, \dots, \gamma^{n-2}\}$$

we can see that in PV encoding of  $f$ , for every  $0 \leq i \leq n/m - 1$  and every  $0 < j < m - 1$ ,  $f(\gamma^{mi+j})$  appears at  $f(\gamma^{mi+j})$  and  $f_1(\gamma^{-1}\gamma^{mi+j})$ ,

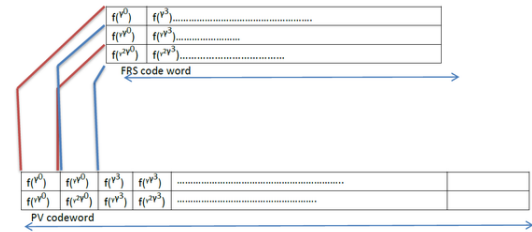


Figure: The correspondence between a folded Reed Solomon code with  $m=3$  and Parvaresh varyd code with  $s=2$  evaluated over  $\{1, \gamma, \gamma^2, \gamma^4, \dots, \gamma^{n-2}\}$ .

#### Relation between PV codes and FRS codes

unlike in the folded FRS encoding in which it appears only once. Thus, the PV and folded RS codes have same information but only the rate of FRS is bigger by a factor of  $2(m-1)/m$  and hence the **list decoding** radius trade-off is better for folded RS code by just using the list decodability of the PV codes. The plus point is in choosing

FRS code in a way that they are compressed forms of suitable PV code with similar error correction performance with better rate than corresponding PV code. One can use this idea to construct a folded RS codes of rate  $R$  that are list decodable up to radius approximately  $1 - R^{s/[s+1]}$  for  $s \geq 1$ .

$$\left( \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{m-1} \end{bmatrix}, \begin{bmatrix} y_m \\ y_{m+1} \\ y_{m+2} \\ \vdots \\ y_{2m-1} \end{bmatrix}, \dots, \begin{bmatrix} y_{n-m} \\ y_{n-m+1} \\ y_{n-m+2} \\ \vdots \\ y_{n-1} \end{bmatrix} \right)$$

We interpolate the nonzero polynomial

### 39.5 Brief overview of list-decoding folded Reed–Solomon codes

A list decoding algorithm which runs in quadratic time to decode FRS code up to radius  $1 - R - \varepsilon$  is presented by Guruswami. The algorithm essentially has three steps namely the interpolation step in which Welch Berlekamp style interpolation is used to interpolate the non-zero polynomial

$$Q(X, Y_1, \dots, Y_s) = A_0(X) + A_1(X)Y_1 + \dots + A_s(X)Y_s, \quad \begin{cases} \deg(A_i) \\ \deg(A_0) \end{cases}$$

by using a carefully chosen degree parameter  $D$ .

$$D = \left\lfloor \frac{N(m-s+1) - k + 1}{s+1} \right\rfloor$$

So the interpolation requirements will be

$$Q(X, Y_1, Y_2, \dots, Y_s) = A_0(X) + A_1(X)Y_1 + A_2(X)Y_2 + \dots + A_s(X)Y_s, \quad Q(\gamma^{im+j} y_{im+j}, y_{im+j+1}, \dots, y_{im+j+s-1}) = 0, \quad \text{for } i = 0, 1, \dots, \frac{n}{m}$$

after which all the polynomials  $f \in \mathbb{F}_q[X]$  with degree  $k-1$  satisfying the equation derived in interpolation are found. In the third step the actual list of close-by codewords are known by pruning the solution subspace which takes  $q^s$  time.

Then the number of monomials in  $Q(X, Y_1, \dots, Y_s)$  is

$$(D+1)s + D + k = (D+1)(s+1) + k - 1 > N(m-s+1)$$

Because the number of monomials in  $Q(X, Y_1, \dots, Y_s)$  is greater than the number of interpolation conditions. We have below lemma

### 39.6 Linear-algebraic list decoding algorithm

Guruswami presents a  $n^{\Omega(1/\varepsilon^2)}$  time list decoding algorithm based on linear-algebra, which can decode folded Reed–Solomon code up to radius  $1 - R - \varepsilon$  with a list-size of  $n^{O(1/\varepsilon^2)}$ . There are three steps in this algorithm: Interpolation Step, Root Finding Step and Prune Step. In the Interpolation step it will try to find the candidate message polynomial  $f(x)$  by solving a linear system. In the Root Finding step, it will try to find the solution subspace by solving another linear system. The last step will try to prune the solution subspace gained in the second step. We will introduce each step in details in the following.

**Lemma 1.**  $0 \neq Q \in \mathbb{F}_q[X, Y_1, \dots, Y_s]$  satisfying the above interpolation condition can be found by solving a homogeneous linear system over  $\mathbb{F}_q$  with at most  $Nm$  constraints and variables. Moreover this interpolation can be performed in  $O(Nm \log^2(Nm) \log \log(Nm))$  operations over  $\mathbb{F}_q$ .<sup>[1]</sup>

This lemma shows us that the interpolation step can be done in near-linear time.

For now, we have talked about everything we need for the multivariate polynomial  $Q(X, Y_1, \dots, Y_s)$ . The remaining task is to focus on the message polynomials  $f(X)$ .

**Lemma 2.** If a candidate message polynomial  $f(X) \in \mathbb{F}[X]$  is a polynomial of degree at most  $k-1$  whose Folded Reed–Solomon encoding agrees with the received word  $y$  in at least  $t$  columns with

$$t > \frac{D + k - 1}{m - s + 1},$$

then  $Q(X, f(X), f(\gamma X), \dots, f(\gamma_{s-1} X)) = 0$ .<sup>[2]</sup>

#### 39.6.1 Step 1: The interpolation step

It is a Welch–Berlekamp-style interpolation (because it can be viewed as the higher-dimensional generalization of the Welch–Berlekamp algorithm). Suppose we received a codeword  $y$  of the  $m$ -folded Reed–Solomon code as shown below

Here “agree” means that all the  $m$  values in a column should match the corresponding values in codeword  $y$ .

This lemma shows us that any such polynomial  $Q(X, Y_1, \dots, Y_s)$  presents an algebraic condition that must be satisfied for those message polynomials  $f(x)$  that we are interested in list decoding.

Combining Lemma 2 and parameter  $D$ , we have

$$t(m - s + 1) > \frac{N(m - s + 1) + s(k - 1)}{s + 1}$$

Further we can get the decoding bound

$$t \geq \frac{N}{s + 1} + \frac{s}{s + 1} \cdot \frac{k}{m - s + 1} = N \left( \frac{1}{s + 1} + \frac{s}{s + 1} \cdot \frac{1}{m - s + 1} \right)$$

We notice that the fractional agreement is

$$\frac{1}{s + 1} + \frac{s}{s + 1} \cdot \frac{mR}{m - s + 1}$$

### 39.6.2 Step 2: The root-finding step

During this step, our task focus on how to find all polynomials  $f \in \mathbb{F}_q[X]$  with degree no more than  $k - 1$  and satisfy the equation we get from Step 1, namely

$$A_0(X) + A_1(X)f(X) + A_2(X)f(\gamma X) + \dots + A_s(X)f(\gamma^s X) = 0$$

Since the above equation forms a linear system equations over  $\mathbb{F}_q$  in the coefficients  $f_0, f_1, \dots, f_{k-1}$  of the polynomial

$$f(X) = f_0 + f_1X + \dots + f_{k-1}X^{k-1},$$

the solutions to the above equation is an affine subspace of  $\mathbb{F}_q^k$ . This fact is the key point that gives rise to an efficient algorithm - we can solve the linear system.

It is natural to ask how large is the dimension of the solution? Is there any upper bound on the dimension? Having an upper bound is very important in constructing an efficient list decoding algorithm because one can simply output all the codewords for any given decoding problem.

Actually it indeed has an upper bound as below lemma argues.

**Lemma 3.** If the order of  $\gamma$  is at least  $k$  (in particular when  $\gamma$  is primitive), then the dimension of the solution is at most  $s - 1$ .<sup>[3]</sup>

This lemma shows us the upper bound of the dimension for the solution space.

Finally, based on the above analysis, we have below theorem

**Theorem 1.** For the folded Reed–Solomon code  $FRS_q^{(m)}[n, k]$  of block length  $N = \frac{n}{m}$  and rate  $R = \frac{k}{n}$ , the following holds for all integers  $s, 1 \leq s \leq m$ . Given a received word  $y \in (\mathbb{F}_q^m)^N$ , in  $O((Nm \log q)^2)$  time, one can find a basis for a subspace of dimension at most  $s - 1$  that contains all message polynomials  $f \in \mathbb{F}_q[X]$  of degree less than  $k$  whose FRS encoding differs from  $y$  in at most a fraction

$$\frac{s}{s + 1} \left( 1 - \frac{mR}{m - s + 1} \right)$$

of the  $N$  codeword positions.

When  $s = m = 1$ , we notice that this reduces to a unique decoding algorithm with up to a fraction  $(1 - R)/2$  of errors. In other words, we can treat unique decoding algorithm as a specialty of list decoding algorithm. The quantity is about  $n^{O(1/\varepsilon)}$  for the parameter choices that achieve a list decoding radius of  $1 - R - \varepsilon$ .

Theorem 1 tells us exactly how large the error radius would be.

Now we finally get the solution subspace. However, there is still one problem standing. The list size in the worst case is  $n^{\Omega(1/\varepsilon)}$ . But the actual list of close-by codewords is only a small set within that subspace. So we need some process to prune the subspace to narrow it down. This prune process takes  $q^s$  time in the worst case. Unfortunately it is not known how to improve the running time because we do not know how to improve the bound of the list size for folded Reed–Solomon code.

Things get better if we change the code by carefully choosing a subset of all possible degree  $k - 1$  polynomials as messages, the list size shows to be much smaller while only losing a little bit in the rate. We will talk about this briefly in next step.

### 39.6.3 Step 3: The prune step

By converting the problem of decoding a folded Reed–Solomon code into two linear systems, one linear system that is used for the interpolation step and another linear system to find the candidate solution subspace, the complexity of the decoding problem is successfully reduced to quadratic. However, in the worst case, the bound of list size of the output is pretty bad.

It was mentioned in Step 2 that if one carefully chooses only a subset of all possible degree  $k - 1$  polynomials as messages, the list size can be much reduced. Here we will expand our discussion.

To achieve this goal, the idea is to limit the coefficient vector  $(f_0, f_1, \dots, f_{k-1})$  to a special subset  $\nu \subseteq \mathbb{F}_q^k$ , which satisfies below two conditions:

**Condition 1.** The set  $\nu$  must be large enough ( $|\nu| \geq q^{(1-\varepsilon)k}$ ).

This is to make sure that the rate will be at most reduced by factor of  $(1 - \varepsilon)$ .

**Condition 2.** The set  $\nu$  should have low intersection with any subspace  $S$  of dimension  $s$  satisfying  $S \subset \mathbb{F}_q^k$  and  $|S \cap \nu| \leq L$ . Such a subset is called subspace-evasive subset.

The bound for the list size at worst case is  $n^{\Omega(1/\varepsilon)}$ , and it can be reduced to a relative small bound  $O(1/\varepsilon^2)$  by using subspace-evasive subsets.

During this step, as it has to check each element of the solution subspace that we get from Step 2, it takes  $q^s$  time in the worst case ( $s$  is the dimension of the solution subspace).

Dvir and Lovett improved the result based on the work of Guruswami, which can reduce the list size to a constant.

Here is only presented the idea that is used to prune the solution subspace. For the details of the prune process, please refer to papers by Guruswami, Dvir and Lovett, which are listed in the reference.

### 39.6.4 Summary

If we don't consider the Step 3, this algorithm can run in quadratic time. A summary for this algorithm is listed below.

## 39.7 See also

- Reed–Solomon error correction
- Singleton bound
- Coding theory
- List decoding

## 39.8 References

- [1] For proof see Proposition 5.11 in Chapter 5 in Brander's thesis listed in the references.
- [2] For the details of the proof, please refer to Guruswami's paper.
- [3] For the details of the proof, please refer to Guruswami's paper.
1. Atri Rudra's Lecture Notes: Folded Reed–Solomon Codes
2. Atri Rudra's Lecture Notes: Bounds
3. A paper by Atri Rudra and Venkatesan Guruswami: Decoding Folded Reed–Solomon Codes
4. A chapter on List Decoding of folded Reed–Solomon codes: List Decoding of Folded Reed–Solomon Codes
5. Venkatesan Guruswami's lecture notes: Elementary bounds on codes
6. Venkatesan Guruswami's lecture notes: List Decoding Folded Reed–Solomon Code
7. Guruswami, Venkatesan. “Linear-algebraic list decoding of folded Reed–Solomon codes”. [arXiv:1106.0436v2](#).
8. Dvir, Zeev; Lovett, Shachar. “Subspace evasive sets”. [arXiv:1110.5696v2](#).
9. PHD Thesis by Kristian Brander: Interpolation and List Decoding of Algebraic Codes
10. Krachkovsky, V. Y. (2003). “Reed–Solomon codes for correcting phased error bursts”. *IEEE Trans. Inform. Theory.* **49** (11): 2975–2984. doi:10.1109/TIT.2003.819333.

# Chapter 40

## Forney algorithm

In **coding theory**, the **Forney algorithm** (or **Forney's algorithm**) calculates the error values at known error locations. It is used as one of the steps in decoding **BCH codes** and **Reed–Solomon codes** (a subclass of BCH codes). George David Forney, Jr. developed the algorithm.<sup>[1]</sup>

$$S(x) = s_0x^0 + s_1x^1 + s_2x^2 + \cdots + s_{2t-1}x^{2t-1}.$$

Then evaluate the error values:<sup>[3]</sup>

### 40.1 Procedure

*Need to introduce terminology and the setup...*

Code words look like polynomials. By design, the generator polynomial has consecutive roots  $\alpha^c, \alpha^{c+1}, \dots, \alpha^{c+d-2}$ .

Syndromes

Error location polynomial<sup>[2]</sup>

$$\Lambda(x) = \prod_{i=1}^{\nu} (1 - x X_i) = 1 + \sum_{i=1}^{\nu} \lambda_i x^i$$

The zeros of  $\Lambda(x)$  are  $X_1^{-1}, \dots, X_{\nu}^{-1}$ . The zeros are the reciprocals of the error locations  $X_j = \alpha^{i_j}$ .

Once the error locations are known, the next step is to determine the error values at those locations. The error values are then used to correct the received values at those locations to recover the original codeword.

In the more general case, the error weights  $e_j$  can be determined by solving the linear system

$$s_0 = e_1 \alpha^{(c+0) i_1} + e_2 \alpha^{(c+0) i_2} + \cdots$$

$$s_1 = e_1 \alpha^{(c+1) i_1} + e_2 \alpha^{(c+1) i_2} + \cdots$$

...

However, there is a more efficient method known as the Forney algorithm, which is based on **Lagrange interpolation**. First calculate the error evaluator polynomial<sup>[3]</sup>

$$\Omega(x) = S(x) \Lambda(x) \pmod{x^{2t}}$$

Where  $S(x)$  is the partial syndrome polynomial:<sup>[4]</sup>

$$e_j = -\frac{X_j^{1-c} \Omega(X_j^{-1})}{\Lambda'(X_j^{-1})}$$

The value  $c$  is often called the “first consecutive root” or “fcr”. Some codes select  $c = 1$ , so the expression simplifies to:

$$e_j = -\frac{\Omega(X_j^{-1})}{\Lambda'(X_j^{-1})}$$

### 40.2 Formal derivative

Main article: **Formal derivative**

$\Lambda'(x)$  is the **formal derivative** of the error locator polynomial  $\Lambda(x)$ .<sup>[3]</sup>

$$\Lambda'(x) = \sum_{i=1}^{\nu} i \cdot \lambda_i x^{i-1}$$

In the above expression, note that  $i$  is an integer, and  $\lambda_i$  would be an element of the finite field. The operator  $\cdot$  represents ordinary multiplication (repeated addition in the finite field) and not the finite field's multiplication operator.

### 40.3 Derivation

**Lagrange interpolation**

Gill (n.d., pp. 52–54) gives a derivation of the Forney algorithm.

## 40.4 Erasures

Define the erasure locator polynomial

$$\Gamma(x) = \prod (1 - x \alpha^{j_i})$$

Where the erasure locations are given by  $j_i$ . Apply the procedure described above, substituting  $\Gamma$  for  $\Lambda$ .

If both errors and erasures are present, use the error-and-erasure locator polynomial

$$\Psi(x) = \Lambda(x) \Gamma(x)$$

## 40.5 See also

- BCH code
- Reed–Solomon error correction

## 40.6 References

- [1] Forney 1965
- [2] Gill n.d., p. 24
- [3] Gill n.d., p. 47
- [4] Gill (n.d., p. 48)
- Forney, G., Jr. (October 1965), “On Decoding BCH Codes”, *IEEE Transactions on Information Theory*, **11** (4): 549–557, doi:10.1109/TIT.1965.1053825, ISSN 0018-9448
- Gill, John (n.d.), *EE387 Notes #7, Handout #28* (PDF), Stanford University, pp. 42–45, retrieved April 21, 2010
- W. Wesley Peterson's book

## 40.7 External links



# Chapter 41

## Fuzzy extractor

**Fuzzy extractors** convert **biometric** data into **random** strings, which makes it possible to apply **cryptographic** techniques for biometric security. They are used to **encrypt** and **authenticate** users records, with biometric inputs as a key. Historically, the first biometric system of this kind was designed by Juels and Wattenberg and was called “Fuzzy commitment”, where the cryptographic key is decommitted using biometric data. “Fuzzy”, in that context, implies that the value close to the original one can extract the committed value. Later, Juels and Sudan came up with **Fuzzy vault** schemes which are order invariant for the fuzzy commitment scheme but uses a **Reed–Solomon code**. Codeword is evaluated by **polynomial** and the secret message is inserted as the coefficients of the polynomial. The polynomial is evaluated for different values of a set of features of the biometric data. So Fuzzy commitment and Fuzzy Vault were precursor to Fuzzy extractors. Fuzzy extractor is a biometric tool to authenticate a user using its own biometric template as a key. They extract uniform and random string  $R$  from its input  $w$  that has tolerance for noise. If the input changes to  $w'$  but is still close to  $w$ , the string  $R$  can still be re-constructed. When  $R$  is used first time to re-construct, it outputs a helper string  $P$  which can be made public without compromising the security of  $R$  (used for encryption and authentication key) and  $P$  (helper string) is stored to recover  $R$ . They remain secure even when the adversary modifies  $P$  (key agreement between a user and a server based only on a biometric input). This article is based on the papers “Fuzzy Extractors: A Brief Survey of Results from 2004 to 2006” and “Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data” by Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin and Adam Smith

### 41.1 Motivation

As fuzzy extractors deal with how to generate strong keys from Biometrics and other Noisy Data, it applies **cryptography** paradigms to biometric data and that means (1) Make little assumptions about the biometric data (these data comes from variety of sources and don't want adversary to exploit that so it is best to assume the in-

put is unpredictable) (2) Apply cryptographic application techniques to the input. (for that fuzzy extractor converts biometric data into secret, uniformly random and reliably reproducible random string). According to “Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data” paper by Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin and Adam Smith – these techniques also have other broader applications (when noisy inputs are used) such as human memory, images used as passwords, keys from quantum channel. Based on the **Differential Privacy** paper by Cynthia Dwork (ICALP 2006) – fuzzy extractors have application in the **proof of impossibility** of strong notions of privacy for statistical databases.

### 41.2 Basic definitions

#### 41.2.1 Predictability

Predictability indicates probability that adversary can guess a secret key. Mathematically speaking, the predictability of a random variable  $A$  is  $\max_a P[A = a]$ . For example, if pair of random variable  $A$  and  $B$ , if the adversary knows  $b$  of  $B$ , then predictability of  $A$  will be  $\max_a P[A = a|B = b]$ . So, Adversary can predict  $A$  with  $E_{b \leftarrow B}[\max_a P[A = a|B = b]]$ . Taking average over  $B$  as it is not under adversary control, but since knowing  $b$  makes  $A$  prediction adversarial, taking the worst case over  $A$ .

#### 41.2.2 Min-entropy

**Min-entropy** indicates worst-case entropy. Mathematically speaking, it is defined as  $H_\infty(A) = -\log(\max_a P[A = a])$ . Random variables with min-entropy at least  $m$  is called  $m$ -source.

#### 41.2.3 Statistical distance

**Statistical distance** is measure of distinguishability. Mathematically speaking, it is between two probability distributions  $A$  and  $B$  is,  $SD[A, B] = \frac{1}{2} \sum_v |P[A =$

$v] - P[B = v]$ . In any system if  $A$  is replaced by  $B$ , it will behave as original system with probability at least  $1 - SD[A, B]$ .

#### 41.2.4 Definition 1 (strong extractor)

Set  $M$  is **strong randomness extractor**. Randomized function  $\text{Ext}: M \rightarrow \{0, 1\}^l$  with randomness of length  $r$  is an  $(m, l, \epsilon)$ -strong extractor if for all  $m$ -sources (Random variables with min-entropy at least  $m$  is called  $m$ -source)  $W$  on  $M(\text{Ext}(W; I), I) \approx_\epsilon (U_l, U_r)$ , where  $I = U_r$  is independent of  $W$ . Output of the extractor is a key generated from  $w \leftarrow W$  with the seed  $i \leftarrow I$ . It behaves independent of other parts of the system with the probability of  $1 - \epsilon$ . Strong extractors can extract at most  $l = m - 2\log \frac{1}{\epsilon} + O(1)$  bits from arbitrary  $m$ -source.

#### 41.2.5 Secure sketch

Secure sketch makes it possible to reconstruct noisy input, so if the input is  $w$  and sketch is  $s$ , given  $s$  and value  $w'$  close to  $w$ , it is possible to recover  $w$ . But sketch  $s$  doesn't give much information about  $w$ , so it is secure. If  $\mathbb{M}$  is a metric space with distance function  $\text{dis}$ . Secure sketch recovers string  $w \in \mathbb{M}$  from any close string  $w' \in \mathbb{M}$  without disclosing  $w$ .

#### 41.2.6 Definition 2 (secure sketch)

An  $(m, \tilde{m}, t)$  secure sketch is a pair of efficient randomized procedures (SS – Sketch, Rec – Recover) such that – (1) The sketching procedure SS on input  $w \in \mathbb{M}$  returns a string  $s \in \{0, 1\}^*$ . The recovery procedure Rec takes an element  $w' \in \mathbb{M}$  and  $s \in \{0, 1\}^*$ . (2) Correctness: If  $\text{dis}(w, w') \leq t$  then  $\text{Rec}(w', \text{SS}(w)) = w$ . (3) Security: For any  $m$ -source over  $M$ , the min-entropy of  $W$  given  $s$  is high: for any  $(W, E)$ , if  $\tilde{H}_\infty(W|E) \geq m$ , then  $\tilde{H}_\infty(W|\text{SS}(W), E) \geq \tilde{m}$ .

#### 41.2.7 Fuzzy extractor

Fuzzy extractors do not recover the original input but generate string  $R$  (which is close to uniform) from  $w$  and its subsequent reproduction (using helper string  $P$ ) given any  $w'$  close to  $w$ . Strong extractors are a special case of fuzzy extractors when  $t = 0$  and  $P = I$ .

#### 41.2.8 Definition 3 (fuzzy extractor)

An  $(m, l, t, \epsilon)$  fuzzy extractor is a pair of efficient randomized procedures (Gen – Generate and Rep – Reproduce) such that: (1) Gen, given  $w \in \mathbb{M}$ , outputs an extracted string  $R \in \{0, 1\}^l$  and a helper string  $P \in \{0, 1\}^*$ . (2) Correctness: If  $\text{dis}(w, w') \leq t$  and

$(R, P) \leftarrow \text{Gen}(w)$ , then  $\text{Rep}(w', P) = R$ . (3) Security: For all  $m$ -sources  $W$  over  $M$ , the string  $R$  is nearly uniform even given  $P$ , So  $\tilde{H}_\infty(W|E) \geq m$ , then  $(R, P, E) \approx (U_l, P, E)$ .

So Fuzzy extractors output almost uniform random bits which is prerequisite for using cryptographic applications (in terms of secret keys). Since output bits are slightly non-uniform, it can decrease security, but not more than the distance  $\epsilon$  from the uniform and as long as that distance is sufficiently small – security still remains robust.

#### 41.2.9 Secure sketches and fuzzy extractors

Secure sketches can be used to construct fuzzy extractors. Like applying SS to  $w$  to obtain  $s$  and strong extractor Ext with randomness  $x$  to  $w$  to get  $R$ .  $(s, x)$  can be stored as helper string  $P$ .  $R$  can be reproduced by  $w'$  and  $P = (s, x)$ .  $\text{Rec}(w', s)$  can recover  $w$  and  $\text{Ext}(w, x)$  can reproduce  $R$ . Following Lemma formalize this.

#### 41.2.10 Lemma 1 (fuzzy extractors from sketches)

Assume (SS, Rec) is an  $(M, m, \tilde{m}, t)$  secure sketch and let Ext be an average-case  $(n, \tilde{m}, l, \epsilon)$  strong extractor. Then the following (Gen, Rep) is an  $(M, m, l, t, \epsilon)$  fuzzy extractor: (1)  $\text{Gen}(w, r, x) : \text{set } P = (\text{SS}(w; r), x), R = \text{Ext}(w; x)$ , and output  $(R, P)$ . (2)  $\text{Rep}(w', (s, x)) : \text{recover } w = \text{Rec}(w', s)$  and output  $R = \text{Ext}(w; x)$ .

Proof: From the definition of secure sketch (Definition 2),  $H_\infty(W|\text{SS}(W)) \geq \tilde{m}$ . And since Ext is an average-case  $(n, m, l, \epsilon)$ -strong extractor.  $SD((\text{Ext}(W; X), \text{SS}(W), X), (U_l, \text{SS}(W), X)) = SD((R, P), (U_l, P)) \leq \epsilon$ .

#### 41.2.11 Corollary 1

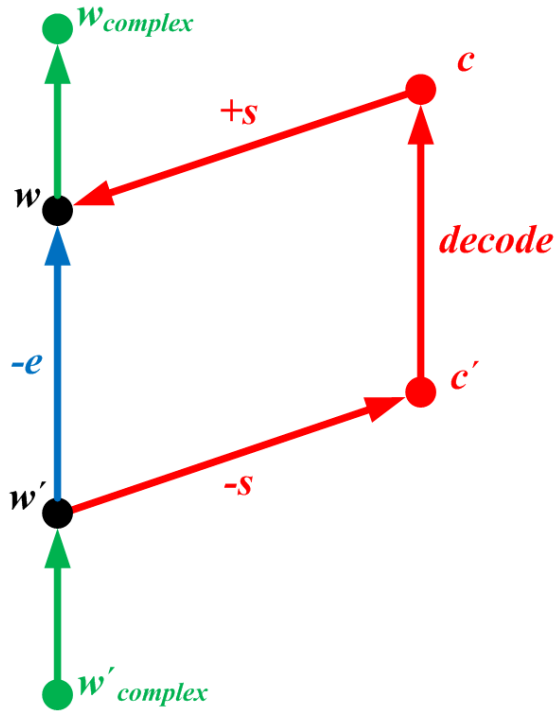
If (SS, Rec) is an  $(M, m, \tilde{m}, t)$  – secure sketch and Ext is an  $(n, \tilde{m} - \log(\frac{1}{\delta}), l, \epsilon)$  – strong extractor, then the above construction (Gen, Rep) is a  $(M, m, l, t, \epsilon + \delta)$  fuzzy extractor.

Reference paper “Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data” by Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin and Adam Smith (2008) includes many generic combinatorial bounds on secure sketches and fuzzy extractors

### 41.3 Basic constructions

Due to their error tolerant properties, a secure sketches can be treated, analyzed, and constructed like a

$(n, k, d)_{\mathcal{F}}$  general **error correcting code** or  $[n, k, d]_{\mathcal{F}}$  for **linear codes**, where  $n$  is the length of codewords,  $k$  is the length of the message to be coded,  $d$  is the distance between codewords, and  $\mathcal{F}$  is the alphabet. If  $\mathcal{F}^n$  is the universe of possible words then it may be possible to find an error correcting code  $C \in \mathcal{F}^n$  that has a unique codeword  $c \in C$  for every  $w \in \mathcal{F}^n$  and have a **Hamming distance** of  $dis_{Ham}(c, w) \leq (d-1)/2$ . The first step for constructing a secure sketch is determining the type of errors that will likely occur and then choosing a distance to measure.



Red is the code-offset construction, blue is the syndrome construction, green represents edit distance and other complex constructions.

### 41.3.1 Hamming distance constructions

When there is no chance of data being deleted and only being corrupted then the best measurement to use for error correction is Hamming distance. There are two common constructions for correcting Hamming errors depending on whether the code is linear or not. Both constructions start with an error correcting code that has a distance of  $2t+1$  where  $t$  is the number of tolerated errors.

#### Code-offset construction

When using a  $(n, k, 2t+1)_{\mathcal{F}}$  general code, assign a uniformly random codeword  $c \in C$  to each  $w$ , then let  $SS(w) = s = w - c$  which is the shift needed to change  $c$  into  $w$ . To fix errors in  $w'$  subtract  $s$  from

$w'$  then correct the errors in the resulting incorrect codeword to get  $c$  and finally add  $s$  to  $c$  to get  $w$ . This means  $Rec(w', s) = s + dec(w' - s) = w$ . This construction can achieve the best possible tradeoff between error tolerance and entropy loss when  $\mathcal{F} \geq n$  and a **Reed-Solomon code** is used resulting in an entropy loss of  $2t \log(\mathcal{F})$ , and the only way to improve upon this is to find a code better than Reed-Solomon.

#### Syndrome construction

When using a  $[n, k, 2t+1]_{\mathcal{F}}$  linear code let the  $SS(w) = s$  be the **syndrome** of  $w$ . To correct  $w'$  find a vector  $e$  such that  $syn(e) = syn(w') - s$ , then  $w = w' - e$ .

### 41.3.2 Set difference constructions

When working with a very large alphabet or very long strings resulting in a very large universe  $\mathcal{U}$ , it may be more efficient to treat  $w$  and  $w'$  as sets and look at **set differences** to correct errors. To work with a large set  $w$  it is useful to look at its characteristic vector  $x_w$ , which is a binary vector of length  $n$  that has a value of 1 when an element  $a \in \mathcal{U}$  and  $a \in w$ , or 0 when  $a \notin w$ . The best way to decrease the size of a secure sketch when  $n$  is large is make  $k$  large since the size is determined by  $n - k$ . A good code to base this construction on is a  $[n, n - t\alpha, 2t+1]_2$  **BCH code** where  $n = 2^\alpha - 1$  and  $t \ll n$  so  $k \leq n - \log \binom{n}{t}$ , it is also useful that BCH codes can be decode in sub-linear time.

#### Pin sketch construction

Let  $SS(w) = s = syn(x_w)$ . To correct  $w'$  first find  $SS(w') = s' = syn(x_{w'})$ , then find a set  $v$  where  $syn(x_v) = s' - s$ , finally compute the **symmetric difference** to get  $Rec(w', s) = w' \triangle v = w$ . While this is not the only construction to use set difference it is the easiest one to use.

### 41.3.3 Edit distance constructions

When data can be corrupted or deleted the best measurement to use is **edit distance**. To make a construction based on edit distance it is easiest to start with a construction for set difference or hamming distance as an intermediate correction step and then build the edit distance construction around that.

### 41.3.4 Other distance measure constructions

There are many other types of errors and distances that can be measured which can be used to model other situations. Most of these other possible constructions are like

edit distance constructions where they build upon simpler constructions.

## 41.4 Improving error-tolerance via relaxed notions of correctness

It is possible to show that the error-tolerance of a secure sketch can be improved by applying a **probabilistic method** to error correction and only needing errors to be correctable with a high probability. This will show that it is possible to exceed the **Plotkin bound** which is limited to correcting  $n/4$  errors, and approach **Shannon's bound** allowing for nearly  $n/2$  corrections. To achieve this better error correction a less restrictive error distribution model must be used.

### 41.4.1 Random errors

For this most restrictive model use a **BSC**  $p$  to create a  $w'$  that a probability  $p$  at each position in  $w'$  that the bit received is wrong. This model can show that entropy loss is limited to  $nH(p) - o(n)$ , where  $H$  is the **binary entropy function**, and if min-entropy  $m \geq n(H(\frac{1}{2} - \gamma)) + \varepsilon$  then  $n(\frac{1}{2} - \gamma)$  errors can be tolerated, for some constant  $\gamma > 0$ .

### 41.4.2 Input-dependent errors

For this model errors do not have a known distribution and can be from an adversary, the only constraints are  $dis_{err} \leq t$  and that a corrupted word depends only on the input  $w$  and not on the secure sketch. It can be shown for this error model that there will never be more than  $t$  errors since this model can account for all complex noise processes, meaning that Shannon's bound can be reached, to do this a random permutation is prepended to the secure sketch that will reduce entropy loss.

### 41.4.3 Computationally bounded errors

This differs from the input dependent model by having errors that depend on both the input  $w$  and the secure sketch, and an adversary is limited to polynomial time algorithms for introducing errors. Since algorithms that can run in better than polynomial time are not currently feasible in the real world, then a positive result using this error model would guarantee that any errors can be fixed. This is the least restrictive model the only known way to approach Shannon's bound is to use **list-decodable codes** although this may not always be useful in practice since returning a list instead of a single codeword may not always be acceptable.

## 41.5 Privacy guarantees

In general a secure system attempts to leak as little information as possible to an **adversary**. In the case of biometrics if information about the biometric reading is leaked the adversary may be able to learn personal information about a user. For example an adversary notices that there is a certain pattern in the helper strings that implies the ethnicity of the user. We can consider this additional information a function  $f(W)$ . If an adversary were to learn a helper string, it must be ensured that, from this data he can not infer any data about the person from which the biometric reading was taken.

### 41.5.1 Correlation between helper string and biometric input

Ideally the helper string  $P$  would reveal no information about the biometric input  $w$ . This is only possible when every subsequent biometric reading  $w'$  is identical to the original  $w$ . In this case there is actually no need for the helper string, so it is easy to generate a string that is in no way correlated to  $w$ .

Since it is desirable to accept biometric input  $w'$  similar to  $w$  the helper string  $P$  must be somehow correlated. The more different  $w$  and  $w'$  are allowed to be, the more correlation there will be between  $P$  and  $w$ , the more correlated they are the more information  $P$  reveals about  $w$ . We can consider this information to be a function  $f(W)$ . The best possible solution is to make sure the adversary can't learn anything useful from the helper string.

### 41.5.2 Gen(W) as a probabilistic map

A probabilistic map  $Y()$  hides the results of functions with a small amount of leakage  $\epsilon$ . The leakage is the difference in probability two adversaries have of guessing some function when one knows the probabilistic map and one does not. Formally:

$$|Pr[A_1(Y(W)) = f(W)] - Pr[A_2() = f(W)]| \leq \epsilon$$

If the function  $Gen(W)$  is a probabilistic map, then even if an adversary knows both the helper string  $P$  and the secret string  $R$  they are only negligibly more likely figure something out about the subject as if they knew nothing. The string  $R$  is supposed to be kept secret, so even if it is leaked (which should be very unlikely) the adversary can still figure out nothing useful about the subject, as long as  $\epsilon$  is small. We can consider  $f(W)$  to be any correlation between the biometric input and some physical characteristic of the person. Setting  $Y = Gen(W) = R, P$  in the above equation changes it to:

$$|Pr[A_1(R, P) = f(W)] - Pr[A_2() = f(W)]| \leq \epsilon$$

This means that if one adversary  $A_1$  has  $(R, P)$  and a second adversary  $A_2$  knows nothing, their best guesses at  $f(W)$  are only  $\epsilon$  apart.

### 41.5.3 Uniform fuzzy extractors

Uniform fuzzy extractors are a special case of fuzzy extractors, where the output  $(R, P)$  of  $Gen(W)$  are negligibly different from strings picked from the uniform distribution, i.e.  $(R, P) \approx_\epsilon (U_\ell, U_{|P|})$

### 41.5.4 Uniform secure sketches

Since secure sketches imply fuzzy extractors, constructing a uniform secure sketch allows for the easy construction of a uniform fuzzy extractor. In a uniform secure sketch the sketch procedure  $SS(w)$  is a **randomness extractor**  $Ext(w; i)$ . Where  $w$  is the biometric input and  $i$  is the **random seed**. Since randomness extractors output a string that appears to be from a uniform distribution they hide all the information about their input.

### 41.5.5 Applications

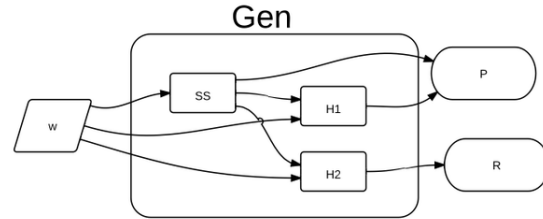
Extractor sketches can be used to construct  $(m, t, \epsilon)$  - fuzzy perfectly one-way hash functions. When used as a hash function the input  $w$  is the object you want to hash. The  $P, R$  that  $Gen(w)$  outputs is the hash value. If one wanted to verify that a  $w'$  within  $t$  from the original  $w$ , they would verify that  $Rep(w', P) = R$ .  $(m, t, \epsilon)$  - fuzzy perfectly one-way hash functions are special **hash functions** where they accept any input with at most  $t$  errors, compared to traditional hash functions which only accept when the input matches the original exactly. Traditional cryptographic hash functions attempt to guarantee that it is computationally infeasible to find two different inputs that hash to the same value. Fuzzy perfectly one-way hash functions make an analogous claim. They make it computationally infeasible to find two inputs, that are more than  $t$  **Hamming distance** apart and hash to the same value.

## 41.6 Protection against active attacks

An active attack could be one where the adversary can modify the helper string  $P$ . If the adversary is able to change  $P$  to another string that is also acceptable to the reproduce function  $Rep(W, P)$ , it cause  $Rep(W, P)$  to output an incorrect secret string  $\tilde{R}$ . Robust fuzzy extractors solve this problem by allowing the reproduce function to fail, if a modified helper string is provided as input.

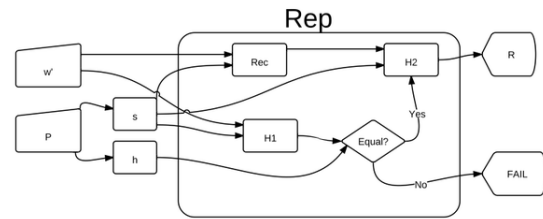
### 41.6.1 Robust fuzzy extractors

One method of constructing robust fuzzy extractors is to use **hash functions**. This construction requires two hash functions  $H_1$  and  $H_2$ . The  $Gen(W)$  functions produces the helper string  $P$  by appending the output of a secure sketch  $s = SS(w)$  to the hash of both the reading  $w$  and secure sketch  $s$ . It generates the secret string  $R$  by applying the second hash function to  $w$  and  $s$ . Formally:  $Gen(w) : s = SS(w), return : P = (s, H_1(w, s)), R = H_2(w, s)$



The reproduce function  $Rep(W, P)$  also makes use of the hash functions  $H_1$  and  $H_2$ . In addition to verifying the biometric input is similar enough to the one recovered using the  $Rec(W, S)$  function, it also verifies that hash in the second part of  $P$  was actually derived from  $w$  and  $s$ . If both of those conditions are met it returns  $R$  which is itself the second hash function applied to  $w$  and  $s$ . Formally:

$Rep(w', \tilde{P})$  : Get  $\tilde{s}$  and  $\tilde{h}$  from  $\tilde{P}$ ;  $\tilde{w} = Rec(w', \tilde{s})$ . If  $\Delta(\tilde{w}, w') \leq t$  and  $\tilde{h} = H_1(\tilde{w}, \tilde{s})$  then return :  $H_2(\tilde{w}, \tilde{s})$  else return : fail



If  $P$  has been tampered with it will be obvious because,  $Rep$  will output fail with very high probability. To cause the algorithm accept a different  $P$  an adversary would have to find a  $\tilde{w}$  such that  $H_1(w, s) = H_1(\tilde{w}, \tilde{s})$ . Since hash function are believed to be **one way functions**, it is computationally infeasible to find such a  $\tilde{w}$ . Seeing  $P$  would provide the adversary with no useful information. Since, again, hash function are one way functions, it is computationally infeasible for the adversary to reverse the hash function and figure out  $w$ . Part of  $P$  is the secure sketch, but by definition the sketch reveals negligible information about its input. Similarly seeing  $R$  (even though it should never see it) would provide the adversary with no useful information as the adversary wouldn't be

able to reverse the hash function and see the biometric input.

## 41.7 References

- Fuzzy Extractors: A Brief Survey of Results from 2004 to 2006
- Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data
- Biometric Fuzzy Extractor Scheme for Iris Templates
- A Fuzzy Vault Scheme



## Chapter 42

# Generalized minimum-distance decoding

In coding theory, **generalized minimum-distance (GMD) decoding** provides an efficient algorithm for decoding concatenated codes, which is based on using an errors-and-erasures decoder for the outer code.

A naive decoding algorithm for concatenated codes can not be an optimal way of decoding because it does not take into account the information that **maximum likelihood decoding (MLD)** gives. In other words, in the naive algorithm, inner received **codewords** are treated the same regardless of the difference between their **hamming distances**. Intuitively, the outer decoder should place higher confidence in symbols whose inner **encodings** are close to the received word. **David Forney** in 1966 devised a better algorithm called generalized minimum distance (GMD) decoding which makes use of those information better. This method is achieved by measuring confidence of each received codeword, and erasing symbols whose confidence is below a desired value. And GMD decoding algorithm was one of the first examples of **soft-decision decoders**. We will present three versions of the GMD decoding algorithm. The first two will be **randomized algorithms** while the last one will be a **deterministic algorithm**.

$(C_{\text{in}}(C_{\text{out}}(m)_1), \dots, C_{\text{in}}(C_{\text{out}}(m)_N))$  where  $C_{\text{out}}(m) = ((C_{\text{out}}(m)_1, \dots, (m)_N))$ . Finally we will take  $C_{\text{out}}$  to be **RS code**, which has an errors and erasure decoder, and  $K = O(\log N)$ , which in turn implies that MLD on the inner code will be polynomial in  $N$  time.

- **Maximum likelihood decoding (MLD)**: MLD is a decoding method for error correcting codes, which outputs the codeword closest to the received word in Hamming distance. The MLD function denoted by  $D_{\text{MLD}} : \Sigma^n \rightarrow C$  is defined as follows. For every  $y \in \Sigma^n$ ,  $D_{\text{MLD}}(y) = \arg \min_{c \in C} \Delta(c, y)$ .
- **Probability density function** : A probability distribution  $\text{Pr}$  on a sample space  $S$  is a mapping from events of  $S$  to **real numbers** such that  $\text{Pr}[A] \geq 0$  for any event  $A$ ,  $\text{Pr}[S] = 1$ , and  $\text{Pr}[A \cup B] = \text{Pr}[A] + \text{Pr}[B]$  for any two mutually exclusive events  $A$  and  $B$
- **Expected value**: The expected value of a discrete random variable  $X$  is

$$\mathbb{E}[X] = \sum_x \text{Pr}[X = x].$$

### 42.1 Setup

- **Hamming distance** : Given two vectors  $u, v \in \Sigma^n$  the Hamming distance between  $u$  and  $v$ , denoted by  $\Delta(u, v)$ , is defined to be the number of positions in which  $u$  and  $v$  differ.
- **Minimum distance**: Let  $C \subseteq \Sigma^n$  be a **code**. The minimum distance of code  $C$  is defined to be  $d = \min \Delta(c_1, c_2)$  where  $c_1 \neq c_2 \in C$
- **Code concatenation**: Given  $m = (m_1, \dots, m_K) \in [Q]^K$ , consider two codes which we call outer code and inner code

$$C_{\text{out}} = [Q]^K \rightarrow [Q]^N, \quad C_{\text{in}} : [q]^k \rightarrow [q]^n,$$

and their distances are  $D$  and  $d$ . A concatenated code can be achieved by  $C_{\text{out}} \circ C_{\text{in}}(m) =$

### 42.2 Randomized algorithm

Consider the received word  $\mathbf{y} = (y_1, \dots, y_N) \in [q^n]^N$  which corrupted by **noisy channel**. The following is the algorithm description for the general case. In this algorithm, we can decode  $\mathbf{y}$  by just declaring an erasure at every bad position and running the errors and erasure decoding algorithm for  $C_{\text{out}}$  on the resulting vector.

#### Randomized\_Decoder

**Given** :  $\mathbf{y} = (y_1, \dots, y_N) \in [q^n]^N$ .

1. For every  $1 \leq i \leq N$ , compute  $y'_i = \text{MLD}_{C_{\text{in}}}(y_i)$ .
2. Set  $\omega_i = \min(\Delta(C_{\text{in}}(y'_i), y_i), \frac{d}{2})$ .



3. For every  $1 \leq i \leq N$ , repeat : With probability  $\frac{2\omega_i}{d}$ , set  $y_i'' \leftarrow ?$ , otherwise set  $y_i'' = y_i'$ .
4. Run errors and erasure algorithm for  $C_{\text{out}}$  on  $\mathbf{y}'' = (y_1'', \dots, y_N'')$ .

**Theorem 1.** Let  $\mathbf{y}$  be a received word such that there exists a codeword  $\mathbf{c} = (c_1, \dots, c_N) \in C_{\text{out}} \circ C_{\text{in}} \subseteq [q^n]^N$  such that  $\Delta(\mathbf{c}, \mathbf{y}) < \frac{Dd}{2}$ . Then the deterministic GMD algorithm outputs  $\mathbf{c}$ .

Note that a naive decoding algorithm for concatenated codes can correct up to  $\frac{Dd}{4}$  errors.

**Lemma 1.** Let the assumption in Theorem 1 hold. And if  $\mathbf{y}''$  has  $e'$  errors and  $s'$  erasures (when compared with  $\mathbf{c}$ ) after **Step 1**, then  $\mathbb{E}[2e' + s'] < D$ .

*Remark.* If  $2e' + s' < D$ , then the algorithm in **Step 2** will output  $\mathbf{c}$ . The lemma above says that in expectation, this is indeed the case. Note that this is not enough to prove **Theorem 1**, but can be crucial in developing future variations of the algorithm.

**Proof of lemma 1.** For every  $1 \leq i \leq N$ , define  $e_i = \Delta(y_i, c_i)$ . This implies that

$$\sum_{i=1}^N e_i < \frac{Dd}{2} \quad (1)$$

Next for every  $1 \leq i \leq N$ , we define two indicator variables:

$$\begin{aligned} X_i^? &= 1 \Leftrightarrow y_i'' = ? \\ X_i^e &= 1 \Leftrightarrow C_{\text{in}}(y_i'') \neq c_i \text{ and } y_i'' \neq ? \end{aligned}$$

We claim that we are done if we can show that for every  $1 \leq i \leq N$ :

$$\mathbb{E}[2X_i^e + X_i^?] \leq \frac{2e_i}{d} \quad (2)$$

Clearly, by definition

$$e' = \sum_i X_i^e \quad \text{and} \quad s' = \sum_i X_i^?.$$

Further, by the linearity of expectation, we get

$$\mathbb{E}[2e' + s'] \leq \frac{2}{d} \sum_i e_i < D.$$

To prove (2) we consider two cases:  $i$ -th block is correctly decoded (**Case 1**),  $i$ -th block is incorrectly decoded (**Case 2**):

**Case 1:** ( $c_i = C_{\text{in}}(y_i')$ )

Note that if  $y_i'' = ?$  then  $X_i^e = 0$ , and  $\Pr[y_i'' = ?] = \frac{2\omega_i}{d}$  implies  $\mathbb{E}[X_i^?] = \Pr[X_i^? = 1] = \frac{2\omega_i}{d}$ , and  $\mathbb{E}[X_i^e] = \Pr[X_i^e = 1] = 0$ .

Further, by definition we have

$$\omega_i = \min(\Delta(C_{\text{in}}(y_i'), y_i), \frac{d}{2}) \leq \Delta(C_{\text{in}}(y_i'), y_i) = \Delta(c_i, y_i) = e_i$$

**Case 2:** ( $c_i \neq C_{\text{in}}(y_i')$ )

In this case,  $\mathbb{E}[X_i^?] = \frac{2\omega_i}{d}$  and  $\mathbb{E}[X_i^e] = \Pr[X_i^e = 1] = 1 - \frac{2\omega_i}{d}$ .

Since  $c_i \neq C_{\text{in}}(y_i')$ ,  $e_i + \omega_i \geq d$ . This follows another case analysis when ( $\omega_i = \Delta(C_{\text{in}}(y_i'), y_i) < \frac{d}{2}$ ) or not.

Finally, this implies

$$\mathbb{E}[2X_i^e + X_i^?] = 2 - \frac{2\omega_i}{d} \leq \frac{2e_i}{d}.$$

In the following sections, we will finally show that the deterministic version of the algorithm above can do unique decoding of  $C_{\text{out}} \circ C_{\text{in}}$  up to half its design distance.

## 42.3 Modified randomized algorithm

Note that, in the previous version of the GMD algorithm in step “3”, we do not really need to use “fresh” randomness for each  $i$ . Now we come up with another randomized version of the GMD algorithm that uses the same randomness for every  $i$ . This idea follows the algorithm below.

### Modified\_Randomized\_Decoder

**Given :**  $\mathbf{y} = (y_1, \dots, y_N) \in [q^n]^N$ , pick  $\theta \in [0, 1]$  at random. Then every for every  $1 \leq i \leq N$ :

1. Set  $y_i' = \text{MLD}_{C_{\text{in}}}(y_i)$ .
2. Compute  $\omega_i = \min(\Delta(C_{\text{in}}(y_i'), y_i), \frac{d}{2})$ .
3. If  $\theta < \frac{2\omega_i}{d}$ , set  $y_i'' \leftarrow ?$ , otherwise set  $y_i'' = y_i'$ .
4. Run errors and erasure algorithm for  $C_{\text{out}}$  on  $\mathbf{y}'' = (y_1'', \dots, y_N'')$ .

For the proof of **Lemma 1**, we only use the randomness to show that

$$\Pr[y_i'' = ?] = \frac{2\omega_i}{d}.$$

In this version of the GMD algorithm, we note that

$$\Pr[y_i'' = ?] = \Pr[\theta \in [0, \frac{2\omega_i}{d}]] = \frac{2\omega_i}{d}.$$

The second equality above follows from the choice of  $\theta$ . The proof of **Lemma 1** can be also used to show  $\mathbb{E}[2e' + s'] < D$  for version2 of GMD. In the next section, we will see how to get a deterministic version of the GMD algorithm by choosing  $\theta$  from a polynomially sized set as opposed to the current infinite set  $[0, 1]$ .

## 42.4 Deterministic algorithm

Let  $Q = \{0, 1\} \cup \{\frac{2\omega_1}{d}, \dots, \frac{2\omega_N}{d}\}$ . Since for each  $i$ ,  $\omega_i = \min(\Delta(\mathbf{y}'_i, \mathbf{y}_i), \frac{d}{2})$ , we have

$$Q = \{0, 1\} \cup \{q_1, \dots, q_m\}$$

where  $q_1 < \dots < q_m$  for some  $m \leq \lfloor \frac{d}{2} \rfloor$ . Note that for every  $\theta \in [q_i, q_{i+1}]$ , the step 1 of the second version of randomized algorithm outputs the same  $\mathbf{y}''$ . Thus, we need to consider all possible value of  $\theta \in Q$ . This gives the deterministic algorithm below.

### Deterministic Decoder

**Given :**  $\mathbf{y} = (y_1, \dots, y_N) \in [q^n]^N$ , for every  $\theta \in Q$ , repeat the following.

1. Compute  $y'_i = MLD_{C_{in}}(y_i)$  for  $1 \leq i \leq N$ .
2. Set  $\omega_i = \min(\Delta(C_{in}(y'_i), y_i), \frac{d}{2})$  for every  $1 \leq i \leq N$ .
3. If  $\theta < \frac{2\omega_i}{d}$ , set  $y''_i \leftarrow ?$ , otherwise set  $y''_i = y'_i$ .
4. Run errors-and-erasures algorithm for  $C_{out}$  on  $\mathbf{y}'' = (y''_1, \dots, y''_N)$ . Let  $c_\theta$  be the codeword in  $C_{out} \circ C_{in}$  corresponding to the output of the algorithm, if any.
5. Among all the  $c_\theta$  output in 4, output the one closest to  $\mathbf{y}$

Every loop of 1~4 can be run in polynomial time, the algorithm above can also be computed in polynomial time. Specifically, each call to an errors and erasures decoder of  $< dD/2$  errors takes  $O(d)$  time. Finally, the runtime of the algorithm above is  $O(NQn^{O(1)} + NT_{out})$  where  $T_{out}$  is the running time of the outer errors and erasures decoder.

## 42.5 See also

1. Concatenated codes
2. Reed Solomon error correction
3. Welch Berlekamp algorithm

## 42.6 References

1. University at Buffalo Lecture Notes on Coding Theory – Atri Rudra
2. MIT Lecture Notes on Essential Coding Theory – Madhu Sudan
3. University of Washington – Venkatesan Guruswami
4. G. David Forney. Generalized Minimum Distance decoding. *IEEE Transactions on Information Theory*, 12:125–131, 1966

# Chapter 43

## Generator matrix

For generator matrices in probability theory, see [transition rate matrix](#).

In coding theory, a **generator matrix** is a matrix whose rows form a basis for a linear code. The codewords are all of the linear combinations of the rows of this matrix, that is, the linear code is the row space of its generator matrix.

### 43.1 Terminology

If  $\mathbf{G}$  is a matrix, it generates the [codewords](#) of a linear code  $C$  by,

$$\mathbf{w} = \mathbf{s} \mathbf{G},$$

where  $\mathbf{w}$  is a codeword of the linear code  $C$ , and  $\mathbf{s}$  is any input vector. Both  $\mathbf{w}$  and  $\mathbf{s}$  are assumed to be row vectors.<sup>[1]</sup> A generator matrix for a linear  $[n, k, d]_q$ -code has format  $k \times n$ , where  $n$  is the length of a codeword,  $k$  is the number of information bits (the dimension of  $C$  as a vector subspace),  $d$  is the minimum distance of the code, and  $q$  is size of the [finite field](#), that is, the number of symbols in the alphabet (thus,  $q = 2$  indicates a [binary code](#), etc.). The number of [redundant bits](#) is denoted by  $r = n - k$ .

The *standard* form for a generator matrix is,<sup>[2]</sup>

$$G = [I_k | P]$$

where  $I_k$  is the  $k \times k$  [identity matrix](#) and  $P$  is a  $k \times r$  matrix. When the generator matrix is in standard form, the code  $C$  is [systematic](#) in its first  $k$  coordinate positions.<sup>[3]</sup>

A generator matrix can be used to construct the [parity check matrix](#) for a code (and vice versa). If the generator matrix  $G$  is in standard form,  $G = [I_k | P]$ , then the parity check matrix for  $C$  is<sup>[4]</sup>

$$H = [-P^T | I_{n-k}]$$

where  $P^T$  is the [transpose](#) of the matrix  $P$ . This is a consequence of the fact that a parity check matrix of  $C$  is a generator matrix of the [dual code](#)  $C^\perp$ .

### 43.2 Equivalent Codes

Codes  $C_1$  and  $C_2$  are *equivalent* (denoted  $C_1 \sim C_2$ ) if one code can be obtained from the other via the following two transformations:<sup>[5]</sup>

1. arbitrarily permute the components, and
2. independently scale by a non-zero element any components.

Equivalent codes have the same minimum distance.

The generator matrices of equivalent codes can be obtained from one another via the following [elementary operations](#):<sup>[6]</sup>

1. permute rows
2. scale rows by a nonzero scalar
3. add rows to other rows
4. permute columns, and
5. scale columns by a nonzero scalar.

Thus, we can perform [Gaussian Elimination](#) on  $G$ . Indeed, this allows us to assume that the generator matrix is in the standard form. More precisely, for any matrix  $G$  we can find a [invertible matrix](#)  $U$  such that  $UG = [I_k | P]$ , where  $G$  and  $[I_k | P]$  generate equivalent codes.

### 43.3 See also

- [Hamming code \(7,4\)](#)

## 43.4 Notes

- [1] MacKay, David, J.C. (2003). *Information Theory, Inference, and Learning Algorithms* (PDF). Cambridge University Press. p. 9. ISBN 9780521642989. Because the Hamming code is a linear code, it can be written compactly in terms of matrices as follows. The transmitted codeword  $\mathbf{t}$  is obtained from the source sequence  $\mathbf{s}$  by a linear operation,

$$\mathbf{t} = \mathbf{G}^T \mathbf{s}$$

where  $\mathbf{G}$  is the *generator matrix* of the code... I have assumed that  $\mathbf{s}$  and  $\mathbf{t}$  are column vectors. If instead they are row vectors, then this equation is replaced by

$$\mathbf{t} = \mathbf{sG}$$

... I find it easier to relate to the right-multiplication (...) than the left-multiplication (...). Many coding theory texts use the left-multiplying conventions (...), however. ...The rows of the generator matrix can be viewed as defining the basis vectors.

- [2] Ling & Xing 2004, p. 52  
 [3] Roman 1992, p. 198  
 [4] Roman 1992, p. 200  
 [5] Pless 1998, p. 8  
 [6] Welsh 1988, pp. 54-55

## 43.5 References

- Ling, San; Xing, Chaoping (2004), *Coding Theory / A First Course*, Cambridge University Press, ISBN 0-521-52923-9
- Pless, Vera (1998), *Introduction to the Theory of Error-Correcting Codes* (3rd ed.), Wiley Interscience, ISBN 0-471-19047-0
- Roman, Steven (1992), *Coding and Information Theory*, GTM, **134**, Springer-Verlag, ISBN 0-387-97812-7
- Welsh, Dominic (1988), *Codes and Cryptography*, Oxford University Press, ISBN 0-19-853287-3

## 43.6 Further reading

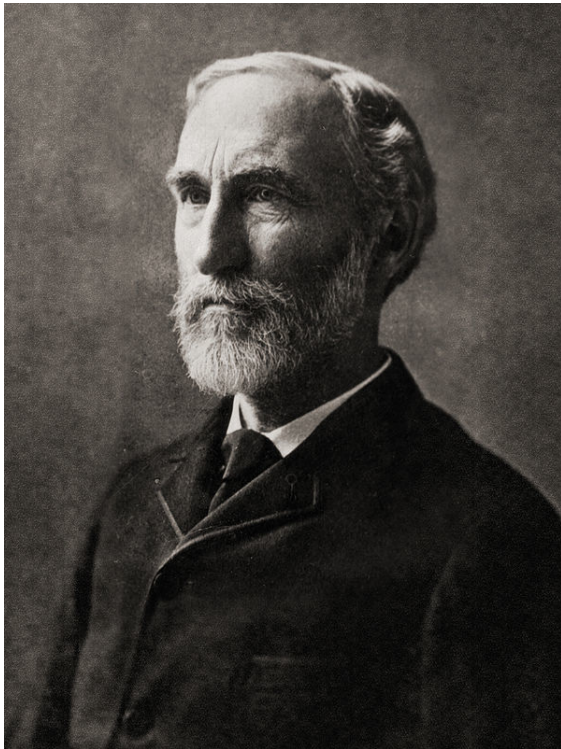
- MacWilliams, F.J.; Sloane, N.J.A. (1977), *The Theory of Error-Correcting Codes*, North-Holland, ISBN 0-444-85193-3

## 43.7 External links

- Generator Matrix at MathWorld

## Chapter 44

# Gibbs' inequality



Josiah Willard Gibbs

In information theory, **Gibbs' inequality** is a statement about the **mathematical entropy** of a discrete **probability distribution**. Several other bounds on the entropy of probability distributions are derived from Gibbs' inequality, including **Fano's inequality**. It was first presented by J. Willard Gibbs in the 19th century.

### 44.1 Gibbs' inequality

Suppose that

$$P = \{p_1, \dots, p_n\}$$

is a probability distribution. Then for any other probability distribution

$$Q = \{q_1, \dots, q_n\}$$

the following inequality between positive quantities (since the  $p_i$  and  $q_i$  are positive numbers less than one) holds<sup>[1]:68</sup>

$$-\sum_{i=1}^n p_i \log_2 p_i \leq -\sum_{i=1}^n p_i \log_2 q_i$$

with equality if and only if

$$p_i = q_i$$

for all  $i$ . Put in words, the **information entropy** of a distribution  $P$  is less than or equal to its **cross entropy** with any other distribution  $Q$ .

The difference between the two quantities is the **Kullback–Leibler divergence** or relative entropy, so the inequality can also be written:<sup>[2]:34</sup>

$$D_{\text{KL}}(P||Q) \equiv \sum_{i=1}^n p_i \log_2 \frac{p_i}{q_i} \geq 0.$$

Note that the use of base-2 logarithms is optional, and allows one to refer to the quantity on each side of the inequality as an “average **surprisal**” measured in **bits**.

### 44.2 Proof

Since

$$\log_2 a = \frac{\ln a}{\ln 2}$$

it is sufficient to prove the statement using the natural logarithm ( $\ln$ ). Note that the natural logarithm satisfies

$$\ln x \leq x - 1$$

for all  $x > 0$  with equality if and only if  $x=1$ .

Let  $I$  denote the set of all  $i$  for which  $p_i$  is non-zero. Then

$$\begin{aligned} -\sum_{i \in I} p_i \ln \frac{q_i}{p_i} &\geq -\sum_{i \in I} p_i \left( \frac{q_i}{p_i} - 1 \right) \\ &= -\sum_{i \in I} q_i + \sum_{i \in I} p_i \\ &\geq 0. \end{aligned}$$

So

$$-\sum_{i \in I} p_i \ln q_i \geq -\sum_{i \in I} p_i \ln p_i$$

and then trivially

$$-\sum_{i=1}^n p_i \ln q_i \geq -\sum_{i=1}^n p_i \ln p_i$$

since the right hand side does not grow, but the left hand side may grow or may stay the same.

For equality to hold, we require:

1.  $\frac{q_i}{p_i} = 1$  for all  $i \in I$  so that the approximation  $\ln \frac{q_i}{p_i} = \frac{q_i}{p_i} - 1$  is exact.
2.  $\sum_{i \in I} q_i = 1$  so that equality continues to hold between the third and fourth lines of the proof.

This can happen if and only if

$$p_i = q_i$$

for  $i = 1, \dots, n$ .

## 44.3 Alternative proofs

The result can alternatively be proved using [Jensen's inequality](#) or [log sum inequality](#).

## 44.4 Corollary

The [entropy](#) of  $P$  is bounded by:<sup>[1]:68</sup>

$$H(p_1, \dots, p_n) \leq \log n.$$

The proof is trivial - simply set  $q_i = 1/n$  for all  $i$ .

## 44.5 See also

- [Information entropy](#)

## 44.6 References

- [1] Pierre Bremaud (6 December 2012). *An Introduction to Probabilistic Modeling*. Springer Science & Business Media. ISBN 978-1-4612-1046-7.
- [2] David J. C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press. ISBN 978-0-521-64298-9.

# Chapter 45

## Gilbert–Varshamov bound

In coding theory, the **Gilbert–Varshamov bound** (due to Edgar Gilbert<sup>[1]</sup> and independently Rom Varshamov<sup>[2]</sup>) is a limit on the parameters of a (not necessarily linear) code. It is occasionally known as the **Gilbert–Shannon–Varshamov bound** (or the **GSV bound**), but the name “Gilbert–Varshamov bound” is by far the most popular. Varshamov proved this bound by using the probabilistic method for linear codes. For more about that proof, see: **GV-linear-code**.

### 45.1 Statement of the bound

Let

$$A_q(n, d)$$

denote the maximum possible size of a  $q$ -ary code  $C$  with length  $n$  and minimum **Hamming weight**  $d$  (a  $q$ -ary code is a code over the field  $\mathbb{F}_q$  of  $q$  elements).

Then:

$$A_q(n, d) \geq \frac{q^n}{\sum_{j=0}^{d-1} \binom{n}{j} (q-1)^j}.$$

### 45.2 Proof

Let  $C$  be a code of length  $n$  and minimum **Hamming distance**  $d$  having maximal size:

$$|C| = A_q(n, d).$$

Then for all  $x \in \mathbb{F}_q^n$ , there exists at least one codeword  $c_x \in C$  such that the Hamming distance  $d(x, c_x)$  between  $x$  and  $c_x$  satisfies

$$d(x, c_x) \leq d-1$$

since otherwise we could add  $x$  to the code whilst maintaining the code’s minimum Hamming distance  $d$  – a contradiction on the maximality of  $|C|$ .

Hence the whole of  $\mathbb{F}_q^n$  is contained in the **union** of all balls of radius  $d-1$  having their **centre** at some  $c \in C$ :

$$\mathbb{F}_q^n = \cup_{c \in C} B(c, d-1).$$

Now each ball has size

$$\sum_{j=0}^{d-1} \binom{n}{j} (q-1)^j$$

since we may allow (or **choose**) up to  $d-1$  of the  $n$  components of a codeword to deviate (from the value of the corresponding component of the ball’s **centre**) to one of  $(q-1)$  possible other values (recall: the code is  $q$ -ary: it takes values in  $\mathbb{F}_q^n$ ). Hence we deduce

$$q^n = |\mathbb{F}_q^n| = \left| \bigcup_{c \in C} B(c, d-1) \right| \leq \sum_{c \in C} |B(c, d-1)| = |C| \sum_{j=0}^{d-1} \binom{n}{j} (q-1)^j$$

That is:

$$A_q(n, d) = |C| \geq \frac{q^n}{\sum_{j=0}^{d-1} \binom{n}{j} (q-1)^j}.$$

### 45.3 An improvement in the prime power case

For  $q$  a prime power, one can improve the bound to  $A_q(n, d) \geq q^k$  where  $k$  is the greatest integer for which

$$q^k < \frac{q^n}{\sum_{j=0}^{d-2} \binom{n-1}{j} (q-1)^j}.$$

### 45.4 See also

- **Singleton bound**



- Hamming bound
- Johnson bound
- Plotkin bound
- Griesmer bound
- Grey–Rankin bound
- Gilbert–Varshamov bound for linear code
- Elias–Bassalygo bound

## 45.5 References

- [1] Gilbert, E. N. (1952), “A comparison of signalling alphabets”, *Bell System Technical Journal*, **31**: 504–522, doi:10.1002/j.1538-7305.1952.tb01393.x.
- [2] Varshamov, R. R. (1957), “Estimate of the number of signals in error correcting codes”, *Dokl. Acad. Nauk SSSR*, **117**: 739–741.

# Chapter 46

## Goppa code

In **mathematics**, an **algebraic geometric code (AG-code)**, otherwise known as a **Goppa code**, is a general type of **linear code** constructed by using an algebraic curve  $X$  over a finite field  $\mathbb{F}_q$ . Such codes were introduced by Valerii Denisovich Goppa. In particular cases, they can have interesting extremal properties. They should not be confused with **Binary Goppa codes** that are used, for instance, in the **McEliece cryptosystem**.

### 46.1 Construction

Traditionally, an AG-code is constructed from a **non-singular projective curve**  $\mathbf{X}$  over a finite field  $\mathbb{F}_q$  by using a number of fixed distinct  $\mathbb{F}_q$ -**rational points** on  $\mathbf{X}$ :

$$\mathcal{P} := \{P_1, \dots, P_n\} \subset \mathbf{X}(\mathbb{F}_q).$$

Let  $G$  be a **divisor** on  $\mathbf{X}$ , with a **support** that consists of only rational points and that is disjoint from the  $P_i$ . Thus  $\mathcal{P} \cap \text{supp}(G) = \emptyset$

By the **Riemann-Roch** theorem, there is a unique finite-dimensional vector space,  $L(G)$ , with respect to the divisor  $G$ . The vector space is a subspace of the **function field** of  $\mathbf{X}$ .

There are two main types of AG-codes that can be constructed using the above information.

### 46.2 Function code

The function code (or dual code) with respect to a curve  $\mathbf{X}$ , a divisor  $G$  and the set  $\mathcal{P}$  is constructed as follows.

Let  $D = P_1 + \dots + P_n$ , be a divisor, with the  $P_i$  defined as above. We usually denote a Goppa code by  $\mathbf{C}(\mathbf{D}, \mathbf{G})$ . We now know all we need to define the Goppa code:

$$C(D, G) = \{(f(P_1), \dots, f(P_n)) : f \in L(G)\} \subset \mathbb{F}_q^n$$

For a fixed basis  $f_1, \dots, f_k$  for  $L(G)$  over  $\mathbb{F}_q$ , the corresponding Goppa code in  $\mathbb{F}_q^n$  is spanned over  $\mathbb{F}_q$  by the vectors

$$(f_i(P_1), \dots, f_i(P_n))$$

Therefore,

$$\begin{bmatrix} f_1(P_1) & \cdots & f_1(P_n) \\ \vdots & & \vdots \\ f_k(P_1) & \cdots & f_k(P_n) \end{bmatrix}$$

is a generator matrix for  $C(D, G)$ .

Equivalently, it is defined as the image of

$$\begin{cases} \alpha : L(G) \rightarrow \mathbb{F}^n \\ f \mapsto (f(P_1), \dots, f(P_n)) \end{cases}$$

The following shows how the parameters of the code relate to classical parameters of **linear systems of divisors**  $D$  on  $C$  (cf. **Riemann-Roch theorem** for more). The notation  $l(D)$  means the dimension of  $L(D)$ .

**Proposition A.** The dimension of the Goppa code  $C(D, G)$  is  $k = l(G) - l(G - D)$ .

**Proof.** Since  $C(D, G) \cong L(G) / \ker(\alpha)$ , we must show that

$$\ker(\alpha) = L(G - D)$$

Let  $f \in \ker(\alpha)$  then  $f(P_1) = \dots = f(P_n) = 0$  so  $\text{div}(f) > D$ . Thus,  $f \in L(G - D)$ . Conversely, suppose  $f \in L(G - D)$ , then  $\text{div}(f) > D$  since

$$P_i < G, \quad i = 1, \dots, n$$

( $G$  doesn't "fix" the problems with the  $-D$ , so  $f$  must do that instead.) It follows that  $f(P_1) = \dots = f(P_n) = 0$ .

**Proposition B.** The minimal distance between two code words is  $d \geq n - \deg(G)$ .

**Proof.** Suppose the **Hamming weight** of  $\alpha(f)$  is  $d$ . That means that for  $n - d$  indices  $i_1, \dots, i_{n-d}$  we have  $f(P_{i_k}) = 0$  for  $k \in \{1, \dots, n - d\}$ . Then  $f \in L(G - P_{i_1} - \dots - P_{i_{n-d}})$ , and

$$\operatorname{div}(f) + G - P_{i_1} - \dots - P_{i_{n-d}} > 0$$

Taking degrees on both sides and noting that

$$\deg(\operatorname{div}(f)) = 0$$

we get

$$\deg(G) - (n - d) \geq 0.$$

so

$$d \geq n - \deg(G).$$

### 46.3 Residue code

The residue code can be defined as the dual of the function code, or as the residue of some functions at the  $P_i$ 's.

### 46.4 References

- Key One Chung, *Goppa Codes*, December 2004, Department of Mathematics, Iowa State University.

### 46.5 External links

- An undergraduate thesis on Algebraic Geometric Coding Theory
- Goppa Codes by Key One Chung

# Chapter 47

## Grammar-based code

**Grammar-based codes** or **Grammar-based compression** are **compression** algorithms based on the idea of constructing a **context-free grammar** (CFG) for the string to be compressed. Examples include universal **lossless data compression** algorithms.<sup>[1]</sup> To compress a data sequence  $x = x_1 \cdots x_n$ , a grammar-based code transforms  $x$  into a context-free grammar  $G$ . The problem of finding a smallest grammar for an input sequence is known to be NP-hard,<sup>[2]</sup> so many grammar-transform algorithms are proposed from theoretical and practical viewpoints. Generally, the produced grammar  $G$  is further compressed by statistical encoders like **arithmetic coding**.

### 47.1 Examples and characteristics

The class of grammar-based codes is very broad. It includes **block codes**, variations of the incremental parsing **Lempel-Ziv code**,<sup>[3]</sup> the multilevel pattern matching (MPM) algorithm,<sup>[4]</sup> and many other new universal lossless compression algorithms. Grammar-based codes are universal in the sense that they can achieve asymptotically the **entropy rate** of any stationary, **ergodic** source with a finite alphabet.

### 47.2 Practical algorithms

The compression programs of the following are available from external links.

- **Sequitur**<sup>[5]</sup> is a classical grammar compression algorithm that sequentially translates an input text into a CFG, and then the produced CFG is encoded by an arithmetic coder.
- **Re-Pair**<sup>[6]</sup> is a greedy algorithm using the strategy of most-frequent-first substitution. The compressive performance is powerful, although the main memory space requirement is very large.

### 47.3 See also

- **Smallest grammar problem**

- **Straight-line grammar**

### 47.4 References

- [1] Kieffer, J. C.; Yang, E.-H. (2000), "Grammar-based codes: A new class of universal lossless source codes", *IEEE Trans. Inform. Theory*, **46** (3): 737–754, doi:10.1109/18.841160
- [2] Charikar, M.; Lehman, E.; Liu, D.; Panigrahy, R.; Prabhakaran, M.; Sahai, A.; Shelat, A. (2005), "The Smallest Grammar Problem", *IEEE Trans. Inform. Theory*, **51** (7): 2554–2576, doi:10.1109/tit.2005.850116
- [3] Kieffer, J. C.; Yang, E.-H.; Nelson, G.; Cosman, P. (2000), "Universal lossless compression via multilevel pattern matching", *IEEE Trans. Inform. Theory*, **46** (4): 1227–1245, doi:10.1109/18.850665
- [4] Ziv, J.; Lempel, A. (1978), "Compression of individual sequences via variable rate coding", *IEEE Trans. Inform. Theory*, **24** (5): 530–536, doi:10.1109/TIT.1978.1055934
- [5] Nevill-Manning, C. G.; Witten, I. H. (1997), "Identifying Hierarchical Structure in Sequences: A linear-time algorithm", *Journal of Artificial Intelligence Research*, **7** (4): 67–82, hdl:10289/1186
- [6] Larsson, N. J.; Moffat, A. (2000), "Offline Dictionary-Based Compression", *IEEE*, **88** (11): 1722–1732, doi:10.1109/5.892708

### 47.5 External links

- **Description of grammar-based codes with example**
- **Sequitur codes**
- **Re-Pair codes**
- **Re-Pair codes** a version of Gonzalo Navarro.
- **GrammarViz 2.0** - implementation of Sequitur, Re-Pair, and parallel Re-Pair in Java.

## Chapter 48

# Gray isometry

The **reflected binary code (RBC)**, also known as **Gray code** after Frank Gray, is a binary numeral system where two successive values differ in only one bit (binary digit). The reflected binary code was originally designed to prevent spurious output from **electromechanical switches**. Today, Gray codes are widely used to facilitate error correction in digital communications such as **digital terrestrial television** and some **cable TV** systems.

### 48.1 Name

received signal.

The binary code with which the present invention deals may take various forms, all of which have the property that the symbol (or pulse group) representing each number (or signal amplitude) differs from the ones representing the next lower and the next higher number (or signal amplitude) in only one digit (or pulse position). Because this code in its primary form may be built up from the conventional binary code by a sort of reflection process and because other forms may in turn be built up from the primary form in similar fashion, the code in question, which has as yet no recognized name, is designated in this specification and in the claims as the “reflected binary code.”

If, at a receiver station, reflected binary code

*Gray’s patent introduces the term “reflected binary code”*

Bell Labs researcher Frank Gray introduced the term *reflected binary code* in his 1947 patent application, remarking that the code had “as yet no recognized name”.<sup>[1]</sup> He derived the name from the fact that it “may be built up from the conventional binary code by a sort of reflection process”.

The code was later named after Gray by others who used it. Two different 1953 patent applications use “Gray code” as an alternative name for the “reflected binary code”;<sup>[2][3]</sup> one of those also lists “minimum error code” and “cyclic permutation code” among the names.<sup>[3]</sup> A 1954 patent application refers to “the Bell Telephone Gray code”.<sup>[4]</sup>

### 48.2 Motivation

Many devices indicate position by closing and opening switches. If that device uses **natural binary codes**, positions 3 and 4 are next to each other but all three bits of the binary representation differ:

The problem with **natural binary codes** is that physical switches are not ideal: it is very unlikely that physical switches will change states exactly in synchrony. In the transition between the two states shown above, all three switches change state. In the brief period while all are changing, the switches will read some spurious position. Even without **keybounce**, the transition might look like 011 — 001 — 101 — 100. When the switches appear to be in position 001, the observer cannot tell if that is the “real” position 001, or a transitional state between two other positions. If the output feeds into a **sequential system**, possibly via **combinational logic**, then the sequential system may store a false value.

The reflected binary code solves this problem by changing only one switch at a time, so there is never any ambiguity of position,

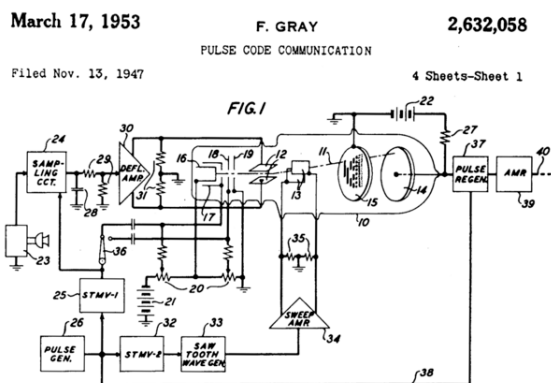
Notice that the Gray code for decimal 7 rolls over to decimal 0 with only one switch change. This is called the “cyclic” property of a Gray code. In the standard Gray coding the least significant bit follows a repetitive pattern of 2 on, 2 off ( ... 11001100 ... ); the next digit a pattern of 4 on, 4 off; and so forth.

More formally, a **Gray code** is a code assigning to each of a contiguous set of **integers**, or to each member of a circular list, a word of symbols such that each two adjacent code words differ by one symbol. These codes are also known as *single-distance codes*, reflecting the **Hamming distance** of 1 between adjacent codes. There can be more than one Gray code for a given word length, but the term was first applied to a particular **binary code** for the non-negative integers, the *binary-reflected Gray code*, or **BRGC**, the three-bit version of which is shown above.

### 48.3 History and practical application

Reflected binary codes were applied to mathematical puzzles before they became known to engineers. **Martin Gardner** wrote a popular account of the Gray code in his August 1972 *Mathematical Games* column in *Scientific American*. The French engineer **Émile Baudot** used Gray codes in *telegraphy* in 1878.<sup>[5]</sup> He received the French **Legion of Honor** medal for his work. The Gray code is sometimes attributed, incorrectly,<sup>[6]</sup> to **Elisha Gray** (in *Principles of Pulse Code Modulation*, K. W. Cattermole,<sup>[7]</sup> for example).

**Frank Gray**, who became famous for inventing the signaling method that came to be used for compatible color television, invented a method to convert analog signals to reflected binary code groups using vacuum tube-based apparatus. The method and apparatus were patented in 1953 and the name of Gray stuck to the codes. The "PCM tube" apparatus that Gray patented was made by Raymond W. Sears of Bell Labs, working with Gray and William M. Goodall, who credited Gray for the idea of the reflected binary code.<sup>[8]</sup>

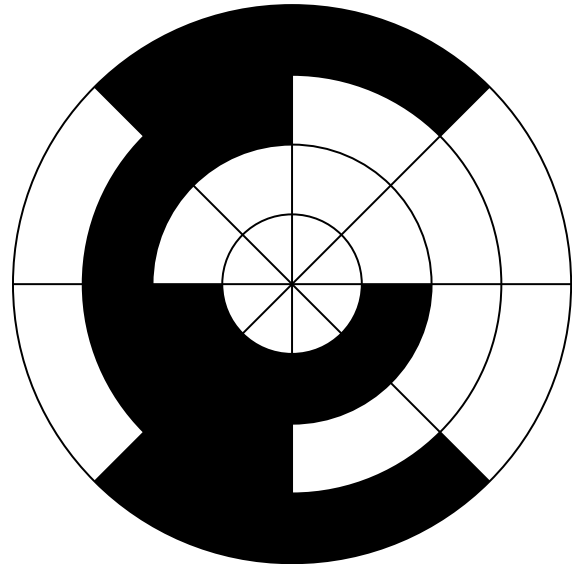


Part of front page of Gray's patent, showing PCM tube (10) with reflected binary code in plate (15)

The use of his eponymous codes that Gray was most interested in was to minimize the effect of error in the conversion of analog signals to digital; his codes are still used today for this purpose, and others.

#### 48.3.1 Position encoders

Gray codes are used in position encoders (**linear encoders** and **rotary encoders**), in preference to straightforward binary encoding. This avoids the possibility that, when several bits change in the binary representation of an angle, a misread will result from some of the bits changing before others. Originally, the code pattern was electrically conductive, supported (in a rotary encoder) by an insulating disk. Each track had its own stationary metal spring contact; one more contact made the connection to the pat-



Rotary encoder for angle-measuring devices marked in 3-bit binary-reflected Gray code (BRGC)

tern. That common contact was connected by the pattern to whichever of the track contacts were resting on the conductive pattern. However, sliding contacts wear out and need maintenance, so non-contact detectors, such as optical or magnetic sensors, are often used instead.

Regardless of the care in aligning the contacts, and accuracy of the pattern, a natural-binary code would have errors at specific disk positions, because it is impossible to make all bits change at exactly the same time as the disk rotates. The same is true of an optical encoder; transitions between opaque and transparent cannot be made to happen simultaneously for certain exact positions. Rotary encoders benefit from the cyclic nature of Gray codes, because consecutive positions of the sequence differ by only one bit. This means that, for a transition from state A to state B, timing mismatches can only affect when the A → B transition occurs, rather than inserting one or more (up to  $N - 1$  for an  $N$ -bit codeword) false intermediate states, as would occur if a standard binary code were used.

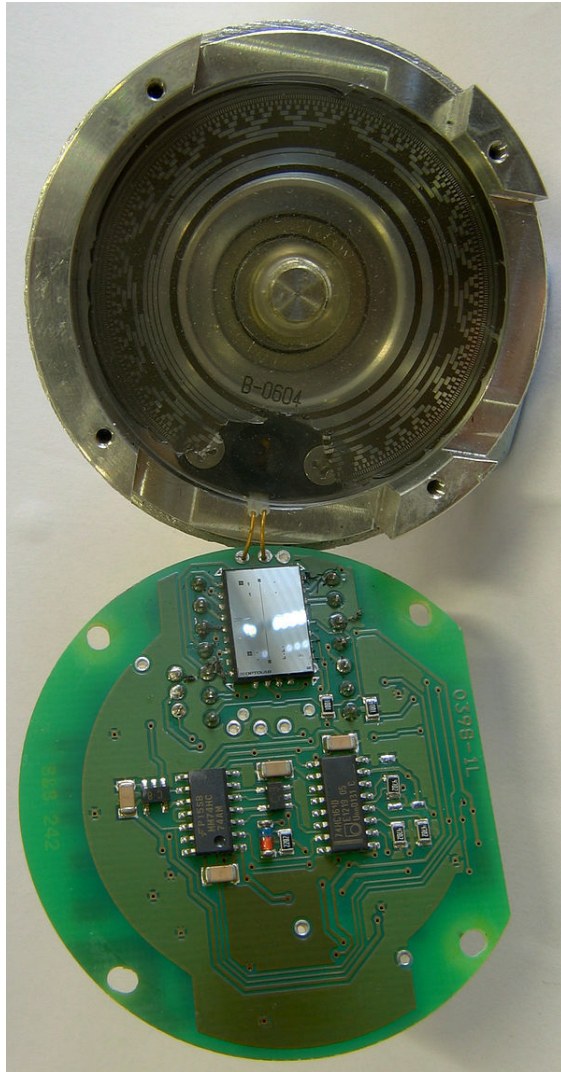
#### 48.3.2 Towers of Hanoi

The binary-reflected Gray code can serve as a solution guide for the **Towers of Hanoi** problem, as well as the classical **Chinese rings puzzle**, a sequential mechanical puzzle mechanism.<sup>[6]</sup> It also forms a **Hamiltonian cycle** on a **hypercube**, where each bit is seen as one dimension.

#### 48.3.3 Genetic algorithms

Due to the **Hamming distance** properties of Gray codes, they are sometimes used in **genetic algorithms**. They are very useful in this field, since mutations in the code allow for mostly incremental changes, but occasionally a single





A Gray code absolute rotary encoder with 13 tracks. At the top can be seen the housing, interrupter disk, and light source; at the bottom can be seen the sensing element and support components.

bit-change can cause a big leap and lead to new properties.

#### 48.3.4 Karnaugh maps

Gray codes are also used in labelling the axes of **Karnaugh maps**.<sup>[9]</sup>

#### 48.3.5 Error correction

In modern **digital communications**, Gray codes play an important role in **error correction**. For example, in a **digital modulation scheme** such as **QAM** where data is typically transmitted in **symbols** of 4 bits or more, the signal's **constellation diagram** is arranged so that the bit patterns conveyed by adjacent constellation points differ by only one bit. By combining this with **forward error correction** capable of correcting single-bit errors, it is possible for a **receiver** to correct any transmission errors that

cause a constellation point to deviate into the area of an adjacent point. This makes the transmission system less susceptible to **noise**.

#### 48.3.6 Communication between clock domains

Main article: [clock domain crossing](#)

Digital logic designers use Gray codes extensively for passing multi-bit count information between synchronous logic that operates at different clock frequencies. The logic is considered operating in different “clock domains”. It is fundamental to the design of large chips that operate with many different clocking frequencies.

#### 48.3.7 Cycling through states with minimal effort

If a system has to cycle through all possible combinations of on-off states of some set of controls, and the changes of the controls require non-trivial expense (e.g. time, wear, human work), a Gray code minimizes the number of setting changes to just one change for each combination of states. An example would be testing a piping system for all combinations of settings of its manually operated valves.

#### Gray code counters and arithmetic

A typical use of Gray code counters is building a **FIFO** (first-in, first-out) data buffer that has read and write ports that exist in different clock domains. The input and output counters inside such a dual-port FIFO are often stored using Gray code to prevent invalid transient states from being captured when the count crosses clock domains.<sup>[10]</sup> The updated read and write pointers need to be passed between clock domains when they change, to be able to track FIFO empty and full status in each domain. Each bit of the pointers is sampled non-deterministically for this clock domain transfer. So for each bit, either the old value or the new value is propagated. Therefore, if more than one bit in the multi-bit pointer is changing at the sampling point, a “wrong” binary value (neither new nor old) can be propagated. By guaranteeing only one bit can be changing, Gray codes guarantee that the only possible sampled values are the new or old multi-bit value. Typically Gray codes of power-of-two length are used.

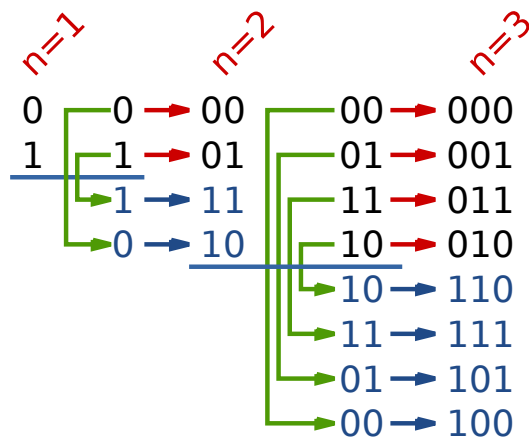
Sometimes digital buses in electronic systems are used to convey quantities that can only increase or decrease by one at a time, for example the output of an event counter which is being passed between clock domains or to a digital-to-analog converter. The advantage of Gray codes in these applications is that differences in the propagation delays of the many wires that represent the bits of the



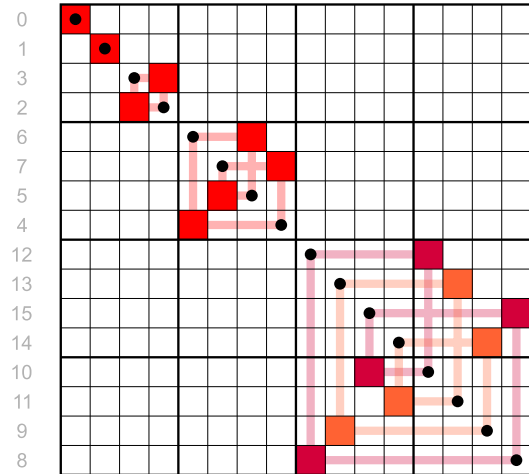
code cannot cause the received value to go through states that are out of the Gray code sequence. This is similar to the advantage of Gray codes in the construction of mechanical encoders, however the source of the Gray code is an electronic counter in this case. The counter itself must count in Gray code, or if the counter runs in binary then the output value from the counter must be reclocked after it has been converted to Gray code, because when a value is converted from binary to Gray code, it is possible that differences in the arrival times of the binary data bits into the binary-to-Gray conversion circuit will mean that the code could go briefly through states that are wildly out of sequence. Adding a clocked register after the circuit that converts the count value to Gray code may introduce a clock cycle of latency, so counting directly in Gray code may be advantageous. A Gray code counter was patented in 1962 [US3020481](#), and there have been many others since. In recent times a Gray code counter can be implemented as a state machine in [Verilog](#). In order to produce the next count value, it is necessary to have some combinational logic that will increment the current count value that is stored in Gray code. Probably the most obvious way to increment a Gray code number is to convert it into ordinary binary code, add one to it with a standard binary adder, and then convert the result back to Gray code. This approach was discussed in a paper in 1996 <sup>[11]</sup> and then subsequently patented by someone else in 1998 [US5754614](#). Other methods of counting in Gray code are discussed in a report by R. W. Doran, including taking the output from the first latches of the master-slave flip flops in a binary ripple counter.<sup>[12]</sup>

Perhaps the most common electronic counter with the “only one bit changes at a time” property is the [Johnson counter](#).

## 48.4 Constructing an $n$ -bit Gray code



The first few steps of the reflect-and-prefix method.



4-bit Gray code permutation

The binary-reflected Gray code list for  $n$  bits can be generated [recursively](#) from the list for  $n - 1$  bits by reflecting the list (i.e. listing the entries in reverse order), concatenating the original list with the reversed list, prefixing the entries in the original list with a binary 0, and then prefixing the entries in the reflected list with a binary 1.<sup>[6]</sup> For example, generating the  $n = 3$  list from the  $n = 2$  list:

The one-bit Gray code is  $G_1 = (0, 1)$ . This can be thought of as built recursively as above from a zero-bit Gray code  $G_0 = (\Lambda)$  consisting of a single entry of zero length. This iterative process of generating  $G_{n+1}$  from  $G_n$  makes the following properties of the standard reflecting code clear:

- $G_n$  is a [permutation](#) of the numbers  $0, \dots, 2^n - 1$ . (Each number appears exactly once in the list.)
- $G_n$  is embedded as the first half of  $G_{n+1}$ .
- Therefore, the coding is *stable*, in the sense that once a binary number appears in  $G_n$  it appears in the same position in all longer lists; so it makes sense to talk about *the* reflective Gray code value of a number:  $G(m)$  = the  $m$ -th reflecting Gray code, counting from 0.
- Each entry in  $G_n$  differs by only one bit from the previous entry. (The Hamming distance is 1.)
- The last entry in  $G_n$  differs by only one bit from the first entry. (The code is cyclic.)

These characteristics suggest a simple and fast method of translating a binary value into the corresponding Gray code. Each bit is inverted if the next higher bit of the input value is set to one. This can be performed in parallel by a bit-shift and exclusive-or operation if they are available: the  $n$ th Gray code is obtained by computing  $n \oplus \lfloor n/2 \rfloor$

A similar method can be used to perform the reverse translation, but the computation of each bit depends on the computed value of the next higher bit so it cannot be performed in parallel. Assuming  $g_i$  is the  $i$ th gray-coded bit ( $g_0$  being the most significant bit), and  $b_i$  is the  $i$ th binary-coded bit ( $b_0$  being the most-significant bit), the reverse translation can be given recursively:  $b_0 = g_0$ , and  $b_i = g_i \oplus b_{i-1}$ . Alternatively, decoding a Gray code into a binary number can be described as a **prefix sum** of the bits in the Gray code, where each individual summation operation in the prefix sum is performed modulo two.

To construct the binary-reflected Gray code iteratively, at step 0 start with the  $code_0 = 0$ , and at step  $i > 0$  find the bit position of the least significant 1 in the binary representation of  $i$  and flip the bit at that position in the previous code  $code_{i-1}$  to get the next code  $code_i$ . The bit positions start 0, 1, 0, 2, 0, 1, 0, 3, ... (sequence A007814 in the OEIS). See **find first set** for efficient algorithms to compute these values.

## 48.5 Converting to and from Gray code

The following functions in C convert between binary numbers and their associated Gray codes. While it may seem that gray-to-binary conversion requires each bit to be handled one at a time, faster algorithms exist.<sup>[13]</sup>

```
/* * This function converts an unsigned binary * number
to reflected binary Gray code. * * The operator >> is
shift right. The operator ^ is exclusive or. */ unsigned int
binaryToGray(unsigned int num) { return num ^ (num
>> 1); } /* * This function converts a reflected binary
* Gray code number to a binary number. * Each Gray
code bit is exclusive-ored with all * more significant
bits. */ unsigned int grayToBinary(unsigned int num) {
unsigned int mask; for (mask = num >> 1; mask != 0;
mask = mask >> 1) { num = num ^ mask; } return num;
} /* * A more efficient version, for Gray codes of 32 or
fewer bits. */ unsigned int grayToBinary32(unsigned int
num) { num = num ^ (num >> 16); num = num ^ (num
>> 8); num = num ^ (num >> 4); num = num ^ (num >>
2); num = num ^ (num >> 1); return num; }
```

## 48.6 Special types of Gray codes

In practice, a “Gray code” almost always refers to a binary-reflected Gray code (BRGC). However, mathematicians have discovered other kinds of Gray codes. Like BRGCs, each consists of a lists of words, where each word differs from the next in only one digit (each word has a **Hamming distance** of 1 from the next word).

### 48.6.1 $n$ -ary Gray code

There are many specialized types of Gray codes other than the binary-reflected Gray code. One such type of Gray code is the  **$n$ -ary Gray code**, also known as a **non-Boolean Gray code**. As the name implies, this type of Gray code uses non-Boolean values in its encodings.

For example, a 3-ary (**ternary**) Gray code would use the values {0, 1, 2}. The  $(n, k)$ -Gray code is the  $n$ -ary Gray code with  $k$  digits.<sup>[14]</sup> The sequence of elements in the (3, 2)-Gray code is: {00, 01, 02, 12, 10, 11, 21, 22, 20}. The  $(n, k)$ -Gray code may be constructed recursively, as the BRGC, or may be constructed **iteratively**. An **algorithm** to iteratively generate the  $(N, k)$ -Gray code is presented (in C):

```
// inputs: base, digits, value // output: gray // Convert
a value to a graycode with the given base and digits.
// Iterating through a sequence of values would result
in a sequence // of Gray codes in which only one digit
changes at a time. void to_gray(unsigned base, unsigned
digits, unsigned value, unsigned gray[digits]) { unsigned
baseN[digits]; // Stores the ordinary base-N number,
one digit per entry unsigned i; // The loop variable //
Put the normal baseN number into the baseN array. For
base 10, 109 // would be stored as [9,0,1] for (i = 0; i
< digits; i++) { baseN[i] = value % base; value = value
/ base; } // Convert the normal baseN number into the
graycode equivalent. Note that // the loop starts at the
most significant digit and goes down. unsigned shift =
0; while (i--) { // The gray digit gets shifted down by
the sum of the higher // digits. gray[i] = (baseN[i] +
shift) % base; shift = shift + base - gray[i]; // Subtract
from base so shift is positive } } // EXAMPLES // input:
value = 1899, base = 10, digits = 4 // output: baseN[] =
[9,9,8,1], gray[] = [0,1,7,1] // input: value = 1900,
base = 10, digits = 4 // output: baseN[] = [0,0,9,1],
gray[] = [0,1,8,1]
```

There are other graycode algorithms for  $(n,k)$ -Gray codes. The  $(n,k)$ -Gray code produced by the above algorithm is always cyclical; some algorithms, such as that by Guan,<sup>[14]</sup> lack this property when  $k$  is odd. On the other hand, while only one digit at a time changes with this method, it can change by wrapping (looping from  $n - 1$  to 0). In Guan’s algorithm, the count alternately rises and falls, so that the numeric difference between two graycode digits is always one.

Gray codes are not uniquely defined, because a permutation of the columns of such a code is a Gray code too. The above procedure produces a code in which the lower the significance of a digit, the more often it changes, making it similar to normal counting methods.

See also **Skew binary number system**, a variant ternary number system where at most 2 digits change on each increment, as each increment can be done with at most one digit carry operation.

### 48.6.2 Balanced Gray code

Although the binary reflected Gray code is useful in many scenarios, it is not optimal in certain cases because of a lack of “uniformity”.<sup>[15]</sup> In **balanced Gray codes**, the number of changes in different coordinate positions are as close as possible. To make this more precise, let  $G$  be an  $R$ -ary complete Gray cycle having transition sequence  $(\delta_k)$ ; the *transition counts* (*spectrum*) of  $G$  are the collection of integers defined by

$$\lambda_k = |\{j \in \mathbb{Z}_{R^n} : \delta_j = k\}|, \text{ for } k \in \mathbb{Z}_R$$

A Gray code is *uniform* or *uniformly balanced* if its transition counts are all equal, in which case we have  $\lambda_k = R^n/n$  for all  $k$ . Clearly, when  $R = 2$ , such codes exist only if  $n$  is a power of 2. Otherwise, if  $n$  does not divide  $R^n$  evenly, it is possible to construct *well-balanced* codes where every transition count is either  $\lfloor R^n/n \rfloor$  or  $\lceil R^n/n \rceil$ . Gray codes can also be *exponentially balanced* if all of their transition counts are adjacent powers of two, and such codes exist for every power of two.<sup>[16]</sup>

For example, a balanced 4-bit Gray code has 16 transitions, which can be evenly distributed among all four positions (four transitions per position), making it uniformly balanced.<sup>[15]</sup>

0 1 1 1 1 1 0 0 0 0 0 1 1 0 0 0 1 1 1 0 0 1 1 1 1 0 0  
0 0 0 0 0 0 1 1 1 1 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0 0 1 1  
1 1 1 1

whereas a balanced 5-bit Gray code has a total of 32 transitions, which cannot be evenly distributed among the positions. In this example, four positions have six transitions each, and one has eight.<sup>[15]</sup>

1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0  
0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1  
1 0 0 0 1 1 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0 1 1 1 1 1 1 0  
0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1  
1 1 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1  
1 0 0 0 1 1 1 1 1 1

We will now show a construction for well-balanced binary Gray codes which allows us to generate an  $n$ -digit balanced Gray code for every  $n$ .<sup>[17]</sup> The main principle is to inductively construct an  $(n+2)$ -digit Gray code  $G'$  given an  $n$ -digit Gray code  $G$  in such a way that the balanced property is preserved. To do this, we consider partitions of  $G = g_0, \dots, g_{2^n-1}$  into an even number  $L$  of non-empty blocks of the form

$$\{g_0\}, \{g_1, \dots, g_{k_2}\}, \{g_{k_2+1}, \dots, g_{k_3}\}, \dots, \{g_{k_{L-2}+1}, \dots, g_{k_{L-1}}\}, \{g_{k_{L-1}+1}, \dots, g_{2^n-1}\}$$

where  $k_1 = 0, k_{L-1} = -2$ , and  $k_L = -1 \pmod{2^n}$ . This partition induces an  $(n+2)$ -digit Gray code given by

$$00g_0,$$

$$00g_1, \dots, 00g_{k_2}, 01g_{k_2}, \dots, 01g_1, 11g_1, \dots, 11g_{k_2}, \\ 11g_{k_2+1}, \dots, 11g_{k_3}, 01g_{k_3}, \dots, 01g_{k_2+1}, 00g_{k_2+1}, \dots, 00g_{k_3}, \dots, \\ 00g_{-2}, 00g_{-1}, 10g_{-1}, 10g_{-2}, \dots, 10g_0, 11g_0, 11g_{-1}, 01g_{-1}, 01g_0$$

If we define the *transition multiplicities*  $m_i = |\{j : \delta_{k_j} = i, 1 \leq j \leq L\}|$  to be the number of times the digit in position  $i$  changes between consecutive blocks in a partition, then for the  $(n+2)$ -digit Gray code induced by this partition the transition spectrum  $\lambda'_i$  is

$$\lambda'_i = \begin{cases} 4\lambda_i - 2m_i, & \text{if } 0 \leq i < n \\ L, & \text{otherwise} \end{cases}$$

The delicate part of this construction is to find an adequate partitioning of a balanced  $n$ -digit Gray code such that the code induced by it remains balanced, but for this only the transition multiplicities matter; joining two consecutive blocks over a digit  $i$  transition and splitting another block at another digit  $i$  transition produces a different Gray code with the exact same transition spectrum  $\lambda'_i$ , so one may for example<sup>[16]</sup> designate the first  $m_i$  transitions at digit  $i$  as those that fall between two blocks. Uniform codes can be found when  $R \equiv 0 \pmod{4}$  and  $R^n \equiv 0 \pmod{n}$ , and this construction can be extended to the  $R$ -ary case as well.<sup>[17]</sup>

### 48.6.3 Monotonic Gray codes

Monotonic codes are useful in the theory of interconnection networks, especially for minimizing dilation for linear arrays of processors.<sup>[18]</sup> If we define the *weight* of a binary string to be the number of 1s in the string, then although we clearly cannot have a Gray code with strictly increasing weight, we may want to approximate this by having the code run through two adjacent weights before reaching the next one.

We can formalize the concept of monotone Gray codes as follows: consider the partition of the hypercube  $Q_n = (V_n, E_n)$  into *levels* of vertices that have equal weight, i.e.

$$V_n(i) = \{v \in V_n : v \text{ weight has } i\}$$

for  $0 \leq i \leq n$ . These levels satisfy  $|V_n(i)| = \binom{n}{i}$ . Let  $Q_n(i)$  be the subgraph of  $Q_n$  induced by  $V_n(i) \cup V_n(i+1)$ , and let  $E_n(i)$  be the edges in  $Q_n(i)$ . A monotonic Gray code is then a Hamiltonian path in  $Q_n$  such that whenever  $\delta_1 \in E_n(i)$  comes before  $\delta_2 \in E_n(j)$  in the path, then  $i \leq j$ .

An elegant construction of monotonic  $n$ -digit Gray codes for any  $n$  is based on the idea of recursively building subpaths  $P_{n,j}$  of length  $2\binom{n}{j}$  having edges in  $E_n(j)$ .<sup>[18]</sup> We define  $P_{1,0} = (0, 1)$ ,  $P_{n,j} = \emptyset$  whenever  $j < 0$  or  $j \geq n$ , and

$$P_{n+1,j} = 1P_{n,j-1}^{\pi_n}, 0P_{n,j}$$

otherwise. Here,  $\pi_n$  is a suitably defined permutation and  $P^\pi$  refers to the path  $P$  with its coordinates permuted by  $\pi$ . These paths give rise to two monotonic  $n$ -digit Gray codes  $G_n^{(1)}$  and  $G_n^{(2)}$  given by

$$G_n^{(1)} = P_{n,0}P_{n,1}^R P_{n,2}P_{n,3}^R \cdots \text{ and } G_n^{(2)} = P_{n,0}^R P_{n,1}P_{n,2}^R P_{n,3} \cdots$$

The choice of  $\pi_n$  which ensures that these codes are indeed Gray codes turns out to be  $\pi_n = E^{-1}(\pi_{n-1}^2)$ . The first few values of  $P_{n,j}$  are shown in the table below.

These monotonic Gray codes can be efficiently implemented in such a way that each subsequent element can be generated in  $O(n)$  time. The algorithm is most easily described using coroutines.

Monotonic codes have an interesting connection to the Lovász conjecture, which states that every connected vertex-transitive graph contains a Hamiltonian path. The “middle-level” subgraph  $Q_{2n+1}(n)$  is vertex-transitive (that is, its automorphism group is transitive, so that each vertex has the same “local environment” and cannot be differentiated from the others, since we can relabel the coordinates as well as the binary digits to obtain an automorphism) and the problem of finding a Hamiltonian path in this subgraph is called the “middle-levels problem”, which can provide insights into the more general conjecture. The question has been answered affirmatively for  $n \leq 15$ , and the preceding construction for monotonic codes ensures a Hamiltonian path of length at least  $0.839N$  where  $N$  is the number of vertices in the middle-level subgraph.<sup>[19]</sup>

#### 48.6.4 Beckett–Gray code

Another type of Gray code, the **Beckett–Gray code**, is named for Irish playwright Samuel Beckett, who was interested in symmetry. His play “Quad” features four actors and is divided into sixteen time periods. Each period ends with one of the four actors entering or leaving the stage. The play begins with an empty stage, and Beckett wanted each subset of actors to appear on stage exactly once.<sup>[20]</sup> Clearly the set of actors currently on stage can be represented by a 4-bit binary Gray code. Beckett, however, placed an additional restriction on the script: he wished the actors to enter and exit so that the actor who had been on stage the longest would always be the one to exit. The actors could then be represented by a first in, first out queue, so that (of the actors onstage) the actor being dequeued is always the one who was enqueued first.<sup>[20]</sup> Beckett was unable to find a Beckett–Gray code for his play, and indeed, an exhaustive listing of all possible sequences reveals that no such code exists for  $n = 4$ . It is known today that such codes do exist for  $n = 2, 5, 6, 7$ ,

and 8, and do not exist for  $n = 3$  or 4. An example of an 8-bit Beckett–Gray code can be found in Donald Knuth’s *Art of Computer Programming*.<sup>[6]</sup> According to Sawada and Wong, the search space for  $n = 6$  can be explored in 15 hours, and more than 9,500 solutions for the case  $n = 7$  have been found.<sup>[21]</sup>

#### 48.6.5 Snake-in-the-box codes

**Snake-in-the-box** codes, or *snakes*, are the sequences of nodes of induced paths in an  $n$ -dimensional hypercube graph, and coil-in-the-box codes, or *coils*, are the sequences of nodes of induced cycles in a hypercube. Viewed as Gray codes, these sequences have the property of being able to detect any single-bit coding error. Codes of this type were first described by W. H. Kautz in the late 1950s;<sup>[22]</sup> since then, there has been much research on finding the code with the largest possible number of codewords for a given hypercube dimension.

#### 48.6.6 Single-track Gray code

Yet another kind of Gray code is the **single-track Gray code** (STGC) developed by N. B. Spedding<sup>[23]</sup> and refined by Hiltgen, Paterson and Brandestini in “Single-track Gray codes” (1996).<sup>[24][25]</sup> The STGC is a cyclical list of  $P$  unique binary encodings of length  $n$  such that two consecutive words differ in exactly one position, and when the list is examined as a  $P \times n$  matrix, each column is a cyclic shift of the first column.<sup>[26]</sup>

The name comes from their use with rotary encoders, where a number of tracks are being sensed by contacts, resulting for each in an output of 0 or 1. To reduce noise due to different contacts not switching at exactly the same moment in time, one preferably sets up the tracks so that the data output by the contacts are in Gray code. To get high angular accuracy, one needs lots of contacts; in order to achieve at least 1 degree accuracy, one needs at least 360 distinct positions per revolution, which requires a minimum of 9 bits of data, and thus the same number of contacts.

If all contacts are placed at the same angular position, then 9 tracks are needed to get a standard BRGC with at least 1 degree accuracy. However, if the manufacturer moves a contact to a different angular position (but at the same distance from the center shaft), then the corresponding “ring pattern” needs to be rotated the same angle to give the same output. If the most significant bit (the inner ring in Figure 1) is rotated enough, it exactly matches the next ring out. Since both rings are then identical, the inner ring can be cut out, and the sensor for that ring moved to the remaining, identical ring (but offset at that angle from the other sensor on that ring). Those two sensors on a single ring make a quadrature encoder. That reduces the number of tracks for a “1 degree resolution” angular encoder to 8 tracks. Reducing the number

of tracks still further can't be done with BRGC.

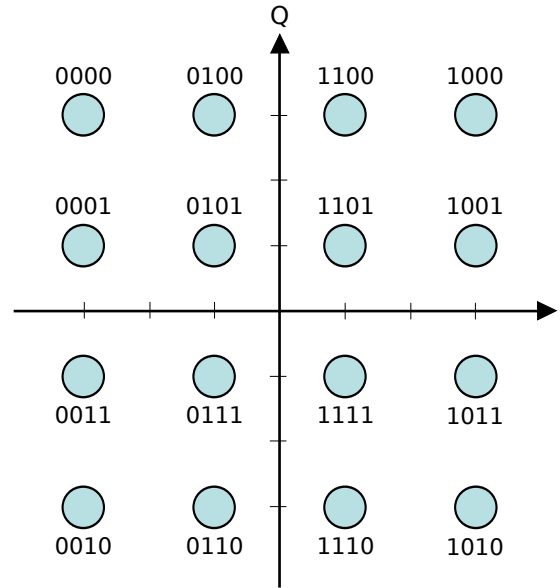
For many years, Torsten Sillke<sup>[27]</sup> and other mathematicians believed that it was impossible to encode position on a single track such that consecutive positions differed at only a single sensor, except for the 2-sensor, 1-track quadrature encoder. So for applications where 8 tracks were too bulky, people used single-track incremental encoders (quadrature encoders) or 2-track “quadrature encoder + reference notch” encoders.

N. B. Spedding, however, registered a patent in 1994 with several examples showing that it was possible.<sup>[23]</sup> Although it is not possible to distinguish  $2^n$  positions with  $n$  sensors on a single track, it *is* possible to distinguish close to that many. For example, when  $n$  is itself a power of 2,  $n$  sensors can distinguish  $2^n - 2n$  positions.<sup>[28]</sup> Hiltgen and Paterson published a paper in 2001 exhibiting a single-track gray code with exactly 360 angular positions, constructed using 9 sensors.<sup>[25]</sup> Since this number is larger than  $2^8 = 256$ , more than 8 sensors are required by any code, although a BRGC could distinguish 512 positions with 9 sensors. An STGC for  $P = 30$  and  $n = 5$  is reproduced here:



Single-track Gray code with 5 sensors.

Each column is a cyclic shift of the first column, and from any row to the next row only one bit changes.<sup>[29]</sup> The single-track nature (like a code chain) is useful in the fabrication of these wheels (compared to BRGC), as only one track is needed, thus reducing their cost and size. The Gray code nature is useful (compared to chain codes, also called **De Bruijn sequences**), as only one sensor will change at any one time, so the uncertainty during a transition between two discrete states will only be plus or minus one unit of angular measurement the device is capable of resolving.<sup>[30]</sup>



A Gray-coded constellation diagram for rectangular 16-QAM.

#### 48.6.7 2-dimensional Gray code

Two-dimensional Gray codes are used in communication to minimize the number of bit errors in **quadrature amplitude modulation** adjacent points in the **constellation**. In a typical encoding the horizontal and vertical adjacent constellation points differ by a single bit, and diagonal adjacent points differ by 2 bits.<sup>[31]</sup>

### 48.7 Gray isometry

The bijective mapping  $\{ 0 \leftrightarrow 00, 1 \leftrightarrow 01, 2 \leftrightarrow 11, 3 \leftrightarrow 10 \}$  establishes an **isometry** between the **metric space** over the **finite field**  $\mathbb{Z}_2^2$  with the metric given by the **Hamming distance** and the metric space over the **finite ring**  $\mathbb{Z}_4$  (the usual **modulo arithmetic**) with the metric given by the **Lee distance**. The mapping is suitably extended to an isometry of the **Hamming spaces**  $\mathbb{Z}_2^{2^m}$  and  $\mathbb{Z}_4^m$ . Its importance lies in establishing a correspondence between various “good” but not necessarily **linear codes** as Gray-map images in  $\mathbb{Z}_2^2$  of **ring-linear codes** from  $\mathbb{Z}_4$ .<sup>[32][33]</sup>

### 48.8 See also

- **Linear feedback shift register**
- **De Bruijn sequence**
- **Gillham code**
- **Steinhaus–Johnson–Trotter algorithm**, an algorithm that generates Gray codes for the factorial number system



## 48.9 References

- [1] F. Gray. *Pulse code communication*, March 17, 1953 (filed Nov. 1947). U.S. Patent 2,632,058
- [2] J. Breckman. *Encoding Circuit*, Jan 31, 1956 (filed Dec. 1953). U.S. Patent 2,733,432
- [3] E. A. Ragland et al. *Direction-Sensitive Binary Code Position Control System*, Feb. 11, 1958 (filed Oct. 1953). U.S. Patent 2,823,345
- [4] S. Reiner et al. *Automatic Rectification System*, Jun 24, 1958 (filed Jan. 1954). U.S. Patent 2,839,974
- [5] Pickover, Clifford A. (2009). *The Math Book: From Pythagoras to the 57th Dimension, 250 Milestones in the History of Mathematics*. Sterling Publishing Company. p. 392. ISBN 9781402757969.
- [6] Knuth, Donald E. “Generating all  $n$ -tuples.” *The Art of Computer Programming, Volume 4A: Enumeration and Backtracking*, pre-fascicle 2a, October 15, 2004.
- [7] Cattermole, K. W. (1969). *Principles of Pulse Code Modulation*. New York: American Elsevier. ISBN 0-444-19747-8.
- [8] Goodall, W. M. (1951). “Television by Pulse Code Modulation”. *Bell Syst. Tech. J.* **30**: 33–49.
- [9] Wakerly, John F (1994). *Digital Design: Principles & Practices*. New Jersey: Prentice Hall. pp. 222, 48–49. ISBN 0-13-211459-3. Note that the two page sections taken together say that K-maps are labeled with Gray code. The first section says that they are labeled with a code that changes only one bit between entries and the second section says that such a code is called Gray code.
- [10] “Synchronization in Digital Logic Circuits by Ryan Donohue
- [11] Mehta, H.; Owens, R.M. & Irwin, M.J. (1996), Some issues in gray code addressing, in the Proceedings of the 6th Great Lakes Symposium on VLSI (GLSVLSI 96), IEEE Computer Society, pp. 178
- [12] The Gray Code by R. W. Doran
- [13] Henry Gordon Dietz. “The Aggregate Magic Algorithms: Gray Code Conversion”
- [14] Guan, Dah-Jyh (1998). “Generalized Gray Codes with Applications”. *Proc. Natl. Sci. Counc. Repub. Of China (A)*. **22**: 841–848. CiteSeerX 10.1.1.119.1344
- [15] Bhat, Girish S.; Savage, Carla D. (1996). “Balanced Gray codes”. *Electronic Journal of Combinatorics*. **3** (1): R25.
- [16] Suparta, I. N. (2005). “A simple proof for the existence of exponentially balanced Gray codes”. *Electronic Journal of Combinatorics*. **12**.
- [17] M. Flahive and B. Bose (2007). “Balancing cyclic R-ary Gray codes”. *Electronic Journal of Combinatorics*. **14**.
- [18] C. D Savage and P. Winkler (1995). “Monotone Gray codes and the middle levels problem”. *Journal of Combinatorial Theory, Series A*. **70** (2): 230–248. doi:10.1016/0097-3165(95)90091-8. ISSN 0097-3165.
- [19] C. D Savage (1997). “Long cycles in the middle two levels of the Boolean lattice”.
- [20] Goddyn, Luis (1999). “MATH 343 Applied Discrete Math Supplementary Materials” (PDF). Dept. of Math, Simon Fraser U.
- [21] Sawada, J.; Wong, D. (2007). “A Fast Algorithm to generate Beckett–Gray codes”. *Electronic Notes in Discrete Mathematics*. **29**: 571–577. doi:10.1016/j.endm.2007.07.091.
- [22] Kautz, W. H. (1958). “Unit-distance error-checking codes”. *IRE Trans. Elect. Comput.* **7**: 177–180.
- [23] NZ 264738, Spedding, Norman Bruce, “A position encoder”, published 1994 A claim is at [http://www.winzurf.co.nz/Single\\_Track\\_Grey\\_Code\\_Patent/Single\\_track\\_Grey\\_code\\_encoder\\_patent.pdf](http://www.winzurf.co.nz/Single_Track_Grey_Code_Patent/Single_track_Grey_code_encoder_patent.pdf)
- [24] Hiltgen, Alain P.; Paterson, Kenneth G.; Brandestini, Marco (1996). “Single-Track Gray Codes” (PDF). *IEEE Transactions on Information Theory*. **42** (5): 1555–1561. doi:10.1109/18.532900.
- [25] Hiltgen, Alain P.; Paterson, Kenneth G. (September 2001). “Single-Track Circuit Codes” (PDF). *IEEE Transactions on Information Theory*. **47** (6): 2587–2595. doi:10.1109/18.945274. (no mention of Spedding)
- [26] Etzion, Tuvi; Schwartz, Moshe (1999). “The Structure of Single-Track Gray Codes” (PDF). *IEEE Transactions on Information Theory*. **45** (7): 2383–2396. doi:10.1109/18.796379.
- [27] Torsten Sillke. “Gray-Codes with few tracks”.
- [28] Etzion, Tuvi; Paterson, Kenneth G. (May 1996). “Near Optimal Single-Track Gray Codes” (PDF). *IEEE Transactions on Information Theory*. **42** (3): 779–789. doi:10.1109/18.490544.
- [29] Ruskey, Frank; Weston, Mark (June 18, 2005). “A Survey of Venn Diagrams: Symmetric Diagrams”. *Electronic Journal of Combinatorics*.
- [30] Alciatore, David G.; Hstand, Michael B. (1999). *Mechatronics*. McGraw–Hill Education – Europe. ISBN 978-0-07-131444-2.
- [31] <http://www.dsprelated.com/showthread/comp.dsp/96917-1.php>
- [32] Marcus Greferath (2009). “An Introduction to Ring-Linear Coding Theory”. In Massimiliano Sala, Teo Mora, Ludovic Perret, Shojiro Sakata, Carlo Traverso. *Gröbner Bases, Coding, and Cryptography*. Springer Science & Business Media. p. 220. ISBN 978-3-540-93806-4.
- [33] [http://www.encyclopediaofmath.org/index.php/Kerdock\\_and\\_Preparata\\_codes](http://www.encyclopediaofmath.org/index.php/Kerdock_and_Preparata_codes)

## Sources

- Black, Paul E. *Gray code*. 25 February 2004. NIST.
- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). “Section 22.3. Gray Codes”. *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.
- Savage, Carla (1997). “A Survey of Combinatorial Gray Codes”. *SIAM Rev.* **39** (4): 605–629. doi:10.1137/S0036144595295272. JSTOR 2132693.
- Wilf, Herbert S. (1989). “Chapters 1–3”. *Combinatorial algorithms: an update*. SIAM. ISBN 0-89871-231-9.

## 48.10 External links

- “Gray Code” demonstration by Michael Schreiber, Wolfram Demonstrations Project (with Mathematica implementation). 2007.
- NIST Dictionary of Algorithms and Data Structures: Gray code
- Hitch Hiker’s Guide to Evolutionary Computation, Q21: What are Gray codes, and why are they used?, including C code to convert between binary and BRGC
- Subsets or Combinations Can generate BRGC strings
- “The Structure of Single-Track Gray Codes” by Moshe Schwartz, Tuvi Etzion
- Single-Track Circuit Codes by Hiltgen, Alain P.; Patterson, Kenneth G.
- Dragos A. Harabor uses Gray codes in a 3D digitizer.
- single-track gray codes, binary chain codes (Lancaster 1994), and linear feedback shift registers are all useful in finding one’s absolute position on a single-track rotary encoder (or other position sensor).
- Computing Binary Combinatorial Gray Codes Via Exhaustive Search With SAT Solvers by Zinovik, I.; Kroening, D.; Chebiryak, Y.
- AMS Column: Gray codes
- Optical Encoder Wheel Generator
- ProtoTalk.net – Understanding Quadrature Encoding – Covers quadrature encoding in more detail with a focus on robotic applications



# Chapter 49

## Griesmer bound

In the **mathematics** of **coding theory**, the **Griesmer bound**, named after James Hugo Griesmer, is a bound on the length of **binary codes** of dimension  $k$  and minimum distance  $d$ . There is also a very similar version for non-binary codes.

so this becomes  $d - w(v) + w(u) \geq d$ . By summing this with  $w(v) + w(u) \geq d$ , we obtain  $d + 2w(u) \geq 2d$ . But  $w(u) = d'$ , so we get  $d' \geq \frac{d}{2}$ . This implies

$$n' \geq N(k-1, \frac{d}{2}),$$

therefore due to the integrality of  $n'$

$$n' \geq \lceil N(k-1, \frac{d}{2}) \rceil,$$

so that

$$N(k, d) \geq \lceil N(k-1, \frac{d}{2}) \rceil + d.$$

By induction over  $k$  we will eventually get

$$N(k, d) \geq \sum_{i=0}^{k-1} \left\lceil \frac{d}{2^i} \right\rceil.$$

Note that at any step the dimension decreases by 1 and the distance is halved, and we use the identity

$$\left\lceil \frac{\lceil a/2^{k-1} \rceil}{2} \right\rceil = \left\lceil \frac{a}{2^k} \right\rceil$$

for any integer  $a$  and positive integer  $k$ .

### 49.1 Statement of the bound

For a binary linear code, the Griesmer bound is:

$$n \geq \sum_{i=0}^{k-1} \left\lceil \frac{d}{2^i} \right\rceil.$$

### 49.2 Proof

Let  $N(k, d)$  denote the minimum length of a binary code of dimension  $k$  and distance  $d$ . Let  $C$  be such a code. We want to show that

$$N(k, d) \geq \sum_{i=0}^{k-1} \left\lceil \frac{d}{2^i} \right\rceil.$$

Let  $G$  be a generator matrix of  $C$ . We can always suppose that the first row of  $G$  is of the form  $r = (1, \dots, 1, 0, \dots, 0)$  with weight  $d$ .

$$G = \begin{bmatrix} 1 & \dots & 1 & 0 & \dots & 0 \\ * & * & * & & G' & \end{bmatrix}$$

The matrix  $G'$  generates a code  $C'$ , which is called the residual code of  $C$ .  $C'$  has obviously dimension  $k' = k - 1$  and length  $n' = N(k, d) - d$ .  $C'$  has a distance  $d'$ , but we don't know it. Let  $u \in C'$  be such that  $w(u) = d'$ . There exists a vector  $v \in \mathbb{F}_2^d$  such that the concatenation  $(v|u) \in C$ . Then  $w(v) + w(u) = w(v|u) \geq d$ . On the other hand, also  $(v|u) + r \in C$ , since  $r \in C$  and  $C$  is linear:  $w((v|u) + r) \geq d$ . But

$w((v|u) + r) = w(((1, \dots, 1) + v)|u) = d - w(v) + w(u)$ , The proof is similar to the binary case and so it is omitted.

### 49.3 The bound for the general case

For a linear code over  $\mathbb{F}_q$ , the Griesmer bound becomes:

$$n \geq \sum_{i=0}^{k-1} \left\lceil \frac{d}{q^i} \right\rceil.$$

## 49.4 See also

- Singleton bound
- Hamming bound
- Gilbert-Varshamov bound
- Johnson bound
- Plotkin bound
- Elias Bassalygo bound

## 49.5 References

- J. H. Griesmer, “A bound for error-correcting codes,” IBM Journal of Res. and Dev., vol. 4, no. 5, pp. 532-542, 1960.

# Chapter 50

## Group code

In computer science, **group codes** are a type of code. Group codes consist of  $n$  linear block codes which are subgroups of  $G^n$ , where  $G$  is a finite abelian group.

A systematic group code  $C$  is a code over  $G^n$  of order  $|G|^k$  defined by  $n - k$  homomorphisms which determine the parity check bits. The remaining  $k$  bits are the information bits themselves.

for codes over finite Abelian groups, *IEEE Trans. Inform. Theory* **42**, No.6, (1996), 1839-1854.

- A. A. Zain, B. Sundar Rajan, "Dual codes of Systematic Group Codes over Abelian Groups", *Appl. Algebra Eng. Commun. Comput.* **8**(1): 71-83 (1996).

### 50.1 Construction

Group codes can be constructed by special generator matrices which resemble generator matrices of linear block codes except that the elements of those matrices are endomorphisms of the group instead of symbols from the code's alphabet. For example, consider the generator matrix

$$G = \begin{pmatrix} \begin{pmatrix} 00 \\ 11 \end{pmatrix} & \begin{pmatrix} 01 \\ 01 \end{pmatrix} & \begin{pmatrix} 11 \\ 01 \end{pmatrix} \\ \begin{pmatrix} 00 \\ 11 \end{pmatrix} & \begin{pmatrix} 11 \\ 11 \end{pmatrix} & \begin{pmatrix} 00 \\ 00 \end{pmatrix} \end{pmatrix}$$

The elements of this matrix are  $2 \times 2$  matrices which are endomorphisms. In this scenario, each codeword can be represented as  $g_1^{m_1} g_2^{m_2} \dots g_r^{m_r}$  where  $g_1, \dots, g_r$  are the generators of  $G$ .

### 50.2 References

- Biglieri, E.; Elia, M. (1993). "Construction of Linear Block Codes Over Groups". *Proceedings. IEEE International Symposium on Information Theory*. p. 360. doi:10.1109/ISIT.1993.748676. ISBN 0-7803-0878-6.
- G. D. Forney, M. Trott, The dynamics of group codes : State spaces, trellis diagrams and canonical encoders, *IEEE Trans. Inform. theory*, Vol **39** (1993), pages 1491-1593.
- V. V. Vazirani, Huzur Saran and B. S. Rajan, An efficient algorithm for constructing minimal trellises

## Chapter 51

# Guruswami–Sudan list decoding algorithm

In coding theory, list decoding is an alternative to unique decoding of error-correcting codes for large error rates. Using unique decoder one can correct up to  $\delta/2$  fraction of errors. But when error rate is greater than  $\delta/2$ , unique decoder will not be able to output the correct result. List decoding overcomes that issue. List decoding can correct more than  $\delta/2$  fraction of errors.

There are many efficient algorithms that can perform List decoding. In this article, we first present an algorithm for Reed–Solomon (RS) codes which corrects up to  $1 - \sqrt{2R}$  errors and is due to Madhu Sudan. Subsequently we describe the improved Guruswami–Sudan list decoding algorithm, which can correct up to  $1 - \sqrt{R}$  errors.

Here is a plot of the rate  $R$  and distance  $\delta$  for different algorithms.

<https://wiki.cse.buffalo.edu/cse545/sites/wiki.cse.buffalo.edu/cse545/files/81/Graph.jpg>

## 51.1 Algorithm 1 (Sudan’s list decoding algorithm)

### 51.1.1 Problem statement

**Input :** A field  $F$ ;  $n$  distinct pairs of elements  $(x_i, y_i)_{i=1}^n$  in  $F \times F$ ; and integers  $d$  and  $t$ .

**Output:** A list of all functions  $f : F \rightarrow F$  satisfying  $f(x)$  is a polynomial in  $x$  of degree at most  $d$

To understand Sudan’s Algorithm better, one may want to first know another algorithm which can be considered as the earlier version or the fundamental version of the algorithms for list decoding RS codes - the Berlekamp–Welch algorithm. Welch and Berlekamp initially came with an algorithm which can solve the problem in polynomial time with best threshold on  $t$  to be  $t \geq (n+d+1)/2$ . The mechanism of Sudan’s Algorithm is almost the same

as the algorithm of Berlekamp–Welch Algorithm, except in the step 1, one wants to compute a bivariate polynomial of bounded  $(1, k)$  degree. Sudan’s list decoding algorithm for Reed–Solomon code which is an improvement on Berlekamp and Welch algorithm, can solve the problem with  $t = (\sqrt{2nd})$ . This bound is better than the unique decoding bound  $1 - (\frac{R}{2})$  for  $R < 0.07$ .

### 51.1.2 Algorithm

#### Definition 1 (weighted degree)

For weights  $w_x, w_y \in \mathbb{Z}^+$ , the  $(w_x, w_y)$  – weighted degree of monomial  $q_{ij}x^i y^j$  is  $iw_x + jw_y$ . The  $(w_x, w_y)$  – weighted degree of a polynomial  $Q(x, y) = \sum_{ij} q_{ij}x^i y^j$  is the maximum, over the monomials with non-zero coefficients, of the  $(w_x, w_y)$  – weighted degree of the monomial.

For example,  $xy^2$  has  $(1, 3)$  -degree 7

#### Algorithm:

Inputs:  $n, d, t$ ;  $\{(x_1, y_1) \cdots (x_n, y_n)\}$  /\* Parameters  $l, m$  to be set later. \*/

Step 1: Find a non-zero bivariate polynomial  $Q : F^2 \mapsto F$  satisfying

- $Q(x, y)$  has  $(1, d)$  -weighted degree at most  $D = m + ld$
- For every  $i \in [n]$ ,

Step 2. Factor  $Q$  into irreducible factors.

Step 3. Output all the polynomials  $f$  such that  $(y - f(x))$  is a factor of  $Q$  and  $f(x_i) = y_i$  for at least  $t$  values of  $i \in [n]$

### 51.1.3 Analysis

One has to prove that the above algorithm runs in polynomial time and outputs the correct result. That can be done by proving following set of claims.

**Claim 1:**

If a function  $Q : F^2 \rightarrow F$  satisfying (2) exists, then one can find it in polynomial time.

**Proof:**

Note that a bivariate polynomial  $Q(x, y)$  of  $(1, d)$ -weighted degree at most  $D$  can be uniquely written as  $Q(x, y) = \sum_{j=0}^l \sum_{k=0}^{m+(l-j)d} q_{kj} x^k y^j$ . Then one has to find the coefficients  $q_{kj}$  satisfying the constraints  $\sum_{j=0}^l \sum_{k=0}^{m+(l-j)d} q_{kj} x_i^k y_i^j = 0$ , for every  $i \in [n]$ . This is a linear set of equations in the unknowns  $\{q_{kj}\}$ . One can find a solution using Gaussian elimination in polynomial time.

**Claim 2:**

If  $(m+1)(l+1) + d \binom{l+1}{2} > n$  then there exists a function  $Q(x, y)$  satisfying (2)

**Proof:**

To ensure a non zero solution exists, the number of coefficients in  $Q(x, y)$  should be greater than the number of constraints. Assume that the maximum degree  $\deg_x(Q)$  of  $x$  in  $Q(x, y)$  is  $m$  and the maximum degree  $\deg_y(Q)$  of  $y$  in  $Q(x, y)$  is  $l$ . Then the degree of  $Q(x, y)$  will be at most  $m + ld$ . One has to see that the linear system is homogenous. The setting  $q_{jk} = 0$  satisfies all linear constraints. However this does not satisfy (2), since the solution can be identically zero. To ensure that a non-zero solution exists, one has to make sure that number of unknowns in the linear system to be  $(m+1)(l+1) + d \binom{l+1}{2} > n$ , so that one can have a non zero  $Q(x, y)$ . Since this value is greater than  $n$ , there are more variables than constraints and therefore a non-zero solution exists.

**Claim 3:**

If  $Q(x, y)$  is a function satisfying (2) and  $f(x)$  is function satisfying (1) and  $t > m + ld$ , then  $(y - f(x))$  divides  $Q(x, y)$

**Proof:**

Consider a function  $p(x) = Q(x, f(x))$ . This is a polynomial in  $x$ , and argue that it has degree at most  $m + ld$ . Consider any monomial  $q_{jk} x^k y^j$  of  $Q(x, y)$ . Since  $Q$  has  $(1, d)$ -weighted degree at most  $m + ld$ , one can say that  $k + jd \leq m + ld$ . Thus the term  $q_{jk} x^k f(x)^j$  is a polynomial in  $x$  of degree at most  $k + jd \leq m + ld$ . Thus  $p(x)$  has degree at most  $m + ld$

Next argue that  $p(x)$  is identically zero. Since  $Q(x_i, f(x_i))$  is zero whenever  $y_i = f(x_i)$ , one can say that  $p(x_i)$  is zero for strictly greater than  $m + ld$  points.

Thus  $p$  has more zeroes than its degree and hence is identically zero, implying  $Q(x, f(x)) \equiv 0$

Finding optimal values for  $m$  and  $l$ . Note that  $m + ld < t$  and  $(m+1)(l+1) + d \binom{l+1}{2} > n$  For a given value  $l$ , one can compute the smallest  $m$  for which the second condition holds By interchanging the second condition one can get  $m$  to be at most  $(n+1 - d \binom{l+1}{2})/2 - 1$  Substituting this value into first condition one can get  $t$  to be at least  $\frac{n+1}{l+1} + \frac{dl}{2}$  Next minimize the above equation of unknown parameter  $l$ . One can do that by taking derivative of the equation and equating that to zero By doing that one will get,  $l = \sqrt{\frac{2(n+1)}{d}} - 1$  Substituting back the  $l$  value into  $m$  and  $t$  one will get  $m \geq \sqrt{\frac{(n+1)d}{2}} - \sqrt{\frac{(n+1)d}{2}} + \frac{d}{2} - 1 = \frac{d}{2} - 1$   $t > \sqrt{\frac{2(n+1)d^2}{d}} - \frac{d}{2} - 1$

## 51.2 Algorithm 2 (Guruswami-Sudan list decoding algorithm)

### 51.2.1 Definition

Consider a  $(n, k)$  Reed-Solomon code over the finite field  $\mathbb{F} = GF(q)$  with evaluation set  $(\alpha_1, \alpha_2, \dots, \alpha_n)$  and a positive integer  $r$ , the Guruswami-Sudan List Decoder accepts a vector  $\beta = (\beta_1, \beta_2, \dots, \beta_n) \in \mathbb{F}^n$  as input, and outputs a list of polynomials of degree  $\leq k$  which are in 1 to 1 correspondence with codewords.

The idea is to add more restrictions on the bi-variate polynomial  $Q(x, y)$  which results in the increment of constraints along with the number of roots.

### 51.2.2 Multiplicity

A bi-variate polynomial  $Q(x, y)$  has a zero of multiplicity  $r$  at  $(0, 0)$  means that  $Q(x, y)$  has no term of degree  $\leq r$ , where the  $x$ -degree of  $f(x)$  is defined as the maximum degree of any  $x$  term in  $f(x)$   $\deg_x f(x) = \max_{i \in I} \{i\}$

For example: Let  $Q(x, y) = y - 4x^2$ .

<https://wiki.cse.buffalo.edu/cse545/sites/wiki.cse.buffalo.edu/cse545/files/76/fig1.jpg>

Hence,  $Q(x, y)$  has a zero of multiplicity 1 at  $(0,0)$ .

Let  $Q(x, y) = y + 6x^2$ .

<https://wiki.cse.buffalo.edu/cse545/sites/wiki.cse.buffalo.edu/cse545/files/76/fig2.jpg>

Hence,  $Q(x, y)$  has a zero of multiplicity 1 at  $(0,0)$ .

Let  $Q(x, y) = (y - 4x^2)(y + 6x^2)$   $Q(x, y) = y^2 + 6x^2y - 4x^2y - 24x^4$

<https://wiki.cse.buffalo.edu/cse545/sites/wiki.cse.buffalo.edu/cse545/files/76/Fig3.jpg>

Hence,  $Q(x, y)$  has a zero of multiplicity 2 at  $(0, 0)$ .

Similarly, if  $Q(x, y) = [(y - \beta) - 4(x - \alpha)^2][(y - \beta) + 6(x - \alpha)^2]$  Then,  $Q(x, y)$  has a zero of multiplicity 2 at  $(\alpha, \beta)$ .

### General definition of multiplicity

$Q(x, y)$  has  $r$  roots at  $(\alpha, \beta)$  if  $Q(x, y)$  has a zero of multiplicity  $r$  at  $(\alpha, \beta)$  when  $(\alpha, \beta) \neq (0, 0)$ .

### 51.2.3 Algorithm

Let the transmitted codeword be  $(f(\alpha_1), f(\alpha_2), \dots, f(\alpha_n))$ ,  $(\alpha_1, \alpha_2, \dots, \alpha_n)$  be the support set of the transmitted codeword & the received word be  $(\beta_1, \beta_2, \dots, \beta_n)$

The algorithm is as follows:

#### • Interpolation step

For a received vector  $(\beta_1, \beta_2, \dots, \beta_n)$ , construct a non-zero bi-variate polynomial  $Q(x, y)$  with  $(1, k)$ -weighted degree of at most  $d$  such that  $Q$  has a zero of multiplicity  $r$  at each of the points  $(\alpha_i, \beta_i)$  where  $1 \leq i \leq n$

$$Q(\alpha_i, \beta_i) = 0$$

#### • Factorization step

Find all the factors of  $Q(x, y)$  of the form  $y - p(x)$  and  $p(\alpha_i) = \beta_i$  for at least  $t$  values of  $i$

where  $0 \leq i \leq n$  &  $p(x)$  is a polynomial of degree  $\leq k$

Recall that polynomials of degree  $\leq k$  are in 1 to 1 correspondence with codewords. Hence, this step outputs the list of codewords.

### 51.2.4 Analysis

#### Interpolation step

**Lemma:** Interpolation step implies  $\binom{r+1}{2}$  constraints on the coefficients of  $a_i$

Let  $Q(x, y) = \sum_{i=0, j=0}^{i=m, j=p} a_{i,j} x^i y^j$  where  $\deg_x Q(x, y) = m$  and  $\deg_y Q(x, y) = p$

Then,  $Q(x + \alpha, y + \beta) = \sum_{u=0, v=0}^r Q_{u,v}(\alpha, \beta) x^u y^v$  .....(Equation 1)

where  $Q_{u,v}(x, y) = \sum_{i=0, j=0}^{i=m, j=p} \binom{i}{u} \binom{j}{v} a_{i,j} x^{i-u} y^{j-v}$

#### Proof of Equation 1:

$$Q(x + \alpha, y + \beta) = \sum_{i,j} a_{i,j} (x + \alpha)^i (y + \beta)^j$$

$$Q(x + \alpha, y + \beta) = \sum_{i,j} a_{i,j} \left( \sum_u \binom{i}{u} x^u \alpha^{i-u} \right) \left( \sum_v \binom{j}{v} y^v \beta^{j-v} \right)$$

$$Q(x + \alpha, y + \beta) = \sum_{u,v} x^u y^v \left( \sum_{i,j} \binom{i}{u} \binom{j}{v} a_{i,j} \alpha^{i-u} \beta^{j-v} \right)$$

$$Q(x + \alpha, y + \beta) = \sum_{u,v} Q_{u,v}(\alpha, \beta) x^u y^v$$

#### Proof of Lemma:

The polynomial  $Q(x, y)$  has a zero of multiplicity  $r$  at  $(\alpha, \beta)$  if

$$Q_{u,v}(\alpha, \beta) \equiv 0 \text{ such that } 0 \leq u + v \leq r - 1$$

$u$  can take  $r - v$  values as  $0 \leq v \leq r - 1$ .

Thus, the total number of constraints is

$$\sum_{v=0}^{r-1} r - v = \binom{r+1}{2}$$

Thus,  $\binom{r+1}{2}$  number of selections can be made for  $(u, v)$  and each selection implies constraints on the coefficients of  $a_i$

### 51.2.5 Factorization step

#### Proposition:

$Q(x, p(x)) \equiv 0$  if  $y - p(x)$  is a factor of  $Q(x, y)$

#### Proof:

Since,  $y - p(x)$  is a factor of  $Q(x, y)$ ,  $Q(x, y)$  can be represented as

$$Q(x, y) = L(x, y)(y - p(x)) + R(x)$$

where,  $L(x, y)$  is the quotient obtained when  $Q(x, y)$  is divided by  $y - p(x)$   $R(x)$  is the remainder

Now, if  $y$  is replaced by  $p(x)$ ,  $Q(x, p(x)) \equiv 0$ , only if  $R(x) \equiv 0$

#### Theorem:

If  $p(\alpha) = \beta$ , then  $(x - \alpha)^r$  is a factor of  $Q(x, p(x))$

#### Proof:

$$Q(x, y) = \sum_{u,v} Q_{u,v}(\alpha, \beta) (x - \alpha)^u (y - \beta)^v$$
 .....From Equation 2

$$Q(x, p(x)) = \sum_{u,v} Q_{u,v}(\alpha, \beta) (x - \alpha)^u (p(x) - \beta)^v$$

$$\text{Given, } p(\alpha) = \beta \pmod{x - \alpha} = 0$$

$$\text{Hence, } (x - \alpha)^u (p(x) - \beta)^v \pmod{x - \alpha}^{u+v} = 0$$

Thus,  $(x - \alpha)^r$  is a factor of  $Q(x, p(x))$ .

As proved above,

$$t \cdot r > D$$

$$t > \frac{D}{r}$$

$\frac{D(D+2)}{2(k-1)} > n \binom{r+1}{2}$  where LHS is the upper bound on the number of coefficients of  $Q(x, y)$  and RHS is the earlier proved Lemma.

$$D = \sqrt{knr(r-1)}$$

$$\text{Therefore, } t = \left\lceil \sqrt{kn(1 - \frac{1}{r})} \right\rceil$$

Substitute  $r = 2kn$ ,

$$t > \left\lceil \sqrt{kn - \frac{1}{2}} \right\rceil > \left\lceil \sqrt{kn} \right\rceil$$

Hence proved, that Guruswami–Sudan List Decoding Algorithm can list decode Reed-Solomon(RS) codes up to  $1 - \sqrt{R}$  errors.

## 51.3 References

- <http://www.cse.buffalo.edu/~{ }atri/courses/coding-theory/>
- <http://www.cs.cmu.edu/~{ }venkatg/pubs/papers/listdecoding-NOW.pdf>
- <http://www.mendeley.com/research/algebraic-softdecision-decoding-reedsolomon-codes/>
- R. J. McEliece. The Guruswami-Sudan Decoding Algorithm for Reed-Solomon Codes.
- M Sudan. Decoding of Reed Solomon codes beyond the error-correction bound.



# Chapter 52

## GV-linear-code

In **coding theory**, the bound of parameters such as rate  $R$ , relative distance, **block length**, etc. is usually concerned. Here **Gilbert–Varshamov bound theorem** claims the lower bound of the rate of the general code. Gilbert–Varshamov bound is the best in terms of relative distance for codes over alphabets of size less than 49.

### 52.1 Gilbert–Varshamov bound theorem

**Theorem:** Let  $q \geq 2$ . For every  $0 \leq \delta < 1 - \frac{1}{q}$  and  $0 < \varepsilon \leq 1 - H_q(\delta)$ , there exists a code with rate  $R \geq 1 - H_q(\delta) - \varepsilon$  and relative distance  $\delta$ .

Here  $H_q$  is the  $q$ -ary entropy function defined as follows:

$$H_q(x) = x \log_q(q-1) - x \log_q x - (1-x) \log_q(1-x).$$

The above result was proved by **Edgar Gilbert** for general code using the **greedy method** as **here**. For **linear code**, **Rom Varshamov** proved using the **probabilistic method** for the random linear code. This proof will be shown in the following part.

#### High-level proof:

To show the existence of the linear code that satisfies those constraints, the **probabilistic method** is used to construct the random linear code. Specifically the linear code is chosen randomly by choosing the **random** generator matrix  $G$  in which the element is chosen uniformly over the field  $\mathbb{F}_q^n$ . Also the **Hamming distance** of the linear code is equal to the minimum weight of the **codeword**. So to prove that the linear code generated by  $G$  has Hamming distance  $d$ , we will show that for any  $m \in \mathbb{F}_q^k \setminus \{0\}$ ,  $wt(mG) \geq d$ . To prove that, we prove the opposite one; that is, the probability that the linear code generated by  $G$  has the Hamming distance less than  $d$  is exponentially small in  $n$ . Then by probabilistic method, there exists the linear code satisfying the theorem.

#### Formal proof:

By using the probabilistic method, to show that there exists a linear code that has a Hamming distance greater than  $d$ , we will show that the **probability** that the random linear code having the distance less than  $d$  is exponentially small in  $n$ .

We know that the linear code is defined using the **generator matrix**. So we use the “random generator matrix”  $G$  as a mean to describe the randomness of the linear code. So a random generator matrix  $G$  of size  $kn$  contains  $kn$  elements which are chosen independently and uniformly over the field  $\mathbb{F}_q$ .

Recall that in a **linear code**, the distance equals the minimum weight of the non-zero codeword. Let  $wt(y)$  be the weight of the codeword  $y$ . So

$$\begin{aligned} P &= \Pr_{\text{random } G} (\text{by generated code linear } G \text{ distance has } < d) \\ &= \Pr_{\text{random } G} (\text{codeword non-zero a exists there } y \text{ by generated code linear } G \text{ such that } wt(y) < d) \\ &= \Pr_{\text{random } G} (\text{exists there } 0 \neq m \in \mathbb{F}_q^k \text{ that such } wt(mG) < d) \end{aligned}$$

The last equality follows from the definition: if a codeword  $y$  belongs to a linear code generated by  $G$ , then  $y = mG$  for some vector  $m \in \mathbb{F}_q^k$ .

By **Boole’s inequality**, we have:

$$P \leq \sum_{0 \neq m \in \mathbb{F}_q^k} \Pr_{\text{random } G} (wt(mG) < d)$$

Now for a given message  $0 \neq m \in \mathbb{F}_q^k$ , we want to compute

$$W = \Pr_{\text{random } G} (wt(mG) < d).$$

Let  $\Delta(m_1, m_2)$  be a Hamming distance of two messages  $m_1$  and  $m_2$ . Then for any message  $m$ , we have:  $wt(m) = \Delta(0, m)$ . Therefore:

$$W = \sum_{\{y \in \mathbb{F}_q^n \mid \Delta(0, y) \leq d-1\}} \Pr_{\text{random } G} (mG = y)$$

Due to the randomness of  $G$ ,  $mG$  is a uniformly random vector from  $\mathbb{F}_q^n$ . So

$$\Pr_{\text{random } G}(mG = y) = q^{-n}$$

Let  $\text{Vol}_q(r, n)$  is a volume of Hamming ball with the radius  $r$ . Then:<sup>[1]</sup>

$$P \leq q^k W = q^k \left( \frac{\text{Vol}_q(d-1, n)}{q^n} \right) \leq q^k \left( \frac{q^{nH_q(\delta)}}{q^n} \right) = q^k q^{-n(1-H_q(\delta))}$$

By choosing  $k = (1 - H_q(\delta) - \varepsilon)n$ , the above inequality becomes

$$P \leq q^{-\varepsilon n}$$

Finally  $q^{-\varepsilon n} \ll 1$ , which is exponentially small in  $n$ , that is what we want before. Then by the probabilistic method, there exists a linear code  $C$  with relative distance  $\delta$  and rate  $R$  at least  $(1 - H_q(\delta) - \varepsilon)$ , which completes the proof.

## 52.2 Comments

1. The Varshamov construction above is not explicit; that is, it does not specify the deterministic method to construct the linear code that satisfies the Gilbert–Varshamov bound. The naive way that we can do is to go over all the generator matrices  $G$  of size  $kn$  over the field  $\mathbb{F}_q$  and check if that linear code has the satisfied Hamming distance. That leads to the exponential time algorithm to implement it.
2. We also have a Las Vegas construction that takes a random linear code and checks if this code has good Hamming distance. Nevertheless, this construction has the exponential running time.

## 52.3 See also

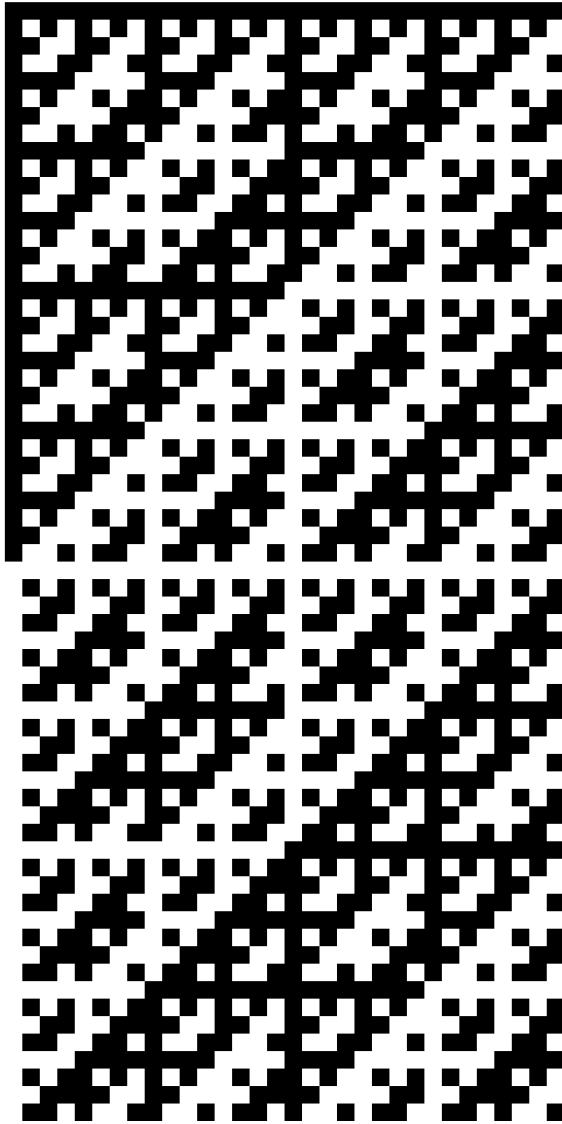
1. Gilbert–Varshamov bound due to Gilbert construction for the general code
2. Hamming Bound
3. Probabilistic method

## 52.4 References

- [1] The later inequality comes from the upper bound of the Volume of Hamming ball

1. Lecture 11: Gilbert–Varshamov Bound. Coding Theory Course. Professor Atri Rudra
2. Lecture 9: Bounds on the Volume of Hamming Ball. Coding Theory Course. Professor Atri Rudra
3. Coding Theory's Notes: Gilbert–Varshamov Bound. Venkatesan ♦Guruswami

# Hadamard code



Matrix of the Punctured Hadamard code (32, 6, 16) for the Reed-Muller code (1, 5) of the NASA space probe Mariner 9

The **Hadamard code** is an error-correcting code that is used for error detection and correction when transmitting messages over very noisy or unreliable channels. In 1971, the code was used to transmit photos of Mars back to Earth from the NASA space probe **Mariner 9**.<sup>[1]</sup> Because of its unique mathematical properties, the Hadamard

[illegible]

**XOR** operations  
Here the white fields stand for 0  
and the red fields for 1

code is not only used by engineers, but also intensely studied in **coding theory**, **mathematics**, and **theoretical computer science**. The Hadamard code is named after the French mathematician **Jacques Hadamard**. It is also known under the names **Walsh code**, **Walsh family**,<sup>[2]</sup> and **Walsh–Hadamard code**<sup>[3]</sup> in recognition of the American mathematician **Joseph Leonard Walsh**.

The Hadamard code is an example of a **linear code** over a **binary alphabet** that maps messages of length  $k$  to codewords of length  $2^k$ . It is unique in that each non-zero codeword has a **Hamming weight** of exactly  $2^{k-1}$ , which implies that the **distance** of the code is also  $2^{k-1}$ . In standard **coding theory notation** for **block codes**, the Hadamard code is a  $[2^k, k, 2^{k-1}]_2$ -code, that is, it is a **linear code** over a **binary alphabet**, has **block length**  $2^k$ , **message length** (or dimension)  $k$ , and **minimum distance**  $2^k/2$ . The block length is very large compared to the message length, but on the other hand, errors can be corrected even in extremely noisy conditions. The **punctured Hadamard code** is a slightly improved version of the Hadamard code; it is a  $[2^k, k+1, 2^{k-1}]_2$ -code and thus has a slightly better **rate** while maintaining the relative distance of  $1/2$ , and is thus preferred in practical applications. The punctured Hadamard code is the same as the first order **Reed–Muller code** over the binary

alphabet.<sup>[4]</sup>

Normally, Hadamard codes are based on Sylvester's construction of Hadamard matrices, but the term "Hadamard code" is also used to refer to codes constructed from arbitrary Hadamard matrices, which are not necessarily of Sylvester type. In general, such a code is not linear. Such codes were first constructed by R. C. Bose and S. S. Shrikhande in 1959.<sup>[5]</sup> If  $n$  is the size of the Hadamard matrix, the code has parameters  $(n, 2n, n/2)_2$ , meaning it is a not-necessarily-linear binary code with  $2n$  codewords of block length  $n$  and minimal distance  $n/2$ . The construction and decoding scheme described below apply for general  $n$ , but the property of linearity and the identification with Reed–Muller codes require that  $n$  be a power of 2 and that the Hadamard matrix be equivalent to the matrix constructed by Sylvester's method.

The Hadamard code is a locally decodable code, which provides a way to recover parts of the original message with high probability, while only looking at a small fraction of the received word. This gives rise to applications in computational complexity theory and particularly in the design of probabilistically checkable proofs. Since the relative distance of the Hadamard code is  $1/2$ , normally one can only hope to recover from at most a  $1/4$  fraction of error. Using list decoding, however, it is possible to compute a short list of possible candidate messages as long as fewer than  $\frac{1}{2} - \epsilon$  of the bits in the received word have been corrupted.

In code division multiple access (CDMA) communication, the Hadamard code is referred to as Walsh Code, and is used to define individual communication channels. It is usual in the CDMA literature to refer to codewords as "codes". Each user will use a different codeword, or "code", to modulate their signal. Because Walsh codewords are mathematically orthogonal, a Walsh-encoded signal appears as random noise to a CDMA capable mobile terminal, unless that terminal uses the same codeword as the one used to encode the incoming signal.<sup>[6]</sup>

## 53.1 History

*Hadamard code* is the name that is most commonly used for this code in the literature. However, in modern use these error correcting codes are referred to as Walsh–Hadamard codes.

There is a reason for this:

Jacques Hadamard did not invent the code himself, but he defined Hadamard matrices around 1893, long before the first error-correcting code, the Hamming code, was developed in the 1940s.

The Hadamard code is based on Hadamard matrices, and while there are many different Hadamard matrices that could be used here, normally only Sylvester's construction of Hadamard matrices is used to obtain the codewords of

the Hadamard code.

James Joseph Sylvester developed his construction of Hadamard matrices in 1867, which actually predates Hadamard's work on Hadamard matrices. Hence the name *Hadamard code* is not undisputed and sometimes the code is called *Walsh code*, honoring the American mathematician Joseph Leonard Walsh.

A Hadamard code was used during the 1971 Mariner 9 mission to correct for picture transmission errors. The data words used during this mission were 6 bits long, which represented 64 grayscale values.

Because of limitations of the quality of the alignment of the transmitter at the time (due to Doppler Tracking Loop issues) the maximum useful data length was about 30 bits. Instead of using a repetition code, a  $[32, 6, 16]$  Hadamard code was used.

Errors of up to 7 bits per word could be corrected using this scheme. Compared to a 5-repetition code, the error correcting properties of this Hadamard code are much better, yet its rate is comparable. The efficient decoding algorithm was an important factor in the decision to use this code.

The circuitry used was called the "Green Machine". It employed the fast Fourier transform which can increase the decoding speed by a factor of three. Since the 1990s use of this code by space programs has more or less ceased, and the Deep Space Network does not support this error correction scheme for its dishes that are greater than 26 m.

## 53.2 Constructions

While all Hadamard codes are based on Hadamard matrices, the constructions differ in subtle ways for different scientific fields, authors, and uses. Engineers, who use the codes for data transmission, and coding theorists, who analyse extremal properties of codes, typically want the rate of the code to be as high as possible, even if this means that the construction becomes mathematically slightly less elegant.

On the other hand, for many applications of Hadamard codes in theoretical computer science it is not so important to achieve the optimal rate, and hence simpler constructions of Hadamard codes are preferred since they can be analyzed more elegantly.

### 53.2.1 Construction using inner products

When given a binary message  $x \in \{0, 1\}^k$  of length  $k$ , the Hadamard code encodes the message into a codeword  $\text{Had}(x)$  using an encoding function  $\text{Had} : \{0, 1\}^k \rightarrow \{0, 1\}^{2^k}$ . This function makes use of the inner product  $\langle x, y \rangle$  of two vectors  $x, y \in \{0, 1\}^k$ , which is defined as

follows:

$$\langle x, y \rangle = \sum_{i=1}^k x_i y_i \bmod 2.$$

Then the Hadamard encoding of  $x$  is defined as the sequence of *all* inner products with  $x$ :

$$\text{Had}(x) = \left( \langle x, y \rangle \right)_{y \in \{0,1\}^k}$$

As mentioned above, the *punctured* Hadamard code is used in practice since the Hadamard code itself is somewhat wasteful. This is because, if the first bit of  $y$  is zero,  $y_1 = 0$ , then the inner product contains no information whatsoever about  $x_1$ , and hence, it is impossible to fully decode  $x$  from those positions of the codeword alone. On the other hand, when the codeword is restricted to the positions where  $y_1 = 1$ , it is still possible to fully decode  $x$ . Hence it makes sense to restrict the Hadamard code to these positions, which gives rise to the *punctured* Hadamard encoding of  $x$ ; that is,

$$\text{pHad}(x) = \left( \langle x, y \rangle \right)_{y \in \{1\} \times \{0,1\}^{k-1}}.$$

### 53.2.2 Construction using a generator matrix

The Hadamard code is a linear code, and all linear codes can be generated by a generator matrix  $G$ . This is a matrix such that  $\text{Had}(x) = x \cdot G$  holds for all  $x \in \{0,1\}^k$ , where the message  $x$  is viewed as a row vector and the vector-matrix product is understood in the **vector space** over the **finite field**  $\mathbb{F}_2$ . In particular, an equivalent way to write the inner product definition for the Hadamard code arises by using the generator matrix whose columns consist of *all* strings  $y$  of length  $k$ , that is,

$$G = \begin{pmatrix} \uparrow & \uparrow & & \uparrow \\ y_1 & y_2 & \dots & y_{2^k} \\ \downarrow & \downarrow & & \downarrow \end{pmatrix}.$$

where  $y_i \in \{0,1\}^k$  is the  $i$ -th binary vector in **lexicographical order**. For example, the generator matrix for the Hadamard code of dimension  $k = 3$  is:

$$G = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

The matrix  $G$  is a  $(k \times 2^k)$ -matrix and gives rise to the **linear operator**  $\text{Had} : \{0,1\}^k \rightarrow \{0,1\}^{2^k}$ .

The generator matrix of the *punctured* Hadamard code is obtained by restricting the matrix  $G$  to the columns whose

first entry is one. For example, the generator matrix for the punctured Hadamard code of dimension  $k = 3$  is:

$$G' = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}.$$

Then  $\text{pHad} : \{0,1\}^k \rightarrow \{0,1\}^{2^{k-1}}$  is a linear mapping with  $\text{pHad}(x) = x \cdot G'$ .

For general  $k$ , the generator matrix of the punctured Hadamard code is a **parity-check matrix** for the **extended Hamming code** of length  $2^{k-1}$  and dimension  $2^{k-1} - k$ , which makes the punctured Hadamard code the **dual code** of the extended Hamming code. Hence an alternative way to define the Hadamard code is in terms of its parity-check matrix: the parity-check matrix of the Hadamard code is equal to the generator matrix of the Hamming code.

### 53.2.3 Construction using general Hadamard matrices

Generalized Hadamard codes are obtained from an  $n$ -by- $n$  **Hadamard matrix**  $H$ . In particular, the  $2n$  codewords of the code are the rows of  $H$  and the rows of  $-H$ . To obtain a code over the alphabet  $\{0,1\}$ , the mapping  $-1 \mapsto 1$ ,  $1 \mapsto 0$ , or, equivalently,  $x \mapsto (1-x)/2$ , is applied to the matrix elements. That the minimum distance of the code is  $n/2$  follows from the defining property of Hadamard matrices, namely that their rows are mutually orthogonal. This implies that two distinct rows of a Hadamard matrix differ in exactly  $n/2$  positions, and, since negation of a row does not affect orthogonality, that any row of  $H$  differs from any row of  $-H$  in  $n/2$  positions as well, except when the rows correspond, in which case they differ in  $n$  positions.

To get the punctured Hadamard code above with  $n = 2^{k-1}$ , the chosen Hadamard matrix  $H$  has to be of Sylvester type, which gives rise to a message length of  $\log_2(2n) = k$ .

## 53.3 Distance

The distance of a code is the minimum **Hamming distance** between any two distinct codewords, i.e., the minimum number of positions at which two distinct codewords differ. Since the Walsh-Hadamard code is a **linear code**, the distance is equal to the minimum **Hamming weight** among all of its non-zero codewords. All non-zero codewords of the Walsh-Hadamard code have a **Hamming weight** of exactly  $2^{k-1}$  by the following argument.

Let  $x \in \{0,1\}^k$  be a non-zero message. Then the following value is exactly equal to the fraction of positions

in the codeword that are equal to one:

$$\Pr_{y \in \{0,1\}^k} [(\text{Had}(x))_y = 1] = \Pr_{y \in \{0,1\}^k} [\langle x, y \rangle = 1].$$

The fact that the latter value is exactly  $1/2$  is called the *random subsum principle*. To see that it is true, assume without loss of generality that  $x_1 = 1$ . Then, when conditioned on the values of  $y_2, \dots, y_k$ , the event is equivalent to  $y_1 \cdot x_1 = b$  for some  $b \in \{0, 1\}$  depending on  $x_2, \dots, x_k$  and  $y_2, \dots, y_k$ . The probability that  $y_1 = b$  happens is exactly  $1/2$ . Thus, in fact, *all* non-zero codewords of the Hadamard code have relative Hamming weight  $1/2$ , and thus, its relative distance is  $1/2$ .

The relative distance of the *punctured* Hadamard code is  $1/2$  as well, but it no longer has the property that every non-zero codeword has weight exactly  $1/2$  since the all 1s vector  $1^{2^k-1}$  is a codeword of the punctured Hadamard code. This is because the vector  $x = 10^{k-1}$  encodes to  $\text{pHad}(10^{k-1}) = 1^{2^k-1}$ . Furthermore, whenever  $x$  is non-zero and not the vector  $10^{k-1}$ , the random subsum principle applies again, and the relative weight of  $\text{Had}(x)$  is exactly  $1/2$ .

## 53.4 Local decodability

A **locally decodable** code is a code that allows a single bit of the original message to be recovered with high probability by only looking at a small portion of the received word.

A code is  $q$ -query **locally decodable** if a message bit,  $x_i$ , can be recovered by checking  $q$  bits of the received word. More formally, a code,  $C : \{0, 1\}^k \rightarrow \{0, 1\}^n$ , is  $(q, \delta \geq 0, \epsilon \geq 0)$ -locally decodable, if there exists a probabilistic decoder,  $D : \{0, 1\}^n \rightarrow \{0, 1\}^k$ , such that (Note:  $\Delta(x, y)$  represents the *Hamming distance* between vectors  $x$  and  $y$ ):

$\forall x \in \{0, 1\}^k, \forall y \in \{0, 1\}^n, \Delta(y, C(x)) \leq \delta n$  implies that  $\Pr[D(y)_i = x_i] \geq \frac{1}{2} + \epsilon, \forall i \in [k]$

**Theorem 1:** The Walsh–Hadamard code is  $(2, \delta, \frac{1}{2} - 2\delta)$ -locally decodable for  $0 \leq \delta \leq \frac{1}{4}$ .

**Lemma 1:** For all codewords,  $c$  in a Walsh–Hadamard code,  $C$ ,  $c_i + c_j = c_{i+j}$ , where  $c_i, c_j$  represent the bits in  $c$  in positions  $i$  and  $j$  respectively, and  $c_{i+j}$  represents the bit at position  $(i + j)$ .

### 53.4.1 Proof of lemma 1

Let  $C(x) = c = (c_0, \dots, c_{2^n-1})$  be the codeword in  $C$  corresponding to message  $x$ .

Let  $G = \begin{pmatrix} \uparrow & \uparrow & & \uparrow \\ g_0 & g_1 & \dots & g_{2^n-1} \\ \downarrow & \downarrow & & \downarrow \end{pmatrix}$  be the generator matrix of  $C$ .

By definition,  $c_i = x \cdot g_i$ . From this,  $c_i + c_j = x \cdot g_i + x \cdot g_j = x \cdot (g_i + g_j)$ . By the construction of  $G$ ,  $g_i + g_j = g_{i+j}$ . Therefore, by substitution,  $c_i + c_j = x \cdot g_{i+j} = c_{i+j}$ .

### 53.4.2 Proof of theorem 1

To prove theorem 1 we will construct a decoding algorithm and prove its correctness.

#### Algorithm

**Input:** Received word  $y = (y_0, \dots, y_{2^n-1})$

For each  $i \in \{1, \dots, n\}$ :

1. Pick  $j \in \{0, \dots, 2^n - 1\}$  uniformly at random
2. Pick  $k \in \{0, \dots, 2^n - 1\}$  such that  $j + k = e_i$  where  $j + k$  is the bitwise *xor* of  $j$  and  $k$ .
3.  $x_i \leftarrow y_j + y_k$

**Output:** Message  $x = (x_1, \dots, x_n)$

#### Proof of correctness

For any message,  $x$ , and received word  $y$  such that  $y$  differs from  $c = C(x)$  on at most  $\delta$  fraction of bits,  $x_i$  can be decoded with probability at least  $\frac{1}{2} + (1 - 2\delta)$ .

By lemma 1,  $c_j + c_k = c_{j+k} = x \cdot g_{j+k} = x \cdot e_i = x_i$ . Since  $j$  and  $k$  are picked uniformly, the probability that  $y_j \neq c_j$  is at most  $\delta$ . Similarly, the probability that  $y_k \neq c_k$  is at most  $\delta$ . By the **union bound**, the probability that either  $y_j$  or  $y_k$  do not match the corresponding bits in  $c$  is at most  $2\delta$ . If both  $y_j$  and  $y_k$  correspond to  $c$ , then lemma 1 will apply, and therefore, the proper value of  $x_i$  will be computed. Therefore, the probability  $x_i$  is decoded properly is at least  $1 - 2\delta$ . Therefore,  $\epsilon = \frac{1}{2} - 2\delta$  and for  $\epsilon$  to be positive,  $0 \leq \delta \leq \frac{1}{4}$ .

Therefore, the Walsh–Hadamard code is  $(2, \delta, \frac{1}{2} - 2\delta)$  locally decodable for  $0 \leq \delta \leq \frac{1}{4}$ .

## 53.5 Optimality


For  $k \leq 7$  the linear Hadamard codes have been proven optimal in the sense of minimum distance.<sup>[7]</sup>

### 53.6 See also

- **Zadoff–Chu sequence** — improve over the Walsh–Hadamard codes



## 53.7 Notes

- [1] <http://www.mcs.csueastbay.edu/~{ }malek/TeX/Hadamard.pdf>
- [2] See, e.g., Amadei, Manzoli & Merani (2002)
- [3] See, e.g., Arora & Barak (2009, Section 19.2.2).
- [4] See, e.g., Guruswami (2009, p. 3).
- [5] Bose, R.C.; Shrikhande, S.S. (1959). “A note on a result in the theory of code construction”. *Information and Control*. **2** (2): 183–194. CiteSeerX 10.1.1.154.2879 . doi:10.1016/S0019-9958(59)90376-6.
- [6] “CDMA Tutorial: Intuitive Guide to Principles of Communications” (PDF). Complex to Real. Retrieved 4 August 2011.
- [7] Jaffe, David B.; Bouyukliev, Iliya, *Optimal binary linear codes of dimension at most seven*

## 53.8 References

- Amadei, M.; Manzoli, U.; Merani, M.L. (2002), “On the assignment of Walsh and quasi-orthogonal codes in a multicarrier DS-CDMA system with multiple classes of users”, *Global Telecommunications Conference, 2002. GLOBECOM'02. IEEE*, **1**, IEEE, pp. 841–5, doi:10.1109/GLOCOM.2002.1188196, ISBN 0-7803-7632-3
- Arora, Sanjeev; Barak, Boaz (2009), *Computational Complexity: A Modern Approach*, Cambridge University Press, ISBN 978-0-521-42426-4
- Guruswami, Venkatesan (2009), *List decoding of binary codes* (PDF)
- Rudra, Atri, “Hamming code and Hamming bound” (PDF), *Lecture notes*



# Chapter 54

## Hamming bound

In mathematics and computer science, in the field of coding theory, the **Hamming bound** is a limit on the parameters of an arbitrary block code: it is also known as the **sphere-packing bound** or the **volume bound** from an interpretation in terms of packing balls in the Hamming metric into the space of all possible words. It gives an important limitation on the efficiency with which any error-correcting code can utilize the space in which its code words are embedded. A code which attains the Hamming bound is said to be a **perfect code**.

### 54.1 Background on error-correcting codes

An original message and an encoded version are both composed in an alphabet of  $q$  letters. Each code word contains  $n$  letters. The original message (of length  $m$ ) is shorter than  $n$  letters. The message is converted into an  $n$ -letter codeword by an encoding algorithm, transmitted over a noisy channel, and finally decoded by the receiver. The decoding process interprets a garbled codeword, referred to as simply a *word*, as the valid codeword “nearest” the  $n$ -letter received string.

Mathematically, there are exactly  $q^m$  possible messages of length  $m$ , and each message can be regarded as a vector of length  $m$ . The encoding scheme converts an  $m$ -dimensional vector into an  $n$ -dimensional vector. Exactly  $q^m$  valid codewords are possible, but any one of  $q^n$  garbled codewords (words) can be received, because the noisy channel might distort one or more of the  $n$  letters while the codeword is being transmitted.

### 54.2 Statement of the bound

Let  $A_q(n, d)$  denote the maximum possible size of a  $q$ -ary block code  $C$  of length  $n$  and minimum Hamming distance  $d$  (a  $q$ -ary block code of length  $n$  is a subset of the strings of  $\mathcal{A}_q^n$ , where the alphabet set  $\mathcal{A}_q$  has  $q$  elements).

Then, the Hamming bound is:

$$A_q(n, d) \leq \frac{q^n}{\sum_{k=0}^t \binom{n}{k} (q-1)^k}$$

where

$$t = \left\lfloor \frac{d-1}{2} \right\rfloor.$$

### 54.3 Proof

By definition of  $d$ , if at most  $t = \lfloor \frac{1}{2}(d-1) \rfloor$  errors are made during transmission of a codeword then minimum distance decoding will decode it correctly (i.e., it decodes the received word as the codeword that was sent). Thus the code is said to be capable of correcting  $t$  errors.

For a given codeword  $c \in C$ , consider the ball of radius  $t$  around  $c$ . Every pair of balls (Hamming spheres) are non-intersecting by the  $t$ -error-correcting property, and each ball contains (in other words, the volume of the ball)  $m$  words. Since we may allow (or choose) up to  $t$  of the  $n$  components of a word to deviate (from the value of the corresponding component of the ball's centre, which is a codeword) to one of  $(q-1)$  possible other values (recall, the code is  $q$ -ary: it takes values in  $\mathcal{A}_q^n$ ), we can define:

$$m = \sum_{k=0}^t \binom{n}{k} (q-1)^k$$

Since  $A_q(n, d)$  is the maximum total number of codewords in  $C$ , and thus the greatest number of balls, and no two balls have a word in common, by taking the union of the words in balls centered at codewords we observe that the resulting set of words, each counted precisely once, is a subset of  $\mathcal{A}_q^n$  (where  $|\mathcal{A}_q^n| = q^n$  words) and deduce:

$$A_q(n, d) \times m = A_q(n, d) \times \sum_{k=0}^t \binom{n}{k} (q-1)^k \leq q^n.$$

Whence:

$$A_q(n, d) \leq \frac{q^n}{\sum_{k=0}^t \binom{n}{k} (q-1)^k}.$$

## 54.4 Covering radius and packing radius

Main article: [Covering radius](#)

For an  $A_q(n, d)$  code  $C$  (a subset of  $\mathcal{A}_q^n$ ), the *covering radius* of  $C$  is the smallest value of  $r$  such that every element of  $\mathcal{A}_q^n$  is contained in at least one ball of radius  $r$  centered at each codeword of  $C$ . The *packing radius* of  $C$  is the largest value of  $s$  such that the set of balls of radius  $s$  centered at each codeword of  $C$  are mutually disjoint.

From the proof of the Hamming bound, it can be seen that for  $t = \lfloor \frac{1}{2}(d-1) \rfloor$ , we have:

$$s \leq t \text{ and } t \leq r.$$

Therefore,  $s \leq r$  and if equality holds then  $s = r = t$ . The case of equality means that the Hamming bound is attained.

## 54.5 Perfect codes

Codes that attain the Hamming bound are called **perfect codes**. Examples include codes that have only one codeword, and codes that are the whole of  $\mathcal{A}_q^n$ . Another example is given by the *repeat codes*, where each symbol of the message is repeated an odd fixed number of times to obtain a codeword where  $q = 2$ . All of these examples are often called the *trivial* perfect codes. In 1973, it was proved that any non-trivial perfect code over a prime-power alphabet has the parameters of a [Hamming code](#) or a [Golay code](#).<sup>[1]</sup>

A perfect code may be interpreted as one in which the balls of Hamming radius  $t$  centered on codewords exactly fill out the space ( $t$  is the covering radius = packing radius). A **quasi-perfect code** is one in which the balls of Hamming radius  $t$  centered on codewords are disjoint and the balls of radius  $t+1$  cover the space, possibly with some overlaps.<sup>[2]</sup> Another way to say this is that a code is *quasi-perfect* if its covering radius is one greater than its packing radius.<sup>[3]</sup>

## 54.6 See also

- [Griesmer bound](#)
- [Singleton bound](#)
- [Gilbert-Varshamov bound](#)
- [Plotkin bound](#)
- [Johnson bound](#)
- [Rate-distortion theory](#)

## 54.7 Notes

- [1] Hill (1988) p. 102
- [2] McWilliams and Sloane, p. 19
- [3] Roman 1992, pg. 140

## 54.8 References

- Raymond Hill (1988). *A First Course In Coding Theory*. Oxford University Press. ISBN 0-19-853803-0.
- F.J. MacWilliams; N.J.A. Sloane (1977). *The Theory of Error-Correcting Codes*. North-Holland. ISBN 0-444-85193-3.
- Vera Pless (1982). *Introduction to the Theory of Error-Correcting Codes*. John Wiley & Sons. ISBN 0-471-08684-3.
- Roman, Steven (1992), *Coding and Information Theory*, GTM, **134**, New York: Springer-Verlag, ISBN 0-387-97812-7
- J.H. van Lint (1992). *Introduction to Coding Theory*. GTM. **86** (2nd ed.). Springer-Verlag. ISBN 3-540-54894-7.
- J.H. van Lint (1975). "A survey of perfect codes". *Rocky Mountain Journal of Mathematics*. **5** (2): 199–224. doi:10.1216/RMJ-1975-5-2-199.
- P. J. Cameron; J. A. Thas; S. E. Payne (1976). "Polarities of generalized hexagons and perfect codes". **5**: 525–528. doi:10.1007/BF00150782.

## Chapter 55

# Hamming code

In telecommunication, **Hamming codes** are a family of linear error-correcting codes that generalize the Hamming(7,4)-code, and were invented by Richard Hamming in 1950. Hamming codes can detect up to two-bit errors or correct one-bit errors without detection of uncorrected errors. By contrast, the simple parity code cannot correct errors, and can detect only an odd number of bits in error. Hamming codes are **perfect codes**, that is, they achieve the highest possible rate for codes with their block length and minimum distance of three.<sup>[1]</sup>

In mathematical terms, Hamming codes are a class of binary linear codes. For each integer  $r \geq 2$  there is a code with block length  $n = 2^r - 1$  and message length  $k = 2^r - r - 1$ . Hence the rate of Hamming codes is  $R = k / n = 1 - r / (2^r - 1)$ , which is the highest possible for codes with minimum distance of three (i.e., the minimal number of bit changes needed to go from any code word to any other code word is three) and block length  $2^r - 1$ . The parity-check matrix of a Hamming code is constructed by listing all columns of length  $r$  that are non-zero, which means that the dual code of the Hamming code is the shortened Hadamard code. The parity-check matrix has the property that any two columns are pairwise linearly independent.

Due to the limited redundancy that Hamming codes add to the data, they can only detect and correct errors when the error rate is low. This is the case in computer memory (**ECC memory**), where bit errors are extremely rare and Hamming codes are widely used. In this context, an extended Hamming code having one extra parity bit is often used. Extended Hamming codes achieve a Hamming distance of four, which allows the decoder to distinguish between when at most one one-bit error occurs and when any two-bit errors occur. In this sense, extended Hamming codes are single-error correcting and double-error detecting, abbreviated as **SECDED**.

### 55.1 History

Richard Hamming, the inventor of Hamming codes, worked at Bell Labs in the 1940s on the Bell Model V computer, an electromechanical relay-based machine

with cycle times in seconds. Input was fed in on punched cards, which would invariably have read errors. During weekdays, special code would find errors and flash lights so the operators could correct the problem. During after-hours periods and on weekends, when there were no operators, the machine simply moved on to the next job.

Hamming worked on weekends, and grew increasingly frustrated with having to restart his programs from scratch due to the unreliability of the card reader. Over the next few years, he worked on the problem of error-correction, developing an increasingly powerful array of algorithms. In 1950, he published what is now known as Hamming Code, which remains in use today in applications such as **ECC memory**.

#### 55.1.1 Codes predating Hamming

A number of simple error-detecting codes were used before Hamming codes, but none were as effective as Hamming codes in the same overhead of space.

#### Parity

Main article: **Parity bit**

Parity adds a single bit that indicates whether the number of ones (bit-positions with values of one) in the preceding data was **even** or **odd**. If an odd number of bits is changed in transmission, the message will change parity and the error can be detected at this point; however, the bit that changed may have been the parity bit itself. The most common convention is that a parity value of one indicates that there is an odd number of ones in the data, and a parity value of zero indicates that there is an even number of ones. If the number of bits changed is even, the check bit will be valid and the error will not be detected.

Moreover, parity does not indicate which bit contained the error, even when it can detect it. The data must be discarded entirely and re-transmitted from scratch. On a noisy transmission medium, a successful transmission could take a long time or may never occur. However, while the quality of parity checking is poor, since it uses

only a single bit, this method results in the least overhead.

### Two-out-of-five code

Main article: [Two-out-of-five code](#)

A two-out-of-five code is an encoding scheme which uses five bits consisting of exactly three 0s and two 1s. This provides ten possible combinations, enough to represent the digits 0–9. This scheme can detect all single bit-errors, all odd numbered bit-errors and some even numbered bit-errors (for example the flipping of both 1-bits). However it still cannot correct any of these errors.

### Repetition

Main article: [Triple modular redundancy](#)

Another code in use at the time repeated every data bit multiple times in order to ensure that it was sent correctly. For instance, if the data bit to be sent is a 1, an  $n = 3$  *repetition code* will send 111. If the three bits received are not identical, an error occurred during transmission. If the channel is clean enough, most of the time only one bit will change in each triple. Therefore, 001, 010, and 100 each correspond to a 0 bit, while 110, 101, and 011 correspond to a 1 bit, as though the bits count as “votes” towards what the intended bit is. A code with this ability to reconstruct the original message in the presence of errors is known as an *error-correcting* code. This triple repetition code is a Hamming code with  $m = 2$ , since there are two parity bits, and  $2^2 - 2 - 1 = 1$  data bit.

Such codes cannot correctly repair all errors, however. In our example, if the channel flips two bits and the receiver gets 001, the system will detect the error, but conclude that the original bit is 0, which is incorrect. If we increase the number of times we duplicate each bit to four, we can detect all two-bit errors but cannot correct them (the votes “tie”); at five repetitions, we can correct all two-bit errors, but not all three-bit errors.

Moreover, the repetition code is extremely inefficient, reducing throughput by three times in our original case, and the efficiency drops drastically as we increase the number of times each bit is duplicated in order to detect and correct more errors.

## 55.2 Hamming codes

If more error-correcting bits are included with a message, and if those bits can be arranged such that different incorrect bits produce different error results, then bad bits could be identified. In a seven-bit message, there are seven possible single bit errors, so three error control bits

could potentially specify not only that an error occurred but also which bit caused the error.

Hamming studied the existing coding schemes, including two-of-five, and generalized their concepts. To start with, he developed a *nomenclature* to describe the system, including the number of data bits and error-correction bits in a block. For instance, parity includes a single bit for any data word, so assuming *ASCII* words with seven bits, Hamming described this as an (8,7) code, with eight bits in total, of which seven are data. The repetition example would be (3,1), following the same logic. The *code rate* is the second number divided by the first, for our repetition example, 1/3.

Hamming also noticed the problems with flipping two or more bits, and described this as the “distance” (it is now called the *Hamming distance*, after him). Parity has a distance of 2, so one bit flip can be detected, but not corrected and any two bit flips will be invisible. The (3,1) repetition has a distance of 3, as three bits need to be flipped in the same triple to obtain another code word with no visible errors. It can correct one-bit errors or detect but not correct two-bit errors. A (4,1) repetition (each bit is repeated four times) has a distance of 4, so flipping three bits can be detected, but not corrected. When three bits flip in the same group there can be situations where attempting to correct will produce the wrong code word. In general, a code with distance  $k$  can detect but not correct  $k - 1$  errors.

Hamming was interested in two problems at once: increasing the distance as much as possible, while at the same time increasing the code rate as much as possible. During the 1940s he developed several encoding schemes that were dramatic improvements on existing codes. The key to all of his systems was to have the parity bits overlap, such that they managed to check each other as well as the data.

### 55.2.1 General algorithm

The following general algorithm generates a single-error correcting (SEC) code for any number of bits.

1. Number the bits starting from 1: bit 1, 2, 3, 4, 5, etc.
2. Write the bit numbers in binary: 1, 10, 11, 100, 101, etc.
3. All bit positions that are powers of two (have only one 1 bit in the binary form of their position) are parity bits: 1, 2, 4, 8, etc. (1, 10, 100, 1000)
4. All other bit positions, with two or more 1 bits in the binary form of their position, are data bits.
5. Each data bit is included in a unique set of 2 or more parity bits, as determined by the binary form of its bit position.

- (a) Parity bit 1 covers all bit positions which have the least significant bit set: bit 1 (the parity bit itself), 3, 5, 7, 9, etc.
- (b) Parity bit 2 covers all bit positions which have the second least significant bit set: bit 2 (the parity bit itself), 3, 6, 7, 10, 11, etc.
- (c) Parity bit 4 covers all bit positions which have the third least significant bit set: bits 4–7, 12–15, 20–23, etc.
- (d) Parity bit 8 covers all bit positions which have the fourth least significant bit set: bits 8–15, 24–31, 40–47, etc.
- (e) In general each parity bit covers all bits where the bitwise AND of the parity position and the bit position is non-zero.

The form of the parity is irrelevant. Even parity is simpler from the perspective of theoretical mathematics, but there is no difference in practice.

This general rule can be shown visually:

Shown are only 20 encoded bits (5 parity, 15 data) but the pattern continues indefinitely. The key thing about Hamming Codes that can be seen from visual inspection is that any given bit is included in a unique set of parity bits. To check for errors, check all of the parity bits. The pattern of errors, called the **error syndrome**, identifies the bit in error. If all parity bits are correct, there is no error. Otherwise, the sum of the positions of the erroneous parity bits identifies the erroneous bit. For example, if the parity bits in positions 1, 2 and 8 indicate an error, then bit  $1+2+8=11$  is in error. If only one parity bit indicates an error, the parity bit itself is in error.

As you can see, if you have  $m$  parity bits, it can cover bits from 1 up to  $2^m - 1$ . If we subtract out the parity bits, we are left with  $2^m - m - 1$  bits we can use for the data. As  $m$  varies, we get all the possible Hamming codes:

If, in addition, an overall parity bit (bit 0) is included, the code can detect (but not correct) any two-bit error, making a SECDED code. The overall parity indicates whether the total number of errors is even or odd. If the basic Hamming code detects an error, but the overall parity says that there are an even number of errors, an uncorrectable 2-bit error has occurred.

### 55.3 Hamming codes with additional parity (SECDED)

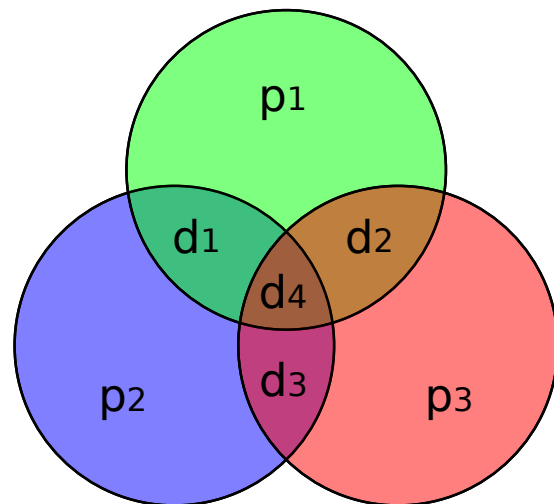
Hamming codes have a minimum distance of 3, which means that the decoder can detect and correct a single error, but it cannot distinguish a double bit error of some

codeword from a single bit error of a different codeword. Thus, they can detect double-bit errors only if correction is not attempted.

To remedy this shortcoming, Hamming codes can be extended by an extra parity bit. This way, it is possible to increase the minimum distance of the Hamming code to 4, which allows the decoder to distinguish between single bit errors and two-bit errors. Thus the decoder can detect and correct a single error and at the same time detect (but not correct) a double error. If the decoder does not attempt to correct errors, it can detect up to three errors.

This extended Hamming code is popular in computer memory systems, where it is known as *SECDED* (abbreviated from *single error correction, double error detection*). Particularly popular is the (72,64) code, a truncated (127,120) Hamming code plus an additional parity bit, which has the same space overhead as a (9,8) parity code.

## 55.4 [7,4] Hamming code



Graphical depiction of the four data bits and three parity bits and which parity bits apply to which data bits

Main article: [Hamming\(7,4\)](#)

In 1950, Hamming introduced the [7,4] Hamming code. It encodes four data bits into seven bits by adding three parity bits. It can detect and correct single-bit errors. With the addition of an overall parity bit, it can also detect (but not correct) double-bit errors.

#### 55.4.1 Construction of G and H

The matrix  $\mathbf{G} := (I_k \mid -A^T)$  is called a (canonical) generator matrix of a linear  $(n,k)$  code,

and  $\mathbf{H} := (A \mid I_{n-k})$  is called a parity-check matrix.

This is the construction of  $\mathbf{G}$  and  $\mathbf{H}$  in standard (or systematic) form. Regardless of form,  $\mathbf{G}$  and  $\mathbf{H}$  for linear block codes must satisfy

$\mathbf{H}\mathbf{G}^T = \mathbf{0}$ , an all-zeros matrix.<sup>[2]</sup>

Since  $[7, 4, 3] = [n, k, d] = [2^m - 1, 2^m - 1 - m, m]$ . The parity-check matrix  $\mathbf{H}$  of a Hamming code is constructed by listing all columns of length  $m$  that are pair-wise independent.

Thus  $\mathbf{H}$  is a matrix whose left side is all of the nonzero  $n$ -tuples where order of the  $n$ -tuples in the columns of matrix does not matter. The right hand side is just the  $(n - k)$ -identity matrix.

So  $\mathbf{G}$  can be obtained from  $\mathbf{H}$  by taking the transpose of the left hand side of  $\mathbf{H}$  with the identity  $k$ -identity matrix on the left hand side of  $\mathbf{G}$ .

The code generator matrix  $\mathbf{G}$  and the parity-check matrix  $\mathbf{H}$  are:

$$\mathbf{G} := \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}_{4,7}$$

and

$$\mathbf{H} := \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}_{3,7}$$

Finally, these matrices can be mutated into equivalent non-systematic codes by the following operations:<sup>[2]</sup>

- Column permutations (swapping columns)
- Elementary row operations (replacing a row with a linear combination of rows)

### 55.4.2 Encoding

#### Example

From the above matrix we have  $2^k = 2^4 = 16$  codewords. The codewords  $\vec{x}$  of this binary code can be obtained from  $\vec{x} = \vec{a}\mathbf{G}$ . With  $\vec{a} = a_1a_2a_3a_4$  with  $a_i$  exist in  $F_2$  (A field with two elements namely 0 and 1).

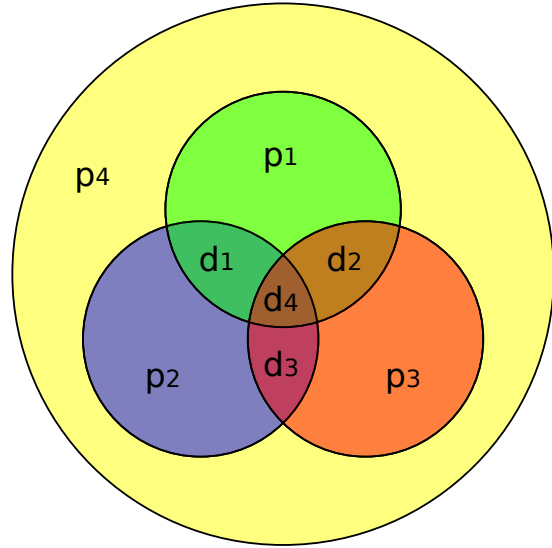
Thus the codewords are all the 4-tuples ( $k$ -tuples).

Therefore,

(1,0,1,1) gets encoded as (1,0,1,1,0,1,0).

### 55.4.3 [7,4] Hamming code with an additional parity bit

The [7,4] Hamming code can easily be extended to an [8,4] code by adding an extra parity bit on top of the (7,4) encoded word (see Hamming(7,4)). This can be summed up with the revised matrices:



The same [7,4] example from above with an extra parity bit. This diagram is not meant to correspond to the matrix  $\mathbf{H}$  for this example.

$$\mathbf{G} := \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}_{4,8}$$

and

$$\mathbf{H} := \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}_{4,8}$$

Note that  $\mathbf{H}$  is not in standard form. To obtain  $\mathbf{G}$ , elementary row operations can be used to obtain an equivalent matrix to  $\mathbf{H}$  in systematic form:

$$\mathbf{H} = \left( \begin{array}{cccc|cccc} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{array} \right)_{4,8}.$$

For example, the first row in this matrix is the sum of the second and third rows of  $\mathbf{H}$  in non-systematic form. Using the systematic construction for Hamming codes from above, the matrix  $\mathbf{A}$  is apparent and the systematic form of  $\mathbf{G}$  is written as

$$\mathbf{G} = \left( \begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{array} \right)_{4,8}.$$



The non-systematic form of  $G$  can be row reduced (using elementary row operations) to match this matrix.

The addition of the fourth row effectively computes the sum of all the codeword bits (data and parity) as the fourth parity bit.

For example, 1011 is encoded (using the non-systematic form of  $G$  at the start of this section) into 01100110 where blue digits are data; red digits are parity bits from the  $[7,4]$  Hamming code; and the green digit is the parity bit added by the  $[8,4]$  code. The green digit makes the parity of the  $[7,4]$  codewords even.

Finally, it can be shown that the minimum distance has increased from 3, in the  $[7,4]$  code, to 4 in the  $[8,4]$  code. Therefore, the code can be defined as  $[8,4]$  Hamming code.

- “Mathematical Challenge April 2013 Error-correcting codes” (PDF). swissQuant Group Leadership Team. April 2013.

## 55.8 External links

- CGI script for calculating Hamming distances (from R. Tervo, UNB, Canada)

## 55.5 See also

- Coding theory
- Golay code
- Reed–Muller code
- Reed–Solomon error correction
- Turbo code
- Low-density parity-check code
- Hamming bound
- Hamming distance

## 55.6 Notes

- [1] See Lemma 12 of
- [2] Moon T. Error correction coding: Mathematical Methods and Algorithms. John Wiley and Sons, 2005.(Cap. 3) ISBN 978-0-471-64800-0

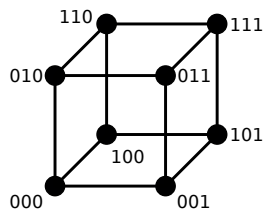
## 55.7 References

- Moon, Todd K. (2005). *Error Correction Coding*. New Jersey: John Wiley & Sons. ISBN 978-0-471-64800-0.
- MacKay, David J.C. (September 2003). *Information Theory, Inference and Learning Algorithms*. Cambridge: Cambridge University Press. ISBN 0-521-64298-1.
- D.K. Bhattacharyya, S. Nandi. “An efficient class of SEC-DED-AUED codes”. *1997 International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '97)*. pp. 410–415. doi:10.1109/ISPAN.1997.645128.



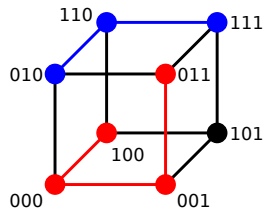
## Chapter 56

# Hamming distance



3-bit binary cube for finding

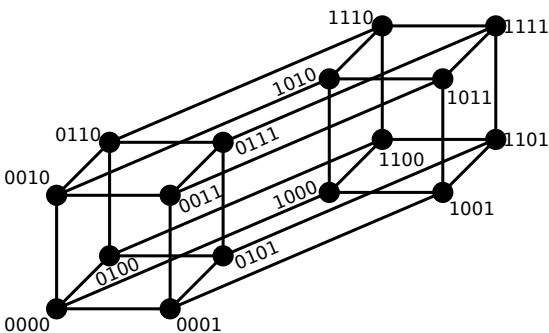
Hamming distance



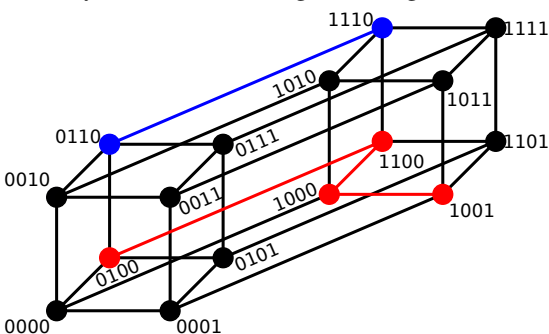
Two example distances:

100→011 has distance 3; 010→111 has distance 2

The minimum distance between any two vertices is the Hamming distance between the two binary strings.



4-bit binary tesseract for finding Hamming distance.



example distances: 0100→1001 has distance 3; 0110→1110 has distance 1

In information theory, the **Hamming distance** between two strings of equal length is the number of positions at which the corresponding symbols are different. In another way, it measures the minimum number of *substitutions* required to change one string into the other, or the minimum number of *errors* that could have transformed one string into the other.

A major application is in coding theory, more specifically to block codes, in which the equal-length strings are vectors over a finite field.

## 56.1 Examples

The Hamming distance between:

- "karolin" and "kathrin" is 3.
- "karolin" and "kerstin" is 3.
- 1011101 and 1001001 is 2.
- 2173896 and 2233796 is 3.

## 56.2 Properties

4- For a fixed length  $n$ , the Hamming distance is a metric on the set of the words of length  $n$  (also known as a Hamming space), as it fulfills the conditions of non-negativity, identity of indiscernibles and symmetry, and it can be shown by complete induction that it satisfies the triangle inequality as well.<sup>[1]</sup> The Hamming distance between two words  $a$  and  $b$  can also be seen as the Hamming weight of  $a-b$  for an appropriate choice of the  $-$  operator.

For binary strings  $a$  and  $b$  the Hamming distance is equal to the number of ones (population count) in  $a \oplus b$ . The metric space of length- $n$  binary strings, with the Hamming distance, is known as the Hamming cube; it is two equivalent as a metric space to the set of distances between vertices in a hypercube graph. One can also view a binary string of length  $n$  as a vector in  $R^n$  by treating each symbol in the string as a real coordinate; with this embedding, the strings form the vertices of an  $n$ -dimensional

hypercube, and the Hamming distance of the strings is equivalent to the Manhattan distance between the vertices.

strings where not just substitutions but also insertions or deletions have to be expected, a more sophisticated metric like the Levenshtein distance is more appropriate.

## 56.3 Error detection and error correction

The Hamming distance is used to define some essential notions in coding theory, such as error detecting and error correcting codes. In particular, a code  $C$  is said to be  $k$ -errors detecting if any two codewords  $c_1$  and  $c_2$  from  $C$  that have a Hamming distance less than  $k$  coincide; otherwise, a code is  $k$ -errors detecting if, and only if, the minimum Hamming distance between any two of its codewords is at least  $k+1$ .<sup>[1]</sup>

A code  $C$  is said to be  $k$ -errors correcting if, for every word  $w$  in the underlying Hamming space  $H$ , there exists at most one codeword  $c$  (from  $C$ ) such that the Hamming distance between  $w$  and  $c$  is less than  $k$ . In other words, a code is  $k$ -errors correcting if, and only if, the minimum Hamming distance between any two of its codewords is at least  $2k+1$ . This is more easily understood geometrically as any closed balls of radius  $k$  centered on distinct codewords being disjoint.<sup>[1]</sup> These balls are also called **Hamming spheres** in this context.<sup>[2]</sup>

Thus a code with minimum Hamming distance  $d$  between its codewords can detect at most  $d-1$  errors and can correct  $\lfloor (d-1)/2 \rfloor$  errors.<sup>[1]</sup> The latter number is also called the *packing radius* or the *error-correcting capability* of the code.<sup>[2]</sup>

## 56.4 History and applications

The Hamming distance is named after Richard Hamming, who introduced the concept in his fundamental paper on Hamming codes *Error detecting and error correcting codes* in 1950.<sup>[3]</sup> Hamming weight analysis of bits is used in several disciplines including information theory, coding theory, and cryptography.

It is used in telecommunication to count the number of flipped bits in a fixed-length binary word as an estimate of error, and therefore is sometimes called the **signal distance**.<sup>[4]</sup> For  $q$ -ary strings over an alphabet of size  $q \geq 2$  the Hamming distance is applied in case of the  $q$ -ary symmetric channel, while the Lee distance is used for phase-shift keying or more generally channels susceptible to synchronization errors because the Lee distance accounts for errors of  $\pm 1$ .<sup>[5]</sup> If  $q = 2$  or  $q = 3$  both distances coincide because  $\mathbb{Z}/2\mathbb{Z}$  and  $\mathbb{Z}/3\mathbb{Z}$  are also fields, but  $\mathbb{Z}/4\mathbb{Z}$  is not a field but only a ring.

The Hamming distance is also used in systematics as a measure of genetic distance.<sup>[6]</sup>

However, for comparing strings of different lengths, or

## 56.5 Algorithm example

The Python3 function `hamming_distance()` computes the Hamming distance between two strings (or other iterable objects) of equal length, by creating a sequence of Boolean values indicating mismatches and matches between corresponding positions in the two inputs, and then summing the sequence with False and True values being interpreted as zero and one.

```
def hamming_distance(s1, s2): """Return the Hamming
distance between equal-length sequences""" if len(s1)
!= len(s2): raise ValueError("Undefined for sequences
of unequal length") return sum(e1 != e2 for e1, e2 in
zip(s1, s2))
```

where the `zip()` function merges two equal-length collections in pairs.

Or in Ruby language the function `hamming_distance()` could be:

```
def hamming_distance(s1, s2) raise "ERROR: Ham-
ming: Non equal lengths" if s1.length != s2.length
(s1.chars.zip(s2.chars)).count {|l, r| l != r} end
```

The following C function will compute the Hamming distance of two integers (considered as binary values, that is, as sequences of bits). The running time of this procedure is proportional to the Hamming distance rather than to the number of bits in the inputs. It computes the bitwise exclusive or of the two inputs, and then finds the Hamming weight of the result (the number of nonzero bits) using an algorithm of Wegner (1960) that repeatedly finds and clears the lowest-order nonzero bit. Some compilers support the `__builtin_popcount` function which can calculate this using specialized processor hardware where available.

```
int hamming_distance(unsigned x, unsigned y) { int dist
= 0; unsigned val = x ^ y; // Count the number of bits set
while (val != 0) { // A bit is set, so increment the count
and clear the bit dist++; val &= val - 1; } // Return the
number of differing bits return dist; }
```

Or, a much faster hardware alternative (for compilers that support builtins) is to use `popcount` like so.

```
int hamming_distance(unsigned x, unsigned y) { return
__builtin_popcount(x ^ y); } //if your compiler supports
64-bit integers int hamming_distance(unsigned long long
x, unsigned long long y) { return __builtin_popcountll(x
^ y); }
```

## 56.6 See also

- Closest string
- Damerau–Levenshtein distance
- Euclidean distance
- Mahalanobis distance
- Jaccard index
- String metric
- Sørensen similarity index
- Word ladder
- Gray code
- Levenshtein distance
- sparse distributed memory
- Wegner, Peter (1960), “A technique for counting ones in a binary computer”, *Communications of the ACM*, **3** (5): 322, doi:10.1145/367236.367286.
- Ayala, Jose (2012), “Fast Propagation of Hamming and Signal Distances for Register-Transfer Level Datapaths”, *Integrated Circuit and System Design*, **1**: 62.

## 56.9 Further reading

- MacKay, David J. C.. *Information Theory, Inference, and Learning Algorithms* Cambridge: Cambridge University Press, 2003. ISBN 0-521-64298-1

## 56.7 Notes

- [1] Derek J.S. Robinson (2003). *An Introduction to Abstract Algebra*. Walter de Gruyter. pp. 255–257. ISBN 978-3-11-019816-4.
- [2] Cohen, G.; Honkala, I.; Litsyn, S.; Lobstein, A. (1997), *Covering Codes*, North-Holland Mathematical Library, **54**, Elsevier, pp. 16–17, ISBN 9780080530079
- [3] Hamming (1950).
- [4] Jose Ayala (2012). *Integrated Circuit and System Design*. Springer. p. 62. ISBN 978-3-642-36156-2.
- [5] Ron Roth (2006). *Introduction to Coding Theory*. Cambridge University Press. p. 298. ISBN 978-0-521-84504-5.
- [6] Pilcher, Wong & Pillai (2008).

## 56.8 References

- This article incorporates public domain material from the General Services Administration document “Federal Standard 1037C”.
- Hamming, Richard W. (1950), “Error detecting and error correcting codes” (PDF), *Bell System Technical Journal*, **29** (2): 147–160, doi:10.1002/j.1538-7305.1950.tb00463.x, MR 0035935.
- Pilcher, C. D.; Wong, J. K.; Pillai, S. K. (March 2008), “Inferring HIV transmission dynamics from phylogenetic sequence relationships”, *PLoS Med.*, **5** (3): e69, doi:10.1371/journal.pmed.0050069, PMC 2267810, PMID 18351799.

## Chapter 57

# Hamming scheme

The **Hamming scheme**, named after **Richard Hamming**, is also known as the **hyper-cubic association scheme**, and it is the most important example for **coding theory**.<sup>[1][2][3]</sup> In this scheme  $X = \mathcal{F}^n$ , the set of binary vectors of length  $n$ , and two vectors  $x, y \in \mathcal{F}^n$  are  $i$ -th associates if they are **Hamming distance**  $i$  apart.

Recall that an **association scheme** is visualized as a **complete graph** with labeled edges. The graph has  $v$  vertices, one for each point of  $X$ , and the edge joining vertices  $x$  and  $y$  is labeled  $i$  if  $x$  and  $y$  are  $i$ -th associates. Each edge has a unique label, and the number of triangles with a fixed base labeled  $k$  having the other edges labeled  $i$  and  $j$  is a constant  $c_{ijk}$ , depending on  $i, j, k$  but not on the choice of the base. In particular, each vertex is incident with exactly  $c_{ii0} = v_i$  edges labeled  $i$ ;  $v_i$  is the **valency** of the **relation**  $R_i$ . The  $c_{ijk}$  in a **Hamming scheme** are given by

$$c_{ijk} = \begin{cases} \binom{k}{\frac{i-j+k}{2}} \binom{n-k}{\frac{i+j-k}{2}}, & \text{if } i+j-k \text{ even, is} \\ 0, & \text{if } i+j-k \text{ odd, is} \end{cases}$$

Here,  $v = |X| = 2^n$  and  $v_i = \binom{n}{i}$ . The **matrices** in the **Bose-Mesner algebra** are  $2^n \times 2^n$  matrices, with rows and columns labeled by vectors  $x \in \mathcal{F}^n$ . In particular the  $(x, y)$ -th entry of  $D_k$  is 1 if and only if  $d_H(x, y) = k$ .

### 57.1 References

- [1] P. Delsarte and V. I. Levenshtein, "Association schemes and coding theory," *IEEE Trans. Inform. Theory*, vol. 44, no. 6, pp. 2477–2504, 1998.
- [2] P. Camion, "Codes and Association Schemes: Basic Properties of Association Schemes Relevant to Coding," in *Handbook of Coding Theory*, V. S. Pless and W. C. Huffman, Eds., Elsevier, The Netherlands, 1998.
- [3] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, Elsevier, New York, 1978.

## Chapter 58

# Hamming space

In statistics and coding theory, a **Hamming space** is usually the set of all  $2^N$  binary strings of length  $N$ .<sup>[1][2]</sup> It is used in the theory of coding signals and transmission.

More generally, a Hamming space can be defined over any alphabet (set)  $Q$  as the set of words of a fixed length  $N$  with letters from  $Q$ .<sup>[3][4]</sup> If  $Q$  is a finite field, then a Hamming space over  $Q$  is an  $N$ -dimensional vector space over  $Q$ . In the typical, binary case, the field is thus  $\text{GF}(2)$  (also denoted by  $\mathbb{Z}_2$ ).<sup>[3]</sup>

In coding theory, if  $Q$  has  $q$  elements, then any subset  $C$  (usually assumed of cardinality at least two) of the  $N$ -dimensional Hamming space over  $Q$  is called a **q-ary code of length  $N$** ; the elements of  $C$  are called **codewords**.<sup>[3][4]</sup> In the case where  $C$  is a linear subspace of its Hamming space, it is called a **linear code**.<sup>[3]</sup> A typical example of linear code is the **Hamming code**. Codes defined via a Hamming space necessarily have the same length for every codeword, so they are called **block codes** when it is necessary to distinguish them from **variable-length codes** that are defined by unique factorization on a monoid.

The **Hamming distance** endows a Hamming space with a **metric**, which is essential in defining basic notions of coding theory such as **error detecting** and **error correcting codes**.<sup>[3]</sup>

Hamming spaces over non-field alphabets have also been considered, especially over finite rings (most notably over  $\mathbb{Z}_4$ ) giving rise to **modules** instead of vector spaces and **ring-linear codes** (identified with **submodules**) instead of linear codes. The typical metric used in this case the **Lee distance**. There exist a **Gray isometry** between  $\mathbb{Z}_2^{2^m}$  (i.e.  $\text{GF}(2^{2^m})$ ) with the Hamming distance and  $\mathbb{Z}_4^m$  (also denoted as  $\text{GR}(4,m)$ ) with the Lee distance.<sup>[5][6][7]</sup>

## 58.1 References

- [1] Baylis, D. J. (1997), *Error Correcting Codes: A Mathematical Introduction*, Chapman Hall/CRC Mathematics Series, **15**, CRC Press, p. 62, ISBN 9780412786907
- [2] Cohen, G.; Honkala, I.; Litsyn, S.; Lobstein, A. (1997), *Covering Codes*, North-Holland Mathematical Library, **54**, Elsevier, p. 1, ISBN 9780080530079
- [3] Derek J.S. Robinson (2003). *An Introduction to Abstract Algebra*. Walter de Gruyter. pp. 254–255. ISBN 978-3-11-019816-4.
- [4] Cohen et al., *Covering Codes*, p. 15
- [5] Marcus Greferath (2009). “An Introduction to Ring-Linear Coding Theory”. In Massimiliano Sala, Teo Mora, Ludovic Perret, Shojiro Sakata, Carlo Traverso. *Gröbner Bases, Coding, and Cryptography*. Springer Science & Business Media. ISBN 978-3-540-93806-4.
- [6] [http://www.encyclopediaofmath.org/index.php/Kerdock\\_and\\_Preparata\\_codes](http://www.encyclopediaofmath.org/index.php/Kerdock_and_Preparata_codes)
- [7] J.H. van Lint (1999). *Introduction to Coding Theory* (3rd ed.). Springer. Chapter 8: Codes over  $\mathbb{Z}_4$ . ISBN 978-3-540-64133-9.

## Chapter 59

# Hamming weight

The **Hamming weight** of a **string** is the number of symbols that are different from the zero-symbol of the alphabet used. It is thus equivalent to the **Hamming distance** from the all-zero string of the same length. For the most typical case, a string of bits, this is the number of 1's in the string. In this binary case, it is also called the **population count**, **popcount**, or **sideways sum**.<sup>[1]</sup> It is the digit sum of the binary representation of a given number and the  $\ell_1$  norm of a bit vector.

### 59.1 History and usage

The Hamming weight is named after **Richard Hamming** although he did not originate the notion.<sup>[2]</sup> The Hamming weight of binary numbers was already used in 1899 by J. W. L. Glaisher to give a formula for the number of odd binomial coefficients in a single row of Pascal's triangle.<sup>[3]</sup> Irving S. Reed introduced a concept, equivalent to Hamming weight in the binary case, in 1954.<sup>[4]</sup>

Hamming weight is used in several disciplines including **information theory**, **coding theory**, and **cryptography**. Examples of applications of the Hamming weight include:

- In modular **exponentiation by squaring**, the number of modular multiplications required for an exponent  $e$  is  $\log_2 e + \text{weight}(e)$ . This is the reason that the public key value  $e$  used in **RSA** is typically chosen to be a number of low Hamming weight.
- The Hamming weight determines path lengths between nodes in **Chord distributed hash tables**.<sup>[5]</sup>
- **IrisCode** lookups in biometric databases are typically implemented by calculating the **Hamming distance** to each stored record.
- In **computer chess** programs using a **bitboard** representation, the Hamming weight of a bitboard gives the number of pieces of a given type remaining in the game, or the number of squares of the board controlled by one player's pieces, and is therefore an important contributing term to the value of a position.

- Hamming weight can be used to efficiently compute **find first set** using the identity  $\text{ffs}(x) = \text{pop}(x \wedge (\sim(-x)))$ . This is useful on platforms such as **SPARC** that have hardware Hamming weight instructions but no hardware find first set instruction.<sup>[6]</sup>
- The Hamming weight operation can be interpreted as a conversion from the **unary numeral system** to **binary numbers**.<sup>[7]</sup>
- In implementation of some **succinct data structures** like **bit vectors** and **wavelet trees**.

### 59.2 Efficient implementation

The population count of a **bitstring** is often needed in cryptography and other applications. The **Hamming distance** of two words  $A$  and  $B$  can be calculated as the Hamming weight of  $A \text{ xor } B$ .

The problem of how to implement it efficiently has been widely studied. Some processors have a single command to calculate it (see below), and some have parallel operations on bit vectors. For processors lacking those features, the best solutions known are based on adding counts in a tree pattern. For example, to count the number of 1 bits in the 16-bit binary number  $a = 0110\ 1100\ 1011\ 1010$ , these operations can be done:

Here, the operations are as in **C programming language**, so  $X \gg Y$  means to shift  $X$  right by  $Y$  bits,  $X \& Y$  means the bitwise AND of  $X$  and  $Y$ , and  $+$  is ordinary addition. The best algorithms known for this problem are based on the concept illustrated above and are given here:

```
//types and constants used in the functions below
//uint64_t is an unsigned 64-bit integer variable type (defined in C99 version of C language)
const uint64_t m1 = 0x5555555555555555; //binary: 0101...
const uint64_t m2 = 0x3333333333333333; //binary: 00110011..
const uint64_t m4 = 0x0f0f0f0f0f0f0f0f; //binary: 4 zeros, 4 ones ...
const uint64_t m8 = 0x00ff00ff00ff00ff; //binary: 8 zeros, 8 ones ...
const uint64_t m16 = 0x0000ffff0000ffff; //binary: 16 zeros, 16 ones ...
const uint64_t m32 = 0x00000000ffffffff; //binary: 32 zeros, 32 ones
const uint64_t hff = 0xffffffffffffffff; //binary:
```



```

all ones const uint64_t h01 = 0x0101010101010101;
//the sum of 256 to the power of 0,1,2,3... //This is
//a naive implementation, shown for comparison, //and
//to help in understanding the better functions. //This
//algorithm uses 24 arithmetic operations (shift, add, and).
int popcount64a(uint64_t x) { x = (x & m1) + ((x >>
1) & m1); //put count of each 2 bits into those 2 bits x
= (x & m2) + ((x >> 2) & m2); //put count of each 4
bits into those 4 bits x = (x & m4) + ((x >> 4) & m4);
//put count of each 8 bits into those 8 bits x = (x &
m8) + ((x >> 8) & m8); //put count of each 16 bits
into those 16 bits x = (x & m16) + ((x >> 16) & m16);
//put count of each 32 bits into those 32 bits x = (x &
m32) + ((x >> 32) & m32); //put count of each 64 bits
into those 64 bits return x; } //This uses fewer arithmetic
operations than any other known //implementation on
machines with slow multiplication. //This algorithm uses
17 arithmetic operations. int popcount64b(uint64_t x) {
x -= (x >> 1) & m1; //put count of each 2 bits into those
2 bits x = (x & m2) + ((x >> 2) & m2); //put count of
each 4 bits into those 4 bits x = (x + (x >> 4)) & m4;
//put count of each 8 bits into those 8 bits x += x >> 8;
//put count of each 16 bits into their lowest 8 bits x += x
>> 16; //put count of each 32 bits into their lowest 8 bits
x += x >> 32; //put count of each 64 bits into their lowest
8 bits return x & 0x7f; } //This uses fewer arithmetic
operations than any other known //implementation on
machines with fast multiplication. //This algorithm uses
12 arithmetic operations, one of which is a multiply. int
popcount64c(uint64_t x) { x -= (x >> 1) & m1; //put
count of each 2 bits into those 2 bits x = (x & m2) +
((x >> 2) & m2); //put count of each 4 bits into those
4 bits x = (x + (x >> 4)) & m4; //put count of each
8 bits into those 8 bits return (x * h01) >> 56; //re-
turns left 8 bits of x + (x<<8) + (x<<16) + (x<<24) + ... }

```

The above implementations have the best worst-case behavior of any known algorithm. However, when a value is expected to have few nonzero bits, it may instead be more efficient to use algorithms that count these bits one at a time. As Wegner (1960) described,<sup>[8]</sup> the bitwise and of  $x$  with  $x - 1$  differs from  $x$  only in zeroing out the least significant nonzero bit: subtracting 1 changes the rightmost string of 0s to 1s, and changes the rightmost 1 to a 0. If  $x$  originally had  $n$  bits that were 1, then after only  $n$  iterations of this operation,  $x$  will be reduced to zero. The following implementation is based on this principle.

```

//This is better when most bits in x are 0 //This is algo-
//rithm works the same for all data sizes. //This algorithm
//uses 3 arithmetic operations and 1 comparison/branch
//per "1" bit in x. int popcount64d(uint64_t x) { int count;
for (count=0; x; count++) x &= x - 1; return count; }

```

If we are allowed greater memory usage, we can calculate the Hamming weight faster than the above methods. With unlimited memory, we could simply create a large lookup table of the Hamming weight of every 64 bit integer. If

we can store a lookup table of the hamming function of every 16 bit integer, we can do the following to compute the Hamming weight of every 32 bit integer.

```

static uint8_t wordbits[65536] = { /* bitcounts of
integers 0 through 65535, inclusive */ }; //This algorithm
//uses 3 arithmetic operations and 2 memory reads. int
popcount32e(uint32_t x) { return wordbits[x & 0xFFFF]
+ wordbits[x >> 16]; }
//Optionally, the wordbits[] table could be filled using
//this function int popcount32e_init(void) { uint32_t
i; uint16_t x; int count; for (i=0; i <= 0xFFFF; i++)
{ x = i; for (count=0; x; count++) // borrowed from
popcount64d() above x &= x - 1; wordbits[i] = count; }
}

```

## 59.3 Language support

Some C compilers provide intrinsic functions that provide bit counting facilities. For example, **GCC** (since version 3.4 in April 2004) includes a builtin function `__builtin_popcount` that will use a processor instruction if available or an efficient library implementation otherwise.<sup>[9]</sup> **LLVM-GCC** has included this function since version 1.5 in June, 2005.<sup>[10]</sup>

In **C++ STL**, the bit-array data structure `bitset` has a `count()` method that counts the number of bits that are set.

In Java, the growable bit-array data structure `BitSet` has a `BitSet.cardinality()` method that counts the number of bits that are set. In addition, there are `Integer.bitCount(int)` and `Long.bitCount(long)` functions to count bits in primitive 32-bit and 64-bit integers, respectively. Also, the `BigInteger` arbitrary-precision integer class also has a `BigInteger.bitCount()` method that counts bits.

In **Common Lisp**, the function `logcount`, given a non-negative integer, returns the number of 1 bits. (For negative integers it returns the number of 0 bits in 2's complement notation.) In either case the integer can be a `BIGNUM`.

Starting in **GHC 7.4**, the **Haskell** base package has a `popCount` function available on all types that are instances of the `Bits` class (available from the `Data.Bits` module).<sup>[11]</sup>

**MySQL** version of **SQL** language provides `BIT_COUNT()` as a standard function.<sup>[12]</sup>

**Fortran 2008** has the standard, intrinsic, elemental function `popcnt` returning the number of nonzero bits within an integer (or integer array), see page 380 in Metcalf, Michael; John Reid; Malcolm Cohen (2011). *Modern Fortran Explained*. Oxford University Press. ISBN 0-19-960142-9.



## 59.4 Processor support

- Cray supercomputers early on featured a population count **machine instruction**, rumoured to have been specifically requested by the U.S. government National Security Agency for cryptanalysis applications.
- Some of Control Data Corporation's CYBER 70/170 series machines included a population count instruction; in **COMPASS**, this instruction was coded as CXi.
- AMD's **Barcelona** architecture introduced the abm (advanced bit manipulation) **ISA** introducing the POPCNT instruction as part of the **SSE4a** extensions.
- Intel Core processors introduced a POPCNT instruction with the **SSE4.2** instruction set extension, first available in a Nehalem-based Core i7 processor, released in November 2008.
- Compaq's Alpha 21264A, released in 1999, was the first Alpha series CPU design that had the count extension (CIX).
- Donald Knuth's model computer **MMIX** that is going to replace **MIX** in his book **The Art of Computer Programming** has an SADD instruction. SADD a,b,c counts all bits that are 1 in b and 0 in c and writes the result to a.
- The **ARM** architecture introduced the VCNT instruction as part of the Advanced SIMD (NEON) extensions.
- Analog Devices' **Blackfin** processors feature the ONES instruction to perform a 32-bit population count.

## 59.5 See also

- **Minimum weight**
- **Two's complement**
- **Most frequent k characters**

## 59.6 References

- [1] D. E. Knuth (2009). *The Art of Computer Programming Volume 4, Fascicle 1: Bitwise tricks & techniques; Binary Decision Diagrams*. Addison-Wesley Professional. ISBN 0-321-58050-8. Draft of Fascicle 1b available for download.

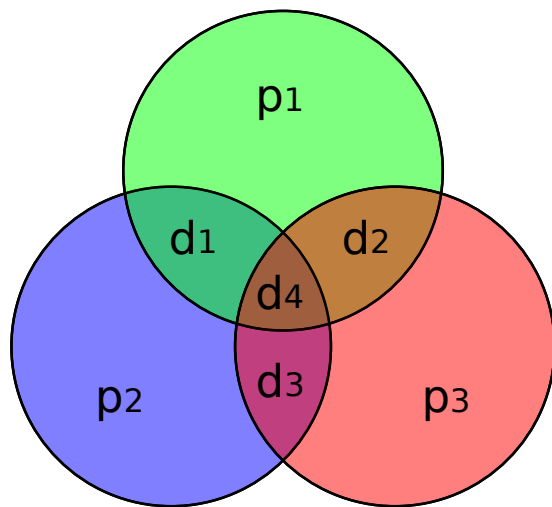
- [2] Thompson, Thomas M. (1983), *From Error-Correcting Codes through Sphere Packings to Simple Groups*, The Carus Mathematical Monographs #21, The Mathematical Association of America, p. 33
- [3] Glaisher, J. W. L. (1899), "On the residue of a binomial-theorem coefficient with respect to a prime modulus", *The Quarterly Journal of Pure and Applied Mathematics*, **30**: 150–156. See in particular the final paragraph of p. 156.
- [4] Reed, I.S. (1954), "A Class of Multiple-Error-Correcting Codes and the Decoding Scheme", *I.R.E. (I.E.E.E.)*, PGIT-4: 38
- [5] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., and Balakrishnan, H. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.* 11, 1 (Feb. 2003), 17-32. Section 6.3: "In general, the number of fingers we need to follow will be the number of ones in the binary representation of the distance from node to query."
- [6] SPARC International, Inc. (1992). *The SPARC architecture manual : version 8* (PDF) (Version 8. ed.). Englewood Cliffs, N.J.: Prentice Hall. p. 231. ISBN 0-13-825001-4. A.41: Population Count. Programming Note.
- [7] Blaxell, David (1978), "Record linkage by bit pattern matching", in Hogben, David; Fife, Dennis W., *Computer Science and Statistics--Tenth Annual Symposium on the Interface*, NBS Special Publication, **503**, U.S. Department of Commerce / National Bureau of Standards, pp. 146–156.
- [8] Wegner, Peter (1960), "A technique for counting ones in a binary computer", *Communications of the ACM*, **3** (5): 322, doi:10.1145/367236.367286
- [9] "GCC 3.4 Release Notes" GNU Project
- [10] "LLVM 1.5 Release Notes" LLVM Project.
- [11] "GHC 7.4.1 release notes". GHC documentation.
- [12] "12.11. Bit Functions — MySQL 5.0 Reference Manual".

## 59.7 External links

- **Aggregate Magic Algorithms**. Optimized population count and other algorithms explained with sample code.
- **HACKMEM item 169**. Population count assembly code for the PDP/6-10.
- **Bit Twiddling Hacks** Several algorithms with code for counting bits set.
- **Necessary and Sufficient** - by Damien Wintour - Has code in C# for various Hamming Weight implementations.
- **Best algorithm to count the number of set bits in a 32-bit integer?** - Stackoverflow

## Chapter 60

# Hamming(7,4)



Graphical depiction of the 4 data bits  $d_1$  to  $d_4$  and 3 parity bits  $p_1$  to  $p_3$  and which parity bits apply to which data bits

In coding theory, **Hamming(7,4)** is a linear error-correcting code that encodes four bits of data into seven bits by adding three parity bits. It is a member of a larger family of Hamming codes, but the term *Hamming code* often refers to this specific code that Richard W. Hamming introduced in 1950. At the time, Hamming worked at Bell Telephone Laboratories and was frustrated with the error-prone punched card reader, which is why he started working on error-correcting codes.<sup>[1]</sup>

The Hamming code adds three additional check bits to every four data bits of the message. Hamming's (7,4) algorithm can correct any single-bit error, or detect all single-bit and two-bit errors. In other words, the minimal Hamming distance between any two correct codewords is 3, and received words can be correctly decoded if they are at a distance of at most one from the codeword that was transmitted by the sender. This means that for transmission medium situations where burst errors do not occur, Hamming's (7,4) code is effective (as the medium would have to be extremely noisy for two out of seven bits to be flipped).

### 60.1 Goal

The goal of the Hamming codes is to create a set of parity bits that overlap such that a single-bit error (the bit is logically flipped in value) in a data bit *or* a parity bit can be detected *and* corrected. While multiple overlaps can be created, the general method is presented in Hamming codes.

This table describes which parity bits cover which transmitted bits in the encoded word. For example,  $p_2$  provides an even parity for bits 2, 3, 6, and 7. It also details which transmitted bit is covered by which parity bit by reading the column. For example,  $d_1$  is covered by  $p_1$  and  $p_2$  but not  $p_3$ . This table will have a striking resemblance to the parity-check matrix (**H**) in the next section.

Furthermore, if the parity columns in the above table were removed

then resemblance to rows 1, 2, and 4 of the code generator matrix (**G**) below will also be evident.

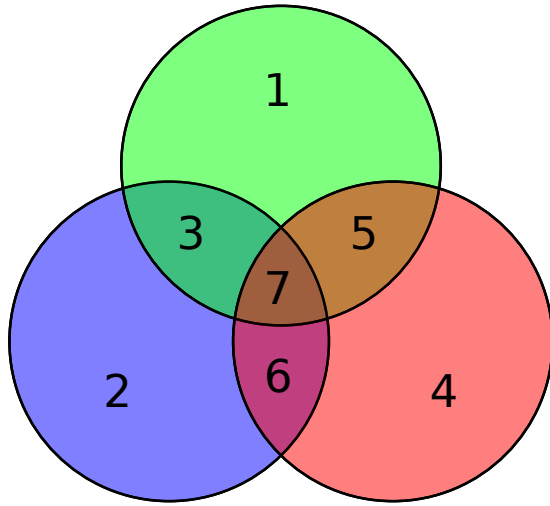
So, by picking the parity bit coverage correctly, all errors with a Hamming distance of 1 can be detected and corrected, which is the point of using a Hamming code.

### 60.2 Hamming matrices

Hamming codes can be computed in linear algebra terms through matrices because Hamming codes are linear codes. For the purposes of Hamming codes, two **Hamming matrices** can be defined: the **code generator matrix G** and the **parity-check matrix H**:

$$\mathbf{G} := \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{H} := \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

As mentioned above, rows 1, 2, and 4 of  $\mathbf{G}$  should look



Bit position of the data and parity bits

familiar as they map the data bits to their parity bits:

- $p_1$  covers  $d_1, d_2, d_4$
- $p_2$  covers  $d_1, d_3, d_4$
- $p_3$  covers  $d_2, d_3, d_4$

The remaining rows (3, 5, 6, 7) map the data to their position in encoded form and there is only 1 in that row so it is an identical copy. In fact, these four rows are **linearly independent** and form the **identity matrix** (by design, not coincidence).

Also as mentioned above, the three rows of  $\mathbf{H}$  should be familiar. These rows are used to compute the **syndrome vector** at the receiving end and if the syndrome vector is the **null vector** (all zeros) then the received word is error-free; if non-zero then the value indicates which bit has been flipped.

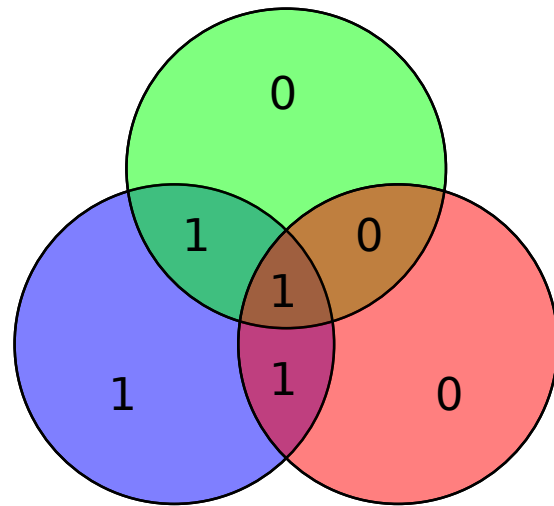
The four data bits — assembled as a vector  $\mathbf{p}$  — is pre-multiplied by  $\mathbf{G}$  (i.e.,  $\mathbf{Gp}$ ) and taken **modulo 2** to yield the encoded value that is transmitted. The original 4 data bits are converted to seven bits (hence the name “Hamming(7,4)”) with three parity bits added to ensure even parity using the above data bit coverages. The first table above shows the mapping between each data and parity bit into its final bit position (1 through 7) but this can also

be presented in a **Venn diagram**. The first diagram in this article shows three circles (one for each parity bit) and encloses data bits that each parity bit covers. The second diagram (shown to the right) is identical but, instead, the bit positions are marked.

For the remainder of this section, the following 4 bits (shown as a column vector) will be used as a running example:

$$\mathbf{p} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

## 60.3 Channel coding



Mapping in the example  $\mathbf{x}$ . The parity of the red, green, and blue circles are even.

Suppose we want to transmit this data (1011) over a noisy **communications channel**. Specifically, a **binary symmetric channel** meaning that error corruption does not favor either zero or one (it is symmetric in causing errors). Furthermore, all source vectors are assumed to be equiprobable. We take the product of  $\mathbf{G}$  and  $\mathbf{p}$ , with entries modulo 2, to determine the transmitted codeword  $\mathbf{x}$ :

$$\mathbf{x} = \mathbf{Gp} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 1 \\ 2 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

This means that 0110011 would be transmitted instead of transmitting 1011.

Programmers concerned about multiplication should observe that each row of the result is the least significant bit of the **Population Count** of set bits resulting from the row and column being **Bitwise ANDed** together rather than multiplied.

In the adjacent diagram, the seven bits of the encoded word are inserted into their respective locations; from inspection it is clear that the parity of the red, green, and blue circles are even:

- red circle has two 1's
- green circle has two 1's
- blue circle has four 1's

What will be shown shortly is that if, during transmission, a bit is flipped then the parity of two or all three circles will be incorrect and the errored bit can be determined (even if one of the parity bits) by knowing that the parity of all three of these circles should be even.

## 60.4 Parity check

If no error occurs during transmission, then the received codeword  $\mathbf{r}$  is identical to the transmitted codeword  $\mathbf{x}$ :

$$\mathbf{r} = \mathbf{x}$$

The receiver multiplies  $\mathbf{H}$  and  $\mathbf{r}$  to obtain the **syndrome** vector  $\mathbf{z}$ , which indicates whether an error has occurred, and if so, for which codeword bit. Performing this multiplication (again, entries modulo 2):

$$\mathbf{z} = \mathbf{H}\mathbf{r} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Since the syndrome  $\mathbf{z}$  is the **null vector**, the receiver can conclude that no error has occurred. This conclusion is based on the observation that when the data vector is multiplied by  $\mathbf{G}$ , a change of basis occurs into a vector subspace that is the **kernel** of  $\mathbf{H}$ . As long as nothing happens during transmission,  $\mathbf{r}$  will remain in the kernel of  $\mathbf{H}$  and the multiplication will yield the null vector.

## 60.5 Error correction

Otherwise, suppose a *single* bit error has occurred. Mathematically, we can write

$$\mathbf{r} = \mathbf{x} + \mathbf{e}_i$$

modulo 2, where  $\mathbf{e}_i$  is the  $i^{th}$  **unit vector**, that is, a zero vector with a 1 in the  $i^{th}$ , counting from 1.

$$\mathbf{e}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Thus the above expression signifies a single bit error in the  $i^{th}$  place.

Now, if we multiply this vector by  $\mathbf{H}$ :

$$\mathbf{H}\mathbf{r} = \mathbf{H}(\mathbf{x} + \mathbf{e}_i) = \mathbf{H}\mathbf{x} + \mathbf{H}\mathbf{e}_i$$

Since  $\mathbf{x}$  is the transmitted data, it is without error, and as a result, the product of  $\mathbf{H}$  and  $\mathbf{x}$  is zero. Thus

$$\mathbf{H}\mathbf{x} + \mathbf{H}\mathbf{e}_i = \mathbf{0} + \mathbf{H}\mathbf{e}_i = \mathbf{H}\mathbf{e}_i$$

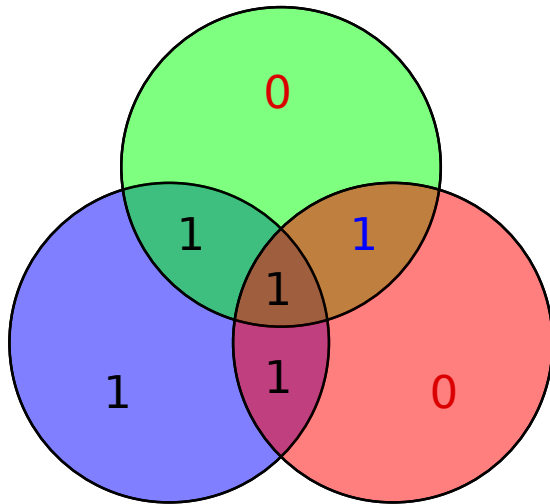
Now, the product of  $\mathbf{H}$  with the  $i^{th}$  standard basis vector picks out that column of  $\mathbf{H}$ , we know the error occurs in the place where this column of  $\mathbf{H}$  occurs.

For example, suppose we have introduced a bit error on bit #5

$$\mathbf{r} = \mathbf{x} + \mathbf{e}_5 = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

The diagram to the right shows the bit error (shown in blue text) and the bad parity created (shown in red text) in the red and green circles. The bit error can be detected by computing the parity of the red, green, and blue circles. If a bad parity is detected then the data bit that overlaps *only* the bad parity circles is the bit with the error. In the above example, the red and green circles have bad parity so the bit corresponding to the intersection of red and green but not blue indicates the errored bit.

Now,



A bit error on bit 5 causes bad parity in the red and green circles

$$\mathbf{R} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Then the received value,  $\mathbf{p}_r$ , is equal to  $\mathbf{R}\mathbf{r}$ . Using the running example from above

$$\mathbf{p}_r = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

## 60.7 Multiple bit errors

$$\mathbf{z} = \mathbf{H}\mathbf{r} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 4 \\ 3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

which corresponds to the fifth column of  $\mathbf{H}$ . Furthermore, the general algorithm used (see *Hamming code#General algorithm*) was intentional in its construction so that the syndrome of 101 corresponds to the binary value of 5, which indicates the fifth bit was corrupted. Thus, an error has been detected in bit 5, and can be corrected (simply flip or negate its value):

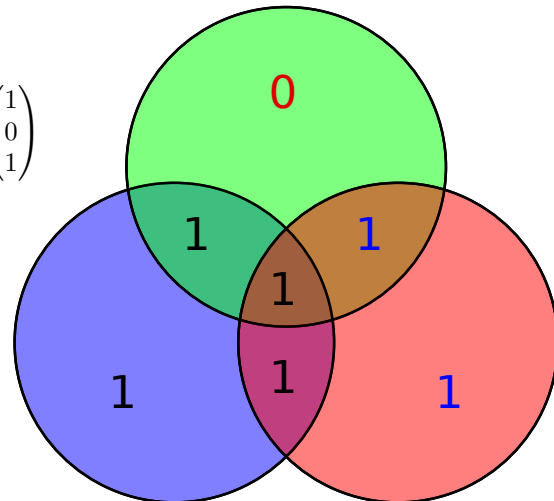
$$\mathbf{r}_{\text{corrected}} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

This corrected received value indeed, now, matches the transmitted value  $\mathbf{x}$  from above.

## 60.6 Decoding

Once the received vector has been determined to be error-free or corrected if an error occurred (assuming only zero or one bit errors are possible) then the received data needs to be decoded back into the original four bits.

First, define a matrix  $\mathbf{R}$ :



A bit error on bit 4 & 5 are introduced (shown in blue text) with a bad parity only in the green circle (shown in red text)

It is not difficult to show that only single bit errors can be corrected using this scheme. Alternatively, Hamming codes can be used to detect single and double bit errors, by merely noting that the product of  $\mathbf{H}$  is nonzero whenever errors have occurred. In the adjacent diagram, bits 4 and 5 were flipped. This yields only one circle (green) with an invalid parity but the errors are not recoverable.

However, the Hamming (7,4) and similar Hamming codes cannot distinguish between single-bit errors and two-bit errors. That is, two-bit errors appear the same as one-bit errors. If error correction is performed on a two-bit error the result will be incorrect.

Similarly, Hamming codes cannot detect or recover from an arbitrary three-bit error; Consider the diagram: if the bit in the green circle (colored red) were 1, the parity checking would return the null vector, indicating that there is no error in the codeword.

## 60.8 All codewords

Since the source is only 4 bits then there are only 16 possible transmitted words. Included is the eight-bit value if an extra parity bit is used (see *Hamming(7,4) code with an additional parity bit*). (The data bits are shown in blue; the parity bits are shown in red; and the extra parity bit shown in green.)

## 60.9 References

[1] “History of Hamming Codes”. Retrieved 2008-04-03.

## 60.10 External links

- A programming problem about the Hamming Code(7,4)

# Chapter 61

## Hexacode

In coding theory, the **hexacode** is a length 6 **linear code** of dimension 3 over the **Galois field**  $GF(4) = \{0, 1, \omega, \omega^2\}$  of 4 elements defined by

$$H = \{(a, b, c, f(1), f(\omega), f(\omega^2)) : f(x) := ax^2 + bx + c; a, b, c \in GF(4)\}.$$

It is a 3-dimensional subspace of the **vector space** of dimension 6 over  $GF(4)$ . Then  $H$  contains 45 **codewords** of **weight** 4, 18 codewords of weight 6 and the zero word. The full **automorphism group** of the hexacode is  $3.S_6$ . The hexacode can be used to describe the **Miracle Octad Generator** of R. T. Curtis.

### 61.1 References

- Conway, John H.; Sloane, Neil J. A. (1998). *Sphere Packings, Lattices and Groups* ((3rd ed.) ed.). New York: Springer-Verlag. ISBN 0-387-98585-9.



## Chapter 62

# Homomorphic signatures for network coding

Network coding has been shown to optimally use bandwidth in a network, maximizing information flow but the scheme is very inherently vulnerable to pollution attacks by malicious nodes in the network. A node injecting garbage can quickly affect many receivers. The pollution of network packets spreads quickly since the output of (even an) honest node is corrupted if at least one of the incoming packets is corrupted. An attacker can easily corrupt a packet even if it is encrypted by either forging the signature or by producing a collision under the hash function. This will give an attacker access to the packets and the ability to corrupt them. Denis Charles, Kamal Jain and Kristin Lauter designed a new homomorphic encryption signature scheme for use with network coding to prevent pollution attacks.<sup>[1]</sup> The homomorphic property of the signatures allows nodes to sign any linear combination of the incoming packets without contacting the signing authority. In this scheme it is computationally infeasible for a node to sign a linear combination of the packets without disclosing what linear combination was used in the generation of the packet. Furthermore, we can prove that the signature scheme is secure under well known cryptographic assumptions of the hardness of the discrete logarithm problem and the computational Elliptic curve Diffie–Hellman.

### 62.1 Network coding

Let  $G = (V, E)$  be a directed graph where  $V$  is a set, whose elements are called vertices or nodes, and  $E$  is a set of ordered pairs of vertices, called arcs, directed edges, or arrows. A source  $s \in V$  wants to transmit a file  $D$  to a set  $T \subseteq V$  of the vertices. One chooses a vector space  $W/\mathbb{F}_p$  (say of dimension  $d$ ), where  $p$  is a prime, and views the data to be transmitted as a bunch of vectors  $w_1, \dots, w_k \in W$ . The source then creates the augmented vectors  $v_1, \dots, v_k$  by setting  $v_i = (0, \dots, 0, 1, \dots, 0, w_{i_1}, \dots, w_{i_d})$  where  $w_{i_j}$  is the  $j$ -th coordinate of the vector  $w_i$ . There are  $(i-1)$  zeros before the first '1' appears in  $v_i$ . One can assume without loss of generality that the vectors  $v_i$  are linearly indepen-

dent. We denote the linear subspace (of  $\mathbb{F}_p^{k+d}$ ) spanned by these vectors by  $V$ . Each outgoing edge  $e \in E$  computes a linear combination,  $y(e)$ , of the vectors entering the vertex  $v = in(e)$  where the edge originates, that is to say

$$y(e) = \sum_{f: out(f)=v} (m_e(f)y(f))$$

where  $m_e(f) \in \mathbb{F}_p$ . We consider the source as having  $k$  input edges carrying the  $k$  vectors  $w_i$ . By induction, one has that the vector  $y(e)$  on any edge is a linear combination  $y(e) = \sum_{1 \leq i \leq k} (g_i(e)v_i)$  and is a vector in  $V$ . The  $k$ -dimensional vector  $g(e) = (g_1(e), \dots, g_k(e))$  is simply the first  $k$  coordinates of the vector  $y(e)$ . We call the matrix whose rows are the vectors  $g(e_1), \dots, g(e_k)$ , where  $e_i$  are the incoming edges for a vertex  $t \in T$ , the global encoding matrix for  $t$  and denote it as  $G_t$ . In practice the encoding vectors are chosen at random so the matrix  $G_t$  is invertible with high probability. Thus any receiver, on receiving  $y_1, \dots, y_k$  can find  $w_1, \dots, w_k$  by solving

$$\begin{bmatrix} y'_1 \\ y'_2 \\ \vdots \\ y'_k \end{bmatrix} = G_t \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix}$$

where the  $y'_i$  are the vectors formed by removing the first  $k$  coordinates of the vector  $y_i$ .

### 62.2 Decoding at the receiver

Each receiver,  $t \in T$ , gets  $k$  vectors  $y_1, \dots, y_k$  which are random linear combinations of the  $v_i$ 's. In fact, if

$$y_i = (\alpha_{i_1}, \dots, \alpha_{i_k}, a_{i_1}, \dots, a_{i_d})$$

then

$$y_i = \sum_{1 \leq j \leq k} (\alpha_{ij} v_j).$$

Thus we can invert the linear transformation to find the  $v_i$ 's with high probability.

## 62.3 History

Krohn, Freedman and Mazieres proposed a theory<sup>[2]</sup> in 2004 that if we have a hash function  $H : V \rightarrow G$  such that:

- $H$  is collision resistant – it is hard to find  $x$  and  $y$  such that  $H(x) = H(y)$  ;
- $H$  is a homomorphism –  $H(x+y) = H(x) + H(y)$  .

Then server can securely distribute  $H(v_i)$  to each receiver, and to check if

$$y = \sum_{1 \leq i \leq k} (\alpha_i v_i)$$

we can check whether

$$H(y) = \sum_{1 \leq i \leq k} (\alpha_i H(v_i))$$

The problem with this method is that the server needs to transfer secure information to each of the receivers. The hash functions  $H$  needs to be transmitted to all the nodes in the network through a separate secure channel.  $H$  is expensive to compute and secure transmission of  $H$  is not economical either.

## 62.4 Advantages of homomorphic signatures

1. Establishes authentication in addition to detecting pollution.
2. No need for distributing secure hash digests.
3. Smaller bit lengths in general will suffice. Signatures of length 180 bits have as much security as 1024 bit RSA signatures.
4. Public information does not change for subsequent file transmission.

## 62.5 Signature scheme

The homomorphic property of the signatures allows nodes to sign any linear combination of the incoming packets without contacting the signing authority.

### 62.5.1 Elliptic curves cryptography over a finite field

Elliptic curve cryptography over a finite field is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields.

Let  $\mathbb{F}_q$  be a finite field such that  $q$  is not a power of 2 or 3. Then an elliptic curve  $E$  over  $\mathbb{F}_q$  is a curve given by an equation of the form

$$y^2 = x^3 + ax + b,$$

where  $a, b \in \mathbb{F}_q$  such that  $4a^3 + 27b^2 \neq 0$

Let  $K \supseteq \mathbb{F}_q$ , then,

$$E(K) = \{(x, y) | y^2 = x^3 + ax + b\} \cup \{O\}$$

forms an abelian group with  $O$  as identity. The group operations can be performed efficiently.

### 62.5.2 Weil pairing

Weil pairing is a construction of roots of unity by means of functions on an elliptic curve  $E$ , in such a way as to constitute a pairing (bilinear form, though with multiplicative notation) on the torsion subgroup of  $E$ . Let  $E/\mathbb{F}_q$  be an elliptic curve and let  $\overline{\mathbb{F}}_q$  be an algebraic closure of  $\mathbb{F}_q$ . If  $m$  is an integer, relatively prime to the characteristic of the field  $\mathbb{F}_q$ , then the group of  $m$ -torsion points,  $E[m] = \{P \in E(\overline{\mathbb{F}}_q) : mP = O\}$ .

If  $E/\mathbb{F}_q$  is an elliptic curve and  $\gcd(m, q) = 1$  then

$$E[m] \cong (\mathbb{Z}/m\mathbb{Z}) * (\mathbb{Z}/m\mathbb{Z})$$

There is a map  $e_m : E[m] * E[m] \rightarrow \mu_m(\mathbb{F}_q)$  such that:

1. (Bilinear)  $e_m(P + R, Q) = e_m(P, Q)e_m(R, Q)$  and  $e_m(P, Q + R) = e_m(P, Q)e_m(P, R)$ .
2. (Non-degenerate)  $e_m(P, Q) = 1$  for all  $P$  implies that  $Q = O$ .
3. (Alternating)  $e_m(P, P) = 1$ .

Also,  $e_m$  can be computed efficiently.<sup>[3]</sup>

### 62.5.3 Homomorphic signatures

Let  $p$  be a prime and  $q$  a prime power. Let  $V/\mathbb{F}_p$  be a vector space of dimension  $D$  and  $E/\mathbb{F}_q$  be an elliptic curve such that  $P_1, \dots, P_D \in E[p]$ . Define  $h : V \rightarrow E[p]$  as follows:  $h(u_1, \dots, u_D) = \sum_{1 \leq i \leq D} (u_i P_i)$ . The function  $h$  is an arbitrary homomorphism from  $V$  to  $E[p]$ .

The server chooses  $s_1, \dots, s_D$  secretly in  $\mathbb{F}_p$  and publishes a point  $Q$  of  $p$ -torsion such that  $e_p(P_i, Q) \neq 1$  and also publishes  $(P_i, s_i Q)$  for  $1 \leq i \leq D$ . The signature of the vector  $v = u_1, \dots, u_D$  is  $\sigma(v) = \sum_{1 \leq i \leq D} (u_i s_i P_i)$ . Note: This signature is homomorphic since the computation of  $h$  is a homomorphism.

### 62.5.4 Signature verification

Given  $v = u_1, \dots, u_D$  and its signature  $\sigma$ , verify that

$$\begin{aligned} e_p(\sigma, Q) &= e_p\left(\sum_{1 \leq i \leq D} (u_i s_i P_i), Q\right) \\ &= \prod_i e_p(u_i s_i P_i, Q) \\ &= \prod_i e_p(u_i P_i, s_i Q) \end{aligned}$$

The verification crucially uses the bilinearity of the Weil-pairing.

## 62.6 System setup

The server computes  $\sigma(v_i)$  for each  $1 \leq i \leq k$ . Transmits  $v_i, \sigma(v_i)$ . At each edge  $e$  while computing  $y(e) = \sum_{f \in E: \text{out}(f) = \text{in}(e)} (m_e(f) y(f))$  also compute  $\sigma(y(e)) = \sum_{f \in E: \text{out}(f) = \text{in}(e)} (m_e(f) \sigma(y(f)))$  on the elliptic curve  $E$ .

The signature is a point on the elliptic curve with coordinates in  $\mathbb{F}_q$ . Thus the size of the signature is  $2 \log q$  bits (which is some constant times  $\log(p)$  bits, depending on the relative size of  $p$  and  $q$ ), and this is the transmission overhead. The computation of the signature  $h(e)$  at each vertex requires  $O(d_{in} \log p \log^{1+\epsilon} q)$  bit operations, where  $d_{in}$  is the in-degree of the vertex  $\text{in}(e)$ . The verification of a signature requires  $O((d+k) \log^{2+\epsilon} q)$  bit operations.

## 62.7 Proof of security

Attacker can produce a collision under the hash function.

If given  $(P_1, \dots, P_r)$  points in  $E[p]$  find  $a = (a_1, \dots, a_r) \in \mathbb{F}_p^r$  and  $b = (b_1, \dots, b_r) \in \mathbb{F}_p^r$  such that  $a \neq b$  and

$$\sum_{1 \leq i \leq r} (a_i P_i) = \sum_{1 \leq j \leq r} (b_j P_j).$$

Proposition: There is a polynomial time reduction from discrete log on the **cyclic group** of order  $p$  on elliptic curves to Hash-Collision.

If  $r = 2$ , then we get  $xP + yQ = uP + vQ$ . Thus  $(x-u)P + (y-v)Q = 0$ . We claim that  $x \neq u$  and  $y \neq v$ . Suppose that  $x = u$ , then we would have  $(y-v)Q = 0$ , but  $Q$  is a point of order  $p$  (a prime) thus  $y-u \equiv 0 \pmod{p}$ . In other words  $y = v$  in  $\mathbb{F}_p$ . This contradicts the assumption that  $(x, y)$  and  $(u, v)$  are distinct pairs in  $\mathbb{F}_2$ . Thus we have that  $Q = -(x-u)(y-v)^{-1}P$ , where the inverse is taken as modulo  $p$ .

If we have  $r > 2$  then we can do one of two things. Either we can take  $P_1 = P$  and  $P_2 = Q$  as before and set  $P_i = O$  for  $i > 2$  (in this case the proof reduces to the case when  $r = 2$ ), or we can take  $P_1 = r_1 P$  and  $P_i = r_i Q$  where  $r_i$  are chosen at random from  $\mathbb{F}_p$ . We get one equation in one unknown (the discrete log of  $Q$ ). It is quite possible that the equation we get does not involve the unknown. However, this happens with very small probability as we argue next. Suppose the algorithm for Hash-Collision gave us that

$$ar_1 P + \sum_{2 \leq i \leq r} (b_i r_i Q) = 0.$$

Then as long as  $\sum_{2 \leq i \leq r} b_i r_i \not\equiv 0 \pmod{p}$ , we can solve for the discrete log of  $Q$ . But the  $r_i$ 's are unknown to the oracle for Hash-Collision and so we can interchange the order in which this process occurs. In other words, given  $b_i$ , for  $2 \leq i \leq r$ , not all zero, what is the probability that the  $r_i$ 's we chose satisfies  $\sum_{2 \leq i \leq r} (b_i r_i) = 0$ ? It is clear that the latter probability is  $\frac{1}{p}$ . Thus with high probability we can solve for the discrete log of  $Q$ .

We have shown that producing hash collisions in this scheme is difficult. The other method by which an adversary can foil our system is by forging a signature. This scheme for the signature is essentially the Aggregate Signature version of the Boneh-Lynn-Shacham signature scheme.<sup>[4]</sup> Here it is shown that forging a signature is at least as hard as solving the **elliptic curve Diffie-Hellman** problem. The only known way to solve this problem on elliptic curves is via computing discrete-logs. Thus forging a signature is at least as hard as solving the computational co-Diffie-Hellman on elliptic curves and probably as hard as computing discrete-logs.

## 62.8 See also

- **Network coding**
- **Homomorphic encryption**

- Elliptic curve cryptography
- Weil pairing
- Elliptic curve Diffie–Hellman
- Elliptic curve DSA
- Digital Signature Algorithm

## 62.9 References

- [1] <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.4738&rep=rep1&type=pdf>
- [2] <http://www.cs.princeton.edu/~{ }mfreed/docs/authcodes-ieee04.pdf>
- [3] <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.8848&rep=rep1&type=pdf>
- [4] <http://cseweb.ucsd.edu/~{ }hovav/dist/sigs.pdf>

## 62.10 External links

1. Comprehensive View of a Live Network Coding P2P System
2. Signatures for Network Coding(presentation) CISS 2006, Princeton
3. University at Buffalo Lecture Notes on Coding Theory – Dr. Atri Rudra

## Chapter 63

# Second Johnson bound

In applied mathematics, the **Johnson bound** (named after **Selmer Martin Johnson**) is a limit on the size of **error-correcting codes**, as used in **coding theory** for **data transmission** or **communications**.

$$A_q(n, d) \leq \frac{q^n}{\sum_{i=0}^t \binom{n}{i} (q-1)^i + \frac{\binom{n}{t+1} (q-1)^{t+1}}{A_q(n, d, t+1)}}.$$

**Theorem 2 (Johnson bound for  $A_q(n, d, w)$ ):**

(i) If  $d > 2w$ ,

$$A_q(n, d, w) = 1.$$

(ii) If  $d \leq 2w$ , then define the variable  $e$  as follows. If  $d$  is even, then define  $e$  through the relation  $d = 2e$ ; if  $d$  is odd, define  $e$  through the relation  $d = 2e - 1$ . Let  $q^* = q - 1$ . Then,

$$A_q(n, d, w) \leq \left\lfloor \frac{nq^*}{w} \left\lfloor \frac{(n-1)q^*}{w-1} \left\lfloor \dots \left\lfloor \frac{(n-w+e)q^*}{e} \right\rfloor \dots \right\rfloor \right\rfloor \right\rfloor$$

where  $\lfloor \cdot \rfloor$  is the **floor function**.

**Remark:** Plugging the bound of Theorem 2 into the bound of Theorem 1 produces a numerical upper bound on  $A_q(n, d)$ .

### 63.1 Definition

Let  $C$  be a  $q$ -ary **code** of length  $n$ , i.e. a subset of  $\mathbb{F}_q^n$ . Let  $d$  be the minimum distance of  $C$ , i.e.

$$d = \min_{x, y \in C, x \neq y} d(x, y),$$

where  $d(x, y)$  is the **Hamming distance** between  $x$  and  $y$ .

Let  $C_q(n, d)$  be the set of all  $q$ -ary codes with length  $n$  and minimum distance  $d$  and let  $C_q(n, d, w)$  denote the set of codes in  $C_q(n, d)$  such that every element has exactly  $w$  nonzero entries.

Denote by  $|C|$  the number of elements in  $C$ . Then, we define  $A_q(n, d)$  to be the largest size of a code with length  $n$  and minimum distance  $d$ :

$$A_q(n, d) = \max_{C \in C_q(n, d)} |C|.$$

Similarly, we define  $A_q(n, d, w)$  to be the largest size of a code in  $C_q(n, d, w)$ :

$$A_q(n, d, w) = \max_{C \in C_q(n, d, w)} |C|.$$

**Theorem 1 (Johnson bound for  $A_q(n, d)$ ):**

If  $d = 2t + 1$ ,

$$A_q(n, d) \leq \frac{q^n}{\sum_{i=0}^t \binom{n}{i} (q-1)^i + \frac{\binom{n}{t+1} (q-1)^{t+1} - \binom{d}{t+1} A_q(n, d, t+1)}{A_q(n, d, t+1)}}.$$

If  $d = 2t$ ,

### 63.2 See also

- Singleton bound
- Hamming bound
- Plotkin bound
- Elias Bassalygo bound
- Gilbert–Varshamov bound
- Griesmer bound

### 63.3 References

- Johnson, Selmer Martin (April 1962). "A new upper bound for error-correcting codes". *IRE Transactions on Information Theory*: 203–207.

- Huffman, William Cary; Pless, Vera S. (2003). *Fundamentals of Error-Correcting Codes*. Cambridge University Press. ISBN 978-0-521-78280-7.

# Chapter 64

## Justesen code

In coding theory, **Justesen codes** form a class of **error-correcting codes** that have a constant rate, constant relative distance, and a constant alphabet size.

Before the Justesen error correction code was discovered, no error correction code was known that had all of these three parameters as a constant.

Subsequently, other ECC codes with this property have been discovered, for example **expander codes**. These codes have important applications in **computer science** such as in the construction of **small-bias sample spaces**.

Justesen codes are derived as the code concatenation of a **Reed–Solomon code** and the **Wozencraft ensemble**.

The Reed–Solomon codes used achieve constant rate and constant relative distance at the expense of an alphabet size that is *linear* in the message length.

The **Wozencraft ensemble** is a family of codes that achieve constant rate and constant alphabet size, but the relative distance is only constant for most of the codes in the family.

The concatenation of the two codes first encodes the message using the Reed–Solomon code, and then encodes each symbol of the codeword further using a code from the **Wozencraft ensemble** – using a different code of the ensemble at each position of the codeword.

This is different from usual code concatenation where the inner codes are the same for each position. The Justesen code can be constructed very efficiently using only **logarithmic space**.

### 64.1 Definition

Justesen code is **concatenation code** with different linear inner codes, which is composed of an  $(N, K, D)_{q^k}$  outer code  $C_{out}$  and different  $(n, k, d)_q$  inner codes  $C_{in}^i$ ,  $1 \leq i \leq N$ .

More precisely, the concatenation of these codes, denoted by  $C_{out} \circ (C_{in}^1, \dots, C_{in}^N)$ , is defined as follows. Given a message  $m \in [q^k]^K$ , we compute the codeword produced by an outer code  $C_{out} : C_{out}(m) = (c_1, c_2, \dots, c_N)$ .

Then we apply each code of  $N$  linear inner codes to each coordinate of that codeword to produce the final codeword; that is,  $C_{out} \circ (C_{in}^1, \dots, C_{in}^N)(m) = (C_{in}^1(c_1), C_{in}^2(c_2), \dots, C_{in}^N(c_N))$ .

Look back to the definition of the outer code and linear inner codes, this definition of the Justesen code makes sense because the codeword of the outer code is a vector with  $N$  elements, and we have  $N$  linear inner codes to apply for those  $N$  elements.

Here for the Justesen code, the outer code  $C_{out}$  is chosen to be **Reed Solomon code** over a field  $\mathbb{F}_{q^k}$  evaluated over  $\mathbb{F}_{q^k} - \{0\}$  of rate  $R$ ,  $0 < R < 1$ .

The outer code  $C_{out}$  have the relative distance  $\delta_{out} = 1 - R$  and block length of  $N = q^k - 1$ . The set of inner codes is the **Wozencraft ensemble**  $\{C_{in}^\alpha\}_{\alpha \in \mathbb{F}_{q^k}^*}$ .

### 64.2 Property of Justesen code

As the linear codes in the Wozencraft ensemble have the rate  $\frac{1}{2}$ , Justesen code is the concatenated code  $C^* = C_{out} \circ (C_{in}^1, C_{in}^2, \dots, C_{in}^N)$  with the rate  $\frac{R}{2}$ . We have the following theorem that estimates the distance of the concatenated code  $C^*$ .

### 64.3 Theorem

Let  $\varepsilon > 0$ . Then  $C^*$  has relative distance of at least  $(1 - R - \varepsilon)H_q^{-1}(\frac{1}{2} - \varepsilon)$ .

#### 64.3.1 Proof

In order to prove a lower bound for the distance of a code  $C^*$  we prove that the Hamming distance of an arbitrary but distinct pair of codewords has a lower bound. So let  $\Delta(c^1, c^2)$  be the Hamming distance of two codewords  $c^1$  and  $c^2$ . For any given

$$m_1 \neq m_2 \in (\mathbb{F}_{q^k})^K,$$

we want a lower bound for  $\Delta(C^*(m_1), C^*(m_2))$ .



Notice that if  $C_{out}(m) = (c_1, \dots, c_N)$ , then  $C^*(m) = (C_{in}^1(c_1), \dots, C_{in}^N(c_N))$ . So for the lower bound  $\Delta(C^*(m_1), C^*(m_2))$ , we need to take into account the distance of  $C_{in}^1, \dots, C_{in}^N$ .

Suppose

$$\begin{aligned} C_{out}(m_1) &= (c_1^1, \dots, c_N^1) \\ C_{out}(m_2) &= (c_1^2, \dots, c_N^2) \end{aligned}$$

Recall that  $\{C_{in}^1, \dots, C_{in}^N\}$  is a **Wozencraft ensemble**. Due to “Wozencraft ensemble theorem”, there are at least  $(1 - \varepsilon)N$  linear codes  $C_{in}^i$  that have distance  $H_q^{-1}(\frac{1}{2} - \varepsilon) \cdot 2k$ . So if for some  $1 \leq i \leq N$ ,  $c_i^1 \neq c_i^2$  and the code  $C_{in}^i$  has distance  $\geq H_q^{-1}(\frac{1}{2} - \varepsilon) \cdot 2k$ , then

$$\Delta(C_{in}^i(c_1^1), C_{in}^i(c_2^2)) \geq H_q^{-1}(\frac{1}{2} - \varepsilon) \cdot 2k.$$

Further, if we have  $T$  numbers  $1 \leq i \leq N$  such that  $c_i^1 \neq c_i^2$  and the code  $C_{in}^i$  has distance  $\geq H_q^{-1}(\frac{1}{2} - \varepsilon) \cdot 2k$ , then

$$\Delta(C^*(m_1), C^*(m_2)) \geq H_q^{-1}(\frac{1}{2} - \varepsilon) \cdot 2k \cdot T.$$

So now the final task is to find a lower bound for  $T$ . Define:

$$S = \{i : 1 \leq i \leq N, c_i^1 \neq c_i^2\}.$$

Then  $T$  is the number of linear codes  $C_{in}^i, i \in S$  having the distance  $H_q^{-1}(\frac{1}{2} - \varepsilon) \cdot 2k$ .

Now we want to estimate  $|S|$ . Obviously  $|S| = \Delta(C_{out}(m_1), C_{out}(m_2)) \geq (1 - R)N$ .

Due to the **Wozencraft Ensemble Theorem**, there are at most  $\varepsilon N$  linear codes having distance less than  $H_q^{-1}(\frac{1}{2} - \varepsilon) \cdot 2k$ , so

$$T \geq |S| - \varepsilon N \geq (1 - R)N - \varepsilon N = (1 - R - \varepsilon)N.$$

Finally, we have

$$\Delta(C^*(m_1), C^*(m_2)) \geq H_q^{-1}(\frac{1}{2} - \varepsilon) \cdot 2k \cdot T \geq H_q^{-1}(\frac{1}{2} - \varepsilon) \cdot 2k \cdot (1 - R - \varepsilon) \cdot N.$$

This is true for any arbitrary  $m_1 \neq m_2$ . So  $C^*$  has the relative distance at least  $(1 - R - \varepsilon)H_q^{-1}(\frac{1}{2} - \varepsilon)$ , which completes the proof.

## 64.4 Comments

We want to consider the “strongly explicit code”. So the question is what the “strongly explicit code” is. Loosely

speaking, for linear code, the “explicit” property is related to the complexity of constructing its generator matrix  $G$ .

That in effect means that we can compute the matrix in logarithmic space without using the brute force algorithm to verify that a code has a given satisfied distance.

For the other codes that are not linear, we can consider the complexity of the encoding algorithm.

So by far, we can see that the Wozencraft ensemble and Reed-Solomon codes are strongly explicit. Therefore, we have the following result:

**Corollary:** The concatenated code  $C^*$  is an asymptotically good code (that is, rate  $R > 0$  and relative distance  $\delta > 0$  for small  $q$ ) and has a strongly explicit construction.

## 64.5 An example of a Justesen code

The following slightly different code is referred to as the Justesen code in MacWilliams/MacWilliams. It is the particular case of the above-considered Justesen code for a very particular Wozencraft ensemble:

Let  $R$  be a Reed-Solomon code of length  $N = 2^m - 1$ , rank  $K$  and minimum weight  $N - K + 1$ .

The symbols of  $R$  are elements of  $F = \text{GF}(2^m)$  and the codewords are obtained by taking every polynomial  $f$  over  $F$  of degree less than  $K$  and listing the values of  $f$  on the non-zero elements of  $F$  in some predetermined order.

Let  $\alpha$  be a **primitive element** of  $F$ . For a codeword  $\mathbf{a} = (a_1, \dots, a_N)$  from  $R$ , let  $\mathbf{b}$  be the vector of length  $2N$  over  $F$  given by

$$\mathbf{b} = (a_1, a_1, a_2, \alpha^1 a_2, \dots, a_N, \alpha^{N-1} a_N)$$

and let  $\mathbf{c}$  be the vector of length  $2N$  obtained from  $\mathbf{b}$  by expressing each element of  $F$  as a binary vector of length  $m$ . The *Justesen code* is the linear code containing all such  $\mathbf{c}$ .

The parameters of this code are length  $2mN$ , dimension  $mK$  and **minimum distance** at least

where  $\ell$  is the greatest integer satisfying  $\sum_{i=1}^{\ell} \binom{2m}{i} \leq N - K + 1$ . (See MacWilliams/MacWilliams for a proof.)

## 64.6 See also

1. **Wozencraft ensemble**

2. Concatenated error correction code
3. Reed-Solomon error correction
4. Linear Code

## 64.7 References

1. Lecture 28: Justesen Code. Coding theory's course. Prof. Atri Rudra.
2. Lecture 6: Concatenated codes. Forney codes. Justesen codes. Essential Coding Theory.
3. J. Justesen (1972). "A class of constructive asymptotically good algebraic codes". *IEEE Trans. Inf. Theory*. **18** (5): 652–656. doi:10.1109/TIT.1972.1054893.
4. F.J. MacWilliams; N.J.A. Sloane (1977). *The Theory of Error-Correcting Codes*. North-Holland. pp. 306–316. ISBN 0-444-85193-3.

## Chapter 65

# Kraft–McMillan inequality

In coding theory, the **Kraft–McMillan inequality** gives a necessary and sufficient condition for the existence of a prefix code<sup>[1]</sup> (in Kraft's version) or a uniquely decodable code (in McMillan's version) for a given set of codeword lengths. Its applications to prefix codes and trees often find use in computer science and information theory.

Kraft's inequality was published in Kraft (1949). However, Kraft's paper discusses only prefix codes, and attributes the analysis leading to the inequality to Raymond Redheffer. The result was independently discovered of the result in McMillan (1956). McMillan proves the result for the general case of uniquely decodable codes, and attributes the version for prefix codes to a spoken observation in 1955 by Joseph Leo Doob.

$$S = \{s_1, s_2, \dots, s_n\}$$

be encoded into a uniquely decodable code over an alphabet of size  $r$  with codeword lengths

$$\ell_1, \ell_2, \dots, \ell_n.$$

Then

$$\sum_{i=1}^n r^{-\ell_i} \leq 1.$$

Conversely, for a given set of natural numbers  $\ell_1, \ell_2, \dots, \ell_n$  satisfying the above inequality, there exists a uniquely decodable code over an alphabet of size  $r$  with those codeword lengths.

### 65.1 Applications and intuitions

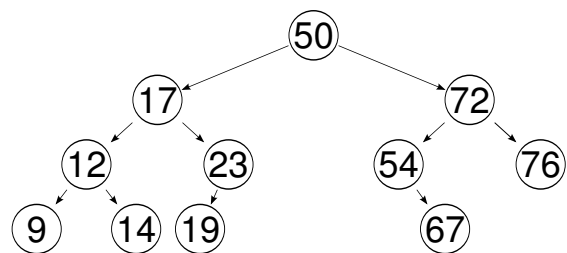
Kraft's inequality limits the lengths of codewords in a prefix code: if one takes an exponential of the length of each valid codeword, the resulting set of values must look like a probability mass function, that is, it must have total measure less than or equal to one. Kraft's inequality can be thought of in terms of a constrained budget to be spent on codewords, with shorter codewords being more expensive. Among the useful properties following from the inequality are the following statements:

- If Kraft's inequality holds with strict inequality, the code has some redundancy.
- If Kraft's inequality holds with equality, the code in question is a complete code.
- If Kraft's inequality does not hold, the code is not uniquely decodable.
- For every uniquely decodable code, there exists a prefix code with the same length distribution.

### 65.2 Formal statement

Let each source symbol from the alphabet

### 65.3 Example: binary trees



9, 14, 19, 67 and 76 are leaf nodes at depths of 3, 3, 3, 3 and 2, respectively.

Any binary tree can be viewed as defining a prefix code for the leaves of the tree. Kraft's inequality states that

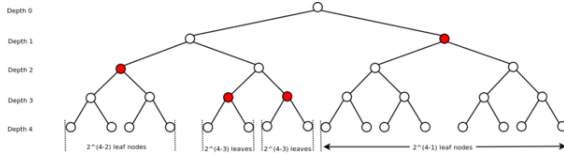
$$\sum_{\ell \in \text{leaves}} 2^{-\text{depth}(\ell)} \leq 1.$$

Here the sum is taken over the leaves of the tree, i.e. the nodes without any children. The depth is the distance to the root node. In the tree to the right, this sum is

$$\frac{1}{4} + 4 \left( \frac{1}{8} \right) = \frac{3}{4} \leq 1.$$

## 65.4 Proof

### 65.4.1 Proof for prefix codes



Example for binary tree. Red nodes represent a prefix tree. The method for calculating the number of descendant leaf nodes in the full tree is shown.

First, let us show that the Kraft inequality holds whenever  $S$  is a prefix code.

Suppose that  $\ell_1 \leq \ell_2 \leq \dots \leq \ell_n$ . Let  $A$  be the full  $r$ -ary tree of depth  $\ell_n$  (thus, every node of  $A$  at level  $< \ell_n$  has  $r$  children, while the nodes at level  $\ell_n$  are leaves). Every word of length  $\ell \leq \ell_n$  over an  $r$ -ary alphabet corresponds to a node in this tree at depth  $\ell$ . The  $i$ th word in the **prefix code** corresponds to a node  $v_i$ ; let  $A_i$  be the set of all leaf nodes (i.e. of nodes at depth  $\ell_n$ ) in the subtree of  $A$  rooted at  $v_i$ . That subtree being of height  $\ell_n - \ell_i$ , we have

$$|A_i| = r^{\ell_n - \ell_i}.$$

Since the code is a prefix code, those subtrees cannot share any leaves, which means that

$$A_i \cap A_j = \emptyset, \quad i \neq j.$$

Thus, given that the total number of nodes at depth  $\ell_n$  is  $r^{\ell_n}$ , we have

$$\left| \bigcup_{i=1}^n A_i \right| = \sum_{i=1}^n |A_i| = \sum_{i=1}^n r^{\ell_n - \ell_i} \leq r^{\ell_n}$$

from which the result follows.

Conversely, given any ordered sequence of  $n$  natural numbers,

$$\ell_1 \leq \ell_2 \leq \dots \leq \ell_n$$

satisfying the Kraft inequality, one can construct a prefix code with codeword lengths equal to each  $\ell_i$  by choosing

a word of length  $\ell_i$  arbitrarily, then ruling out all words of greater length that have it as a prefix. There again, we shall interpret this in terms of leaf nodes of an  $r$ -ary tree of depth  $\ell_n$ . First choose any node from the full tree at depth  $\ell_1$ ; it corresponds to the first word of our new code. Since we are building a prefix code, all the descendants of this node (i.e., all words that have this first word as a prefix) become unsuitable for inclusion in the code. We consider the descendants at depth  $\ell_n$  (i.e., the leaf nodes among the descendants); there are  $r^{\ell_n - \ell_1}$  such descendant nodes that are removed from consideration. The next iteration picks a (surviving) node at depth  $\ell_2$  and removes  $r^{\ell_n - \ell_2}$  further leaf nodes, and so on. After  $n$  iterations, we have removed a total of

$$\sum_{i=1}^n r^{\ell_n - \ell_i}$$

nodes. The question is whether we need to remove more leaf nodes than we actually have available —  $r^{\ell_n}$  in all — in the process of building the code. Since the Kraft inequality holds, we have indeed

$$\sum_{i=1}^n r^{\ell_n - \ell_i} \leq r^{\ell_n}$$

and thus a prefix code can be built. Note that as the choice of nodes at each step is largely arbitrary, many different suitable prefix codes can be built, in general.

### 65.4.2 Proof of the general case

Now, we will prove that the Kraft inequality holds whenever  $S$  is a uniquely decodable code. (The converse needs not be proven, since we have already proven it for prefix codes, which is a stronger claim.)

Consider the generating function in inverse of  $x$  for the code  $S$

$$F(x) = \sum_{i=1}^n x^{-|s_i|} = \sum_{\ell=\min}^{\max} p_\ell x^{-\ell}$$

in which  $p_\ell$  —the coefficient in front of  $x^{-\ell}$ —is the number of distinct codewords of length  $\ell$ . Here min is the length of the shortest codeword in  $S$ , and max is the length of the longest codeword in  $S$ .

Consider all  $m$ -powers  $S^m$ , in the form of words  $s_{i_1} s_{i_2} \dots s_{i_m}$ , where  $i_1, i_2, \dots, i_m$  are indices between 1 and  $n$ . Note that, since  $S$  was assumed to uniquely decodable,  $s_{i_1} s_{i_2} \dots s_{i_m} = s_{j_1} s_{j_2} \dots s_{j_m}$  implies  $i_1 = j_1, i_2 = j_2, \dots, i_m = j_m$ . Because of this property, one can compute the generating function  $G(x)$  for  $S^m$  from the generating function  $F(x)$  as

$$\begin{aligned}
G(x) &= (F(x))^m = \left( \sum_{i=1}^n x^{-|s_i|} \right)^m \\
&= \sum_{i_1=1}^n \sum_{i_2=1}^n \cdots \sum_{i_m=1}^n x^{-(|s_{i_1}| + |s_{i_2}| + \cdots + |s_{i_m}|)} \\
&= \sum_{i_1=1}^n \sum_{i_2=1}^n \cdots \sum_{i_m=1}^n x^{-|s_{i_1} s_{i_2} \cdots s_{i_m}|} = \sum_{\ell=m \cdot \min}^{m \cdot \max} q_\ell x^{-\ell}
\end{aligned}$$

Here, similarly as before,  $q_\ell$  — the coefficient in front of  $x^{-\ell}$  in  $G(x)$  — is the number of words of length  $\ell$  in  $S^m$ . Clearly,  $q_\ell$  cannot exceed  $r^\ell$ . Hence for any positive  $x$ ,

$$(F(x))^m \leq \sum_{\ell=m \cdot \min}^{m \cdot \max} r^\ell x^{-\ell}.$$

Substituting the value  $x = r$  we have

$$(F(r))^m \leq m \cdot (\max - \min) + 1$$

for any positive integer  $m$ . The left side of the inequality grows exponentially in  $m$  and the right side only linearly. The only possibility for the inequality to be valid for all  $m$  is that  $F(r) \leq 1$ . Looking back on the definition of  $F(x)$  we finally get the inequality.

$$\sum_{i=1}^n r^{-\ell_i} = \sum_{i=1}^n r^{-|s_i|} = F(r) \leq 1.$$

### 65.4.3 Alternative construction for the converse

Given a sequence of  $n$  natural numbers,

$$\ell_1 \leq \ell_2 \leq \cdots \leq \ell_n$$

satisfying the Kraft inequality, we can construct a prefix code as follows. Define the  $i^{\text{th}}$  codeword,  $C_i$ , to be the first  $\ell_i$  digits after the radix point (e.g. decimal point) in the base  $r$  representation of

$$\sum_{j=1}^{i-1} r^{-\ell_j}.$$

Note that by Kraft's inequality, this sum is never more than 1. Hence the codewords capture the entire value of the sum. Therefore, for  $j > i$ , the first  $\ell_i$  digits of  $C_j$  form a larger number than  $C_i$ , so the code is prefix free.

## 65.5 Notes

- [1] Cover, Thomas M.; Thomas, Joy A. (2006), *Elements of Information Theory* (PDF) (2nd ed.), John Wiley & Sons, Inc, pp. 108–109, doi:10.1002/047174882X.ch5, ISBN 0-471-24195-4

## 65.6 References

- Kraft, Leon G. (1949), *A device for quantizing, grouping, and coding amplitude modulated pulses*, Cambridge, MA: MS Thesis, Electrical Engineering Department, Massachusetts Institute of Technology.
- McMillan, Brockway (1956), “Two inequalities implied by unique decipherability”, *IEEE Trans. Information Theory*, **2** (4): 115–116, doi:10.1109/TIT.1956.1056818.

## 65.7 See also

Chaitin's constant, Canonical Huffman code.

# Chapter 66

## Lee distance

In coding theory, the **Lee distance** is a distance between two strings  $x_1x_2 \dots x_n$  and  $y_1y_2 \dots y_n$  of equal length  $n$  over the  $q$ -ary alphabet  $\{0, 1, \dots, q-1\}$  of size  $q \geq 2$ . It is a metric, defined as

$$\sum_{i=1}^n \min((x_i - y_i), q - (x_i - y_i)).^{[1]}$$

Considering the alphabet as the additive group  $\mathbf{Z}_q$ , the Lee distance between two single letters  $x$  and  $y$  is the length of shortest path in the **Cayley graph** (which is circular since the group is cyclic) between them.<sup>[2]</sup>

If  $q = 2$  or  $q = 3$  the Lee distance coincides with the **Hamming distance**, because both distances are 0 for two single equal symbols and 1 for two single non-equal symbols. For  $q > 3$  this is not the case anymore, the Lee distance can become bigger than 1.

The metric space induced by the Lee distance is a discrete analog of the elliptic space.<sup>[1]</sup>

### 66.1 Example

If  $q = 6$ , then the Lee distance between 3140 and 2543 is  $1 + 2 + 0 + 3 = 6$ .

### 66.2 History and application

The Lee distance is named after C. Y. Lee. It is applied for phase modulation while the Hamming distance is used in case of orthogonal modulation.

The **Berlekamp code** is an example of code in the Lee metric.<sup>[3]</sup> Other significant examples are the **Preparata code** and **Kerdock code**; these codes are non-linear when considered over a field, but are linear over a ring.<sup>[4]</sup>

Also, there exists a **Gray isometry** (bijection preserving weight) between  $\mathbb{Z}_4$  with the Lee weight and  $\mathbb{Z}_2^2$  with the Hamming weight.<sup>[4]</sup>

### 66.3 References

- [1] Deza, Elena; Deza, Michel (2014), *Dictionary of Distances* (3rd ed.), Elsevier, p. 52, ISBN 9783662443422
- [2] Blahut, Richard E. (2008). *Algebraic Codes on Lines, Planes, and Curves: An Engineering Approach*. Cambridge University Press. p. 108. ISBN 978-1-139-46946-3.
- [3] Roth, Ron (2006). *Introduction to Coding Theory*. Cambridge University Press. p. 314. ISBN 978-0-521-84504-5.
- [4] Greferath, Marcus (2009). "An Introduction to Ring-Linear Coding Theory". In Sala, Massimiliano; Mora, Teo; Perret, Ludovic; Sakata, Shojiro; and Traverso, Carlo. *Gröbner Bases, Coding, and Cryptography*. Springer Science & Business Media. p. 220. ISBN 978-3-540-93806-4.
- Lee, C. Y. (1958), "Some properties of non-binary error-correcting codes", *IRE Transactions on Information Theory*, **4** (2): 77–82, doi:10.1109/TIT.1958.1057446
- Berlekamp, Elwyn R. (1968), *Algebraic Coding Theory*, McGraw-Hill
- Voloch, Jose Felipe; Walker, Judy L. (1998). "Lee Weights of Codes from Elliptic Curves". In Vardy, Alexander. *Codes, Curves, and Signals: Common Threads in Communications*. Springer Science & Business Media. ISBN 978-1-4615-5121-8.

# Chapter 67

## Linear code

In coding theory, a **linear code** is an error-correcting code for which any linear combination of codewords is also a codeword. Linear codes are traditionally partitioned into block codes and convolutional codes, although turbo codes can be seen as a hybrid of these two types.<sup>[1]</sup> Linear codes allow for more efficient encoding and decoding algorithms than other codes (cf. syndrome decoding).

Linear codes are used in forward error correction and are applied in methods for transmitting symbols (e.g., bits) on a communications channel so that, if errors occur in the communication, some errors can be corrected or detected by the recipient of a message block. The codewords in a linear block code are blocks of symbols that are encoded using more symbols than the original value to be sent.<sup>[2]</sup> A linear code of length  $n$  transmits blocks containing  $n$  symbols. For example, the [7,4,3] Hamming code is a linear binary code which represents 4-bit messages using 7-bit codewords. Two distinct codewords differ in at least three bits. As a consequence, up to two errors per codeword can be detected while a single error can be corrected.<sup>[3]</sup> This code contains  $2^4=16$  codewords.

### 67.1 Definition and parameters

A linear code of length  $n$  and rank  $k$  is a linear subspace  $C$  with dimension  $k$  of the vector space  $\mathbb{F}_q^n$  where  $\mathbb{F}_q$  is the finite field with  $q$  elements. Such a code is called a  $q$ -ary code. If  $q = 2$  or  $q = 3$ , the code is described as a binary code, or a ternary code respectively. The vectors in  $C$  are called codewords. The size of a code is the number of codewords and equals  $q^k$ .

The weight of a codeword is the number of its elements that are nonzero and the distance between two codewords is the Hamming distance between them, that is, the number of elements in which they differ. The distance  $d$  of a linear code is the minimum weight of its nonzero codewords, or equivalently, the minimum distance between distinct codewords. A linear code of length  $n$ , dimension  $k$ , and distance  $d$  is called an  $[n,k,d]$  code.

We want to give  $\mathbb{F}_q^n$  the standard basis because each coordinate represents a “bit” that is transmitted across a “noisy

channel” with some small probability of transmission error (a binary symmetric channel). If some other basis is used then this model cannot be used and the Hamming metric does not measure the number of errors in transmission, as we want it to.

### 67.2 Generator and check matrices

As a linear subspace of  $\mathbb{F}_q^n$ , the entire code  $C$  (which may be very large) may be represented as the span of a minimal set of codewords (known as a basis in linear algebra). These basis codewords are often collated in the rows of a matrix  $G$  known as a generating matrix for the code  $C$ . When  $G$  has the block matrix form  $G = (I_k | A)$ , where  $I_k$  denotes the  $k \times k$  identity matrix and  $A$  is some  $k \times (n - k)$  matrix, then we say  $G$  is in standard form.

A matrix  $H$  representing a linear function  $\phi : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^{n-k}$  whose kernel is  $C$  is called a check matrix of  $C$  (or sometimes a parity check matrix). Equivalently,  $H$  is a matrix whose null space is  $C$ . If  $C$  is a code with a generating matrix  $G$  in standard form,  $G = (I_k | A)$ , then  $H = (-A^t | I_{n-k})$  is a check matrix for  $C$ . The code generated by  $H$  is called the dual code of  $C$ .

Linearity guarantees that the minimum Hamming distance  $d$  between a codeword  $c_0$  and any of the other codewords  $c \neq c_0$  is independent of  $c_0$ . This follows from the property that the difference  $c - c_0$  of two codewords in  $C$  is also a codeword (i.e., an element of the subspace  $C$ ), and the property that  $d(c, c_0) = d(c - c_0, 0)$ . These properties imply that

$$\min_{c \in C, c \neq c_0} d(c, c_0) = \min_{c \in C, c \neq c_0} d(c - c_0, 0) = \min_{c \in C, c \neq 0} d(c, 0) = d.$$

In other words, in order to find out the minimum distance between the codewords of a linear code, one would only need to look at the non-zero codewords. The non-zero codeword with the smallest weight has then the minimum distance to the zero codeword, and hence determines the minimum distance of the code.

The distance  $d$  of a linear code  $C$  also equals the minimum number of linearly dependent columns of the check matrix  $H$ .



*Proof:* Because  $\mathbf{H} \cdot \mathbf{c}^T = \mathbf{0}$ , which is equivalent to  $\sum_{i=1}^n (c_i \cdot \mathbf{H}_i) = \mathbf{0}$ , where  $\mathbf{H}_i$  is the  $i^{th}$  column of  $\mathbf{H}$ . Remove those items with  $c_i = 0$ , those  $\mathbf{H}_i$  with  $c_i \neq 0$  are linearly dependent. Therefore,  $d$  is at least the minimum number of linearly dependent columns. On another hand, consider the minimum set of linearly dependent columns  $\{\mathbf{H}_j | j \in S\}$  where  $S$  is the column index set.  $\sum_{i=1}^n (c_i \cdot \mathbf{H}_i) = \sum_{j \in S} (c_j \cdot \mathbf{H}_j) + \sum_{j \notin S} (c_j \cdot \mathbf{H}_j) = \mathbf{0}$ . Now consider the vector  $\mathbf{c}'$  such that  $c'_j = 0$  if  $j \notin S$ . Note  $\mathbf{c}' \in C$  because  $\mathbf{H} \cdot \mathbf{c}'^T = \mathbf{0}$ . Therefore, we have  $d \leq wt(\mathbf{c}')$ , which is the minimum number of linearly dependent columns in  $\mathbf{H}$ . The claimed property is therefore proved.

### 67.3 Example: Hamming codes

Main article: [Hamming code](#)

As the first class of linear codes developed for error correction purpose, the *Hamming codes* has been widely used in digital communication systems. For any positive integer  $r \geq 2$ , there exists a  $[2^r - 1, 2^r - r - 1, 3]_2$  Hamming code. Since  $d = 3$ , this Hamming code can correct a 1-bit error.

**Example :** The linear block code with the following generator matrix and parity check matrix is a  $[7, 4, 3]_2$  Hamming code.

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}, \quad \mathbf{H} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

### 67.4 Example: Hadamard codes

Main article: [Hadamard code](#)

**Hadamard code** is a  $[2^r, r, 2^{r-1}]_2$  linear code and is capable of correcting many errors. Hadamard code could be constructed column by column : the  $i^{th}$  column is the bits of the binary representation of integer  $i$ , as shown in the following example. Hadamard code has minimum distance  $2^{r-1}$  and therefore can correct  $2^{r-2} - 1$  errors.

**Example :** The linear block code with the following generator matrix is a  $[8, 3, 4]_2$  Hadamard code:  $\mathbf{G}_{Had} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$ .

**Hadamard code** is a special case of **Reed-Muller code**. If we take the first column (the all-zero column) out from

$\mathbf{G}_{Had}$ , we get  $[7, 3, 4]_2$  *simplex code*, which is the *dual code* of Hamming code.

### 67.5 Nearest neighbor algorithm

The parameter  $d$  is closely related to the error correcting ability of the code. The following construction/algorithm illustrates this (called the nearest neighbor decoding algorithm):

Input: A "received vector"  $\mathbf{v}$  in  $\mathbb{F}_q^n$ .

Output: A codeword  $\mathbf{w}$  in  $C$  closest to  $\mathbf{v}$ .

- Enumerate the elements of the ball of (Hamming) radius  $t$  around the received word  $\mathbf{v}$ , denoted  $B_t(\mathbf{v})$ .
- For each  $\mathbf{w}$  in  $B_t(\mathbf{v})$ , check if  $\mathbf{w}$  in  $C$ . If so, return  $\mathbf{w}$  as the solution!
- Fail when enumeration is complete and no solution has been found.

Note: "fail" is not returned unless  $t > (d - 1)/2$ . We say that a linear  $C$  is  $t$ -error correcting if there is at most one codeword in  $B_t(\mathbf{v})$ , for each  $\mathbf{v}$  in  $\mathbb{F}_q^n$ .

### 67.6 Popular notation

Main article: [Block\\_code § Popular\\_notation](#)

**Codes** in general are often denoted by the letter  $C$ , and a code of length  $n$  and of **rank**  $k$  (i.e., having  $k$  code words in its basis and  $k$  rows in its *generating matrix*) is generally referred to as an  $(n, k)$  code. Linear block codes are frequently denoted as  $[n, k, d]$  codes, where  $d$  refers to the code's minimum Hamming distance between any two code words.

(The  $[n, k, d]$  notation should not be confused with the  $(n, M, d)$  notation used to denote a *non-linear* code of length  $n$ , size  $M$  (i.e., having  $M$  code words), and minimum Hamming distance  $d$ .)

### 67.7 Singleton bound

**Lemma (Singleton bound):** Every linear  $[n, k, d]$  code  $C$  satisfies  $k + d \leq n + 1$ .

A code  $C$  whose parameters satisfy  $k + d = n + 1$  is called **maximum distance separable** or **MDS**. Such codes, when they exist, are in some sense best possible.

If  $C_1$  and  $C_2$  are two codes of length  $n$  and if there is a permutation  $p$  in the **symmetric group**  $S_n$  for which  $(c_1, \dots, c_n)$  in  $C_1$  if and only if  $(c_{p(1)}, \dots, c_{p(n)})$  in  $C_2$ , then we say  $C_1$  and  $C_2$  are **permutation equivalent**. In

more generality, if there is an  $n \times n$  monomial matrix  $M: \mathbb{F}_q^n \rightarrow \mathbb{F}_q^n$  which sends  $C_1$  isomorphically to  $C_2$  then we say  $C_1$  and  $C_2$  are **equivalent**.

*Lemma:* Any linear code is permutation equivalent to a code which is in standard form.

## 67.8 Examples

Some examples of linear codes include:

- Repetition codes
- Parity codes
- Cyclic codes
- Hamming codes
- Golay code, both the binary and ternary versions
- Polynomial codes, of which BCH codes are an example
- Reed–Solomon codes
- Reed–Muller codes
- Goppa codes
- Low-density parity-check codes
- Expander codes
- Multidimensional parity-check codes

## 67.9 Generalization

Hamming spaces over non-field alphabets have also been considered, especially over finite rings (most notably over  $\mathbb{Z}_4$ ) giving rise to modules instead of vector spaces and ring-linear codes (identified with submodules) instead of linear codes. The typical metric used in this case the Lee distance. There exist a Gray isometry between  $\mathbb{Z}_2^{2m}$  (i.e.  $\text{GF}(2^{2m})$ ) with the Hamming distance and  $\mathbb{Z}_4^m$  (also denoted as  $\text{GR}(4,m)$ ) with the Lee distance; its main attraction is that it establishes a correspondence between some “good” codes that are not linear over  $\mathbb{Z}_2^{2m}$  as images of ring-linear codes from  $\mathbb{Z}_4^m$ .<sup>[4][5][6]</sup>

More recently, some authors have referred to such codes over rings simply as linear codes as well.<sup>[7]</sup>

## 67.10 See also

- Decoding methods

## 67.11 References

- [1] William E. Ryan and Shu Lin (2009). *Channel Codes: Classical and Modern*. Cambridge University Press. p. 4. ISBN 978-0-521-84868-8.
  - [2] MacKay, David, J.C. (2003). *Information Theory, Inference, and Learning Algorithms* (PDF). Cambridge University Press. p. 9. ISBN 9780521642989. In a linear block code, the extra  $N - K$  bits are linear functions of the original  $K$  bits; these extra bits are called *parity-check bits*
  - [3] Thomas M. Cover and Joy A. Thomas (1991). *Elements of Information Theory*. John Wiley & Sons, Inc. pp. 210–211. ISBN 0-471-06259-6.
  - [4] Marcus Greferath (2009). “An Introduction to Ring-Linear Coding Theory”. In Massimiliano Sala, Teo Mora, Ludovic Perret, Shojiro Sakata, Carlo Traverso. *Gröbner Bases, Coding, and Cryptography*. Springer Science & Business Media. ISBN 978-3-540-93806-4.
  - [5] [http://www.encyclopediaofmath.org/index.php/Kerdock\\_and\\_Preparata\\_codes](http://www.encyclopediaofmath.org/index.php/Kerdock_and_Preparata_codes)
  - [6] J.H. van Lint (1999). *Introduction to Coding Theory* (3rd ed.). Springer. Chapter 8: Codes over  $\mathbb{Z}_4$ . ISBN 978-3-540-64133-9.
  - [7] S.T. Dougherty, J.-L. Kim, P. Sole (2015). “Open Problems in Coding Theory”. In Steven Dougherty, Alberto Facchini, Andre Gerard Leroy, Edmund Puczyłowski, Patrick Sole. *Noncommutative Rings and Their Applications*. American Mathematical Soc. p. 80. ISBN 978-1-4704-1032-2.
- J. F. Humphreys; M. Y. Prest (2004). *Numbers, Groups and Codes* (2nd ed.). Cambridge University Press. ISBN 978-0-511-19420-7. Chapter 5 contains a more gentle introduction (than this article) to the subject of linear codes.

## 67.12 External links

- $q$ -ary code generator program
- Code Tables: Bounds on the parameters of various types of codes, *IAKS, Fakultät für Informatik, Universität Karlsruhe (TH)*. Online, up to date table of the optimal binary codes, includes non-binary codes.
- The database of  $\mathbb{Z}_4$  codes Online, up to date database of optimal  $\mathbb{Z}_4$  codes.

# Chapter 68

## List decoding

In computer science, particularly in coding theory, **list decoding** is an alternative to unique decoding of error-correcting codes for large error rates. The notion was proposed by Elias in the 1950s. The main idea behind list decoding is that the decoding algorithm instead of outputting a single possible message outputs a list of possibilities one of which is correct. This allows for handling a greater number of errors than that allowed by unique decoding.

The unique decoding model in coding theory, which is constrained to output a single valid codeword from the received word could not tolerate greater fraction of errors. This resulted in a gap between the error-correction performance for stochastic noise models (proposed by Shannon) and the adversarial noise model (considered by Richard Hamming). Since the mid 90s, significant algorithmic progress by the coding theory community has bridged this gap. Much of this progress is based on a relaxed error-correction model called list decoding, wherein the decoder outputs a list of codewords for worst-case pathological error patterns where the actual transmitted codeword is included in the output list. In case of typical error patterns though, the decoder outputs a unique single codeword, given a received word, which is almost always the case (However, this is not known to be true for all codes). The improvement here is significant in that the error-correction performance doubles. This is because now the decoder is not confined by the half-the-minimum distance barrier. This model is very appealing because having a list of codewords is certainly better than just giving up. The notion of list-decoding has many interesting applications in complexity theory.

The way the channel noise is modeled plays a crucial role in that it governs the rate at which reliable communication is possible. There are two main schools of thought in modeling the channel behavior:

- Probabilistic noise model studied by Shannon in which the channel noise is modeled precisely in the sense that the probabilistic behavior of the channel is well known and the probability of occurrence of too many or too few errors is low
- Worst-case or adversarial noise model considered by Hamming in which the channel acts as an adversary

that arbitrarily corrupts the codeword subject to a bound on the total number of errors.

The highlight of list-decoding is that even under adversarial noise conditions, it is possible to achieve the information-theoretic optimal trade-off between rate and fraction of errors that can be corrected. Hence, in a sense this is like improving the error-correction performance to that possible in case of a weaker, stochastic noise model.

### 68.1 Mathematical formulation

Let  $\mathcal{C}$  be a  $(n, k, d)_q$  error-correcting code; in other words,  $\mathcal{C}$  is a code of length  $n$ , dimension  $k$  and minimum distance  $d$  over an alphabet  $\Sigma$  of size  $q$ . The list-decoding problem can now be formulated as follows:

**Input:** Received word  $x \in \Sigma^n$ , error bound  $e$

**Output:** A list of all codewords  $x_1, x_2, \dots, x_m \in \mathcal{C}$  whose hamming distance from  $x$  is at most  $e$ .

### 68.2 Motivation for list decoding

Given a received word  $y$ , which is a noisy version of some transmitted codeword  $c$ , the decoder tries to output the transmitted codeword by placing its bet on a codeword that is “nearest” to the received word. The Hamming distance between two codewords is used as a metric in finding the nearest codeword, given the received word by the decoder. If  $d$  is the minimum Hamming distance of a code  $\mathcal{C}$ , then there exists two codewords  $c_1$  and  $c_2$  that differ in exactly  $d$  positions. Now, in the case where the received word  $y$  is equidistant from the codewords  $c_1$  and  $c_2$ , unambiguous decoding becomes impossible as the decoder cannot decide which one of  $c_1$  and  $c_2$  to output as the original transmitted codeword. As a result, the half-the minimum distance acts as a combinatorial barrier beyond which unambiguous error-correction is impossible, if we only insist on unique decoding. However, received words such as  $y$  considered above occur only in the worst-case and if one looks at the way Hamming balls are packed in high-dimensional space, even for error pat-

terns  $e$  beyond half-the minimum distance, there is only a single codeword  $c$  within Hamming distance  $e$  from the received word. This claim has been shown to hold with high probability for a random code picked from a natural ensemble and more so for the case of **Reed–Solomon codes** which is well studied and quite ubiquitous in the real world applications. In fact, Shannon’s proof of the capacity theorem for  $q$ -ary symmetric channels can be viewed in light of the above claim for random codes.

Under the mandate of list-decoding, for worst-case errors, the decoder is allowed to output a small list of codewords. With some context specific or side information, it may be possible to prune the list and recover the original transmitted codeword. Hence, in general, this seems to be a stronger error-recovery model than unique decoding.

### 68.3 List-decoding potential

For a polynomial-time list-decoding algorithm to exist, we need the combinatorial guarantee that any Hamming ball of radius  $pn$  around a received word  $r$  (where  $p$  is the fraction of errors in terms of the block length  $n$ ) has a small number of codewords. This is because the list size itself is clearly a lower bound on the running time of the algorithm. Hence, we require the list size to be a polynomial in the block length  $n$  of the code. A combinatorial consequence of this requirement is that it imposes an upper bound on the rate of a code. List decoding promises to meet this upper bound. It has been shown non-constructively that codes of rate  $R$  exist that can be list decoded up to a fraction of errors approaching  $1 - R$ . The quantity  $1 - R$  is referred to in the literature as the list-decoding capacity. This is a substantial gain compared to the unique decoding model as we now have the potential to correct twice as many errors. Naturally, we need to have at least a fraction  $R$  of the transmitted symbols to be correct in order to recover the message. This is an information-theoretic lower bound on the number of correct symbols required to perform decoding and with list decoding, we can potentially achieve this information-theoretic limit. However, to realize this potential, we need explicit codes (codes that can be constructed in polynomial time) and efficient algorithms to perform encoding and decoding.

### 68.4 $(p, L)$ -list-decodability

For any error fraction  $0 \leq p \leq 1$  and an integer  $L \geq 1$ , a code  $\mathcal{C} \subseteq \Sigma^n$  is said to be list decodable up to a fraction  $p$  of errors with list size at most  $L$  or  $(p, L)$ -list-decodable if for every  $y \in \Sigma^n$ , the number of codewords  $c \in \mathcal{C}$  within Hamming distance  $pn$  from  $y$  is at most  $L$ .

## 68.5 Combinatorics of list decoding

The relation between list decodability of a code and other fundamental parameters such as minimum distance and rate have been fairly well studied. It has been shown that every code can be list decoded using small lists beyond half the minimum distance up to a bound called the Johnson radius. This is quite significant because it proves the existence of  $(p, L)$ -list-decodable codes of good rate with a list-decoding radius much larger than  $\frac{d}{2}$ . In other words, the **Johnson bound** rules out the possibility of having a large number of codewords in a Hamming ball of radius slightly greater than  $\frac{d}{2}$  which means that it is possible to correct far more errors with list decoding.

### 68.6 List-decoding capacity

**Theorem (List-Decoding Capacity).** Let  $q \geq 2$ ,  $0 \leq p \leq 1 - \frac{1}{q}$  and  $\epsilon \geq 0$ . The following two statements hold for large enough block length  $n$ .

- i) If  $R \leq 1 - H_q(p) - \epsilon$ , then there exists a  $(p, O(1/\epsilon))$ -list decodable code.
- ii) If  $R \geq 1 - H_q(p) + \epsilon$ , then every  $(p, L)$ -list-decodable code has  $L = q^{\Omega(n)}$ .

Where

$$H_q(p) = p \log_q(q-1) - p \log_q p - (1-p) \log_q(1-p)$$

is the  $q$ -ary entropy function defined for  $p \in (0, 1)$  and extended by continuity to  $[0, 1]$ .

What this means is that for rates approaching the channel capacity, there exists list decodable codes with polynomial sized lists enabling efficient decoding algorithms whereas for rates exceeding the channel capacity, the list size becomes exponential which rules out the existence of efficient decoding algorithms.

The proof for list-decoding capacity is a significant one in that it exactly matches the capacity of a  $q$ -ary symmetric channel  $qSC_p$ . In fact, the term “list-decoding capacity” should actually be read as the capacity of an adversarial channel under list decoding. Also, the proof for list-decoding capacity is an important result that pin points the optimal trade-off between rate of a code and the fraction of errors that can be corrected under list decoding.

#### 68.6.1 Sketch of proof

The idea behind the proof is similar to that of Shannon’s proof for capacity of the binary symmetric channel





give explicit codes that achieve list-decoding capacity, that is, they can be list decoded up to the radius  $1 - R - \epsilon$  for any  $\epsilon > 0$ . In other words, this is error-correction with optimal redundancy. This answered a question that had been open for about 50 years. This work has been invited to the Research Highlights section of the Communications of the ACM (which is “devoted to the most important research results published in Computer Science in recent years”) and was mentioned in an article titled “Coding and Computing Join Forces” in the Sep 21, 2007 issue of the Science magazine. The codes that they are given are called **folded Reed-Solomon codes** which are nothing but plain Reed-Solomon codes but viewed as a code over a larger alphabet by careful bundling of codeword symbols.

Because of their ubiquity and the nice algebraic properties they possess, list-decoding algorithms for Reed-Solomon codes were a main focus of researchers. The list-decoding problem for Reed-Solomon codes can be formulated as follows:

**Input:** For an  $[n, k + 1]_q$  Reed-Solomon code, we are given the pair  $(\alpha_i, y_i)$  for  $1 \leq i \leq n$ , where  $y_i$  is the  $i$ th bit of the received word and the  $\alpha_i$ ’s are distinct points in the finite field  $F_q$  and an error parameter  $e = n - t$ .

**Output:** The goal is to find all the polynomials  $P(X) \in F_q[X]$  of degree at most  $k$  which is the message length such that  $p(\alpha_i) = y_i$  for at least  $t$  values of  $i$ . Here, we would like to have  $t$  as small as possible so that greater number of errors can be tolerated.

With the above formulation, the general structure of list-decoding algorithms for Reed-Solomon codes is as follows:

**Step 1:** (Interpolation) Find a non-zero bivariate polynomial  $Q(X, Y)$  such that  $Q(\alpha_i, y_i) = 0$  for  $1 \leq i \leq n$ .

**Step 2:** (Root finding/Factorization) Output all degree  $k$  polynomials  $p(X)$  such that  $Y - p(X)$  is a factor of  $Q(X, Y)$  i.e.  $Q(X, p(X)) = 0$ . For each of these polynomials, check if  $p(\alpha_i) = y_i$  for at least  $t$  values of  $i \in [n]$ . If so, include such a polynomial  $p(X)$  in the output list.

Given the fact that bivariate polynomials can be factored efficiently, the above algorithm runs in polynomial time.

- Construction of **hard-core predicates** from **one-way permutations**.
- Predicting witnesses for NP-search problems.
- Amplifying hardness of Boolean functions.
- Average case hardness of **permanent** of random matrices.
- **Extractors** and **Pseudorandom generators**.
- Efficient traitor tracing.

## 68.9 External links

- A Survey on list decoding by Madhu Sudan
- Notes from a course taught by Madhu Sudan
- Notes from a course taught by Luca Trevisan
- Notes from a course taught by Venkatesan Guruswami
- Notes from a course taught by Atri Rudra
- P. Elias, “List decoding for noisy channels,” Technical Report 335, Research Laboratory of Electronics, MIT, 1957.
- P. Elias, “Error-correcting codes for list decoding,” IEEE Transactions on Information Theory, vol. 37, pp. 5–12, 1991.
- J. M. Wozencraft, “List decoding,” Quarterly Progress Report, Research Laboratory of Electronics, MIT, vol. 48, pp. 90–95, 1958.
- Venkatesan Guruswami’s PhD thesis
- Algorithmic Results in List Decoding
- Folded Reed-Solomon code

## 68.8 Applications in complexity theory and cryptography

Algorithms developed for list decoding of several interesting code families have found interesting applications in **computational complexity** and the field of **cryptography**. Following is a sample list of applications outside of coding theory:

# Chapter 69

## Long code (mathematics)

In theoretical computer science and coding theory, the **long code** is an error-correcting code that is locally decodable. Long codes have an extremely poor rate, but play a fundamental role in the theory of hardness of approximation.

Notes)

### 69.1 Definition

Let  $f_1, \dots, f_{2^k} : \{0, 1\}^k \rightarrow \{0, 1\}$  for  $k = \log n$  be the list of *all* functions from  $\{0, 1\}^k \rightarrow \{0, 1\}$ . Then the long code encoding of a message  $x \in \{0, 1\}^k$  is the string  $f_1(x) \circ f_2(x) \circ \dots \circ f_{2^k}(x)$  where  $\circ$  denotes concatenation of strings. This string has length  $2^k = 2^{\log n}$ .

The **Walsh-Hadamard code** is a subcode of the long code, and can be obtained by only using functions  $f_i$  that are **linear functions** when interpreted as functions  $\mathbb{F}_2^k \rightarrow \mathbb{F}_2$  on the **finite field** with two elements. Since there are only  $2^k$  such functions, the block length of the Walsh-Hadamard code is  $2^k$ .

An equivalent definition of the long code is as follows: The Long code encoding of  $j \in [n]$  is defined to be the truth table of the Boolean dictatorship function on the  $j$ th coordinate, i.e., the truth table of  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  with  $f(x_1, \dots, x_n) = x_i$ .<sup>[1]</sup> Thus, the Long code encodes a  $(\log n)$ -bit string as a  $2^n$ -bit string.

### 69.2 Properties

The long code does not contain repetitions, in the sense that the function  $f_i$  computing the  $i$ th bit of the output is different from any function  $f_j$  computing the  $j$ th bit of the output for  $j \neq i$ . Among all codes that do not contain repetitions, the long code has the longest possible output. Moreover, it contains all non-repeating codes as a subcode.

### 69.3 References

- [1] Definition 7.3.1 in Limits of Approximation Algorithms: PCPs and Unique Games (DIMACS Tutorial Lecture



## Chapter 70

# Low-density parity-check code

In information theory, a **low-density parity-check (LDPC) code** is a linear error correcting code, a method of transmitting a message over a noisy transmission channel.<sup>[1][2]</sup> An LDPC is constructed using a sparse bipartite graph.<sup>[3]</sup> LDPC codes are capacity-approaching codes, which means that practical constructions exist that allow the noise threshold to be set very close (or even *arbitrarily* close on the **BEC**) to the theoretical maximum (the **Shannon limit**) for a symmetric memoryless channel. The noise threshold defines an upper bound for the channel noise, up to which the probability of lost information can be made as small as desired. Using iterative **belief propagation** techniques, LDPC codes can be decoded in time linear to their block length.

LDPC codes are finding increasing use in applications requiring reliable and highly efficient information transfer over bandwidth or return channel-constrained links in the presence of corrupting noise. Implementation of LDPC codes has lagged behind that of other codes, notably **turbo codes**. The fundamental patent for Turbo Codes expired on August 29, 2013.<sup>[US5446747]<sup>[4]</sup></sup>

LDPC codes are also known as **Gallager codes**, in honor of **Robert G. Gallager**, who developed the LDPC concept in his doctoral dissertation at the **Massachusetts Institute of Technology** in 1960.<sup>[5]</sup>

## 70.1 History

Impractical to implement when first developed by Gallager in 1963,<sup>[6]</sup> LDPC codes were forgotten until his work was rediscovered in 1996.<sup>[7]</sup> **Turbo codes**, another class of capacity-approaching codes discovered in 1993, became the coding scheme of choice in the late 1990s, used for applications such as the **Deep Space Network** and **satellite communications**. However, the advances in low-density parity-check codes have seen them surpass turbo codes in terms of **error floor** and performance in the higher **code rate** range, leaving turbo codes better suited for the lower code rates only.<sup>[8]</sup>

## 70.2 Applications

In 2003, an irregular repeat accumulate (IRA) style LDPC code beat six turbo codes to become the error correcting code in the new **DVB-S2** standard for the satellite transmission of **digital television**.<sup>[9]</sup> The DVB-S2 selection committee made decoder complexity estimates for the Turbo Code proposals using a much less efficient serial decoder architecture rather than a parallel decoder architecture.

This forced the Turbo Code proposals to use frame sizes on the order of one half the frame size of the LDPC proposals. In 2008, LDPC beat convolutional turbo codes as the **forward error correction (FEC)** system for the **ITU-T G.hn** standard.<sup>[10]</sup> G.hn chose LDPC codes over turbo codes because of their lower decoding complexity (especially when operating at data rates close to 1.0 Gbit/s) and because the proposed turbo codes exhibited a significant **error floor** at the desired range of operation.<sup>[11]</sup>

LDPC codes are also used for **10GBase-T Ethernet**, which sends data at 10 gigabits per second over twisted-pair cables. As of 2009, LDPC codes are also part of the **Wi-Fi 802.11** standard as an optional part of **802.11n** and **802.11ac**, in the **High Throughput (HT) PHY** specification.<sup>[12]</sup>

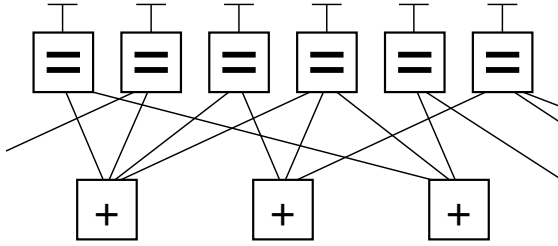
Some **OFDM** systems add an additional outer error correction that fixes the occasional errors (the “error floor”) that get past the LDPC correction inner code even at low **bit error rates**. For example: The **Reed-Solomon code** with LDPC Coded Modulation (RS-LCM) uses a Reed-Solomon outer code.<sup>[13]</sup> The **DVB-S2**, the **DVB-T2** and the **DVB-C2** standards all use a **BCH code** outer code to mop up residual errors after LDPC decoding.<sup>[14]</sup>

## 70.3 Operational use

LDPC codes functionally are defined by a sparse **parity-check matrix**. This sparse matrix is often randomly generated, subject to the **sparsity** constraints—LDPC code construction is discussed later. These codes were first designed by Robert Gallager in 1962.

Below is a graph fragment of an example LDPC code using **Forney's factor graph notation**. In this graph,  $n$  variable nodes in the top of the graph are connected to  $(n-k)$  constraint nodes in the bottom of the graph.

This is a popular way of graphically representing an  $(n, k)$  LDPC code. The bits of a valid message, when placed on the **T's** at the top of the graph, satisfy the graphical constraints. Specifically, all lines connecting to a variable node (box with an '=' sign) have the same value, and all values connecting to a factor node (box with a '+' sign) must sum, **modulo two**, to zero (in other words, they must sum to an even number; or there must be an even number of odd values).



Ignoring any lines going out of the picture, there are eight possible six-bit strings corresponding to valid codewords: (i.e., 000000, 011001, 110010, 101011, 111100, 100101, 001110, 010111). This LDPC code fragment represents a three-bit message encoded as six bits. Redundancy is used, here, to increase the chance of recovering from channel errors. This is a  $(6, 3)$  **linear code**, with  $n = 6$  and  $k = 3$ .

Once again ignoring lines going out of the picture, the parity-check matrix representing this graph fragment is

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

In this matrix, each row represents one of the three parity-check constraints, while each column represents one of the six bits in the received codeword.

In this example, the eight codewords can be obtained by putting the **parity-check matrix**  $\mathbf{H}$  into this form  $[-P^T | I_{n-k}]$  through basic **row operations in GF(2)**:

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}_1 \sim \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}_2 \xrightarrow{\text{Fig. 70.4}} \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}_3 \sim \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}_4.$$

Step 1: H.

Step 2: Row 1 is added to row 3.

Step 3: Row 2 and 3 are swapped.

Step 4: Row 1 is added to row 3.

From this, the **generator matrix**  $\mathbf{G}$  can be obtained as  $[I_k | P]$  (noting that in the special case of this being a binary code  $P = -P$ ), or specifically:

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

Finally, by multiplying all eight possible 3-bit strings by  $\mathbf{G}$ , all eight valid codewords are obtained. For example, the codeword for the bit-string '101' is obtained by:

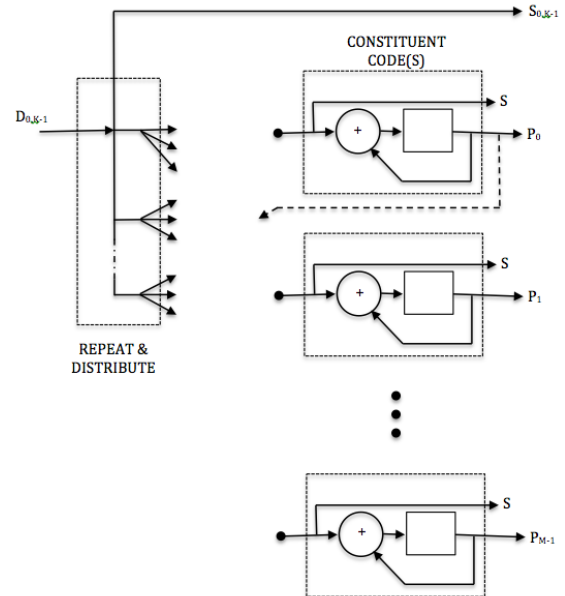
$$\begin{pmatrix} 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

As a check, the row space of  $\mathbf{G}$  is orthogonal to  $\mathbf{H}$  such that  $\mathbf{GH}^T = 0$

The bit-string '101' is found in as the first 3 bits of the codeword '101011'.

## 70.4 Example Encoder

Figure 1 illustrates the functional components of most LDPC encoders.



During the encoding of a frame, the input data bits ( $D$ ) are repeated and distributed to a set of constituent encoders. The constituent encoders are typically accumulators and each accumulator is used to generate a parity symbol. A single copy of the original data ( $S_{0,K-1}$ ) is transmitted with the parity bits ( $P$ ) to make up the code

symbols. The  $S$  bits from each constituent encoder are discarded.

The parity bit may be used within another constituent code.

In an example using the DVB-S2 rate  $2/3$  code the encoded block size is 64800 symbols ( $N=64800$ ) with 43200 data bits ( $K=43200$ ) and 21600 parity bits ( $M=21600$ ). Each constituent code (check node) encodes 16 data bits except for the first parity bit which encodes 8 data bits. The first 4680 data bits are repeated 13 times (used in 13 parity codes), while the remaining data bits are used in 3 parity codes (irregular LDPC code).

For comparison, classic turbo codes typically use two constituent codes configured in parallel, each of which encodes the entire input block ( $K$ ) of data bits. These constituent encoders are recursive convolutional codes (RSC) of moderate depth (8 or 16 states) that are separated by a code interleaver which interleaves one copy of the frame.

The LDPC code, in contrast, uses many low depth constituent codes (accumulators) in parallel, each of which encode only a small portion of the input frame. The many constituent codes can be viewed as many low depth (2 state) 'convolutional codes' that are connected via the repeat and distribute operations. The repeat and distribute operations perform the function of the interleaver in the turbo code.

The ability to more precisely manage the connections of the various constituent codes and the level of redundancy for each input bit give more flexibility in the design of LDPC codes, which can lead to better performance than turbo codes in some instances. Turbo codes still seem to perform better than LDPCs at low code rates, or at least the design of well performing low rate codes is easier for Turbo Codes.

As a practical matter, the hardware that forms the accumulators is reused during the encoding process. That is, once a first set of parity bits are generated and the parity bits stored, the same accumulator hardware is used to generate a next set of parity bits.

## 70.5 Decoding

As with other codes, the **maximum likelihood decoding** of an LDPC code on the **binary symmetric channel** is an **NP-complete** problem. Performing optimal decoding for a NP-complete code of any useful size is not practical.

However, sub-optimal techniques based on iterative **belief propagation** decoding give excellent results and can be practically implemented. The sub-optimal decoding techniques view each parity check that makes up the LDPC as an independent single parity check (SPC) code. Each SPC code is decoded separately using **soft-in-soft-out** (SISO) techniques such as **SOVA**, **BCJR**, **MAP**, and other derivatives thereof. The soft decision information

from each SISO decoding is cross-checked and updated with other redundant SPC decodings of the same information bit. Each SPC code is then decoded again using the updated soft decision information. This process is iterated until a valid code word is achieved or decoding is exhausted. This type of decoding is often referred to as **sum-product decoding**.

The decoding of the SPC codes is often referred to as the "check node" processing, and the cross-checking of the variables is often referred to as the "variable-node" processing.

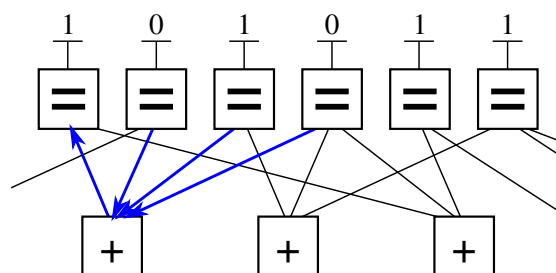
In a practical LDPC decoder implementation, sets of SPC codes are decoded in parallel to increase throughput.

In contrast, belief propagation on the **binary erasure channel** is particularly simple where it consists of iterative constraint satisfaction.

For example, consider that the valid codeword, 101011, from the example above, is transmitted across a binary erasure channel and received with the first and fourth bit erased to yield ?01?11. Since the transmitted message must have satisfied the code constraints, the message can be represented by writing the received message on the top of the factor graph.

In this example, the first bit cannot yet be recovered, because all of the constraints connected to it have more than one unknown bit. In order to proceed with decoding the message, constraints connecting to only one of the erased bits must be identified. In this example, only the second constraint suffices. Examining the second constraint, the fourth bit must have been zero, since only a zero in that position would satisfy the constraint.

This procedure is then iterated. The new value for the fourth bit can now be used in conjunction with the first constraint to recover the first bit as seen below. This means that the first bit must be a one to satisfy the leftmost constraint.



Thus, the message can be decoded iteratively. For other channel models, the messages passed between the variable nodes and check nodes are **real numbers**, which express probabilities and likelihoods of belief.

This result can be validated by multiplying the corrected codeword  $\mathbf{r}$  by the parity-check matrix  $\mathbf{H}$ :

$$\mathbf{z} = \mathbf{H}\mathbf{r} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

Because the outcome  $\mathbf{z}$  (the **syndrome**) of this operation is the three  $\times$  one zero vector, the resulting codeword  $\mathbf{r}$  is successfully validated.

After the decoding is completed, the original message bits '101' can be extracted by looking at the first 3 bits of the codeword.

While illustrative, this erasure example does not show the use of soft-decision decoding or soft-decision message passing, which is used in virtually all commercial LDPC decoders.

### 70.5.1 Updating node information

In recent years, there has also been a great deal of work spent studying the effects of alternative schedules for variable-node and constraint-node update. The original technique that was used for decoding LDPC codes was known as *flooding*. This type of update required that, before updating a variable node, all constraint nodes needed to be updated and vice versa. In later work by Vila Casado *et al.*,<sup>[15]</sup> alternative update techniques were studied, in which variable nodes are updated with the newest available check-node information.

The intuition behind these algorithms is that variable nodes whose values vary the most are the ones that need to be updated first. Highly reliable nodes, whose **log-likelihood ratio** (LLR) magnitude is large and does not change significantly from one update to the next, do not require updates with the same frequency as other nodes, whose sign and magnitude fluctuate more widely. These scheduling algorithms show greater speed of convergence and lower error floors than those that use flooding. These lower error floors are achieved by the ability of the Informed Dynamic Scheduling (IDS)<sup>[15]</sup> algorithm to overcome trapping sets of near codewords.<sup>[16]</sup>

When nonflooding scheduling algorithms are used, an alternative definition of iteration is used. For an  $(n, k)$  LDPC code of rate  $k/n$ , a full *iteration* occurs when  $n$  variable and  $n - k$  constraint nodes have been updated, no matter the order in which they were updated.

### 70.5.2 Lookup table decoding

It is possible to decode LDPC codes on a relatively low-power microprocessor by the use of **lookup tables**.

While codes such as the LDPC are generally implemented on high-power processors, with long block

lengths, there are also applications which use lower-power processors and short block lengths (1024).

Therefore, it is possible to precalculate the output bit based upon predetermined input bits. A table is generated which contains  $n$  entries (for a block length of 1024 bits, this would be 1024 bits long), and contains all possible entries for different input states (both errored and nonerrored).

As a bit is input, it is then added to a FIFO register, and the value of the FIFO register is then used to look up in the table the relevant output from the precalculated values.

By this method, very high iterations can be used, with little processor overhead, the only cost being that of the memory for the lookup table, such that LDPC decoding is possible even on a 4.0 MHz PIC chip.

## 70.6 Code construction

For large block sizes, LDPC codes are commonly constructed by first studying the behaviour of decoders. As the block size tends to infinity, LDPC decoders can be shown to have a noise threshold below which decoding is reliably achieved, and above which decoding is not achieved,<sup>[17]</sup> colloquially referred to as the **cliff effect**. This threshold can be optimised by finding the best proportion of arcs from check nodes and arcs from variable nodes. An approximate graphical approach to visualising this threshold is an **EXIT chart**.

The construction of a specific LDPC code after this optimization falls into two main types of techniques:

- Pseudorandom approaches
- Combinatorial approaches

Construction by a pseudo-random approach builds on theoretical results that, for large block size, a random construction gives good decoding performance.<sup>[7]</sup> In general, pseudorandom codes have complex encoders, but pseudorandom codes with the best decoders can have simple encoders.<sup>[18]</sup> Various constraints are often applied to help ensure that the desired properties expected at the theoretical limit of infinite block size occur at a finite block size.

Combinatorial approaches can be used to optimize the properties of small block-size LDPC codes or to create codes with simple encoders.

Some LDPC codes are based on **Reed–Solomon codes**, such as the RS-LDPC code used in the 10 Gigabit Ethernet standard.<sup>[19]</sup> Compared to randomly generated LDPC codes, structured LDPC codes—such as the LDPC code used in the **DVB-S2** standard—can have simpler and therefore lower-cost hardware—in particular, codes constructed such that the  $\mathbf{H}$  matrix is a **circulant matrix**.<sup>[20]</sup>

Yet another way of constructing LDPC codes is to use finite geometries. This method was proposed by Y. Kou *et al.* in 2001.<sup>[21]</sup>

## 70.7 See also

### 70.7.1 People

- Robert G. Gallager
- Richard Hamming
- Claude Shannon
- David J. C. MacKay
- Irving S. Reed

### 70.7.2 Theory

- Belief propagation
- Graph theory
- Hamming code
- Linear code
- Sparse graph code
- Expander code

### 70.7.3 Applications

- G.hn/G.9960 (ITU-T Standard for networking over power lines, phone lines and coaxial cable)
- 802.3an (10 Giga-bit/s Ethernet over Twisted pair)
- CMMB(China Multimedia Mobile Broadcasting)
- DVB-S2 / DVB-T2 / DVB-C2 (Digital video broadcasting, 2nd Generation)
- DMB-T/H (Digital video broadcasting)<sup>[22]</sup>
- WiMAX (IEEE 802.16e standard for microwave communications)
- IEEE 802.11n-2009 (Wi-Fi standard)
- DOCSIS 3.1

## 70.7.4 Other capacity-approaching codes

- Turbo codes
- Serial concatenated convolutional codes
- Online codes
- Fountain codes
- LT codes
- Raptor codes
- Repeat-accumulate codes (a class of simple turbo codes)
- Tornado codes (LDPC codes designed for erasure decoding)
- Polar codes

## 70.8 References

- [1] David J.C. MacKay (2003) Information theory, Inference and Learning Algorithms, CUP, ISBN 0-521-64298-1, (also available online)
- [2] Todd K. Moon (2005) Error Correction Coding, Mathematical Methods and Algorithms. Wiley, ISBN 0-471-64800-0 (Includes code)
- [3] Amin Shokrollahi (2003) LDPC Codes: An Introduction
- [4] NewScientist, *Communication speed nears terminal velocity*, by Dana Mackenzie, 9 July 2005
- [5] Larry Hardesty (January 21, 2010). "Explained: Gallager codes". *MIT News*. Retrieved August 7, 2013.
- [6] Robert G. Gallager (1963). *Low Density Parity Check Codes* (PDF). Monograph, M.I.T. Press. Retrieved August 7, 2013.
- [7] David J.C. MacKay and Radford M. Neal, "Near Shannon Limit Performance of Low Density Parity Check Codes," *Electronics Letters*, July 1996
- [8] *Telemetry Data Decoding, Design Handbook*
- [9] Presentation by Hughes Systems
- [10] HomePNA Blog: G.hn, a PHY For All Seasons
- [11] IEEE Communications Magazine paper on G.hn
- [12] IEEE Standard, section 20.3.11.6 "802.11n-2009", IEEE, October 29, 2009, accessed March 21, 2011.
- [13] Chih-Yuan Yang, Mong-Kai Ku. <http://123seminaronly.com/Seminar-Reports/029/26540350-Ldpc-Coded-Ofdm-Modulation.pdf> "LDPC coded OFDM modulation for high spectral efficiency transmission"
- [14] Nick Wells. "DVB-T2 in relation to the DVB-x2 Family of Standards"



- [15] A.I. Vila Casado, M. Griot, and R. Wesel, “Informed dynamic scheduling for belief propagation decoding of LDPC codes,” *Proc. IEEE Int. Conf. on Comm. (ICC)*, June 2007.
- [16] T. Richardson, “Error floors of LDPC codes,” in *Proc. 41st Allerton Conf. Comm., Control, and Comput.*, Monticello, IL, 2003.
- [17] Thomas J. Richardson and M. Amin Shokrollahi and Rüdiger L. Urbanke, “Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes,” *IEEE Transactions in Information Theory*, 47(2), February 2001
- [18] Thomas J. Richardson and Rüdiger L. Urbanke, “Efficient Encoding of Low-Density Parity-Check Codes,” *IEEE Transactions in Information Theory*, 47(2), February 2001
- [19] Ahmad Darabiha, Anthony Chan Carusone, Frank R. Kschischang. “Power Reduction Techniques for LDPC Decoders”
- [20] Zhengya Zhang, Venkat Anantharam, Martin J. Wainwright, and Borivoje Nikolic. “An Efficient 10GBASE-T Ethernet LDPC Decoder Design With Low Error Floors”.
- [21] Y. Kou, S. Lin and M. Fossorier, “Low-Density Parity-Check Codes Based on Finite Geometries: A Rediscovery and New Results,” *IEEE Transactions on Information Theory*, vol. 47, no. 7, November 2001, pp. 2711- 2736.
- [22] <http://spectrum.ieee.org/consumer-electronics/standards/does-china-have-the-best-digital-television-standard-on-the-planet/>  
2

## 70.9 External links

- The on-line textbook: *Information Theory, Inference, and Learning Algorithms*, by David J.C. MacKay, discusses LDPC codes in Chapter 47.
- LDPC Codes: An Introduction
- LDPC codes and performance results
- Online density evolution for LDPC codes
- LDPC Codes – a brief Tutorial (by Bernhard Leiner, 2005)
- Iterative Error Correction: Turbo, Low-Density Parity-Check and Repeat-Accumulate Codes
- Source code for encoding, decoding, and simulating LDPC codes is available from a variety of locations:
  - Binary LDPC codes in C
  - Binary LDPC codes for Python (core algorithm in C)
  - LDPC codes over GF(q) in MATLAB
  - LDPC encoder and LDPC decoder in MATLAB

# Chapter 71

## Luby transform code

In computer science, **Luby transform codes (LT codes)** are the first class of practical fountain codes that are near-optimal erasure correcting codes. They were invented by Michael Luby in 1998 and published in 2002.<sup>[1]</sup> Like some other fountain codes, LT codes depend on sparse bipartite graphs to trade reception overhead for encoding and decoding speed. The distinguishing characteristic of LT codes is in employing a particularly simple algorithm based on the exclusive or operation ( $\oplus$ ) to encode and decode the message.<sup>[2]</sup>

LT codes are *rateless* because the encoding algorithm can in principle produce an infinite number of message packets (i.e., the percentage of packets that must be received to decode the message can be arbitrarily small). They are *erasure correcting codes* because they can be used to transmit digital data reliably on an erasure channel.

The next generation beyond LT codes are **raptor codes** (see for example IETF RFC 5053 or IETF RFC 6330), which have linear time encoding and decoding. Raptor codes use two encoding stages for encoding, where the second stage is an LT encoding.

### 71.1 Why use an LT code?

The traditional scheme for transferring data across an erasure channel depends on continuous two-way communication.

- The sender encodes and sends a packet of information.
- The receiver attempts to decode the received packet. If it can be decoded, the receiver sends an acknowledgment back to the transmitter. Otherwise, the receiver asks the transmitter to send the packet again.
- This two-way process continues until all the packets in the message have been transferred successfully.

Certain networks, such as ones used for cellular wireless broadcasting, do not have a feedback channel. Applications on these networks still require reliability. Fountain codes in general, and LT codes in particular, get around

this problem by adopting an essentially one-way communication protocol.

- The sender encodes and sends packet after packet of information.
- The receiver evaluates each packet as it is received. If there is an error, the erroneous packet is discarded. Otherwise the packet is saved as a piece of the message.
- Eventually the receiver has enough valid packets to reconstruct the entire message. When the entire message has been received successfully the receiver signals that transmission is complete.

### 71.2 LT encoding

The encoding process begins by dividing the uncoded message into  $n$  blocks of roughly equal length. Encoded packets are then produced with the help of a pseudorandom number generator.

- The degree  $d$ ,  $1 \leq d \leq n$ , of the next packet is chosen at random.
- Exactly  $d$  blocks from the message are randomly chosen.
- If  $M_i$  is the  $i$ th block of the message, the data portion of the next packet is computed as

$$M_{i_1} \oplus M_{i_2} \oplus \dots \oplus M_{i_d}$$

where  $\{i_1, i_2, \dots, i_d\}$  are the randomly chosen indices for the  $d$  blocks included in this packet.

- A prefix is appended to the encoded packet, defining how many blocks  $n$  are in the message, how many blocks  $d$  have been exclusive-ored into the data portion of this packet, and the list of indices  $\{i_1, i_2, \dots, i_d\}$ .



- Finally, some form of error-detecting code (perhaps as simple as a **cyclic redundancy check**) is applied to the packet, and the packet is transmitted.

This process continues until the receiver signals that the message has been received and successfully decoded.

### 71.3 LT decoding

The decoding process uses the "exclusive or" operation to retrieve the encoded message.

- If the current packet isn't clean, or if it replicates a packet that has already been processed, the current packet is discarded.
- If the current cleanly received packet is of degree  $d > 1$ , it is first processed against all the fully decoded blocks in the message queuing area (as described more fully in the next step), then stored in a buffer area if its reduced degree is greater than 1.
- When a new, clean packet of degree  $d = 1$  (block  $M_i$ ) is received (or the degree of the current packet is reduced to 1 by the preceding step), it is moved to the message queuing area, and then matched against all the packets of degree  $d > 1$  residing in the buffer. It is exclusive-ored into the data portion of any buffered packet that was encoded using  $M_i$ , the degree of that matching packet is decremented, and the list of indices for that packet is adjusted to reflect the application of  $M_i$ .
- When this process unlocks a block of degree  $d = 2$  in the buffer, that block is reduced to degree 1 and is in its turn moved to the message queuing area, and then processed against the packets remaining in the buffer.
- When all  $n$  blocks of the message have been moved to the message queuing area, the receiver signals the transmitter that the message has been successfully decoded.

This decoding procedure works because  $A \oplus A = 0$  for any bit string  $A$ . After  $d - 1$  distinct blocks have been exclusive-ored into a packet of degree  $d$ , the original unencoded content of the unmatched block is all that remains. In symbols we have

$$\begin{aligned}
 & (M_{i_1} \oplus \cdots \oplus M_{i_d}) \oplus (M_{i_1} \oplus \cdots \oplus M_{i_{k-1}} \oplus M_{i_{k+1}} \oplus \cdots \oplus M_{i_d}) \\
 &= M_{i_1} \oplus M_{i_1} \oplus \cdots \oplus M_{i_{k-1}} \oplus M_{i_{k-1}} \oplus M_{i_k} \oplus M_{i_{k+1}} \oplus \cdots \oplus M_{i_d} \oplus \cdots \oplus M_{i_d} \\
 &= 0 \oplus \cdots \oplus 0 \oplus M_{i_k} \oplus 0 \oplus \cdots \oplus 0 \\
 &= M_{i_k}
 \end{aligned}$$

### 71.4 Variations

Several variations of the encoding and decoding processes described above are possible. For instance, instead of prefixing each packet with a list of the actual message block indices  $\{i_1, i_2, \dots, i_d\}$ , the encoder might simply send a short "key" which served as the seed for the **pseudorandom number generator** (PRNG) or index table used to construct the list of indices. Since the receiver equipped with the same RNG or index table can reliably recreate the "random" list of indices from this seed, the decoding process can be completed successfully. Alternatively, by combining a simple LT code of low average degree with a robust error-correcting code, a **raptor code** can be constructed that will outperform an optimized LT code in practice.<sup>[3]</sup>

### 71.5 Optimization of LT codes

There is only one parameter that can be used to optimize a straight LT code: the degree distribution function (described as a pseudorandom number generator for the degree  $d$  in the **LT encoding** section above). In practice the other "random" numbers (the list of indices  $\{i_1, i_2, \dots, i_d\}$ ) are invariably taken from a uniform distribution on  $[0, n)$ , where  $n$  is the number of blocks into which the message has been divided.<sup>[4]</sup>

Luby himself<sup>[1]</sup> discussed the "ideal soliton distribution" defined by

$$\begin{aligned}
 P\{d = 1\} &= \frac{1}{n} \\
 P\{d = k\} &= \frac{1}{k(k-1)} \quad (k = 2, 3, \dots, n).
 \end{aligned}$$

This degree distribution theoretically minimizes the expected number of redundant code words that will be sent before the decoding process can be completed. However the ideal soliton distribution does not work well in practice because any fluctuation around the expected behavior makes it likely that at some step in the decoding process there will be no available packet of (reduced) degree 1 so decoding will fail. Furthermore, some of the original blocks will not be xor-ed into any of the transmission packets. Therefore, in practice, a modified distribution, the "robust soliton distribution", is substituted for the ideal distribution. The effect of the modification is, generally, to produce more packets of very small degree (around 1) and fewer packets of degree greater than 1, except for a spike of packets at a fairly large quantity chosen to ensure that all original blocks will be included in some packet.<sup>[4]</sup>

## 71.6 See also

- [Online codes](#)
- [Raptor codes](#)
- [Tornado codes](#)

## 71.7 Notes and references

- [1] M.Luby, “LT Codes”, The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002.
- [2] The *exclusive or* (XOR) operation, symbolized by  $\oplus$ , has the very useful property that  $A \oplus A = 0$ , where  $A$  is an arbitrary string of bits.
- [3] Fountain codes, by D.J.C. MacKay, first published in IEEE Proc.-Commun., Vol. 152, No. 6, December 2005.
- [4] Optimizing the Degree Distribution of LT Codes with an Importance Sampling Approach, by Esa Hyttiä, Tuomas Tirronen, and Jorma Virtamo (2006).

## 71.8 External links

- [“Implementation of Luby transform Code in C#”](#)

## Chapter 72

# Minimum weight

In error-correcting coding, the minimum Hamming weight, commonly referred to as the **minimum weight**  $w_{\min}$  of a code is the weight of the lowest-weight non-zero code word. The weight  $w$  of a code word is the number of 1s in the word. For example the word 11001010 has a weight of 4.

In a linear block code the minimum weight is also the minimum Hamming distance ( $d_{\min}$ ) and defines the error correction capability of the code. If  $w_{\min} = n$ , then  $d_{\min} = n$  and the code will correct up to  $d_{\min}/2$  errors.<sup>[1]</sup>

### 72.1 References

- [1] Stern & Mahmoud, *Communications System Design*, Prentice Hall, 2004, p 477ff.

## Chapter 73

# Multiple description coding

**Multiple description coding (MDC)** is a coding technique that fragments a single media stream into  $n$  substreams ( $n \geq 2$ ) referred to as descriptions. The packets of each description are routed over multiple, (partially) disjoint paths. In order to decode the media stream, any description can be used, however, the quality improves with the number of descriptions received in parallel. The idea of MDC is to provide error resilience to media streams. Since an arbitrary subset of descriptions can be used to decode the original stream, network congestion or packet loss — which are common in best-effort networks such as the Internet — will not interrupt the stream but only cause a (temporary) loss of quality. The quality of a stream can be expected to be roughly proportional to data rate sustained by the receiver.

MDC is a form of data partitioning, thus comparable to layered coding as it is used in MPEG-2 and MPEG-4. Yet, in contrast to MDC, layered coding mechanisms generate a base layer and  $n$  enhancement layers. The base layer is necessary for the media stream to be decoded, enhancement layers are applied to improve stream quality. However, the first enhancement layer depends on the base layer and each enhancement layer  $n + 1$  depends on its subordinate layer  $n$ , thus can only be applied if  $n$  was already applied. Hence, media streams using the layered approach are interrupted whenever the base layer is missing and, as a consequence, the data of the respective enhancement layers is rendered useless. The same applies for missing enhancement layers. In general, this implies that in lossy networks the quality of a media stream is not proportional to the amount of correctly received data.

Besides increased fault tolerance, MDC allows for rate-adaptive streaming: Content providers send all descriptions of a stream without paying attention to the download limitations of clients. Receivers that cannot sustain the data rate only subscribe to a subset of these streams, thus freeing the content provider from sending additional streams at lower data rates.

The vast majority of state-of-the-art codecs uses single description (SD) video coding. This approach does not partition any data at all. Despite the aforementioned advantages of MDC, SD codecs are still predominant. The reasons are probably the comparatively high complexity of codec development, the loss of some compression effi-

ciency as well as the caused transmission overhead.

Though MDC has its practical roots in media communication, it is widely researched in the area of Information Theory.

### 73.1 References

- V. K. Goyal, “Multiple Description Coding: Compression Meets the Network,” *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 74–94, Sept. 2001.
- R. Puri and K. Ramchandran, “Multiple description source coding through forward error correction codes,” *IEEE Proceedings Asilomar Conference on Signals, Systems, and Computers*, Asilomar, CA, October 1999.
- A. Farzamnia, S. K. Syed-Yusof, N. Fisal, and S. A. Abu-Bakar, “Investigation of Error Concealment Using Different Transform Codings and Multiple Description Codings,” *Journal of Electrical Engineering*, vol. 63, pp. 171–179, 2012.
- Ilan Sadeh, “The rate distortion region for coding in stationary systems”, *Journal of Applied Mathematics and Computer Science*, vol. 6, No. 1, 123-136, 1996.

## Chapter 74

# Linear network coding

**Linear network coding** is a technique which can be used to improve a network's throughput, efficiency and scalability, as well as resilience to attacks and eavesdropping. Instead of simply relaying the packets of information they receive, the nodes of a network take several packets and combine them together for transmission. This can be used to attain the maximum possible information flow in a network.

It has been proven that linear coding is enough to achieve the upper bound in multicast problems with one source.<sup>[1]</sup> However linear coding is not sufficient in general (e.g. multisource, multisink with arbitrary demands), even for more general versions of linearity such as convolutional coding and filter-bank coding.<sup>[2]</sup> Finding optimal coding solutions for general network problems with arbitrary demands remains an open problem.

### 74.1 Encoding and decoding

In a linear network coding problem, a group of nodes  $P$  are involved in moving the data from  $S$  source nodes to  $K$  sink nodes. Each node generates new packets which are linear combinations of earlier received packets, multiplying them by coefficients chosen from a finite field, typically of size  $GF(2^s)$ .

Each node,  $p_k$  with indegree,  $InDeg(p_k) = S$ , generates a message  $X_k$  from the linear combination of received messages  $\{M_i\}_{i=1}^S$  by the relation:

$$X_k = \sum_{i=1}^S g_k^i \cdot M_i$$

where the values  $g_k^i$  are the coefficients selected from  $GF(2^s)$ . Note that, since operations are computed in a finite field, the generated message is of the same length as the original messages. Each node forwards the computed value  $X_k$  along with the coefficients,  $g_k^i$ , used in the  $k^{\text{th}}$  level,  $g_k^i$ .

Sink nodes receive these network coded messages, and collect them in a matrix. The original messages can be recovered by performing Gaussian elimination on the

matrix.<sup>[3]</sup> In reduced row echelon form, decoded packets correspond to the rows of the form  $e_i = [0...010...0]$ .

### 74.2 A brief history

A network is represented by a directed graph  $\mathcal{G} = (V, E, C)$ .  $V$  is the set of nodes or vertices,  $E$  is the set of directed links (or edges), and  $C$  gives the capacity of each link of  $E$ . Let  $T(s, t)$  be the maximum possible throughput from node  $s$  to node  $t$ . By the max-flow min-cut theorem,  $T(s, t)$  is upper bounded by the minimum capacity of all cuts, which is the sum of the capacities of the edges on a cut, between these two nodes.

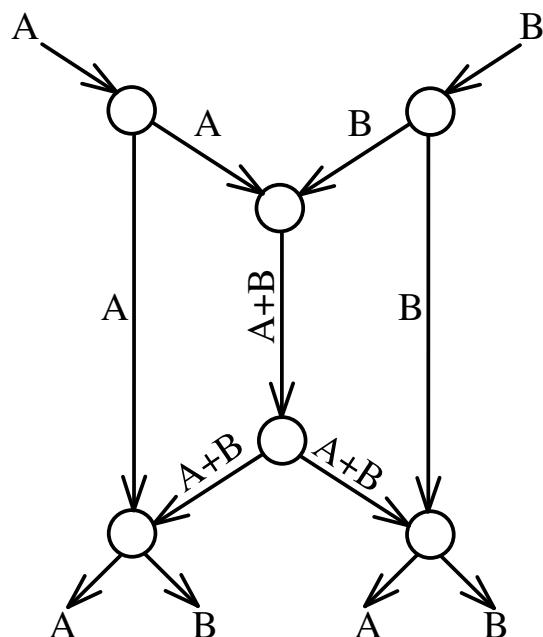
Karl Menger proved that there is always a set of edge-disjoint paths achieving the upper bound in a unicast scenario, known as the max-flow min-cut theorem. Later, the Ford-Fulkerson algorithm was proposed to find such paths in polynomial time. Then, Edmonds proved in the paper "Edge-Disjoint Branchings" the upper bound in the broadcast scenario is also achievable, and proposed a polynomial time algorithm.

However, the situation in the multicast scenario is more complicated, and in fact, such an upper bound can't be reached using traditional routing ideas. Ahlswede, et al. proved that it can be achieved if additional computing tasks (incoming packets are combined into one or several outgoing packets) can be done in the intermediate nodes.<sup>[4]</sup>

### 74.3 The butterfly network example

The butterfly network<sup>[4]</sup> is often used to illustrate how linear network coding can outperform routing. Two source nodes (at the top of the picture) have information A and B that must be transmitted to the two destination nodes (at the bottom), which each want to know both A and B. Each edge can carry only a single value (we can think of an edge transmitting a bit in each time slot).

If only routing were allowed, then the central link would



Butterfly Network.

be only able to carry A or B, but not both. Suppose we send A through the center; then the left destination would receive A twice and not know B at all. Sending B poses a similar problem for the right destination. We say that routing is insufficient because no routing scheme can transmit both A and B simultaneously to both destinations.

Using a simple code, as shown, A and B can be transmitted to both destinations simultaneously by sending the sum of the symbols through the center – in other words, we encode A and B using the formula “ $A+B$ ”. The left destination receives A and  $A+B$ , and can calculate B by subtracting the two values. Similarly, the right destination will receive B and  $A+B$ , and will also be able to determine both A and B.

## 74.4 Random Linear Network Coding

Random linear network coding <sup>[5]</sup> is a simple yet powerful encoding scheme, which in broadcast transmission schemes allows close to optimal throughput using a decentralized algorithm. Nodes transmit random linear combinations of the packets they receive, with coefficients chosen from a Galois field. If the field size is sufficiently large, the probability that the receiver(s) will obtain linearly independent combinations (and therefore obtain innovative information) approaches 1. It should however be noted that, although random linear network coding has excellent throughput performance, if a receiver obtains an insufficient number of packets, it is extremely unlikely that they can recover any of the original pack-

ets. This can be addressed by sending additional random linear combinations until the receiver obtains the appropriate number of packets.

### 74.4.1 Open issues

Based on previous studies, there are three important open issues in RLNC:

1. High decoding computational complexity due to using the Gauss-Jordan elimination method
2. High transmission overhead due to attaching large coefficients vectors to encoded blocks
3. Linear dependency among coefficients vectors which can reduce the number of innovative encoded blocks

## 74.5 Wireless Network Coding

The broadcast nature of wireless (coupled with network topology) determines the nature of **interference**. Simultaneous transmissions in a wireless network typically result in all of the packets being lost (i.e., collision, see **Multiple Access with Collision Avoidance for Wireless**). A wireless network therefore requires a scheduler (as part of the **MAC** functionality) to minimize such interference. Hence any gains from network coding are strongly impacted by the underlying scheduler and will deviate from the gains seen in wired networks. Further, wireless links are typically half-duplex due to hardware constraints; i.e., a node can not simultaneously transmit and receive due to the lack of sufficient isolation between the two paths.

Although, originally network coding was proposed to be used at Network layer (see **OSI model**), in wireless networks, network coding has been widely used in either MAC layer or **PHY** layer.<sup>[6]</sup> It has been shown that in both cases, network coding can increase the end-to-end throughput.<sup>[7]</sup>

## 74.6 Applications

Network coding is seen to be useful in the following areas:

- Alternative to **forward error correction** and **ARQ** in traditional and wireless networks with packet loss. e.g.: **Coded TCP**,<sup>[8]</sup> **Multi-user ARQ**<sup>[9]</sup>
- Robust and resilient to network attacks like snooping, eavesdropping, replay or data corruption attacks.<sup>[10][11]</sup>
- Digital file distribution and P2P file sharing. e.g.: **Avalanche** from Microsoft

- Distributed storage.<sup>[12][13]</sup>
- Throughput increase in wireless mesh networks. e.g. : COPE,<sup>[14]</sup> CORE,<sup>[15]</sup> Coding-aware routing,<sup>[16][17]</sup> B.A.T.M.A.N.<sup>[18]</sup>
- Bidirectional low energy transmission in wireless sensor networks.
- Many-to-many broadcast network capacity augmentations.
- Buffer and Delay reduction in spatial sensor networks: Spatial buffer multiplexing<sup>[19]</sup>
- Reduce the number of packet retransmission for a single-hop wireless multicast transmission, and hence improve network bandwidth.<sup>[20]</sup>
- Distributed file sharing<sup>[21]</sup>

## 74.7 See also

- Secret sharing protocol
- Homomorphic signatures for network coding
- Triangular network coding

## 74.8 References

- [1] S. Li, R. Yeung, and N. Cai, "Linear Network Coding"(PDF), in IEEE Transactions on Information Theory, Vol 49, No. 2, pp. 371–381, 2003
- [2] R. Dougherty, C. Freiling, and K. Zeger, "Insufficiency of Linear Coding in Network Information Flow" (PDF), in IEEE Transactions on Information Theory, Vol. 51, No. 8, pp. 2745–2759, August 2005 ( erratum)
- [3] Chou, Philip A.; Wu, Yunnan; Jain, Kamal (October 2003), "Practical network coding", *Allerton Conference on Communication, Control, and Computing*, Any receiver can then recover the source vectors using Gaussian elimination on the vectors in its  $h$  (or more) received packets.
- [4] Ahlswede, Rudolf; N. Cai; Shuo-Yen Robert Li; Raymond Wai-Ho Yeung (2000). "Network Information Flow". *IEEE Transactions on Information Theory*, IT-46. **46** (4): 1204–1216. doi:10.1109/18.850663.
- [5] T. Ho, R. Koetter, M. Medard, D. R. Karger and M. Effros, "The Benefits of Coding over Routing in a Randomized Setting" in 2003 IEEE International Symposium on Information Theory. doi:10.1109/ISIT.2003.1228459
- [6] M.H.Firooz, Z. Chen, S. Roy and H. Liu, (Wireless Network Coding via Modified 802.11 MAC/PHY: Design and Implementation on SDR) in IEEE Journal on Selected Areas in Communications, 2013.
- [7] XORs in The Air: Practical Wireless Network Coding
- [8] <http://arxiv.org/abs/1212.2291>
- [9] [http://www.ericsson.com/technology/research\\_papers/wireless\\_access/doc/Multi-User%20ARQ.pdf](http://www.ericsson.com/technology/research_papers/wireless_access/doc/Multi-User%20ARQ.pdf)
- [10] <http://securenetworkcoding.wikidot.com/>
- [11] [http://home.eng.iastate.edu/~{ }yuzhen/publications/ZhenYu\\_INFocom\\_2008.pdf](http://home.eng.iastate.edu/~{ }yuzhen/publications/ZhenYu_INFocom_2008.pdf)
- [12] <http://netcod.org/papers/11AcedanskiDMK-final.pdf>
- [13] <http://arxiv.org/pdf/cs/0702015.pdf>
- [14] <http://people.csail.mit.edu/rahul/papers/cope-ton2008.pdf>
- [15] "CORE: COPE with MORE in Wireless Meshed Networks". *2013 IEEE 77th Vehicular Technology Conference (VTC Spring)*. doi:10.1109/VTCSpring.2013.6692495.
- [16] <http://arena.cse.sc.edu/papers/rocx.secon06.pdf>
- [17] <http://www.cs.wisc.edu/~{ }shravan/infocom-07-2.pdf>
- [18] "NetworkCoding - batman-adv - Open Mesh". [www.open-mesh.org](http://www.open-mesh.org). Retrieved 2015-10-28.
- [19] Welcome to IEEE Xplore 2.0: Looking at Large Networks: Coding vs. Queueing
- [20] <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4549741>
- [21] Data dissemination in wireless networks with network coding
- Fragouli, C.; Le Boudec, J. & Widmer, J. "Network coding: An instant primer" in *Computer Communication Review*, 2006.

Ali Farzamnia, Sharifah K. Syed-Yusof, Norsheila Fisa "Multicasting Multiple Description Coding Using p-Cycle Network Coding", KSII Transactions on Internet and Information Systems, Vol 7, No 12, 2013.

## 74.9 External links

- Network Coding Homepage
- A network coding bibliography
- Raymond W. Yeung, Information Theory and Network Coding, Springer 2008, <http://iest2.ie.cuhk.edu.hk/~{ }whyueung/book2/>
- Raymond W. Yeung et al., Network Coding Theory, now Publishers, 2005, <http://iest2.ie.cuhk.edu.hk/~{ }whyueung/netcode/monograph.html>
- Christina Fragouli et al., Network Coding: An Instant Primer, ACM SIGCOMM 2006, <http://infoscience.epfl.ch/getfile.py?mode=best&recid=58339>.



- Avalanche Filesystem, <http://research.microsoft.com/en-us/projects/avalanche/default.aspx>
- Random Network Coding, <http://www.mit.edu/~medard/coding1.htm>
- Digital Fountain Codes, <http://www.icsi.berkeley.edu/~luby/>
- Coding-Aware Routing, <http://arena.cse.sc.edu/papers/rocx.secon06.pdf>
- MIT offers a course: Introduction to Network Coding
- Network coding: Networking's next revolution?
- Coding-aware protocol design for wireless networks: <http://scholarcommons.sc.edu/etd/230/>

# Chapter 75

## Noisy text

The **noise** can be seen as all the differences between the surface form of a coded representation of the **text** and the intended, correct, or original text.<sup>[1]</sup> It can be due to e.g. **typographic errors** or **colloquialisms** always present in **natural language** and usually lowers the **data quality** in a way that makes the text less accessible to automated processing by computers such as **natural language processing**. The **noise** can also get introduced through an extraction process (i.e. **transcription**, **OCR**) from media other than original **electronic texts**.<sup>[2]</sup>

Language usage over computer mediated discourses, like **chats**, **emails** and **SMS** texts, significantly differs from the standard form of the language. An urge towards shorter message length facilitating **faster typing** and the need for **semantic** clarity, shape the structure of this text used in such discourses.

Various business analysts estimate that **unstructured data** constitutes around 80% of the whole **enterprise data**. A great proportion of this data comprises chat transcripts, emails and other informal and semi-formal internal and external communications. Usually such text is meant for human consumption, but - given the amount of data - manual processing and evaluation of those resources is not practically feasible anymore. This raises the need for robust **text mining** methods.<sup>[3]</sup>

### 75.1 Techniques for noise reduction

To reduce the amount of noise in typed text as it is produced, **spell checkers** and **grammar checkers** available today. Many **word processors** like **MS Word** include this in the editing tool. Online, **Google search** includes a search term suggestion engine to guide users when they make mistakes with their queries.

### 75.2 See also

- **natural language understanding**
- **jargon**, **Leet speak**

- **noisy channel**
- **data corruption**

### 75.3 References

- [1] Knoblock, C., Lopresti, D., Roy, S., Subramaniam, L. V. (2007). "Special Issue on Noisy Text Analytics". *International Journal on Document Analysis and Recognition*.
- [2] Vinciarelli, A. (2005). "Noisy text categorization". *IEEE Transactions on Pattern Analysis and Machine Intelligence* Volume. **27** (12). doi:10.1109/TPAMI.2005.248.
- [3] Subramaniam, L. V., Roy, S., Faruque, T. A., Negi, S. (2009). *A survey of types of text noise and techniques to handle noisy text*. Third Workshop on Analytics for Noisy Unstructured Text Data (AND).

## Chapter 76

# Noisy-channel coding theorem

“Shannon’s theorem” redirects here. Shannon’s name is also associated with the [sampling theorem](#).

In [information theory](#), the **noisy-channel coding theorem** (sometimes **Shannon’s theorem**), establishes that for any given degree of [noise contamination of a communication channel](#), it is possible to communicate discrete data ([digital information](#)) nearly error-free up to a computable maximum rate through the channel. This result was presented by [Claude Shannon](#) in 1948 and was based in part on earlier work and ideas of [Harry Nyquist](#) and [Ralph Hartley](#).

The **Shannon limit** or **Shannon capacity** of a communications channel is the theoretical maximum information transfer rate of the channel, for a particular noise level.

## 76.1 Overview

Stated by [Claude Shannon](#) in 1948, the theorem describes the maximum possible efficiency of [error-correcting methods](#) versus levels of noise interference and data corruption. Shannon’s theorem has wide-ranging applications in both communications and [data storage](#). This theorem is of foundational importance to the modern field of [information theory](#). Shannon only gave an outline of the proof. The first rigorous proof for the discrete case is due to [Amiel Feinstein](#) in 1954.

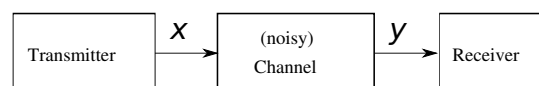
The Shannon theorem states that given a noisy channel with [channel capacity](#)  $C$  and information transmitted at a rate  $R$ , then if  $R < C$  there exist [codes](#) that allow the [probability of error](#) at the receiver to be made arbitrarily small. This means that, theoretically, it is possible to transmit information nearly without error at any rate below a limiting rate,  $C$ .

The converse is also important. If  $R > C$ , an arbitrarily small probability of error is not achievable. All codes will have a probability of error greater than a certain positive minimal level, and this level increases as the rate increases. So, information cannot be guaranteed to be transmitted reliably across a channel at rates beyond the channel capacity. The theorem does not address the rare situation in which rate and capacity are equal.

The channel capacity  $C$  can be calculated from the physical properties of a channel; for a band-limited channel with Gaussian noise, using the [Shannon–Hartley theorem](#).

Simple schemes such as “send the message 3 times and use a best 2 out of 3 voting scheme if the copies differ” are inefficient error-correction methods, unable to asymptotically guarantee that a block of data can be communicated free of error. Advanced techniques such as [Reed–Solomon codes](#) and, more recently, [low-density parity-check \(LDPC\)](#) codes and [turbo codes](#), come much closer to reaching the theoretical Shannon limit, but at a cost of high computational complexity. Using these highly efficient codes and with the computing power in today’s [digital signal processors](#), it is now possible to reach very close to the Shannon limit. In fact, it was shown that LDPC codes can reach within 0.0045 dB of the Shannon limit (for binary AWGN channels, with very long block lengths).<sup>[1]</sup>

## 76.2 Mathematical statement



Theorem (Shannon, 1948):

1. For every discrete memoryless channel, the [channel capacity](#)

$$C = \sup_{p_X} I(X; Y) \quad [2]$$

has the following property. For any  $\epsilon > 0$  and  $R < C$ , for large enough  $N$ , there exists a code of length  $N$  and rate  $\geq R$  and a decoding algorithm, such that the maximal probability of block error is  $\leq \epsilon$ .

2. If a probability of bit error  $pb$  is acceptable, rates up to  $R(pb)$  are achievable, where

$$R(p_b) = \frac{C}{1 - H_2(p_b)}.$$

and  $H_2(p_b)$  is the *binary entropy function*

$$H_2(p_b) = -[p_b \log_2 p_b + (1 - p_b) \log_2 (1 - p_b)]$$

3. For any  $p_b$ , rates greater than  $R(p_b)$  are not achievable.

(MacKay (2003), p. 162; cf Gallager (1968), ch.5; Cover and Thomas (1991), p. 198; Shannon (1948) thm. 11)

## 76.3 Outline of proof

As with several other major results in information theory, the proof of the noisy channel coding theorem includes an achievability result and a matching converse result. These two components serve to bound, in this case, the set of possible rates at which one can communicate over a noisy channel, and matching serves to show that these bounds are tight bounds.

The following outlines are only one set of many different styles available for study in information theory texts.

### 76.3.1 Achievability for discrete memory-less channels

This particular proof of achievability follows the style of proofs that make use of the *asymptotic equipartition property* (AEP). Another style can be found in information theory texts using *error exponents*.

Both types of proofs make use of a random coding argument where the codebook used across a channel is randomly constructed - this serves to make the analysis simpler while still proving the existence of a code satisfying a desired low probability of error at any data rate below the *channel capacity*.

By an AEP-related argument, given a channel, length  $n$  strings of source symbols  $X_1^n$ , and length  $n$  strings of channel outputs  $Y_1^n$ , we can define a *jointly typical set* by the following:

$$A_\varepsilon^{(n)} = \{(x^n, y^n) \in \mathcal{X}^n \times \mathcal{Y}^n$$

$$2^{-n(H(X)+\varepsilon)} \leq p(X_1^n) \leq 2^{-n(H(X)-\varepsilon)}$$

$$2^{-n(H(Y)+\varepsilon)} \leq p(Y_1^n) \leq 2^{-n(H(Y)-\varepsilon)}$$

$$2^{-n(H(X,Y)+\varepsilon)} \leq p(X_1^n, Y_1^n) \leq 2^{-n(H(X,Y)-\varepsilon)}\}$$

We say that two sequences  $X_1^n$  and  $Y_1^n$  are *jointly typical* if they lie in the jointly typical set defined above.

#### Steps

1. In the style of the random coding argument, we randomly generate  $2^{nR}$  codewords of length  $n$  from a probability distribution  $Q$ .
2. This code is revealed to the sender and receiver. It is also assumed that one knows the transition matrix  $p(y|x)$  for the channel being used.
3. A message  $W$  is chosen according to the uniform distribution on the set of codewords. That is,  $Pr(W = w) = 2^{-nR}, w = 1, 2, \dots, 2^{nR}$ .
4. The message  $W$  is sent across the channel.
5. The receiver receives a sequence according to  $P(y^n|x^n(w)) = \prod_{i=1}^n p(y_i|x_i(w))$
6. Sending these codewords across the channel, we receive  $Y_1^n$ , and decode to some source sequence if there exists exactly 1 codeword that is jointly typical with  $Y$ . If there are no jointly typical codewords, or if there are more than one, an error is declared. An error also occurs if a decoded codeword doesn't match the original codeword. This is called *typical set decoding*.

The probability of error of this scheme is divided into two parts:

1. First, error can occur if no jointly typical  $X$  sequences are found for a received  $Y$  sequence
  2. Second, error can occur if an incorrect  $X$  sequence is jointly typical with a received  $Y$  sequence.
- By the randomness of the code construction, we can assume that the average probability of error averaged over all codes does not depend on the index sent. Thus, without loss of generality, we can assume  $W = 1$ .
  - From the joint AEP, we know that the probability that no jointly typical  $X$  exists goes to 0 as  $n$  grows large. We can bound this error probability by  $\varepsilon$ .
  - Also from the joint AEP, we know the probability that a particular  $X_1^n(i)$  and the  $Y_1^n$  resulting from  $W = 1$  are jointly typical is  $\leq 2^{-n(I(X;Y)-3\varepsilon)}$ .

Define:  $E_i = \{(X_1^n(i), Y_1^n) \in A_\varepsilon^{(n)}\}, i = 1, 2, \dots, 2^{nR}$

as the event that message  $i$  is jointly typical with the sequence received when message 1 is sent.

$$\begin{aligned} P(\text{error}) &= P(\text{error}|W=1) \leq P(E_1^c) + \sum_{i=2}^{2^{nR}} P(E_i) \\ &\leq P(E_1^c) + (2^{nR} - 1)2^{-n(I(X;Y)-3\varepsilon)} \\ &\leq \varepsilon + 2^{-n(I(X;Y)-R-3\varepsilon)}. \end{aligned}$$

We can observe that as  $n$  goes to infinity, if  $R < I(X;Y)$  for the channel, the probability of error will go to 0.

Finally, given that the average codebook is shown to be “good,” we know that there exists a codebook whose performance is better than the average, and so satisfies our need for arbitrarily low error probability communicating across the noisy channel.

### 76.3.2 Weak converse for discrete memoryless channels

Suppose a code of  $2^{nR}$  codewords. Let  $W$  be drawn uniformly over this set as an index. Let  $X^n$  and  $Y^n$  be the codewords and received codewords, respectively.

1.  $nR = H(W) = H(W|Y^n) + I(W; Y^n)$  using identities involving entropy and **mutual information**
2.  $\leq H(W|Y^n) + I(X^n(W); Y^n)$  since  $X$  is a function of  $W$
3.  $\leq 1 + P_e^{(n)}nR + I(X^n(W); Y^n)$  by the use of **Fano's Inequality**
4.  $\leq 1 + P_e^{(n)}nR + nC$  by the fact that capacity is maximized mutual information.

The result of these steps is that  $P_e^{(n)} \geq 1 - \frac{1}{nR} - \frac{C}{R}$ . As the block length  $n$  goes to infinity, we obtain  $P_e^{(n)}$  is bounded away from 0 if  $R$  is greater than  $C$  - we can get arbitrarily low rates of error only if  $R$  is less than  $C$ .

### 76.3.3 Strong converse for discrete memoryless channels

A strong converse theorem, proven by Wolfowitz in 1957,<sup>[3]</sup> states that,

$$P_e \geq 1 - \frac{4A}{n(R-C)^2} - e^{-\frac{n(R-C)}{2}}$$

for some finite positive constant  $A$ . While the weak converse states that the error probability is bounded away

from zero as  $n$  goes to infinity, the strong converse states that the error goes to 1. Thus,  $C$  is a sharp threshold between perfectly reliable and completely unreliable communication.

## 76.4 Channel coding theorem for non-stationary memoryless channels

We assume that the channel is memoryless, but its transition probabilities change with time, in a fashion known at the transmitter as well as the receiver.

Then the channel capacity is given by

$$C = \liminf \max_{p(X_1), p(X_2), \dots} \frac{1}{n} \sum_{i=1}^n I(X_i; Y_i).$$

The maximum is attained at the capacity achieving distributions for each respective channel. That is,  $C = \liminf \frac{1}{n} \sum_{i=1}^n C_i$  where  $C_i$  is the capacity of the  $i$ th channel.

### 76.4.1 Outline of the proof

The proof runs through in almost the same way as that of channel coding theorem. Achievability follows from random coding with each symbol chosen randomly from the capacity achieving distribution for that particular channel. Typicality arguments use the definition of typical sets for non-stationary sources defined in the **asymptotic equipartition property** article.

The technicality of  $\liminf$  comes into play when  $\frac{1}{n} \sum_{i=1}^n C_i$  does not converge.

## 76.5 See also

- **Asymptotic equipartition property (AEP)**
- **Fano's Inequality**
- **Rate–distortion theory**
- **Shannon's source coding theorem**
- **Shannon–Hartley theorem**
- **Turbo code**

## 76.6 Notes

- [1] Sae-Young Chung, G. David Forney, Jr., Thomas J. Richardson, and Rüdiger Urbanke, "On the Design of

Low-Density Parity-Check Codes within 0.0045 dB of the Shannon Limit", *IEEE Communications Letters*, 5: 58-60, Feb. 2001. ISSN 1089-7798

- [2] For a description of the “sup” function, see [Supremum](#)
- [3] Robert Gallager. *Information Theory and Reliable Communication*. New York: John Wiley & Sons, 1968. ISBN 0-471-29048-3

## 76.7 References

- Cover T. M., Thomas J. A., *Elements of Information Theory*, John Wiley & Sons, 1991. ISBN 0-471-06259-6
- Fano, R. A., *Transmission of information; a statistical theory of communications*, MIT Press, 1961. ISBN 0-262-06001-9
- Feinstein, Amiel, “A New basic theorem of information theory”, *IEEE Transactions on Information Theory*, 4(4): 2-22, 1954.
- MacKay, David J. C., *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, 2003. ISBN 0-521-64298-1 [free online]
- Shannon, C. E., *A Mathematical Theory of Communication* Urbana, IL: University of Illinois Press, 1949 (reprinted 1998).
- Wolfowitz, J., “The coding of messages subject to chance errors”, *Illinois J. Math.*, 1: 591–606, 1957.

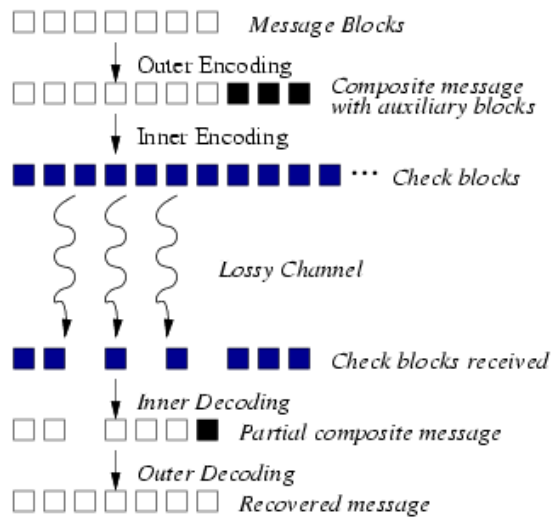
## 76.8 External links

- [On Shannon and Shannon’s law](#)
- [Shannon’s Noisy Channel Coding Theorem](#)

# Chapter 77

## Online codes

In computer science, **online codes** are an example of **rateless erasure codes**. These codes can encode a message into a number of symbols such that knowledge of any fraction of them allows one to recover the original message (with high probability). *Rateless* codes produce an arbitrarily large number of symbols which can be broadcast until the receivers have enough symbols.



High level view of the use of online codes

The online encoding **algorithm** consists of several phases. First the message is split into  $n$  fixed size message blocks. Then the *outer encoding* is an **erasure code** which produces auxiliary blocks that are appended to the message blocks to form a composite message.

From this the inner encoding generates check blocks. Upon receiving a certain number of check blocks some fraction of the composite message can be recovered. Once enough has been recovered the outer decoding can be used to recover the original message.

### 77.1 Detailed discussion

Online codes are parameterised by the block size and two scalars,  $q$  and  $\epsilon$ . The authors suggest  $q=3$  and  $\epsilon=0.01$ . These parameters set the balance between the complexity

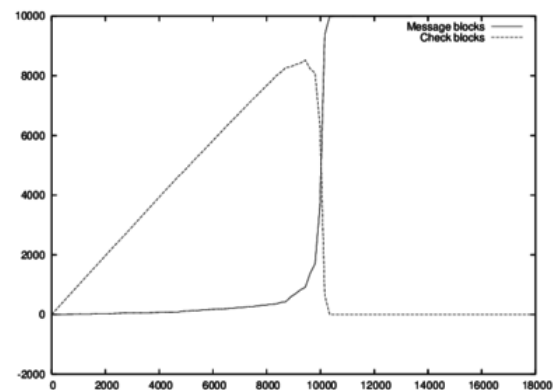
and performance of the encoding. A message of  $n$  blocks can be recovered, **with high probability**, from  $(1+3\epsilon)n$  check blocks. The probability of failure is  $(\epsilon/2)^{q+1}$ .

#### 77.1.1 Outer encoding

Any erasure code may be used as the outer encoding, but the author of online codes suggest the following.

For each message block, pseudo-randomly choose  $q$  auxiliary blocks (from a total of  $0.55q\epsilon n$  auxiliary blocks) to attach it to. Each auxiliary block is then the XOR of all the message blocks which have been attached to it.

#### 77.1.2 Inner encoding



A graph of check blocks received against number of message blocks fixed for a 10000 block message.

The inner encoding takes the composite message and generates a stream of check blocks. A check block is the XOR of all the blocks from the composite message that it is attached to.

The *degree* of a check block is the number of blocks that it is attached to. The degree is determined by sampling a random distribution,  $p$ , which is defined as:

$$F = \left\lceil \frac{\ln(\epsilon^2/4)}{\ln(1 - \epsilon/2)} \right\rceil$$



$$p_1 = 1 - \frac{1 + 1/F}{1 + \epsilon}$$

$$p_i = \frac{(1-p_1)^F}{(F-1)^{i(i-1)}} \text{ for } 2 \leq i \leq F$$

Once the degree of the check block is known, the blocks from the composite message which it is attached to are chosen uniformly.

### 77.1.3 Decoding

Obviously the decoder of the inner stage must hold check blocks which it cannot currently decode. A check block can only be decoded when all but one of the blocks which it is attached to are known. The graph to the left shows the progress of an inner decoder. The x-axis plots the number of check blocks received and the dashed line shows the number of check blocks which cannot currently be used. This climbs almost linearly at first as many check blocks with degree  $> 1$  are received but unusable. At a certain point, some of the check blocks are suddenly usable, resolving more blocks which then causes more check blocks to be usable. Very quickly the whole file can be decoded.

As the graph also shows the inner decoder falls just shy of decoding everything for a little while after having received  $n$  check blocks. The outer encoding ensures that a few elusive blocks from the inner decoder are not an issue, as the file can be recovered without them.

## 77.2 External links

- [Original paper](#)
- [Rateless Codes and Big Downloads](#) (A more accessible paper by the same author)
- [Papers by Petar Maymounkov](#)
- [A Ruby project hosted at RubyForge](#) containing a Ruby library for Online Coding

## Chapter 78

# Package-merge algorithm

The **package-merge algorithm** is an  $O(nL)$ -time algorithm for finding an optimal **length-limited Huffman code** for a given distribution on a given alphabet of size  $n$ , where no code word is longer than  $L$ . It is a **greedy algorithm**, and a generalization of **Huffman's original algorithm**. Package-merge works by reducing the code construction problem to the binary *coin collector's problem*.<sup>[1]</sup>

### 78.1 The coin collector's problem

Suppose a coin collector has a number of coins of various denominations, each of which has a **numismatic value** unrelated to its denomination. The coin collector has run out of money and needs to use some of his coin collection to buy something of cost  $N$ . He wishes to select a subset of coins from his collection of minimum numismatic value whose denominations total  $N$ .

The binary version of this problem is that all denominations are powers of 2, that is, 1, 1/2, 1/4, etc. dollars.

### 78.2 Description of the package-merge algorithm

Assume that the largest denomination is 1 dollar, and that  $N$  is an integer. (The algorithm works even if these assumptions do not hold, by trivial modifications.) The coin collector first separates his coins into lists, one for each denomination, sorted by numismatic value. He then **packages** the smallest denomination coins in pairs, starting from the pair of smallest total numismatic value. If there is one coin left over, it will be the coin of highest numismatic value of that denomination, and it is set aside and ignored henceforth. These packages are then **merged** into the list of coins of the next smallest denomination, again in order of numismatic value. The items in that list are then **packaged** in pairs, and merged into the next smallest list, and so forth.

Finally, there is a list of items, each of which is a 1 dollar coin or a package consisting of two or more smaller coins whose denominations total 1 dollar. They are also sorted

in order of numismatic value. The coin collector then selects the least value  $N$  of them.

Note that the time of the algorithm is linear in the number of coins.

### 78.3 Reduction of length-limited Huffman coding to the coin collector's problem

Let  $L$  be the maximum length any code word is permitted to have. Let  $p_1, \dots, p_n$  be the frequencies of the symbols of the alphabet to be encoded. We first sort the symbols so that  $p_i \leq p_{i+1}$ . Create  $L$  coins for each symbol, of denominations  $2^{-1}, \dots, 2^{-L}$ , each of numismatic value  $p_i$ . Use the package-merge algorithm to select the set of coins of minimum numismatic value whose denominations total  $n - 1$ . Let  $h_i$  be the number of coins of numismatic value  $p_i$  selected. The optimal length-limited Huffman code will encode symbol  $i$  with a bit string of length  $h_i$ . The **canonical Huffman code** can easily be constructed by a simple bottom-up greedy method, given that the  $h_i$  are known, and this can be the basis for fast **data compression**.<sup>[2]</sup>

### 78.4 Performance improvements and generalizations

With this reduction, the algorithm is  $O(nL)$ -time and  $O(nL)$ -space. However, the original paper, "A fast algorithm for optimal length-limited Huffman codes", shows how this can be improved to  $O(nL)$ -time and  $O(n)$ -space. The idea is to run the algorithm a first time, only keeping enough data to be able to determine two equivalent subproblems that sum to half the size of the original problem. This is done recursively, resulting in an algorithm that takes about twice as long but requires only linear space.<sup>[1]</sup>

Many other improvements have been made to the package-merge algorithm to reduce the **multiplicative constant** and to make it faster in special cases, such


as those problems having repeated *pis*.<sup>[3]</sup> The package-merge approach has also been adapted to related problems such as *alphabetic coding*.<sup>[4]</sup>

Methods involving *graph theory* have been shown to have better asymptotic space complexity than the package-merge algorithm, but these have not seen as much practical application.

## 78.5 References

- [1] Larmore, Lawrence L.; Hirschberg, Daniel S. (1990). “A fast algorithm for optimal length-limited Huffman codes”. *Journal of the Association for Computing Machinery*. **37** (3): 464–473. doi:10.1145/79147.79150.
- [2] Moffat, Alistair; Turpin, Andrew (Oct 1997). “On the implementation of minimum redundancy prefix codes”. *IEEE Transactions on Communications*. **45** (10): 1200–1207. doi:10.1109/26.634683.
- [3] Witten, Ian H.; Moffat, Alistair; Bell, Timothy Clinton (1999). *Managing Gigabytes: Compressing and indexing documents and images* (2 ed.). Morgan Kaufmann Publishers. ISBN 978-1-55860-570-1. 1558605703.
- [4] Larmore, Lawrence L.; Przytycka, Teresa M. (1994). “A Fast Algorithm for Optimal Height-Limited Alphabetic Binary-Trees”. *SIAM Journal on Computing*. **23** (6): 1283–1312. doi:10.1137/s0097539792231167.

## 78.6 External links

- Baer, Michael B. “Twenty (or so) Questions: *D*-ary Length-Bounded Prefix Coding”. arXiv:cs.IT/0602085 .
- Moffat, Alistair; Turpin, Andrew; Katajainen, Jyrki (March 1995). *Space-Efficient Construction of Optimal Prefix Codes*. IEEE Data Compression Conference. Snowbird, Utah, USA. doi:10.1109/DCC.1995.515509.
- An implementation of the package-merge algorithm  
""

## Chapter 79

# Packet erasure channel

The **packet erasure channel** is a communication channel model where sequential packets are either received or lost (at a known location). This channel model is closely related to the binary erasure channel.

An erasure code can be used for forward error correction on such a channel.

### 79.1 See also

- Network traffic simulation
- Traffic generation model

### 79.2 References

- Philip A. Chou, Mihaela van der Schaar: Multi-media over IP and wireless networks, ISBN 0-12-088480-1

## Chapter 80

# Parity-check matrix

In coding theory, a **parity-check matrix** of a linear block code  $C$  is a matrix which describes the linear relations that the components of a **codeword** must satisfy. It can be used to decide whether a particular vector is a codeword and is also used in decoding algorithms.

### 80.1 Definition

Formally, a parity check matrix,  $H$  of a linear code  $C$  is a **generator matrix** of the dual code,  $C^\perp$ . This means that a codeword  $\mathbf{c}$  is in  $C$  if and only if the matrix-vector product  $H\mathbf{c}^\top = \mathbf{0}$  (some authors<sup>[1]</sup> would write this in an equivalent form,  $\mathbf{c}H^\top = \mathbf{0}$ .)

The rows of a parity check matrix are the coefficients of the parity check equations.<sup>[2]</sup> That is, they show how linear combinations of certain digits (components) of each codeword equal zero. For example, the parity check matrix

$$H = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

compactly represents the parity check equations,

$$c_3 + c_4 = 0$$

$$c_1 + c_2 = 0$$

that must be satisfied for the vector  $(c_1, c_2, c_3, c_4)$  to be a codeword of  $C$ .

From the definition of the parity-check matrix it directly follows the minimum distance of the code is the minimum number  $d$  such that every  $d - 1$  columns of a parity-check matrix  $H$  are linearly independent while there exist  $d$  columns of  $H$  that are linearly dependent.

### 80.2 Creating a parity check matrix

The parity check matrix for a given code can be derived from its **generator matrix** (and vice versa).<sup>[3]</sup> If the gen-

erator matrix for an  $[n, k]$ -code is in standard form

$$G = [I_k | P]$$

then the parity check matrix is given by

$$H = [-P^\top | I_{n-k}]$$

because

$$GH^\top = P - P = 0$$

Negation is performed in the finite field  $\mathbb{F}_q$ . Note that if the **characteristic** of the underlying field is 2 (i.e.,  $1 + 1 = 0$  in that field), as in **binary codes**, then  $-P = P$ , so the negation is unnecessary.

For example, if a binary code has the generator matrix

$$G = \left[ \begin{array}{cc|cc} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{array} \right]$$

then its parity check matrix is

$$H = \left[ \begin{array}{cc|ccc} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{array} \right]$$

### 80.3 Syndromes

For any (row) vector  $\mathbf{x}$  of the ambient vector space,  $\mathbf{s} = H\mathbf{x}^\top$  is called the **syndrome** of  $\mathbf{x}$ . The vector  $\mathbf{x}$  is a codeword if and only if  $\mathbf{s} = \mathbf{0}$ . The calculation of syndromes is the basis for the **syndrome decoding algorithm**.<sup>[4]</sup>

### 80.4 See also

- Hamming code

## 80.5 Notes

- [1] for instance, Roman 1992, p. 200
- [2] Roman 1992, p. 201
- [3] Pless 1998, p. 9
- [4] Pless 1998, p. 20

## 80.6 References

- Hill, Raymond (1986). *A first course in coding theory*. Oxford Applied Mathematics and Computing Science Series. Oxford University Press. p. 69. ISBN 0-19-853803-0.
- Pless, Vera (1998), *Introduction to the Theory of Error-Correcting Codes* (3rd ed.), Wiley Interscience, ISBN 0-471-19047-0
- Roman, Steven (1992), *Coding and Information Theory*, GTM, **134**, Springer-Verlag, ISBN 0-387-97812-7
- J.H. van Lint (1992). *Introduction to Coding Theory*. GTM. **86** (2nd ed.). Springer-Verlag. p. 34. ISBN 3-540-54894-7.

# Chapter 81

## Parvaresh–Vardy code

**Parvaresh–Vardy codes** are a family of **error-correcting codes** first described in 2005 by Farzad Parvaresh and Alexander Vardy.<sup>[1]</sup> They can be used for efficient list-decoding.

### 81.1 See also

- Reed–Solomon code
- Folded Reed–Solomon code

### 81.2 References

- [1] Parvaresh, Farzad; Alexander Vardy (October 2005). “Correcting Errors Beyond the Guruswami-Sudan Radius in Polynomial Time”. *Proceedings of the 2005 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS’05)*: 285–294. doi:10.1109/SFCS.2005.29.



# Chapter 82

## Plotkin bound

In the **mathematics** of **coding theory**, the **Plotkin bound**, named after Morris Plotkin, is a limit (or bound) on the maximum possible number of codewords in **binary codes** of given length  $n$  and given minimum distance  $d$ .

### 82.1 Statement of the bound

A code is considered “binary” if the codewords use symbols from the binary **alphabet**  $\{0, 1\}$ . In particular, if all codewords have a fixed length  $n$ , then the binary code has length  $n$ . Equivalently, in this case the codewords can be considered elements of **vector space**  $\mathbb{F}_2^n$  over the finite field  $\mathbb{F}_2$ . Let  $d$  be the minimum distance of  $C$ , i.e.

$$d = \min_{x, y \in C, x \neq y} d(x, y)$$

where  $d(x, y)$  is the **Hamming distance** between  $x$  and  $y$ . The expression  $A_2(n, d)$  represents the maximum number of possible codewords in a binary code of length  $n$  and minimum distance  $d$ . The Plotkin bound places a limit on this expression.

**Theorem (Plotkin bound):**

i) If  $d$  is even and  $2d > n$ , then

$$A_2(n, d) \leq 2 \left\lfloor \frac{d}{2d - n} \right\rfloor.$$

ii) If  $d$  is odd and  $2d + 1 > n$ , then

$$A_2(n, d) \leq 2 \left\lfloor \frac{d + 1}{2d + 1 - n} \right\rfloor.$$

iii) If  $d$  is even, then

$$A_2(2d, d) \leq 4d.$$

iv) If  $d$  is odd, then

$$A_2(2d + 1, d) \leq 4d + 4$$

where  $\lfloor \cdot \rfloor$  denotes the **floor function**.

### 82.2 Proof of case i)

Let  $d(x, y)$  be the **Hamming distance** of  $x$  and  $y$ , and  $M$  be the number of elements in  $C$  (thus,  $M$  is equal to  $A_2(n, d)$ ). The bound is proved by bounding the quantity  $\sum_{(x, y) \in C^2, x \neq y} d(x, y)$  in two different ways.

On the one hand, there are  $M$  choices for  $x$  and for each such choice, there are  $M - 1$  choices for  $y$ . Since by definition  $d(x, y) \geq d$  for all  $x$  and  $y$  ( $x \neq y$ ), it follows that

$$\sum_{(x, y) \in C^2, x \neq y} d(x, y) \geq M(M - 1)d.$$

On the other hand, let  $A$  be an  $M \times n$  matrix whose rows are the elements of  $C$ . Let  $s_i$  be the number of zeros contained in the  $i$ 'th column of  $A$ . This means that the  $i$ 'th column contains  $M - s_i$  ones. Each choice of a zero and a one in the same column contributes exactly 2 (because  $d(x, y) = d(y, x)$ ) to the sum  $\sum_{x, y \in C} d(x, y)$  and therefore

$$\sum_{x, y \in C} d(x, y) = \sum_{i=1}^n 2s_i(M - s_i).$$

The quantity on the right is maximized if and only if  $s_i = M/2$  holds for all  $i$  (at this point of the proof we ignore the fact, that the  $s_i$  are integers), then

$$\sum_{x, y \in C} d(x, y) \leq \frac{1}{2}nM^2.$$

Combining the upper and lower bounds for  $\sum_{x, y \in C} d(x, y)$  that we have just derived,

$$M(M - 1)d \leq \frac{1}{2}nM^2$$

which given that  $2d > n$  is equivalent to

$$M \leq \frac{2d}{2d - n}.$$

Since  $M$  is even, it follows that

$$M \leq 2 \left\lfloor \frac{d}{2d-n} \right\rfloor.$$

This completes the proof of the bound.

### 82.3 See also

- Singleton bound
- Hamming bound
- Elias-Bassalygo bound
- Gilbert-Varshamov bound
- Johnson bound
- Griesmer bound

### 82.4 References

- Plotkin, M. (1960), “Binary codes with specified minimum distance”, *IRE Transactions on Information Theory*, **6**: 445–450, doi:10.1109/TIT.1960.1057584

## Chapter 83

# Polar code (coding theory)

In information theory, a **polar code** is a linear block error correcting code. The code construction is based on a multiple recursive concatenation of a short kernel code which transforms the physical channel into virtual outer channels. When the number of recursions becomes large, the virtual channels tend to either have high reliability or low reliability (in other words, they polarize), and the data bits are allocated to the most reliable channels. The construction itself was first described by Stolz,<sup>[1]</sup> and later independently by Erdal Arıkan.<sup>[2]</sup> It is the first code with an explicit construction to provably achieve the channel capacity for symmetric binary-input, discrete, memoryless channels (B-DMC) with polynomial dependence on the gap to capacity. Notably, polar codes have modest encoding and decoding complexity  $O(n \log n)$ , which renders them attractive for many applications.

### 83.1 Simulating Polar Codes

One can implement a simulation environment of polar codes in any programming language such as MATLAB, C++ etc.

It typically involves modelling an encoder, a decoder, a channel (such as AWGN, BSC, BEC), and a code-construction module.

An example MATLAB implementation is available at,<sup>[3]</sup> including a series of introductory video tutorials.

### 83.2 Industrial Applications

There are many aspects that polar codes should investigate further before considering for industry applications. Especially, the original design of the polar codes achieves capacity when block sizes are asymptotically large with successive cancellation decoder. However, in block sizes that industry applications are operating, the performance of the successive cancellation is poor compared to the well-defined and implemented coding schemes such as LDPC and Turbo. Polar performance can be improved with successive cancellation list decoding, but, their usability in real applications still questionable due to very


poor implementation efficiencies.<sup>[4]</sup>

In October 2016 Huawei announced that it had achieved 27Gbps in 5G field trial tests using the Polar codes for channel coding. The improvements has been introduced so that the channel performance is now has almost closed the gap between the Shannon limit which sets the bar for the maximum rate for the given bandwidth for a given noise level/<sup>[5]</sup>

### 83.3 See also

- Category:Capacity-achieving codes
- Category:Capacity-approaching codes

### 83.4 References

- [1] Stolz, N. (January 2002). "Chapter 6.1: Optimierte Konstruktion für bitweise Mehrstufendecodierung". *Rekursive codes mit der Plotkin-Konstruktion und ihre Decodierung* (Ph.D. dissertation, Technische Universität Darmstadt).
- [2] Arıkan, E. (July 2009). "Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels". *IEEE Transactions on Information Theory*. **55** (7): 3051–73. arXiv:0807.3917v5  doi:10.1109/TIT.2009.2021379.
- [3] "www.polarcodes.com". *Resources on Polar Codes*.
- [4] Arıkan, Erdal, et al. "Challenges and some new directions in channel coding." arXiv:1504.03916 (2015).
- [5] "Huawei achieves 27Gbps 5G speeds with Polar Code". Retrieved 2016-10-10.

# Chapter 84

## Polynomial code

In coding theory, a **polynomial code** is a type of linear code whose set of valid code words consists of those polynomials (usually of some fixed length) that are divisible by a given fixed polynomial (of shorter length, called the *generator polynomial*).

### 84.1 Definition

Fix a finite field  $GF(q)$ , whose elements we call *symbols*. For the purposes of constructing polynomial codes, we identify a string of  $n$  symbols  $a_{n-1} \dots a_0$  with the polynomial

$$a_{n-1}x^{n-1} + \dots + a_1x + a_0.$$

Fix integers  $m \leq n$  and let  $g(x)$  be some fixed polynomial of degree  $m$ , called the *generator polynomial*. The polynomial code generated by  $g(x)$  is the code whose code words are precisely the polynomials of degree less than  $n$  that are divisible (without remainder) by  $g(x)$ .

### 84.2 Example

Consider the polynomial code over  $GF(2) = \{0, 1\}$  with  $n = 5$ ,  $m = 2$ , and generator polynomial  $g(x) = x^2 + x + 1$ . This code consists of the following code words:

$$0 \cdot g(x), \quad 1 \cdot g(x), \quad x \cdot g(x), \quad (x+1) \cdot g(x), \\ x^2 \cdot g(x), \quad (x^2+1) \cdot g(x), \quad (x^2+x) \cdot g(x), \quad (x^2+x+1) \cdot g(x).$$

Or written explicitly:

$$0, \quad x^2 + x + 1, \quad x^3 + x^2 + x, \quad x^3 + 2x^2 + 2x + 1, \\ x^4 + x^3 + x^2, \quad x^4 + x^3 + 2x^2 + x + 1, \quad x^4 + 2x^3 + 2x^2 + x,$$

Since the polynomial code is defined over the Binary Galois Field  $GF(2) = \{0, 1\}$ , polynomial elements are represented as a modulo-2 sum and the final polynomials are:

$$0, \quad x^2 + x + 1, \quad x^3 + x^2 + x, \quad x^3 + 1, \\ x^4 + x^3 + x^2, \quad x^4 + x^3 + x + 1, \quad x^4 + x, \quad x^4 + x^2 + 1.$$

Equivalently, expressed as strings of binary digits, the codewords are:

$$00000, \quad 00111, \quad 01110, \quad 01001, \\ 11100, \quad 11011, \quad 10010, \quad 10101.$$

Note that this, as every polynomial code, is indeed a **linear code**, i.e., linear combinations of code words are again code words. In a case like this where the field is  $GF(2)$ , linear combinations are found by taking the XOR of the codewords expressed in binary form (e.g.  $00111 \text{ XOR } 10010 = 10101$ ).

### 84.3 Encoding

In a polynomial code over  $GF(q)$  with code length  $n$  and generator polynomial  $g(x)$  of degree  $m$ , there will be exactly  $q^{n-m}$  code words. Indeed, by definition,  $p(x)$  is a code word if and only if it is of the form  $p(x) = g(x) \cdot q(x)$ , where  $q(x)$  (the *quotient*) is of degree less than  $n - m$ . Since there are  $q^{n-m}$  such quotients available, there are the same number of possible code words. Plain (unencoded) data words should therefore be of length  $n - m$ .

Some authors, such as (Lidl & Pilz, 1999), only discuss the mapping  $q(x) \mapsto g(x) \cdot q(x)$  as the assignment from data words to code words. However, this has the disadvantage that the data word does not appear as part of the code word.

Instead, the following method is often used to create a **systematic code**: given a data word  $d(x)$  of length  $n - m$ , first multiply  $d(x)$  by  $x^m$ , which has the effect of shifting  $d(x)$  by  $m$  places to the left. In general,  $x^m d(x)$  will not be divisible by  $g(x)$ , i.e., it will not be a valid code word. However, there is a unique code word that can be obtained by adjusting the rightmost  $m$  symbols of  $x^m d(x)$ . To

calculate it, compute the remainder of dividing  $x^m d(x)$  by  $g(x)$  :

$$x^m d(x) = g(x) \cdot q(x) + r(x),$$

where  $r(x)$  is of degree less than  $m$  . The code word corresponding to the data word  $d(x)$  is then defined to be

$$p(x) := x^m d(x) - r(x),$$

Note the following properties:

1.  $p(x) = g(x) \cdot q(x)$  , which is divisible by  $g(x)$  . In particular,  $p(x)$  is a valid code word.
2. Since  $r(x)$  is of degree less than  $m$  , the leftmost  $n-m$  symbols of  $p(x)$  agree with the corresponding symbols of  $x^m d(x)$  . In other words, the first  $n-m$  symbols of the code word are the same as the original data word. The remaining  $m$  symbols are called *checksum digits* or *check bits*.

### 84.3.1 Example

For the above code with  $n = 5$  ,  $m = 2$  , and generator polynomial  $g(x) = x^2 + x + 1$  , we obtain the following assignment from data words to codewords:

- $000 \mapsto 00000$
- $001 \mapsto 00111$
- $010 \mapsto 01001$
- $011 \mapsto 01110$
- $100 \mapsto 10010$
- $101 \mapsto 10101$
- $110 \mapsto 11011$
- $111 \mapsto 11100$

## 84.4 Decoding

An erroneous message can be detected in a straightforward way through polynomial division by the generator polynomial resulting in a non-zero remainder.

Assuming that the code word is free of errors, a systematic code can be decoded simply by stripping away the  $m$  checksum digits.

If there are errors, then error correction should be performed before decoding. Efficient decoding algorithms exist for specific polynomial codes, such as **BCH codes**.

## 84.5 Properties of polynomial codes

As for all digital codes, the **error detection and correction** abilities of polynomial codes are determined by the minimum **Hamming distance** of the code. Since polynomial codes are linear codes, the minimum Hamming distance is equal to the minimum weight of any non-zero codeword. In the example above, the minimum Hamming distance is 2, since 01001 is a codeword, and there is no nonzero codeword with only one bit set.

More specific properties of a polynomial code often depend on particular algebraic properties of its generator polynomial. Here are some examples of such properties:

- A polynomial code is **cyclic** if and only if the generator polynomial divides  $x^n - 1$  .
- If the generator polynomial is **primitive**, then the resulting code has Hamming distance at least 3, provided that  $n \leq 2^m - 1$  .
- In **BCH codes**, the generator polynomial is chosen to have specific roots in an extension field, in a way that achieves high Hamming distance.

The algebraic nature of polynomial codes, with cleverly chosen generator polynomials, can also often be exploited to find efficient error correction algorithms. This is the case for **BCH codes**.

## 84.6 Specific families of polynomial codes

- **Cyclic codes** – every cyclic code is also a polynomial code; a popular example is the **CRC** code.
- **BCH codes** – a family of cyclic codes with high Hamming distance and efficient algebraic error correction algorithms.
- **Reed–Solomon codes** – an important subset of BCH codes with particularly efficient structure.

## 84.7 References

- W.J. Gilbert and W.K. Nicholson: *Modern Algebra with Applications*, 2nd edition, Wiley, 2004.
- R. Lidl and G. Pilz. *Applied Abstract Algebra*, 2nd edition. Wiley, 1999.

# Chapter 85

## Prefix code

A **prefix code** is a type of **code** system (typically a **variable-length code**) distinguished by its possession of the “prefix property”, which requires that there is no whole **code word** in the system that is a **prefix** (initial segment) of any other code word in the system. For example, a code with code words {9, 55} has the prefix property; a code consisting of {9, 5, 59, 55} does not, because “5” is a prefix of “59” and also of “55”. A prefix code is a **uniquely decodable code**: a receiver can identify each word without requiring a special marker between words.

Prefix codes are also known as **prefix-free codes**, **prefix condition codes** and **instantaneous codes**. Although **Huffman coding** is just one of many algorithms for deriving prefix codes, prefix codes are also widely referred to as “Huffman codes”, even when the code was not produced by a Huffman algorithm. The term **comma-free code** is sometimes also applied as a synonym for prefix-free codes<sup>[1][2]</sup> but in most mathematical books and articles (e.g.<sup>[3][4]</sup>) a comma-free code is used to mean a **self-synchronizing code**, a subclass of prefix codes.

Using prefix codes, a message can be transmitted as a sequence of concatenated code words, without any **out-of-band** markers or (alternatively) special markers between words to **frame** the words in the message. The recipient can decode the message unambiguously, by repeatedly finding and removing sequences that form valid code words. This is not generally possible with codes that lack the prefix property, for example {0, 1, 10, 11}: a receiver reading a “1” at the start of a code word would not know whether that was the complete code word “1”, or merely the prefix of the code word “10” or “11”; so the string “10” could be interpreted either as a single codeword or as the concatenation of the words “1” then “0”.

The variable-length **Huffman codes**, **country calling codes**, the country and publisher parts of **ISBNs**, the Secondary Synchronization Codes used in the **UMTS W-CDMA 3G Wireless Standard**, and the **instruction sets** (machine language) of most computer microarchitectures are prefix codes.

Prefix codes are not **error-correcting codes**. In practice, a message might first be compressed with a prefix code, and then encoded again with **channel coding** (including error correction) before transmission.

**Kraft’s inequality** characterizes the sets of code word lengths that are possible in a uniquely decodable code.<sup>[5]</sup>

### 85.1 Techniques

If every word in the code has the same length, the code is called a **fixed-length code**, or a **block code** (though the term **block code** is also used for fixed-size **error-correcting codes** in **channel coding**). For example, **ISO 8859-15** letters are always 8 bits long. **UTF-32/UCS-4** letters are always 32 bits long. **ATM cells** are always 424 bits (53 bytes) long. A fixed-length code of fixed length  $k$  bits can encode up to  $2^k$  source symbols.

A fixed-length code is necessarily a prefix code. It is possible to turn any code into a fixed-length code by padding fixed symbols to the shorter prefixes in order to meet the length of the longest prefixes. Alternately, such padding codes may be employed to introduce redundancy that allows autocorrection and/or synchronisation. However, fixed length encodings are inefficient in situations where some words are much more likely to be transmitted than others.

**Truncated binary encoding** is a straightforward generalization of fixed-length codes to deal with cases where the number of symbols  $n$  is not a power of two. Source symbols are assigned codewords of length  $k$  and  $k+1$ , where  $k$  is chosen so that  $2^k < n \leq 2^{k+1}$ .

**Huffman coding** is a more sophisticated technique for constructing variable-length prefix codes. The Huffman coding algorithm takes as input the frequencies that the code words should have, and constructs a prefix code that minimizes the weighted average of the code word lengths. (This is closely related to minimizing the entropy.) This is a form of **lossless data compression** based on **entropy encoding**.

Some codes mark the end of a code word with a special “comma” symbol, different from normal data.<sup>[6]</sup> This is somewhat analogous to the spaces between words in a sentence; they mark where one word ends and another begins. If every code word ends in a comma, and the comma does not appear elsewhere in a code word, the code is automatically prefix-free. However, modern com-

munication systems send everything as sequences of “1” and “0” – adding a third symbol would be expensive, and using it only at the ends of words would be inefficient. **Morse code** is an everyday example of a variable-length code with a comma. The long pauses between letters, and the even longer pauses between words, help people recognize where one letter (or word) ends, and the next begins. Similarly, **Fibonacci coding** uses a “11” to mark the end of every code word.

Self-synchronizing codes are prefix codes that allow frame synchronization.

## 85.2 Related concepts

A **suffix code** is a set of words none of which is a suffix of any other; equivalently, a set of words which are the reverse of a prefix code. As with a prefix code, the representation of a string as a concatenation of such words is unique. A **bifix code** is a set of words which is both a prefix and a suffix code.<sup>[7]</sup> An **optimal prefix code** is a prefix code with minimal average length. That is, assume an alphabet of  $n$  symbols with probabilities  $p(A_i)$  for a prefix code  $C$ . If  $C'$  is another prefix code and  $\lambda'_i$  are the lengths of the codewords of  $C'$ , then  $\sum_{i=1}^n \lambda_i p(A_i) \leq \sum_{i=1}^n \lambda'_i p(A_i)$ .<sup>[8]</sup>

## 85.3 Prefix codes in use today

Examples of prefix codes include:

- variable-length **Huffman codes**
- country calling codes
- **Chen–Ho encoding**
- the country and publisher parts of **ISBNs**
- the Secondary Synchronization Codes used in the **UMTS W-CDMA 3G Wireless Standard**
- **VCR Plus+ codes**
- **Unicode Transformation Format**, in particular the **UTF-8** system for encoding **Unicode** characters, which is both a prefix-free code and a self-synchronizing code<sup>[9]</sup>

### 85.3.1 Techniques

Commonly used techniques for constructing prefix codes include **Huffman codes** and the earlier **Shannon-Fano codes**, and universal codes such as:

- **Elias delta coding**

- **Elias gamma coding**
- **Elias omega coding**
- **Fibonacci coding**
- **Levenshtein coding**
- **Unary coding**
- **Golomb Rice code**
- **Straddling checkerboard** (simple cryptography technique which produces prefix codes)

## 85.4 Notes

- [1] US Federal Standard 1037C
- [2] *ATIS Telecom Glossary 2007*, retrieved December 4, 2010
- [3] Berstel, Jean; Perrin, Dominique (1985), *Theory of Codes*, Academic Press
- [4] Golomb, S. W.; Gordon, Basil; Welch, L. R. (1958), “Comma-Free Codes”, *Canadian Journal of Mathematics*, **10** (2): 202–209, doi:10.4153/CJM-1958-023-9
- [5] Berstel et al (2010) p.75
- [6] “Development of Trigger and Control Systems for CMS” by J. A. Jones: “Synchronisation” p. 70
- [7] Berstel et al (2010) p.58
- [8] McGill COMP 423 Lecture notes
- [9] Pike, Rob (2003-04-03). “UTF-8 history”.

## 85.5 References

- Berstel, Jean; Perrin, Dominique; Reutenauer, Christophe (2010). *Codes and automata*. Encyclopedia of Mathematics and its Applications. **129**. Cambridge: Cambridge University Press. ISBN 978-0-521-88831-8. Zbl 1187.94001.
- Elias, Peter (1975). “Universal codeword sets and representations of the integers”. *IEEE Trans. Inform. Theory*. **21** (2): 194–203. ISSN 0018-9448. Zbl 0298.94011.
- D.A. Huffman, “A method for the construction of minimum-redundancy codes”, Proceedings of the I.R.E., Sept. 1952, pp. 1098–1102 (Huffman’s original article)
- Profile: David A. Huffman, Scientific American, Sept. 1991, pp. 54–58 (Background story)



- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 16.3, pp. 385–392.
- This article incorporates public domain material from the General Services Administration document “Federal Standard 1037C”.

## 85.6 External links

- Codes, trees and the prefix property by Kona Macphee

## Chapter 86

# Preparata code

In coding theory, the **Preparata codes** form a class of non-linear double-error-correcting codes. They are named after Franco P. Preparata who first described them in 1968.

Although non-linear over  $\text{GF}(2)$  the Preparata codes are linear over  $\mathbf{Z}_4$  with the Lee distance.

- J.H. van Lint (1992). *Introduction to Coding Theory*. GTM. **86** (2nd ed.). Springer-Verlag. pp. 111–113. ISBN 3-540-54894-7.
- [http://www.encyclopediaofmath.org/index.php/Preparata\\_code](http://www.encyclopediaofmath.org/index.php/Preparata_code)
- [http://www.encyclopediaofmath.org/index.php/Kerdock\\_and\\_Preparata\\_codes](http://www.encyclopediaofmath.org/index.php/Kerdock_and_Preparata_codes)

### 86.1 Construction

Let  $m$  be an odd number, and  $n = 2^m - 1$ . We first describe the **extended Preparata code** of length  $2n + 2 = 2^{m+1}$ : the Preparata code is then derived by deleting one position. The words of the extended code are regarded as pairs  $(X, Y)$  of  $2^m$ -tuples, each corresponding to subsets of the finite field  $\text{GF}(2^m)$  in some fixed way.

The extended code contains the words  $(X, Y)$  satisfying three conditions

1.  $X, Y$  each have even weight;
2.  $\sum_{x \in X} x = \sum_{y \in Y} y$ ;
3.  $\sum_{x \in X} x^3 + \left(\sum_{x \in X} x\right)^3 = \sum_{y \in Y} y^3$ .

The Preparata code is obtained by deleting the position in  $X$  corresponding to 0 in  $\text{GF}(2^m)$ .

### 86.2 Properties

The Preparata code is of length  $2^{m+1} - 1$ , size  $2^k$  where  $k = 2^{m+1} - 2m - 2$ , and minimum distance 5.

When  $m = 3$ , the Preparata code of length 15 is also called the **Nordstrom–Robinson code**.

### 86.3 References

- F.P. Preparata (1968). “A class of optimum non-linear double-error-correcting codes”. *Information and Control*. **13** (4): 378–400. doi:10.1016/S0019-9958(68)90874-7.

# Chapter 87

## Puncturing

In coding theory, **puncturing** is the process of removing some of the parity bits after encoding with an error-correction code. This has the same effect as encoding with an error-correction code with a higher rate, or less redundancy. However, with puncturing the same decoder can be used regardless of how many bits have been punctured, thus puncturing considerably increases the flexibility of the system without significantly increasing its complexity.

In some cases, a pre-defined pattern of puncturing is used in an encoder. Then, the inverse operation, known as depuncturing, is implemented by the decoder.

Puncturing is used in **UMTS** during the rate matching process. It is also used in **Wi-Fi**, **GPRS** and **EDGE**, as well as in the **DVB-T** and **DRM** Standards.

Puncturing is often used with the **Viterbi algorithm** in coding systems.

During **Radio Resource Control (RRC)** Connection set procedure, during sending NBAP radio link setup message the uplink puncturing limit will send to NODE B, along with U/L spreading factor & U/L scrambling code.<sup>[1]</sup>

### 87.1 See also

- **Singleton bound**

### 87.2 References

[1] Chris Johnson. "Radio Access Networks for UMTS: Principles and Practice". 2011.

- Pless, Vera (2011). *Introduction to the Theory of Error-Correcting Codes*. Wiley Series in Discrete Mathematics and Optimization. **48** (Third ed.). John Wiley & Sons. ISBN 1118030990.

# Chapter 88

## Quadratic residue code

A **quadratic residue code** is a type of **cyclic code**.

### 88.1 Examples

Examples of quadratic residue codes include the  $(7, 4)$  **Hamming code** over  $GF(2)$ , the  $(23, 12)$  **binary Golay code** over  $GF(2)$  and the  $(11, 6)$  **ternary Golay code** over  $GF(3)$ .

### 88.2 Constructions

There is a quadratic residue code of length  $p$  over the finite field  $GF(l)$  whenever  $p$  and  $l$  are primes,  $p$  is odd, and  $l$  is a **quadratic residue** modulo  $p$ . Its generator polynomial as a cyclic code is given by

$$f(x) = \prod_{j \in Q} (x - \zeta^j)$$

where  $Q$  is the set of quadratic residues of  $p$  in the set  $\{1, 2, \dots, p-1\}$  and  $\zeta$  is a primitive  $p$ th root of unity in some finite extension field of  $GF(l)$ . The condition that  $l$  is a quadratic residue of  $p$  ensures that the coefficients of  $f$  lie in  $GF(l)$ . The dimension of the code is  $(p+1)/2$ . Replacing  $\zeta$  by another primitive  $p$ -th root of unity  $\zeta^r$  either results in the same code or an equivalent code, according to whether or not  $r$  is a quadratic residue of  $p$ .

An alternative construction avoids roots of unity. Define

$$g(x) = c + \sum_{j \in Q} x^j$$

for a suitable  $c \in GF(l)$ . When  $l = 2$  choose  $c$  to ensure that  $g(1) = 1$ . If  $l$  is odd, choose  $c = (1 + \sqrt{p^*})/2$ , where  $p^* = p$  or  $-p$  according to whether  $p$  is congruent to 1 or 3 modulo 4. Then  $g(x)$  also generates a quadratic residue code; more precisely the ideal of  $F_l[X]/\langle X^p - 1 \rangle$  generated by  $g(x)$  corresponds to the quadratic residue code.

### 88.3 Weight

The **minimum weight** of a quadratic residue code of length  $p$  is greater than  $\sqrt{p}$ ; this is the **square root bound**.

### 88.4 Extended code

Adding an overall parity-check digit to a quadratic residue code gives an **extended quadratic residue code**. When  $p \equiv 3 \pmod{4}$  an extended quadratic residue code is self-dual; otherwise it is equivalent but not equal to its dual. By the **Gleason–Prange theorem** (named for **Andrew Gleason** and **Eugene Prange**), the automorphism group of an extended quadratic residue code has a subgroup which is isomorphic to either  $PSL_2(p)$  or  $SL_2(p)$ .

### 88.5 References

- F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland Publishing Co., Amsterdam-New York-Oxford, 1977.
- Blahut, R. E. (September 2006), “The Gleason–Prange theorem”, *IEEE Trans. Inf. Theory*, Piscataway, NJ, USA: IEEE Press, **37** (5): 1269–1273, doi:10.1109/18.133245.

## Chapter 89

# Rank error-correcting code

In coding theory, **rank codes** (also called **Gabidulin codes**) are non-binary<sup>[1]</sup> linear error-correcting codes over not Hamming but *rank* metric. They described a systematic way of building codes that could detect and correct multiple random *rank* errors. By adding redundancy with coding  $k$ -symbol word to a  $n$ -symbol word, a rank code can correct any errors of rank up to  $t = \lfloor (d - 1) / 2 \rfloor$ , where  $d$  is a code distance. As an **erasure code**, it can correct up to  $d - 1$  known erasures.

A **rank code** is an algebraic linear code over the finite field  $GF(q^N)$  similar to **Reed–Solomon code**.

The rank of the vector over  $GF(q^N)$  is the maximum number of linearly independent components over  $GF(q)$ . The rank distance between two vectors over  $GF(q^N)$  is the rank of the difference of these vectors.

The rank code corrects all errors with rank of the error vector not greater than  $t$ .

### 89.1 Rank metric

Let  $X^n$  —  $n$ -dimensional vector space over the finite field  $GF(q^N)$ , where  $q$  is a power of a prime,  $N$  is an integer and  $(u_1, u_2, \dots, u_N)$  with  $u_i \in GF(q)$  is a base of the vector space over the field  $GF(q)$ .

Every element  $x_i \in GF(q^N)$  can be represented as  $x_i = a_{1i}u_1 + a_{2i}u_2 + \dots + a_{Ni}u_N$ . Hence, every vector  $\vec{x} = (x_1, x_2, \dots, x_n)$  over  $GF(q^N)$  can be written as matrix:

$$\vec{x} = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \dots & \dots & \dots & \dots \\ a_{N,1} & a_{N,2} & \dots & a_{N,n} \end{pmatrix}$$

Rank of the vector  $\vec{x}$  over the field  $GF(q^N)$  is a rank of the corresponding matrix  $A(\vec{x})$  over the field  $GF(q)$  denoted by  $r(\vec{x}; q)$ .

The set of all vectors  $\vec{x}$  is a space  $X^n = A_N^n$ . The map  $\vec{x} \rightarrow r(\vec{x}; q)$  defines a norm over  $X^n$  and a *rank metric*:

$$d(\vec{x}; \vec{y}) = r(\vec{x} - \vec{y}; q)$$

### 89.2 Rank code

A set  $\{x_1, x_2, \dots, x_n\}$  of vectors from  $X^n$  is called a code with code distance  $d = \min d(x_i, x_j)$  and a  $k$ -dimensional subspace of  $X^n$  — a linear  $(n, k)$ -code with distance  $d \leq n - k + 1$ .

### 89.3 Generating matrix

There is known the only construction of rank code, which is a *maximum rank distance* MRD-code with  $d = n - k + 1$ .

Let's define a Frobenius power  $[i]$  of the element  $x \in GF(q^N)$  as

$$x^{[i]} = x^{q^{i \bmod N}}.$$

Then, every vector  $\vec{g} = (g_1, g_2, \dots, g_n)$ ,  $g_i \in GF(q^N)$ ,  $n \leq N$ , linearly independent over  $GF(q)$ , defines a generating matrix of the MRD  $(n, k, d = n - k + 1)$ -code.

$$G = \begin{pmatrix} g_1 & g_2 & \dots & g_n \\ g_1^{[m]} & g_2^{[m]} & \dots & g_n^{[m]} \\ g_1^{[2m]} & g_2^{[2m]} & \dots & g_n^{[2m]} \\ \dots & \dots & \dots & \dots \\ g_1^{[km]} & g_2^{[km]} & \dots & g_n^{[km]} \end{pmatrix},$$

where  $\gcd(m, N) = 1$ .

### 89.4 Applications

There are several proposals for public-key cryptosystems based on rank codes. However, most of them have been

proven insecure (see e.g. Journal of Cryptology, April 2008<sup>[2]</sup>).

Rank codes are also useful for error and erasure correction in network coding.

## 89.5 See also

- Linear code
- Reed–Solomon error correction
- Berlekamp–Massey algorithm
- Network coding

## 89.6 Notes

- [1] Codes for which each input symbol is from a set of size greater than 2.
- [2] “Structural Attacks for Public Key Cryptosystems based on Gabidulin Codes”. *Journal of Cryptology*. **21**: 280–301. doi:10.1007/s00145-007-9003-9.

## 89.7 References

- Gabidulin, Ernst M. (1985), “Theory of codes with maximum rank distance”, *Problems of Information Transmission*, **21** (1): 1–12
- Kshevetskiy, Alexander; Gabidulin, Ernst M. (4–9 Sept. 2005), “The new construction of rank codes”, *Proceedings of IEEE International Symposium on Information Theory (ISIT) 2005*: 2105–2108, ISBN 0-7803-9151-9 Check date values in: |date= (help)
- Gabidulin, Ernst M.; Pilipchuk, Nina I. (June 29 – July 4, 2003), “A new method of erasure correction by rank codes”, *Proceedings of the 2003 IEEE International Symposium on Information Theory*: 423, ISBN 0-7803-7728-1

## 89.8 External links

- MATLAB implementation of a Rank–metric codec

# Chapter 90

## Raptor code

In computer science, **raptor codes** (*rapid tornado*;<sup>[1]</sup> see Tornado codes) are the first known class of fountain codes with linear time encoding and decoding. They were invented by Amin Shokrollahi in 2000/2001 and were first published in 2004 as an extended abstract. Raptor codes are a significant theoretical and practical improvement over LT codes, which were the first practical class of fountain codes.

Raptor codes, as with fountain codes in general, encode a given message consisting of a number of symbols,  $k$ , into a potentially limitless sequence of encoding symbols such that knowledge of any  $k$  or more encoding symbols allows the message to be recovered with some non-zero probability. The probability that the message can be recovered increases with the number of symbols received above  $k$  becoming very close to 1, once the number of received symbols is only very slightly larger than  $k$ . For example, with the latest generation of Raptor codes, the RaptorQ codes, the chance of decoding failure when  $k$  symbols have been received is less than 1%, and the chance of decoding failure when  $k+2$  symbols have been received is less than one in a million. A symbol can be any size, from a single byte to hundreds or thousands of bytes.

Raptor codes may be systematic or non-systematic. In the systematic case, the symbols of the original message are included within the set of encoding symbols. An example of a systematic raptor code is the code defined by the 3rd Generation Partnership Project for use in mobile cellular wireless broadcast and multicast and also used by DVB-H standards for IP datacast to handheld devices (see external links). The Raptor codes in these standards is defined also in IETF RFC 5053. The most advanced version of a practical Raptor code is RaptorQ defined in IETF RFC 6330.

Online codes are another example of a non-systematic raptor code.

is applied as a 'pre-code' or 'outer code'. This pre-code may itself be a concatenation of multiple codes, for example in the code standardized by 3GPP a high density parity check code derived from the binary Gray sequence is concatenated with a simple regular low density parity check code. Another possibility would be a concatenation of a Hamming code with a low density parity check code.

The inner code takes the result of the pre-coding operation and generates a sequence of encoding symbols. The inner code is a form of LT codes. Each encoding symbol is the XOR of a pseudo-randomly chosen set of symbols from the pre-code output. The number of symbols which are XOR'ed together to form an output symbol is chosen pseudo-randomly for each output symbol according to a specific probability distribution.

This distribution, as well as the mechanism for generating pseudo-random numbers for sampling this distribution and for choosing the symbols to be XOR'ed, must be known to both sender and receiver. In one approach, each symbol is accompanied with an identifier which can be used as a seed to a pseudo-random number generator to generate this information, with the same process being followed by both sender and receiver.

In the case of non-systematic raptor codes, the source data to be encoded is used as the input to the pre-coding stage.

In the case of systematic raptor codes, the input to the pre-coding stage is obtained by first applying the inverse of the encoding operation that generates the first  $k$  output symbols to the source data. Thus, applying the normal encoding operation to the resulting symbols causes the original source symbols to be regenerated as the first  $k$  output symbols of the code. It is necessary to ensure that the pseudo-random processes which generate the first  $k$  output symbols generate an operation which is invertible.

### 90.1 Overview

Raptor codes are formed by the concatenation of two codes.

A fixed rate erasure code, usually with a fairly high rate,

### 90.2 Decoding

Two approaches are possible for decoding raptor codes. In a concatenated approach, the inner code is decoded first, using a belief propagation algorithm, as used for the



LT codes. Decoding succeeds if this operation recovers a sufficient number of symbols, such that the outer code can recover the remaining symbols using the decoding algorithm appropriate for that code.

In a combined approach, the relationships between symbols defined by both the inner and outer codes are considered as a single combined set of simultaneous equations which can be solved by the usual means, typically by **Gaussian elimination**.

### 90.3 Computational complexity

Raptor codes require  $O(1)$  time to generate an encoding symbol. Decoding a message of length  $k$  with a belief propagation decoding algorithm requires  $O(k)$  time for the appropriate choice of inner/outer codes.

### 90.4 Legal complexity

Raptor codes are heavily covered with patents in various jurisdictions. For the specific instances specified in RFCs, statements by patent owners may or may not provide some leeway, depending among other things on whether the device carrying the implementation also implements a “wide-area wireless” specification.

### 90.5 See also

- Erasure code
- LT code
- Fountain codes
- Michael Luby
- Tornado codes

### 90.6 Notes

- [1] Amin Shokrollahi (31 January 2011). *The Development of Raptor Codes* (Speech). Invited talk at the Kungliga Tekniska högskolan. Retrieved 24 February 2012.

2. The Open Source Implementation of Raptor Code RFC5053 can be found here : <https://code.google.com/p/raptor-code-rfc/>

### 90.7 References

- Amin Shokrollahi, “Raptor Codes,” IEEE Transactions on Information Theory, vol. 52, pp. 2551-2567, 2006. PDF (requires login)

- **3GPP** The 3rd Generation Partnership Project
- **DVB** Digital Video Broadcasting
- **3GPP TS26.346** 3GPP Technical Specification for Multimedia Broadcast/Multicast Service: Protocols and Codecs.
- **RFC5053** Raptor Forward Error Correction Scheme for Object Delivery
- **DVB-H IP** Datacasting specifications
- **RFC6330** RaptorQ Forward Error Correction Scheme for Object Delivery
- “IPR” Search Result for **RFC 5053**, with statements by some patent owners
- “IPR” Search Result for **RFC 6330**, with statements by some patent owners

# Chapter 91

## Recursive indexing

When number (generally large number) is represented in a finite alphabet set, and it cannot be represented by just one member of the set, **Recursive indexing** is used.

Recursive indexing itself is a method to write the successive differences of the number after extracting the maximum value of the alphabet set from the number, and continuing recursively till the difference falls in the range of the set.

Recursive indexing with a 2-letter alphabet is called **unary code**.

### 91.1 Encoding

To encode a number  $N$ , keep reducing the maximum element of this set ( $S_{\max}$ ) from  $N$  and output  $S_{\max}$  for each such difference, stopping when the number lies in the half closed half open range  $[0 - S_{\max})$ .

#### 91.1.1 Example

Let  $S = [0 \ 1 \ 2 \ 3 \ 4 \ \dots \ 10]$ , be an 11-element set, and we have to recursively index the value  $N=49$ .

According to this method, we need to keep removing 10 from 49, and keep proceeding till we reach a number in the 0–10 range.

So the values are 10 ( $N = 49 - 10 = 39$ ), 10 ( $N = 39 - 10 = 29$ ), 10 ( $N = 29 - 10 = 19$ ), 10 ( $N = 19 - 10 = 9$ ), 9. Hence the recursively indexed sequence for  $N = 49$  with set  $S$ , is 10, 10, 10, 10, 9.

### 91.2 Decoding

Keep adding all the elements of the index, stopping when the index value is between (inclusive of ends) the least and penultimate elements of the set  $S$ .

#### 91.2.1 Example

Continuing from above example we have  $10 + 10 + 10 + 10 + 9 = 49$ .

### 91.3 Uses

This technique is most commonly used in **run-length encoding** systems to encode longer runs than the alphabet sizes permit.

### 91.4 References

- Khalid Sayood, Data Compression 3rd ed, Morgan Kaufmann.

## Chapter 92

# Reed–Muller code

**Reed–Muller codes** are a family of **linear error-correcting codes** used in communications. Reed–Muller codes belong to the classes of **locally testable codes** and **locally decodable codes**, which is why they are useful in the design of **probabilistically checkable proofs** in **computational complexity theory**. They are named after **Irving S. Reed** and **David E. Muller**. Muller discovered the codes, and Reed proposed the majority logic decoding for the first time. Special cases of Reed–Muller codes include the **Walsh–Hadamard code** and the **Reed–Solomon code**.

Reed–Muller codes are listed as  $RM(r, m)$ , where  $r$  is the order of the code,  $0 \leq r \leq m$ , and  $m$  determines the block length  $N = 2^m$ . RM codes are related to binary functions on the field  $GF(2^m)$  over the elements  $\{0, 1\}$ .

$RM(0, m)$  codes are repetition codes of length  $N = 2^m$ , rate  $R = \frac{1}{N}$  and minimum distance  $d_{\min} = N$ .

$RM(1, m)$  codes are parity check codes of length  $N = 2^m$ , rate  $R = \frac{m+1}{N}$  and minimum distance  $d_{\min} = \frac{N}{2}$ .

$RM(m-1, m)$  codes are single parity check codes of length  $N = 2^m$ , rate  $R = \frac{N-1}{N}$  and minimum distance  $d_{\min} = 2$ .

$RM(m-2, m)$  codes are the family of extended Hamming codes of length  $N = 2^m$  with minimum distance  $d_{\min} = 4$ .<sup>[1]</sup>

### 92.1 Construction

A generator matrix for a Reed–Muller code  $RM(r, m)$  of length  $N = 2^m$  can be constructed as follows. Let us write the set of all  $m$ -dimensional binary vectors as:

$$X = \mathbb{F}_2^m = \{x_1, \dots, x_m\}.$$

We define in  $N$ -dimensional space  $\mathbb{F}_2^N$  the **indicator vectors**

$$\mathbb{I}_A \in \mathbb{F}_2^N$$

on subsets  $A \subset X$  by:

$$(\mathbb{I}_A)_i = \begin{cases} 1 & \text{if } x_i \in A \\ 0 & \text{otherwise} \end{cases}$$

together with, also in  $\mathbb{F}_2^N$ , the binary operation

$$w \wedge z = (w_1 \cdot z_1, \dots, w_N \cdot z_N),$$

referred to as the **wedge product** (this wedge product is not to be confused with the **wedge product** defined in exterior algebra). Here,  $w = (w_1, w_2, \dots, w_N)$  and  $z = (z_1, z_2, \dots, z_N)$  are points in  $\mathbb{F}_2^N$  ( $N$ -dimensional binary vectors), and the operation  $\cdot$  is the usual multiplication in the field  $\mathbb{F}_2$ .

$\mathbb{F}_2^m$  is an  $m$ -dimensional vector space over the field  $\mathbb{F}_2$ , so it is possible to write

$$(\mathbb{F}_2)^m = \{(y_m, \dots, y_1) \mid y_i \in \mathbb{F}_2\}$$

We define in  $N$ -dimensional space  $\mathbb{F}_2^N$  the following vectors with length  $N$ :  $v_0 = (1, 1, \dots, 1)$  and

$$v_i = \mathbb{I}_{H_i}$$

where  $1 \leq i \leq m$  and the  $H_i$  are **hyperplanes** in  $(\mathbb{F}_2)^m$  (with dimension  $m-1$ ):

$$H_i = \{y \in (\mathbb{F}_2)^m \mid y_i = 0\}$$

#### 92.1.1 Building a generator matrix

The **Reed–Muller  $RM(r, m)$  code** of order  $r$  and length  $N = 2^m$  is the code generated by  $v_0$  and the wedge products of up to  $r$  of the  $v_i$ ,  $1 \leq i \leq m$  (where by convention a wedge product of fewer than one vector is the identity for the operation). In other words, we can build a generator matrix for the  $RM(r, m)$  code, using vectors and their wedge product permutations up to  $r$  at a time  $v_0, v_1, \dots, v_n, \dots, (v_{i_1} \wedge v_{i_2}), \dots, (v_{i_1} \wedge v_{i_2} \wedge \dots \wedge v_{i_r})$ , as the rows of the generator matrix, where  $1 \leq i_k \leq m$ .

## 92.2 Example 1

Let  $m = 3$ . Then  $N = 8$ , and

$$X = \mathbb{F}_2^3 = \{(0, 0, 0), (0, 0, 1), \dots, (1, 1, 1)\},$$

and

$$v_0 = (1, 1, 1, 1, 1, 1, 1, 1)$$

$$v_1 = (1, 0, 1, 0, 1, 0, 1, 0)$$

$$v_2 = (1, 1, 0, 0, 1, 1, 0, 0)$$

$$v_3 = (1, 1, 1, 1, 0, 0, 0, 0).$$

The RM(1,3) code is generated by the set

$$\{v_0, v_1, v_2, v_3\},$$

or more explicitly by the rows of the matrix:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

## 92.3 Example 2

The RM(2,3) code is generated by the set:

$$\{v_0, v_1, v_2, v_3, v_1 \wedge v_2, v_1 \wedge v_3, v_2 \wedge v_3\}$$

or more explicitly by the rows of the matrix:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

## 92.4 Properties

The following properties hold:

**1** The set of all possible wedge products of up to  $m$  of the  $v_i$  form a basis for  $\mathbb{F}_2^N$ .

**2** The RM  $(r, m)$  code has rank

$$\sum_{s=0}^r \binom{m}{s}.$$

**3**  $\text{RM}(r, m) = \text{RM}(r, m-1) \mid \text{RM}(r-1, m-1)$  where  $\mid$  denotes the **bar product** of two codes.

**4** RM  $(r, m)$  has minimum **Hamming weight**  $2^{m-r}$ .

### 92.4.1 Proof

**1**

There are

$$\sum_{s=0}^m \binom{m}{s} = 2^m = N$$

such vectors and  $\mathbb{F}_2^N$  have dimension  $N$  so it is sufficient to check that the  $N$  vectors span; equivalently it is sufficient to check that  $\text{RM}(m, m) = \mathbb{F}_2^N$ .

Let  $x$  be a binary vector of length  $m$ , an element of  $X$ . Let  $(x)_i$  denote the  $i^{\text{th}}$  element of  $x$ . Define

$$y_i = \begin{cases} v_i & \text{if } (x)_i = 0 \\ v_0 + v_i & \text{if } (x)_i = 1 \end{cases}$$

where  $1 \leq i \leq m$ .

$$\mathbb{I}_{\{x\}} = y_1 \wedge \dots \wedge y_m$$

Expansion via the distributivity of the wedge product gives  $\mathbb{I}_{\{x\}} \in \text{RM}(m, m)$ . Then since the vectors  $\{\mathbb{I}_{\{x\}} \mid x \in X\}$  span  $\mathbb{F}_2^N$  we have  $\text{RM}(m, m) = \mathbb{F}_2^N$ .

**2**

By **1**, all such wedge products must be linearly independent, so the rank of  $\text{RM}(r, m)$  must simply be the number of such vectors.

**3**

Omitted.

**4**

By induction.

The  $RM(0, m)$  code is the repetition code of length  $N = 2^m$  and weight  $N = 2^{m-0} = 2^{m-r}$ . By 1  $RM(m, m) = \mathbb{F}_2^n$  and has weight  $1 = 2^0 = 2^{m-r}$ .

The article [bar product \(coding theory\)](#) gives a proof that the weight of the bar product of two codes  $C_1, C_2$  is given by

$$\min\{2w(C_1), w(C_2)\}$$

If  $0 < r < m$  and if

- i)  $RM(r, m-1)$  has weight  $2^{m-1-r}$
- ii)  $RM(r-1, m-1)$  has weight  $2^{m-1-(r-1)} = 2^{m-r}$

then the bar product has weight

$$\min\{2 \times 2^{m-1-r}, 2^{m-r}\} = 2^{m-r}.$$

## 92.5 Alternative construction

A Reed–Muller code  $RM(r, m)$  exists for any integers  $m \geq 0$  and  $0 \leq r \leq m$ .  $RM(m, m)$  is defined as the universe  $(2^m, 2^m, 1)$  code.  $RM(-1, m)$  is defined as the trivial code  $(2^m, 0, \infty)$ . The remaining RM codes may be constructed from these elementary codes using the length-doubling construction

$$RM(r, m) = \{(\mathbf{u}, \mathbf{u} + \mathbf{v}) \mid \mathbf{u} \in RM(r, m-1), \mathbf{v} \in RM(r-1, m-1)\}.$$

From this construction,  $RM(r, m)$  is a binary linear block code  $(n, k, d)$  with length  $n = 2^m$ , dimension  $k(r, m) = k(r, m-1) + k(r-1, m-1)$  and minimum distance  $d = 2^{m-r}$  for  $r \geq 0$ . The dual code to  $RM(r, m)$  is  $RM(m-r-1, m)$ . This shows that repetition and SPC codes are duals, biorthogonal and extended Hamming codes are duals and that codes with  $k = n/2$  are self-dual.

## 92.6 Construction based on low-degree polynomials over a finite field

There is another, simple way to construct Reed–Muller codes based on low-degree polynomials over a finite field. This construction is particularly suited for their application as locally testable codes and locally decodable codes.<sup>[2]</sup>

Let  $\mathbb{F}$  be a finite field and let  $m$  and  $d$  be positive integers, where  $m$  should be thought of as larger than  $d$ . We are going to encode messages consisting of  $\binom{m+d}{m}$  elements of  $\mathbb{F}$  as codewords of length  $|\mathbb{F}|^m$  as follows: We interpret the message as an  $m$ -variate polynomial  $f$  of degree at most  $d$  with coefficient from  $\mathbb{F}$ . Such a polynomial has  $\binom{m+d}{m}$  coefficients. The Reed–Muller encoding of  $f$  is the list of the evaluations of  $f$  on all  $x \in \mathbb{F}^m$ ; the codeword at the position indexed by  $x \in \mathbb{F}^m$  has value  $f(x)$ .

## 92.7 Table of Reed–Muller codes

The table below lists the  $RM(r, m)$  codes of lengths up to 32 for alphabet of size 2 annotated with standard coding theory notation for block codes. The Reed–Muller code is a  $[2^m, k, 2^{m-r}]_2$ -code, that is, it is a linear code over a binary alphabet, has block length  $2^m$ , message length (or dimension)  $k$ , and minimum distance  $2^{m-r}$ .

## 92.8 Decoding RM codes

$RM(r, m)$  codes can be decoded using majority logic decoding. The basic idea of majority logic decoding is to build several checksums for each received code word element. Since each of the different checksums must all have the same value (i.e. the value of the message word element weight), we can use a majority logic decoding to decipher the value of the message word element. Once each order of the polynomial is decoded, the received word is modified accordingly by removing the corresponding codewords weighted by the decoded message contributions, up to the present stage. So for a  $r$ th order RM code, we have to decode iteratively  $r+1$  times before we arrive at the final received code-word. Also, the values of the message bits are calculated through this scheme; finally we can calculate the codeword by multiplying the message word (just decoded) with the generator matrix.

One clue if the decoding succeeded, is to have an all-zero modified received word, at the end of  $(r+1)$ -stage decoding through the majority logic decoding. This technique was proposed by Irving S. Reed, and is more general when applied to other finite geometry codes.

## 92.9 Notes

- [1] Trellis and Turbo Coding, C. Schlegel & L. Perez, Wiley Interscience, 2004, p149.
- [2] Prahladh Harsha et al., Limits of Approximation Algorithms: PCPs and Unique Games (DIMACS Tutorial Lecture Notes), Section 5.2.1.

## 92.10 References

### Research Articles:

- D. E. Muller. Application of boolean algebra to switching circuit design and to error detection. IRE Transactions on Electronic Computers, 3:6–12, 1954.
- Irving S. Reed. A class of multiple-error-correcting codes and the decoding scheme. Transactions of the IRE Professional Group on Information Theory, 4:38–49, 1954.

### Textbooks:

- Shu Lin; Daniel Costello (2005). *Error Control Coding* (2 ed.). Pearson. ISBN 0-13-017973-6. Chapter 4.
- J.H. van Lint (1992). *Introduction to Coding Theory*. GTM. **86** (2 ed.). Springer-Verlag. ISBN 3-540-54894-7. Chapter 4.5.

## 92.11 External links

- MIT OpenCourseWare, 6.451 Principles of Digital Communication II, Lecture Notes section 6.4
- GPL Matlab-implementation of RM-codes
- Source GPL Matlab-implementation of RM-codes

## Chapter 93

# Reed–Solomon error correction

**Reed–Solomon codes** are a group of **error-correcting codes** that were introduced by **Irving S. Reed** and **Gustave Solomon** in 1960.<sup>[1]</sup> They have many applications, the most prominent of which include consumer technologies such as **CDs**, **DVDs**, **Blu-ray Discs**, **QR Codes**, data transmission technologies such as **DSL** and **WiMAX**, broadcast systems such as **DVB** and **ATSC**, and storage systems such as **RAID 6**. They are also used in satellite communication.

In coding theory, the Reed–Solomon code belongs to the class of non-binary cyclic error-correcting codes. The Reed–Solomon code is based on univariate polynomials over finite fields.

It is able to detect and correct multiple symbol errors. By adding  $t$  check symbols to the data, a Reed–Solomon code can detect any combination of up to  $t$  erroneous symbols, or correct up to  $\lfloor t/2 \rfloor$  symbols. As an **erasure code**, it can correct up to  $t$  known erasures, or it can detect and correct combinations of errors and erasures. Furthermore, Reed–Solomon codes are suitable as multiple-burst bit-error correcting codes, since a sequence of  $b + 1$  consecutive bit errors can affect at most two symbols of size  $b$ . The choice of  $t$  is up to the designer of the code, and may be selected within wide limits.

### 93.1 History

Reed–Solomon codes were developed in 1960 by **Irving S. Reed** and **Gustave Solomon**, who were then staff members of MIT Lincoln Laboratory. Their seminal article was titled “Polynomial Codes over Certain Finite Fields.” (Reed & Solomon 1960). The original encoding scheme described in the Reed Solomon article used a variable polynomial based on the message to be encoded, which made decoding impractical for all but the simplest of cases. This was resolved by changing the encoding scheme to use a fixed polynomial known to both encoder and decoder. A practical decoder developed by **Daniel Gorenstein** and **Neal Zierler** was described in an MIT Lincoln Laboratory report by Zierler in January 1960 and later in a paper in June 1961.<sup>[2]</sup> During the same period, work was also being done on **BCH codes** and Reed–Solomon codes were considered as a special class of BCH

codes at the time. The Gorenstein–Zierler decoder and the related work on BCH codes are described in a book **Error Correcting Codes** by **W. Wesley Peterson** (1961).<sup>[3]</sup>

An improved decoder was developed in 1969 by **Elwyn Berlekamp** and **James Massey**, and is since known as the **Berlekamp–Massey decoding algorithm**. Another improved decoder was developed in 1975 by **Yasuo Sugiyama**, based on the **extended Euclidean algorithm**.<sup>[4]</sup>

In 1977, Reed–Solomon codes were implemented in the **Voyager program** in the form of **concatenated error correction codes**. The first commercial application in mass-produced consumer products appeared in 1982 with the **compact disc**, where two **interleaved Reed–Solomon codes** are used. Today, Reed–Solomon codes are widely implemented in **digital storage devices** and **digital communication standards**, though they are being slowly replaced by more modern **low-density parity-check (LDPC) codes** or **turbo codes**. For example, Reed–Solomon codes are used in the **Digital Video Broadcasting (DVB) standard DVB-S**, but LDPC codes are used in its successor, **DVB-S2**.

### 93.2 Applications

#### 93.2.1 Data storage

Reed–Solomon coding is very widely used in mass storage systems to correct the burst errors associated with media defects.

Reed–Solomon coding is a key component of the **compact disc**. It was the first use of strong error correction coding in a mass-produced consumer product, and **DAT** and **DVD** use similar schemes. In the CD, two layers of Reed–Solomon coding separated by a 28-way **convolutional interleaver** yields a scheme called **Cross-Interleaved Reed–Solomon Coding (CIRC)**. The first element of a CIRC decoder is a relatively weak inner (32,28) Reed–Solomon code, shortened from a (255,251) code with 8-bit symbols. This code can correct up to 2 byte errors per 32-byte block. More importantly, it flags as erasures any uncorrectable blocks, i.e., blocks with more than 2 byte errors. The decoded 28-byte blocks, with era-



sure indications, are then spread by the deinterleaver to different blocks of the (28,24) outer code. Thanks to the deinterleaving, an erased 28-byte block from the inner code becomes a single erased byte in each of 28 outer code blocks. The outer code easily corrects this, since it can handle up to 4 such erasures per block.

The result is a CIRC that can completely correct error bursts up to 4000 bits, or about 2.5 mm on the disc surface. This code is so strong that most CD playback errors are almost certainly caused by tracking errors that cause the laser to jump track, not by uncorrectable error bursts.<sup>[5]</sup>

DVDs use a similar scheme, but with much larger blocks, a (208,192) inner code, and a (182,172) outer code.

Reed–Solomon error correction is also used in *archive* files which are commonly posted accompanying multimedia files on USENET. The Distributed online storage service *Wuala* (discontinued in 2015) also used to make use of Reed–Solomon when breaking up files.

### 93.2.2 Bar code

Almost all two-dimensional bar codes such as *PDF-417*, *MaxiCode*, *Datamatrix*, *QR Code*, and *Aztec Code* use Reed–Solomon error correction to allow correct reading even if a portion of the bar code is damaged. When the bar code scanner cannot recognize a bar code symbol, it will treat it as an erasure.

Reed–Solomon coding is less common in one-dimensional bar codes, but is used by the *PostBar* symbology.

### 93.2.3 Data transmission

Specialized forms of Reed–Solomon codes, specifically *Cauchy-RS* and *Vandermonde-RS*, can be used to overcome the unreliable nature of data transmission over *erasure channels*. The encoding process assumes a code of  $RS(N, K)$  which results in  $N$  codewords of length  $N$  symbols each storing  $K$  symbols of data, being generated, that are then sent over an erasure channel.

Any combination of  $K$  codewords received at the other end is enough to reconstruct all of the  $N$  codewords. The code rate is generally set to 1/2 unless the channel's erasure likelihood can be adequately modelled and is seen to be less. In conclusion,  $N$  is usually  $2K$ , meaning that at least half of all the codewords sent must be received in order to reconstruct all of the codewords sent.

Reed–Solomon codes are also used in xDSL systems and CCSDS's Space Communications Protocol Specifications as a form of forward error correction.

### 93.2.4 Space transmission

One significant application of Reed–Solomon coding was to encode the digital pictures sent back by the *Voyager* space probe.

Voyager introduced Reed–Solomon coding *concatenated* with *convolutional codes*, a practice that has since become very widespread in deep space and satellite (e.g., direct digital broadcasting) communications.

*Viterbi decoders* tend to produce errors in short bursts. Correcting these burst errors is a job best done by short or simplified Reed–Solomon codes.

Modern versions of concatenated Reed–Solomon/Viterbi-decoded convolutional coding were and are used on the *Mars Pathfinder*, *Galileo*, *Mars Exploration Rover* and *Cassini* missions, where they perform within about 1–1.5 dB of the ultimate limit, being the *Shannon capacity*.

These concatenated codes are now being replaced by more powerful *turbo codes*.

## 93.3 Constructions

The Reed–Solomon code is actually a family of codes: For every choice of the three parameters  $k < n \leq q$ , there is a Reed–Solomon code that has an *alphabet* of size  $q$ , a *block length*  $n$ , and a *message length*  $k$ . Moreover, the alphabet is interpreted as the *finite field* of order  $q$ , and thus,  $q$  has to be a prime power. In the most useful parameterizations of the Reed–Solomon code, the block length is usually some constant multiple of the message length, that is, the *rate*  $R = k/n$  is some constant, and furthermore, the block length is equal to or one less than the alphabet size, that is,  $n = q$  or  $n = q - 1$ .

### 93.3.1 Reed & Solomon's original view: The codeword as a sequence of values

There are different encoding procedures for the Reed–Solomon code, and thus, there are different ways to describe the set of all codewords. In the original view of *Reed & Solomon (1960)*, every codeword of the Reed–Solomon code is a sequence of function values of a polynomial of degree less than  $k$ . One issue with this view is that decoding and checking for errors is not practical except for the simplest of cases. In order to obtain a codeword of the Reed–Solomon code, the message is interpreted as the description of a polynomial  $p$  of degree less than  $k$  over the finite field  $F$  with  $q$  elements. In turn, the polynomial  $p$  is evaluated at  $n$  distinct points  $a_1, \dots, a_n$  of the field  $F$ , and the sequence of values is the corresponding codeword. Formally, the set  $C$  of codewords of the Reed–Solomon code is defined as follows:

$$\mathbf{C} = \left\{ (p(a_1), p(a_2), \dots, p(a_n)) \mid p \text{ over polynomial of degree } \leq k \right\}$$

Since any two *distinct* polynomials of degree less than  $k$  agree in at most  $k - 1$  points, this means that any two codewords of the Reed–Solomon code disagree in at least  $n - (k - 1) = n - k + 1$  positions. Furthermore, there are two polynomials that do agree in  $k - 1$  points but are not equal, and thus, the **distance** of the Reed–Solomon code is exactly  $d = n - k + 1$ . Then the relative distance is  $\delta = d/n = 1 - k/n + 1/n \sim 1 - R$ , where  $R = k/n$  is the rate. This trade-off between the relative distance and the rate is asymptotically optimal since, by the **Singleton bound**, every code satisfies  $\delta + R \leq 1$ . Being a code that achieves this optimal trade-off, the Reed–Solomon code belongs to the class of **maximum distance separable codes**.

While the number of different polynomials of degree less than  $k$  and the number of different messages are both equal to  $q^k$ , and thus every message can be uniquely mapped to such a polynomial, there are different ways of doing this encoding. The original construction of **Reed & Solomon (1960)** interprets the message  $x$  as the *coefficients* of the polynomial  $p$ , whereas subsequent constructions interpret the message as the *values* of the polynomial at the first  $k$  points  $a_1, \dots, a_k$  and obtain the polynomial  $p$  by interpolating these values with a polynomial of degree less than  $k$ . The latter encoding procedure, while being slightly less efficient, has the advantage that it gives rise to a **systematic code**, that is, the original message is always contained as a subsequence of the codeword.

In many contexts it is convenient to choose the sequence  $a_1, \dots, a_n$  of evaluation points so that they exhibit some additional structure. In particular, it is useful to choose the sequence of successive powers of a **primitive root**  $\alpha$  of the field  $F$ , that is,  $\alpha$  is generator of the finite field's **multiplicative group** and the sequence is defined as  $a_i = \alpha^i$  for all  $i = 1, \dots, q - 1$ . This sequence contains all elements of  $F$  except for 0, so in this setting, the block length is  $n = q - 1$ . Then it follows that, whenever  $p(a)$  is a polynomial over  $F$ , then the function  $p(\alpha a)$  is also a polynomial of the same degree, which gives rise to a codeword that is a **cyclic left-shift** of the codeword derived from  $p(a)$ ; thus, this construction of Reed–Solomon codes gives rise to a **cyclic code**.

#### Simple encoding procedure: The message as a sequence of coefficients

In the original construction of **Reed & Solomon (1960)**, the message  $x = (x_1, \dots, x_k) \in F^k$  is mapped to the polynomial  $p_x$  with

$$p_x(a) = \sum_{i=1}^k x_i a^{i-1}.$$

As described above, the codeword is then obtained by evaluating  $p$  at  $n$  different points  $a_1, \dots, a_n$  of the field  $F$ . Thus the classical encoding function  $C : F^k \rightarrow F^n$  for the Reed–Solomon code is defined as follows:

$$C(x) = (p_x(a_1), \dots, p_x(a_n)).$$

This function  $C$  is a **linear mapping**, that is, it satisfies  $C(x) = x \cdot A$  for the following  $(k \times n)$ -matrix  $A$  with elements from  $F$ :

$$A = \begin{bmatrix} 1 & \dots & 1 \\ a_1 & \dots & a_n \\ a_1^2 & \dots & a_n^2 \\ \vdots & \ddots & \vdots \\ a_1^{k-1} & \dots & a_n^{k-1} \end{bmatrix}$$

This matrix is the transpose of a **Vandermonde matrix** over  $F$ . In other words, the Reed–Solomon code is a **linear code**, and in the classical encoding procedure, its **generator matrix** is  $A$ .

#### Systematic encoding procedure: The message as an initial sequence of values

As mentioned above, there is an alternative way to map codewords  $x$  to polynomials  $p_x$ . In this alternative encoding procedure, the polynomial  $p_x$  is the unique polynomial of degree less than  $k$  such that

$$p_x(a_i) = x_i \text{ holds for all } i = 1, \dots, k.$$

To compute this polynomial  $p_x$  from  $x$ , one can use **Lagrange interpolation**. Once it has been found, it is evaluated at the other points  $a_{k+1}, \dots, a_n$  of the field. The alternative encoding function  $C : F^k \rightarrow F^n$  for the Reed–Solomon code is then again just the sequence of values:

$$C(x) = (p_x(a_1), \dots, p_x(a_n)).$$

This time, however, the first  $k$  entries of the codeword are exactly equal to  $x$ , so this encoding procedure gives rise to a **systematic code**. It can be checked that the alternative encoding function is a linear mapping as well.

#### Theoretical decoding procedure

**Reed & Solomon (1960)** described a theoretical decoder that corrected errors by finding the most popular message polynomial. The decoder only knows the set of values  $a_1$  to  $a_n$  and which encoding method was used to generate the codeword's sequence of values. The original message,

the polynomial, and any errors are unknown. A decoding procedure could use a method like Lagrange interpolation on various subsets of  $n$  codeword values taken  $k$  at a time to repeatedly produce potential polynomials, until a sufficient number of matching polynomials are produced to reasonably eliminate any errors in the received codeword. Once a polynomial is determined, then any errors in the codeword can be corrected, by recalculating the corresponding codeword values. Unfortunately, in all but the simplest of cases, there are too many subsets, so the algorithm is impractical. The number of subsets is the **binomial coefficient**,  $\binom{n}{k} = \frac{n!}{(n-k)!k!}$ , and the number of subsets is infeasible for even modest codes. For a (255, 249) code that can correct 3 errors, the naive theoretical decoder would examine 359 billion subsets. Practical decoding involved changing the view of codewords to be a sequence of coefficients as explained in the next section.

### 93.3.2 The BCH view: The codeword as a sequence of coefficients

In this view, the sender again maps the message  $x$  to a polynomial  $p_x$ , and for this, any of the two mappings above can be used (where the message is either interpreted as the coefficients of  $p_x$  or as the initial sequence of values of  $p_x$ ). Once the sender has constructed the polynomial  $p_x$  in some way, however, instead of sending the *values* of  $p_x$  at all points, the sender computes some related polynomial  $s$  of degree at most  $n - 1$  for  $n = q - 1$  and sends the  $n$  *coefficients* of that polynomial. The polynomial  $s(a)$  is constructed by multiplying the message polynomial  $p_x(a)$ , which has degree at most  $k - 1$ , with a **generator polynomial**  $g(a)$  of degree  $n - k$  that is known to both the sender and the receiver. The generator polynomial  $g(x)$  is defined as the polynomial whose roots are exactly  $\alpha, \alpha^2, \dots, \alpha^{n-k}$ , i.e.,

$$g(x) = (x - \alpha)(x - \alpha^2) \cdots (x - \alpha^{n-k}) = g_0 + g_1x + \cdots + g_{n-k}x^{n-k}$$

The transmitter sends the  $n = q - 1$  coefficients of  $s(a) = p_x(a) \cdot g(a)$ . Thus, in the BCH view of Reed Solomon codes, the set  $\mathbf{C}'$  of codewords is defined for  $n = q - 1$  as follows:<sup>[6]</sup>

$$\mathbf{C}' = \left\{ (s_1, s_2, \dots, s_n) \left| s(a) = \sum_{i=1}^n s_i a^{i-1} \text{ roots the least at } \alpha^1, \alpha^2, \dots, \alpha^{n-k} \right. \right\}.$$

With this definition of the codewords, it can be immediately seen that a Reed–Solomon code is a **polynomial code**, and in particular a **BCH code**. The generator polynomial  $g(a)$  is the minimal polynomial with roots  $\alpha, \alpha^2, \dots, \alpha^{n-k}$  as defined above, and the codewords are exactly the polynomials that are divisible by  $g(a)$ .

Since Reed–Solomon codes are a special case of **BCH codes**, the practical decoders designed for BCH codes are

applicable to Reed–Solomon codes: The receiver interprets the received word as the coefficients of a polynomial  $r(a)$ . If no error has occurred during the transmission, that is, if  $r(a) = s(a)$ , then the receiver can use **polynomial division** to determine the message polynomial  $p_x(a) = r(a)/g(a)$ . In general, the receiver can use polynomial division to construct the unique polynomials  $p(a)$  and  $e(a)$ , such that  $e(a)$  has degree less than the degree of  $g(a)$  and

$$r(a) = p(a) \cdot g(a) + e(a).$$

If the remainder polynomial  $e(a)$  is not identically zero, then an error has occurred during the transmission. The receiver can evaluate  $r(a)$  at the roots of  $g(a)$  and build a system of equations that eliminates  $s(a)$  and identifies which coefficients of  $r(a)$  are in error, and the magnitude of each coefficient's error. If the system of equations can be solved, then the receiver knows how to modify the received word  $r(a)$  to get the most likely codeword  $s(a)$  that was sent.

#### Systematic encoding procedure

The above encoding procedure for the BCH view of Reed–Solomon codes is classical, but does not give rise to a **systematic encoding procedure**, i.e., the codewords do not necessarily contain the message as a subsequence. To remedy this fact, instead of sending  $s(x) = p(x)g(x)$ , the encoder constructs the transmitted polynomial  $s(x)$  such that the coefficients of the  $k$  largest monomials are equal to the corresponding coefficients of  $p(x)$ , and the lower-order coefficients of  $s(x)$  are chosen exactly in such a way that  $s(x)$  becomes divisible by  $g(x)$ . Then the coefficients of  $p(x)$  are a subsequence of the coefficients of  $s(x)$ . To get a code that is overall systematic, we construct the message polynomial  $p(x)$  by interpreting the message as the sequence of its coefficients.

Formally, the construction is done by multiplying  $p(x)$  by  $x^t$  to make room for the  $t = n - k$  check symbols, dividing that product by  $g(x)$  to find the remainder, and then compensating for that remainder by subtracting it. The  $t$  check symbols are created by computing the remainder  $s_r(x)$ :

Note that the remainder has degree at most  $t - 1$ , whereas the coefficients of  $x^{t-1}, x^{t-2}, \dots, x^1, x^0$  in the polynomial  $p(x) \cdot x^t$  are zero. Therefore, the following definition of the codeword  $s(x)$  has the property that the first  $k$  coefficients are identical to the coefficients of  $p(x)$ :

$$s(x) = p(x) \cdot x^t - s_r(x).$$

As a result, the codewords  $s(x)$  are indeed elements of  $\mathbf{C}'$ , that is, they are divisible by the generator polynomial  $g(x)$ :<sup>[7]</sup>

$$s(x) \equiv p(x) \cdot x^t - s_r(x) \equiv s_r(x) - s_r(x) \equiv 0 \pmod{g(x)}.$$

### 93.3.3 Duality of the two views - discrete Fourier transform

At first sight, the two views of Reed–Solomon codes above seem very different. In the first definition, codewords in the set  $\mathbf{C}$  are *values* of polynomials, whereas in the second set  $\mathbf{C}'$ , they are *coefficients*. Moreover, the generator polynomials in the first definition are of degree less than  $k$ , are variable, and unknown to the decoder, whereas those in the second definition are of degree  $n - k$ , are required to have specific roots, and are known to both encoder and decoder. Although the codewords as produced by the above encoder schemes are not the same, there is a duality between the coefficients of polynomials and their values that would allow the same codeword to be considered as a set of coefficients or a set of values.

The equivalence of the two definitions can be proved using the **discrete Fourier transform**. This transform, which exists in all finite fields as well as the complex numbers, establishes a duality between the coefficients of polynomials and their values. This duality can be approximately summarized as follows: Let  $p(x)$  and  $q(x)$  be two polynomials of degree less than  $n$ . If the *values* of  $p(x)$  are the *coefficients* of  $q(x)$ , then (up to a scalar factor and re-ordering), the *values* of  $q(x)$  are the *coefficients* of  $p(x)$ . For this to make sense, the values must be taken at locations  $x = \alpha^i$ , for  $i = 0, \dots, n - 1$ , where  $\alpha$  is a primitive  $n$ th root of unity.

To be more precise, let

$$p(x) = v_0 + v_1x + v_2x^2 + \dots + v_{n-1}x^{n-1},$$

$$q(x) = f_0 + f_1x + f_2x^2 + \dots + f_{n-1}x^{n-1}$$

and assume  $p(x)$  and  $q(x)$  are related by the discrete Fourier transform. Then the coefficients and values of  $p(x)$  and  $q(x)$  are related as follows: for all  $i = 0, \dots, n - 1$ ,  $f_i = p(\alpha^i)$  and  $v_i = \frac{1}{n}q(\alpha^{n-i})$  or in the case of a binary field where addition is effectively xor,  $v_i = q(\alpha^{n-i})$ .

Using these facts, we have:  $(f_0, \dots, f_{n-1})$  is a code word of the Reed–Solomon code according to the first definition

- if and only if  $p(x)$  is of degree less than  $k$  (because  $f_0, \dots, f_{n-1}$  are the values of  $p(x)$ ),
- if and only if  $v_i = 0$  for  $i = k, \dots, n - 1$ ,

- if and only if  $q(\alpha^i) = 0$  for  $i = 1, \dots, n - k$  (because  $q(\alpha^i) = nv_{n-i}$  or for binary field  $q(\alpha^i) = v_{n-i}$ ),
- if and only if  $(f_0, \dots, f_{n-1})$  is a code word of the Reed–Solomon code according to the second definition.

This shows that the two definitions are equivalent. However, the practical decoders described below require a generator polynomial known to the decoder, and view a codeword as a set of coefficients.

### 93.3.4 Remarks

Designers are not required to use the “natural” sizes of Reed–Solomon code blocks. A technique known as “shortening” can produce a smaller code of any desired size from a larger code. For example, the widely used (255,223) code can be converted to a (160,128) code by padding the unused portion of the source block with 95 binary zeroes and not transmitting them. At the decoder, the same portion of the block is loaded locally with binary zeroes. The Delsarte-Goethals-Seidel<sup>[8]</sup> theorem illustrates an example of an application of shortened Reed–Solomon codes. In parallel to shortening, a technique known as **puncturing** allows omitting some of the encoded parity symbols.

## 93.4 Properties

The Reed–Solomon code is a  $[n, k, n - k + 1]$  code; in other words, it is a **linear block code** of length  $n$  (over  $F$ ) with **dimension**  $k$  and minimum **Hamming distance**  $n - k + 1$ . The Reed–Solomon code is optimal in the sense that the minimum distance has the maximum value possible for a linear code of size  $(n, k)$ ; this is known as the **Singleton bound**. Such a code is also called a **maximum distance separable (MDS) code**.

The error-correcting ability of a Reed–Solomon code is determined by its minimum distance, or equivalently, by  $n - k$ , the measure of redundancy in the block. If the locations of the error symbols are not known in advance, then a Reed–Solomon code can correct up to  $(n - k)/2$  erroneous symbols, i.e., it can correct half as many errors as there are redundant symbols added to the block. Sometimes error locations are known in advance (e.g., “side information” in **demodulator signal-to-noise ratios**)—these are called **erasures**. A Reed–Solomon code (like any **MDS code**) is able to correct twice as many erasures as errors, and any combination of errors and erasures can be corrected as long as the relation  $2E + S \leq n - k$  is satisfied, where  $E$  is the number of errors and  $S$  is the number of erasures in the block.

For practical uses of Reed–Solomon codes, it is common to use a finite field  $F$  with  $2^m$  elements. In this case,



each symbol can be represented as an  $m$ -bit value. The sender sends the data points as encoded blocks, and the number of symbols in the encoded block is  $n = 2^m - 1$ . Thus a Reed–Solomon code operating on 8-bit symbols has  $n = 2^8 - 1 = 255$  symbols per block. (This is a very popular value because of the prevalence of **byte-oriented** computer systems.) The number  $k$ , with  $k < n$ , of *data* symbols in the block is a design parameter. A commonly used code encodes  $k = 223$  eight-bit data symbols plus 32 eight-bit parity symbols in an  $n = 255$ -symbol block; this is denoted as a  $(n, k) = (255, 223)$  code, and is capable of correcting up to 16 symbol errors per block.

The Reed–Solomon code properties discussed above make them especially well-suited to applications where errors occur in **bursts**. This is because it does not matter to the code how many bits in a symbol are in error — if multiple bits in a symbol are corrupted it only counts as a single error. Conversely, if a data stream is not characterized by error bursts or drop-outs but by random single bit errors, a Reed–Solomon code is usually a poor choice compared to a binary code.

The Reed–Solomon code, like the **convolutional code**, is a transparent code. This means that if the channel symbols have been **inverted** somewhere along the line, the decoders will still operate. The result will be the inversion of the original data. However, the Reed–Solomon code loses its transparency when the code is shortened. The “missing” bits in a shortened code need to be filled by either zeros or ones, depending on whether the data is complemented or not. (To put it another way, if the symbols are inverted, then the zero-fill needs to be inverted to a one-fill.) For this reason it is mandatory that the sense of the data (i.e., true or complemented) be resolved before Reed–Solomon decoding.

## 93.5 Error correction algorithms

The decoders described below use the **BCH view** of the codeword as **sequence of coefficients**.

### 93.5.1 Peterson–Gorenstein–Zierler decoder

Main article: **Peterson–Gorenstein–Zierler algorithm**

Daniel Gorenstein and Neal Zierler developed a practical decoder that was described in a MIT Lincoln Laboratory report by Zierler in January 1960 and later in a paper in June 1961.<sup>[9]</sup> The Gorenstein–Zierler decoder and the related work on BCH codes are described in a book *Error Correcting Codes* by **W. Wesley Peterson** (1961).<sup>[10]</sup>

### Syndrome decoding

The transmitted message is viewed as the coefficients of a polynomial  $s(x)$  that is divisible by a generator polynomial  $g(x)$ .

$$s(x) = \sum_{i=0}^{n-1} c_i x^i$$

$$g(x) = \prod_{j=1}^{n-k} (x - \alpha^j),$$

where  $\alpha$  is a primitive root.

Since  $s(x)$  is divisible by generator  $g(x)$ , it follows that

$$s(\alpha^i) = 0, \quad i = 1, 2, \dots, n - k$$

The transmitted polynomial is corrupted in transit by an error polynomial  $e(x)$  to produce the received polynomial  $r(x)$ .

$$r(x) = s(x) + e(x)$$

$$e(x) = \sum_{i=0}^{n-1} e_i x^i$$

where  $e_i$  is the coefficient for the  $i$ -th power of  $x$ . Coefficient  $e_i$  will be zero if there is no error at that power of  $x$  and nonzero if there is an error. If there are  $\nu$  errors at distinct powers  $ik$  of  $x$ , then

$$e(x) = \sum_{k=1}^{\nu} e_{i_k} x^{i_k}$$

The goal of the decoder is to find the number of errors ( $\nu$ ), the positions of the errors ( $ik$ ), and the error values at those positions ( $e_{ik}$ ). From those,  $e(x)$  can be calculated and subtracted from  $r(x)$  to get the original message  $s(x)$ .

The syndromes  $S_j$  are defined as

$$S_j = r(\alpha^j) = s(\alpha^j) + e(\alpha^j) = 0 + e(\alpha^j) = e(\alpha^j), \quad j = 1, 2, \dots, n - k$$

$$= \sum_{k=1}^{\nu} e_{i_k} (\alpha^j)^{i_k}$$

The advantage of looking at the syndromes is that the message polynomial drops out. Another possible way of calculating  $e(x)$  is using **polynomial interpolation** to find the only polynomial that passes through the points  $(\alpha^j, S_j)$  (Because  $S_j = e(\alpha^j)$ ), however, this is not used widely because polynomial interpolation is not always feasible in the fields used in Reed–Solomon error correction. For example, it is feasible over the integers (of course), but it is infeasible over the integers modulo a prime.

### Error locators and error values

For convenience, define the **error locators**  $X_k$  and **error values**  $Y_k$  as:

$$X_k = \alpha^{i_k}, Y_k = e_{i_k}$$

Then the syndromes can be written in terms of the error locators and error values as

$$S_j = \sum_{k=1}^{\nu} Y_k X_k^j$$

The syndromes give a system of  $n - k \geq 2\nu$  equations in  $2\nu$  unknowns, but that system of equations is nonlinear in the  $X_k$  and does not have an obvious solution. However, if the  $X_k$  were known (see below), then the syndrome equations provide a linear system of equations that can easily be solved for the  $Y_k$  error values.

$$\begin{bmatrix} X_1^1 & X_2^1 & \cdots & X_{\nu}^1 \\ X_1^2 & X_2^2 & \cdots & X_{\nu}^2 \\ \vdots & \vdots & & \vdots \\ X_1^{n-k} & X_2^{n-k} & \cdots & X_{\nu}^{n-k} \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_{\nu} \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{n-k} \end{bmatrix}$$

Consequently, the problem is finding the  $X_k$ , because then the leftmost matrix would be known, and both sides of the equation could be multiplied by its inverse, yielding  $Y_k$

### Error locator polynomial

There is a linear recurrence relation that gives rise to a system of linear equations. Solving those equations identifies the error locations.

Define the **error locator polynomial**  $\Lambda(x)$  as

$$\Lambda(x) = \prod_{k=1}^{\nu} (1 - xX_k) = 1 + \Lambda_1 x^1 + \Lambda_2 x^2 + \cdots + \Lambda_{\nu} x^{\nu}$$

The zeros of  $\Lambda(x)$  are the reciprocals  $X_k^{-1}$ :

$$\Lambda(X_k^{-1}) = 0$$

$$\Lambda(X_k^{-1}) = 1 + \Lambda_1 X_k^{-1} + \Lambda_2 X_k^{-2} + \cdots + \Lambda_{\nu} X_k^{-\nu} = 0$$

Multiply both sides by  $Y_k X_k^{j+\nu}$  and it will still be zero.  $j$  is any number such that  $1 \leq j \leq \nu$ .

$$Y_k X_k^{j+\nu} \Lambda(X_k^{-1}) = 0.$$

$$\text{Hence } Y_k X_k^{j+\nu} + \Lambda_1 Y_k X_k^{j+\nu} X_k^{-1} + \Lambda_2 Y_k X_k^{j+\nu} X_k^{-2} + \cdots + \Lambda_{\nu} Y_k X_k^{j+\nu} X_k^{-\nu} = 0$$

$$\text{so and } Y_k X_k^{j+\nu} + \Lambda_1 Y_k X_k^{j+\nu-1} + \Lambda_2 Y_k X_k^{j+\nu-2} + \cdots + \Lambda_{\nu} Y_k X_k^j = 0$$

Sum for  $k = 1$  to  $\nu$

$$\sum_{k=1}^{\nu} (Y_k X_k^{j+\nu} + \Lambda_1 Y_k X_k^{j+\nu-1} + \Lambda_2 Y_k X_k^{j+\nu-2} + \cdots + \Lambda_{\nu} Y_k X_k^j) = 0$$

$$\sum_{k=1}^{\nu} (Y_k X_k^{j+\nu}) + \Lambda_1 \sum_{k=1}^{\nu} (Y_k X_k^{j+\nu-1}) + \Lambda_2 \sum_{k=1}^{\nu} (Y_k X_k^{j+\nu-2}) + \cdots + \Lambda_{\nu} \sum_{k=1}^{\nu} (Y_k X_k^j) = 0$$

This reduces to

$$S_{j+\nu} + \Lambda_1 S_{j+\nu-1} + \cdots + \Lambda_{\nu-1} S_{j+1} + \Lambda_{\nu} S_j = 0$$

$$S_j \Lambda_{\nu} + S_{j+1} \Lambda_{\nu-1} + \cdots + S_{j+\nu-1} \Lambda_1 = -S_{j+\nu}$$

This yields a system of linear equations that can be solved for the coefficients  $\Lambda_i$  of the error location polynomial:

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_{\nu} \\ S_2 & S_3 & \cdots & S_{\nu+1} \\ \vdots & \vdots & & \vdots \\ S_{\nu} & S_{\nu+1} & \cdots & S_{2\nu-1} \end{bmatrix} \begin{bmatrix} \Lambda_{\nu} \\ \Lambda_{\nu-1} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{\nu+1} \\ -S_{\nu+2} \\ \vdots \\ -S_{2\nu+\nu} \end{bmatrix}$$

The above assumes the decoder knows the number of errors  $\nu$ , but that number has not been determined yet. The PGZ decoder does not determine  $\nu$  directly but rather searches for it by trying successive values. The decoder first assumes the largest value for a trial  $\nu$  and sets up the linear system for that value. If the equations can be solved (i.e., the matrix determinant is nonzero), then that trial value is the number of errors. If the linear system cannot be solved, then the trial  $\nu$  is reduced by one and the next smaller system is examined. (Gill n.d., p. 35)

### Obtain the error locators from the error locator polynomial

Use the coefficients  $\Lambda_i$  found in the last step to build the error location polynomial. The roots of the error location polynomial can be found by exhaustive search. The error locators are the reciprocals of those roots. **Chien search** is an efficient implementation of this step.

### Calculate the error locations

Calculate  $i_k$  by taking the log base  $a$  of  $X_k$ . This is generally done using a precomputed lookup table.

### Calculate the error values

Once the error locators are known, the error values can be determined. This can be done by direct solution for  $Y_k$  in the error equations given above, or using the **Forney**

algorithm

**Fix the errors**

Finally,  $e(x)$  is generated from  $i_k$  and  $e_{ik}$  and then is subtracted from  $r(x)$  to get the sent message  $s(x)$ .

**93.5.2 Berlekamp–Massey decoder**

The **Berlekamp–Massey algorithm** is an alternate iterative procedure for finding the error locator polynomial. During each iteration, it calculates a discrepancy based on a current instance of  $\Lambda(x)$  with an assumed number of errors  $e$ :

$$\Delta = S_i + \Lambda_1 S_{i-1} + \cdots + \Lambda_e S_{i-e}$$

and then adjusts  $\Lambda(x)$  and  $e$  so that a recalculated  $\Delta$  would be zero. The article **Berlekamp–Massey algorithm** has a detailed description of the procedure. In the following example,  $C(x)$  is used to represent  $\Lambda(x)$ .

**Example**

Consider the Reed–Solomon code defined in  $GF(929)$  with  $\alpha = 3$  and  $t = 4$  (this is used in **PDF417** barcodes). The generator polynomial is

$$g(x) = (x-3)(x-3^2)(x-3^3)(x-3^4) = x^4 + 809x^3 + 723x^2 + 568x + 520(x)x^t + \Omega(x)$$

If the message polynomial is  $p(x) = 3x^2 + 2x + 1$ , then the codeword is calculated as follows.

$$s_r(x) = p(x)x^t \mod g(x) = 547x^3 + 738x^2 + 442x + 455$$

$$s(x) = p(x)x^t - s_r(x) = 3x^6 + 2x^5 + 1x^4 + 382x^3 + 191x^2 + 487x + 454$$

Errors in transmission might cause this to be received instead.

$$r(x) = s(x) + e(x) = 3x^6 + 2x^5 + 123x^4 + 456x^3 + 191x^2 + 487x + 474$$

The syndromes are calculated by evaluating  $r$  at powers of  $\alpha$ .

$$S_1 = r(3^1) = 3 \cdot 3^6 + 2 \cdot 3^5 + 123 \cdot 3^4 + 456 \cdot 3^3 + 191 \cdot 3^2 + 487 \cdot 3 + 474 = 733$$

$$S_2 = r(3^2) = 637, S_3 = r(3^3) = 762, S_4 = r(3^4) = 925$$

To correct the errors, first use the **Berlekamp–Massey algorithm** to calculate the error locator polynomial.

The final value of  $C$  is the error locator polynomial,  $\Lambda(x)$ . The zeros can be found by trial substitution. They are  $x_1 = 757 = 3^{-3}$  and  $x_2 = 562 = 3^{-4}$ , corresponding to the

error locations. To calculate the error values, apply the **Forney algorithm**.

$$\Omega(x) = S(x)\Lambda(x) \mod x^4 = 546x + 732$$

$$\Lambda'(x) = 658x + 821$$

$$e_1 = -\Omega(x_1)/\Lambda'(x_1) = -649/54 = 280 \times 843 = 74$$

$$e_2 = -\Omega(x_2)/\Lambda'(x_2) = 122$$

Subtracting  $e_1x^3$  and  $e_2x^4$  from the received polynomial  $r$  reproduces the original codeword  $s$ .

**93.5.3 Euclidean decoder**

Another iterative method for calculating both the error locator polynomial and the error value polynomial is based on Sugiyama's adaptation of the **Extended Euclidean algorithm**.

Define  $S(x)$ ,  $\Lambda(x)$ , and  $\Omega(x)$  for  $t$  syndromes and  $e$  errors:

$$S(x) = S_t x^{t-1} + S_{t-1} x^{t-2} + \cdots + S_2 x + S_1$$

$$\Lambda(x) = \Lambda_e x^e + \Lambda_{e-1} x^{e-1} + \cdots + \Lambda_1 x + 1$$

$$\Omega(x) = \Omega_e x^e + \Omega_{e-1} x^{e-1} + \cdots + \Omega_1 x + \Omega_0$$

The key equation is:

For  $t = 6$  and  $e = 3$ :

$$\begin{bmatrix} \Lambda_3 S_6 & x^8 \\ \Lambda_2 S_6 + \Lambda_3 S_5 & x^7 \\ \Lambda_1 S_6 + \Lambda_2 S_5 + \Lambda_3 S_4 & x^6 \\ S_6 + \Lambda_1 S_5 + \Lambda_2 S_4 + \Lambda_3 S_3 & x^5 \\ S_5 + \Lambda_1 S_4 + \Lambda_2 S_3 + \Lambda_3 S_2 & x^4 \\ S_4 + \Lambda_1 S_3 + \Lambda_2 S_2 + \Lambda_3 S_1 & x^3 \\ S_3 + \Lambda_1 S_2 + \Lambda_2 S_1 & x^2 \\ S_2 + \Lambda_1 S_1 & x \\ S_1 & 1 \end{bmatrix} = \begin{bmatrix} Q_2 x^8 \\ Q_1 x^7 \\ Q_0 x^6 \\ 0 \\ 0 \\ 0 \\ \Omega_2 x^2 \\ \Omega_1 x \\ \Omega_0 \end{bmatrix}$$

The middle terms are zero due to the relationship between  $\Lambda$  and syndromes.

The extended Euclidean algorithm can find a series of polynomials of the form

$$A_i(x) S(x) + B_i(x) x^t = R_i(x)$$

where the degree of  $R$  decreases as  $i$  increases. Once the degree of  $R_i(x) < t/2$ , then

$$A_i(x) = \Lambda(x)$$

$$B_i(x) = -Q(x)$$

$$R_i(x) = \Omega(x).$$

$B(x)$  and  $Q(x)$  don't need to be saved, so the algorithm becomes:



$R_{-1} = x^t$   
 $R_0 = S(x)$   
 $A_{-1} = 0$   
 $A_0 = 1$   
 $i = 0$   
 while degree of  $R_i \geq t/2$   
      $i = i + 1$   
      $Q = R_{i-2} / R_{i-1}$   
      $R_i = R_{i-2} - Q R_{i-1}$   
      $A_i = A_{i-2} - Q A_{i-1}$

to set low order term of  $\Lambda(x)$  to 1, divide  $\Lambda(x)$  and  $\Omega(x)$  by  $A_i(0)$ :

$$\Lambda(x) = A_i / A_i(0)$$

$$\Omega(x) = R_i / A_i(0)$$

$A_i(0)$  is the constant (low order) term of  $A_i$ .

### Example

Using the same data as the Berlekamp Massey example above:

$$\Lambda(x) = A_2 / 544 = 329 x^2 + 821 x + 001$$

$$\Omega(x) = R_2 / 544 = 546 x + 732$$

### 93.5.4 Decoder using discrete Fourier transform

A discrete Fourier transform can be used for decoding.<sup>[11]</sup> To avoid conflict with syndrome names, let  $c(x) = s(x)$  the encoded codeword.  $r(x)$  and  $e(x)$  are the same as above. Define  $C(x)$ ,  $E(x)$ , and  $R(x)$  as the discrete Fourier transforms of  $c(x)$ ,  $e(x)$ , and  $r(x)$ . Since  $r(x) = c(x) + e(x)$ , and since a discrete Fourier transform is a linear operator,  $R(x) = C(x) + E(x)$ .

Transform  $r(x)$  to  $R(x)$  using discrete Fourier transform. Since the calculation for a discrete Fourier transform is the same as the calculation for syndromes,  $t$  coefficients of  $R(x)$  and  $E(x)$  are the same as the syndromes:

$$R_j = E_j = S_j = r(\alpha^j)$$

for  $1 \leq j \leq t$

Use  $R_1$  through  $R_t$  as syndromes (they're the same) and generate the error locator polynomial using the methods from any of the above decoders.

Let  $v$  = number of errors. Generate  $E(x)$  using the known coefficients  $E_1$  to  $E_t$ , the error locator polynomial, and these formulas

$$E_0 = -\frac{1}{\sigma_v}(E_v + \sigma_1 E_{v-1} + \cdots + \sigma_{v-1} E_1)$$

$$E_j = -(\sigma_1 E_{j-1} + \sigma_2 E_{j-2} + \cdots + \sigma_v E_{j-v})$$

for  $t < j < n$

Then calculate  $C(x) = R(x) - E(x)$  and take the inverse transform of  $C(x)$  to produce  $c(x)$ .

### 93.5.5 Decoding beyond the error-correction bound

The **Singleton bound** states that the minimum distance  $d$  of a linear block code of size  $(n, k)$  is upper-bounded by  $n - k + 1$ . The distance  $d$  was usually understood to limit the error-correction capability to  $\lfloor d/2 \rfloor$ . The Reed–Solomon code achieves this bound with equality, and can thus correct up to  $\lfloor (n - k + 1)/2 \rfloor$  errors. However, this error-correction bound is not exact.

In 1999, **Madhu Sudan** and **Venkatesan Guruswami** at MIT published “Improved Decoding of Reed–Solomon and Algebraic-Geometry Codes” introducing an algorithm that allowed for the correction of errors beyond half the minimum distance of the code.<sup>[12]</sup> It applies to Reed–Solomon codes and more generally to **algebraic geometric codes**. This algorithm produces a list of codewords (it is a **list-decoding** algorithm) and is based on interpolation and factorization of polynomials over  $GF(2^m)$  and its extensions.

### 93.5.6 Soft-decoding

The algebraic decoding methods described above are hard-decision methods, which means that for every symbol a hard decision is made about its value. For example, a decoder could associate with each symbol an additional value corresponding to the channel **demodulator's** confidence in the correctness of the symbol. The advent of **LDPC** and **turbo codes**, which employ iterated **soft-decision** belief propagation decoding methods to achieve error-correction performance close to the **theoretical limit**, has spurred interest in applying soft-decision decoding to conventional algebraic codes. In 2003, Ralf Koetter and **Alexander Vardy** presented a polynomial-time soft-decision algebraic list-decoding algorithm for Reed–Solomon codes, which was based upon the work by Sudan and Guruswami.<sup>[13]</sup>

### 93.5.7 Matlab Example

#### Encoder

Here we present a simple Matlab implementation for an encoder.

```
function [ encoded ] = rsEncoder( msg, m, prim_poly,
n, k ) %RSENCODER Encode message with the
Reed-Solomon algorithm % m is the number of bits per
symbol % prim_poly: Primitive polynomial p(x). Ie for
DM is 301 % k is the size of the message % n is the
total size (k+redundant) % Example: msg = uint8('Test')
% enc_msg = rsEncoder(msg, 8, 301, 12, numel(msg));
% Get the alpha alpha = gf(2, m, prim_poly); % Get
the reed-solomon generating polynomial g(x) g_x =
genpoly(k, n, alpha); % Multiply the information by
X^(n-k), or just pad with zeros at the end to % get
space to add the redundant information msg_padded =
gf([msg zeros(1, n-k)], m, prim_poly); % Get the
remainder of the division of the extended message by
the % reed-solomon generating polynomial g(x) [~,
remainder] = deconv(msg_padded, g_x); % Now return
the message with the redundant information encoded =
msg_padded - remainder; end % Find the reed-solomon
generating polynomial g(x), by the way this is the %
same as the rsgenpoly function on matlab function g =
genpoly(k, n, alpha) g = 1; % A multiplication on the
galois field is just a convolution for k = mod(1 : n-k, n)
g = conv(g, [1 alpha .^ (k)]); end end
```

## Decoder

Now the decoding part:

```
function [ decoded, error_pos, error_mag, g, S ] = rsDe-
coder( encoded, m, prim_poly, n, k ) %RSDECODER
Decode a reed-solomon encoded message % Example:
% [dec, ~, ~, ~, ~] = rsDecoder(enc_msg, 8, 301, 12,
numel(msg)) max_errors = floor((n-k)/2); orig_vals =
encoded.x; % Initialize the error vector errors = zeros(1,
n); g = []; S = []; % Get the alpha alpha = gf(2, m,
prim_poly); % Find the syndromes (Check if dividing the
message by the generator % polynomial the result is zero)
Synd = polyval(encoded, alpha .^ (1:n-k)); Syndromes
= trim(Synd); % If all syndromes are zeros (perfectly
divisible) there are no errors if isempty(Syndromes.x)
decoded = orig_vals(1:k); error_pos = []; error_mag
= []; g = []; S = Synd; return; end % Prepare for the
euclidean algorithm (Used to find the error locating %
polynomials) r0 = [1, zeros(1, 2*max_errors)]; r0 =
gf(r0, m, prim_poly); r0 = trim(r0); size_r0 = length(r0);
r1 = Syndromes; f0 = gf([zeros(1, size_r0-1) 1], m,
prim_poly); f1 = gf(zeros(1, size_r0), m, prim_poly);
g0 = f1; g1 = f0; % Do the euclidian algorithm on the
polynomials r0(x) and Syndromes(x) in % order to find
the error locating polynomial while true % Do a long
division [quotient, remainder] = deconv(r0, r1); % Add
some zeros quotient = pad(quotient, length(g1)); % Find
quotient*g1 and pad c = conv(quotient, g1); c = trim(c);
c = pad(c, length(g0)); % Update g as g0-quotient*g1
g = g0 - c; % Check if the degree of remainder(x) is less
than max_errors if all(remainder(1:end - max_errors)
== 0) break; end % Update r0, r1, g0, g1 and remove
```

```
leading zeros r0 = trim(r1); r1 = trim(remainder); g0 =
g1; g1 = g; end % Remove leading zeros g = trim(g); %
Find the zeros of the error polynomial on this galois field
evalPoly = polyval(g, alpha .^ (n-1 : -1 : 0)); error_pos
= gf(find(evalPoly == 0), m); % If no error position is
found we return the received work, because % basically
is nothing that we could do and we return the received
message if isempty(error_pos) decoded = orig_vals(1:k);
error_mag = []; return; end % Prepare a linear system to
solve the error polynomial and find the error % magni-
tudes size_error = length(error_pos); Syndrome_Vals =
Syndromes.x; b(:, 1) = Syndrome_Vals(1:size_error); for
idx = 1 : size_error e = alpha .^ (idx*(n-error_pos.x));
err = e.x; er(idx, :) = err; end % Solve the linear
system error_mag = (gf(er, m, prim_poly) \ gf(b, m,
prim_poly))'; % Put the error magnitude on the error
vector errors(error_pos.x) = error_mag.x; % Bring
this vector to the galois field errors_gf = gf(errors,
m, prim_poly); % Now to fix the errors just add with
the encoded code decoded_gf = encoded(1:k) + er-
rors_gf(1:k); decoded = decoded_gf.x; end % Remove
leading zeros from galois array function gt = trim(g) gx
= g.x; gt = gf(gx(find(gx, 1) : end), g.m, g.prim_poly);
end % Add leading zeros function xpad = pad(x,k) len
= length(x); if (len<k) xpad = [zeros(1, k-len) x]; end end
```

## 93.6 See also

- BCH code
- Cyclic code
- Chien search
- Berlekamp–Massey algorithm
- Forward error correction
- Berlekamp–Welch algorithm
- Folded Reed–Solomon code

## 93.7 Notes

- [1] Reed & Solomon (1960)
- [2] D. Gorenstein and N. Zierler, “A class of cyclic linear error-correcting codes in  $p^m$  symbols,” J. SIAM, vol. 9, pp. 207-214, June 1961
- [3] Error Correcting Codes by W\_Wesley\_Peterson, 1961
- [4] Yasuo Sugiyama, Masao Kasahara, Shigeichi Hirasawa, and Toshihiko Namekawa. A method for solving key equation for decoding Goppa codes. Information and Control, 27:87–99, 1975.

- [5] Immink, K. A. S. (1994), “Reed–Solomon Codes and the Compact Disc”, in Wicker, Stephen B.; Bhargava, Vijay K., *Reed–Solomon Codes and Their Applications*, IEEE Press, ISBN 978-0-7803-1025-4
- [6] Lidl, Rudolf; Pilz, Günter (1999). *Applied Abstract Algebra* (2nd ed.). Wiley. p. 226.
- [7] See Lin & Costello (1983, p. 171), for example.
- [8] Pfender, Florian; Ziegler, Günter M. (September 2004), “Kissing Numbers, Sphere Packings, and Some Unexpected Proofs” (PDF), *Notices of the American Mathematical Society*, **51** (8): 873–883. Explains the Delsarte–Goethals–Seidel theorem as used in the context of the error correcting code for compact disc.
- [9] D. Gorenstein and N. Zierler, “A class of cyclic linear error-correcting codes in  $p^m$  symbols,” *J. SIAM*, vol. 9, pp. 207–214, June 1961
- [10] Error Correcting Codes by W\_Wesley\_Peterson, 1961
- [11] Shu Lin and Daniel J. Costello Jr, “Error Control Coding” second edition, pp. 255–262, 1982, 2004
- [12] Guruswami, V.; Sudan, M. (September 1999), “Improved decoding of Reed–Solomon codes and algebraic geometry codes”, *IEEE Transactions on Information Theory*, **45** (6): 1757–1767, doi:10.1109/18.782097
- [13] Koetter, Ralf; Vardy, Alexander (2003). “Algebraic soft-decision decoding of Reed–Solomon codes”. *IEEE Transactions on Information Theory*. **49** (11): 2809–2825. doi:10.1109/TIT.2003.819332.

## 93.8 References

- Gill, John (n.d.), *EE387 Notes #7, Handout #28* (PDF), Stanford University, retrieved April 21, 2010
- Hong, Jonathan; Vetterli, Martin (August 1995), “Simple Algorithms for BCH Decoding”, *IEEE Transactions on Communications*, **43** (8): 2324–2333, doi:10.1109/26.403765
- Lin, Shu; Costello, Jr., Daniel J. (1983), *Error Control Coding: Fundamentals and Applications*, New Jersey, NJ: Prentice-Hall, ISBN 0-13-283796-X
- Massey, J. L. (1969), “Shift-register synthesis and BCH decoding” (PDF), *IEEE Transactions on Information Theory*, IT-15 (1): 122–127, doi:10.1109/tit.1969.1054260
- Peterson, Wesley W. (1960), “Encoding and Error Correction Procedures for the Bose–Chaudhuri Codes”, *IRE Transactions on Information Theory*, Institute of Radio Engineers, **IT-6**: 459–470
- Reed, Irving S.; Solomon, Gustave (1960), “Polynomial Codes over Certain Finite Fields”, *Journal of the Society for Industrial and Applied Mathematics (SIAM)*, **8** (2): 300–304, doi:10.1137/0108018

- Welch, L. R. (1997), *The Original View of Reed–Solomon Codes* (PDF), Lecture Notes

## 93.9 Further reading

- Berlekamp, Elwyn R. (1967), *Nonbinary BCH decoding*, International Symposium on Information Theory, San Remo, Italy
- Berlekamp, Elwyn R. (1984) [1968], *Algebraic Coding Theory* (Revised ed.), Laguna Hills, CA: Aegean Park Press, ISBN 0-89412-063-8
- Cipra, Barry A. (1993), “The Ubiquitous Reed–Solomon Codes”, *SIAM News*, **26** (1)
- Forney, Jr., G. (October 1965), “On Decoding BCH Codes”, *IEEE Transactions on Information Theory*, **11** (4): 549–557, doi:10.1109/TIT.1965.1053825
- Koetter, Ralf (2005), *Reed–Solomon Codes*, MIT Lecture Notes 6.451 (Video)
- MacWilliams, F. J.; Sloane, N. J. A. (1977), *The Theory of Error-Correcting Codes*, New York, NY: North-Holland Publishing Company
- Reed, Irving S.; Chen, Xuemin (1999), *Error-Control Coding for Data Networks*, Boston, MA: Kluwer Academic Publishers

## 93.10 External links

### 93.10.1 Information and Tutorials

- Introduction to Reed–Solomon codes: principles, architecture and implementation (CMU)
- A Tutorial on Reed–Solomon Coding for Fault-Tolerance in RAID-like Systems
- Algebraic soft-decoding of Reed–Solomon codes
- Wikiversity: Reed–Solomon codes for coders
- BBC R&D White Paper WHP031
- Geisel, William A. (August 1990), *Tutorial on Reed–Solomon Error Correction Coding* (PDF), Technical Memorandum, NASA, TM-102162

### 93.10.2 Code

- Schifra Open Source C++ Reed–Solomon Codec
- Henry Minsky’s RSCode library, Reed–Solomon encoder/decoder
- Open Source C++ Reed–Solomon Soft Decoding library

- Matlab implementation of errors and-erasures Reed–Solomon decoding
- Pure-Python implementation of a Reed–Solomon codec

## Chapter 94

# Sardinas–Patterson algorithm

In **coding theory**, the **Sardinas–Patterson algorithm** is a classical algorithm for determining in **polynomial time** whether a given **variable-length code** is uniquely decodable, named after August Albert Sardinas and George W. Patterson, who published it in 1953.<sup>[1]</sup> The algorithm carries out a systematic search for a string which admits two different decompositions into codewords. As **Knuth** reports, the algorithm was rediscovered about ten years later in 1963 by **Floyd**, despite the fact that it was at the time already well known in coding theory.<sup>[2]</sup>

In the second round, we try out two different approaches: the first trial is to look for a codeword that has  $w$  as prefix. Then we obtain a new dangling suffix  $w'$ , with which we can continue our search. If we eventually encounter a dangling suffix that is itself a codeword (or the empty word), then the search will terminate, as we know there exists a string with two decompositions. The second trial is to seek for a codeword that is itself a prefix of  $w$ . In our example, we have  $w = 10$ , and the sequence  $I$  is a codeword. We can thus also continue with  $w'=0$  as the new dangling suffix.

### 94.1 Idea of the algorithm

Consider the code  $\{a \mapsto 1, b \mapsto 011, c \mapsto 01110, d \mapsto 1110, e \mapsto 10011\}$ . This code, which is based on an example by Berstel,<sup>[3]</sup> is an example of a code which is not uniquely decodable, since the string

011101110011

can be interpreted as the sequence of codewords

01110 – 1110 – 011,

but also as the sequence of codewords

011 – 1 – 011 – 10011.

Two possible decodings of this encoded string are thus given by *cdb* and *babe*.

In general, a codeword can be found by the following idea: In the first round, we choose two codewords  $x_1$  and  $y_1$  such that  $x_1$  is a **prefix** of  $y_1$ , that is,  $x_1 w = y_1$  for some “dangling suffix”  $w$ . If one tries first  $x_1 = 011$  and  $y_1 = 01110$ , the dangling **suffix** is  $w = 10$ . If we manage to find two sequences  $x_2, \dots, x_p$  and  $y_2, \dots, y_q$  of codewords such that  $x_2 \cdots x_p = w y_2 \cdots y_q$ , then we are finished: For then the string  $x = x_1 x_2 \cdots x_p$  can alternatively be decomposed as  $y_1 y_2 \cdots y_q$ , and we have found the desired string having at least two different decompositions into codewords.

### 94.2 Precise description of the algorithm

The algorithm is described most conveniently using quotients of **formal languages**. In general, for two sets of strings  $D$  and  $N$ , the (left) quotient  $N^{-1}D$  is defined as the residual words obtained from  $D$  by removing some prefix in  $N$ . Formally,  $N^{-1}D = \{y \mid xy \in D \text{ and } x \in N\}$ . Now let  $C$  denote the (finite) set of codewords in the given code.

The algorithm proceeds in rounds, where we maintain in each round not only one dangling suffix as described above, but the (finite) set of all potential dangling suffixes. Starting with round  $i = 1$ , the set of potential dangling suffixes will be denoted by  $S_i$ . The sets  $S_i$  are defined **inductively** as follows:

$S_1 = C^{-1}C \setminus \{\varepsilon\}$ . Here, the symbol  $\varepsilon$  denotes the **empty word**.

$S_{i+1} = C^{-1}S_i \cup S_i^{-1}C$ , for all  $i \geq 1$ .

The algorithm computes the sets  $S_i$  in increasing order of  $i$ . As soon as one of the  $S_i$  contains a word from  $C$  or the empty word, then the algorithm terminates and answers that the given code is not uniquely decodable. Otherwise, once a set  $S_i$  equals a previously encountered set  $S_j$  with  $j < i$ , then the algorithm would enter in principle an endless loop. Instead of continuing endlessly, it answers that the given code is uniquely decodable.

### 94.3 Termination and correctness of the algorithm

Since all sets  $S_i$  are sets of suffixes of a finite set of codewords, there are only finitely many different candidates for  $S_i$ . Since visiting one of the sets for the second time will cause the algorithm to stop, the algorithm cannot continue endlessly and thus must always **terminate**. More precisely, the total number of dangling suffixes that the algorithm considers is at most equal to the total of the lengths of the codewords in the input, so the algorithm runs in **polynomial time** as a function of this input length. By using a **suffix tree** to speed the comparison between each dangling suffix and the codewords, the time for the algorithm can be bounded by  $O(nk)$ , where  $n$  is the total length of the codewords and  $k$  is the number of codewords.<sup>[4]</sup> The algorithm can be implemented using a pattern matching machine.<sup>[5]</sup> The algorithm can also be implemented to run on a **nondeterministic turing machine** that uses only **logarithmic space**; the problem of testing unique decipherability is **NL-complete**, so this space bound is optimal.<sup>[6]</sup>

A proof that the algorithm is **correct**, i.e. that it always gives the correct answer, is found in the textbooks by Salomaa<sup>[7]</sup> and by Berstel et al.<sup>[8]</sup>

### 94.4 See also

- **Kraft's inequality** in some cases provides a quick way to exclude the possibility that a given code is uniquely decodable.
- **Prefix codes** and **block codes** are important classes of codes which are uniquely decodable by definition.
- **Timeline of information theory**

### 94.5 Notes

- [1] Sardinas & Patterson (1953).
- [2] Knuth (2003), p. 2
- [3] Berstel et al. (2009), Example 2.3.1 p. 63
- [4] Rodeh (1982).
- [5] Apostolico and Giancarlo (1984).
- [6] Rytter (1986) proves that the complementary problem, of testing for the existence of a string with two decodings, is NL-complete, and therefore that unique decipherability is co-NL-complete. The equivalence of NL-completeness and co-NL-completeness follows from the Immerman–Szelepcsényi theorem.
- [7] Salomaa (1981)
- [8] Berstel et al. (2009), Chapter 2.3

### 94.6 References

- Berstel, Jean; Perrin, Dominique; Reutenauer, Christophe (2010). *Codes and automata*. Encyclopedia of Mathematics and its Applications. **129**. Cambridge: Cambridge University Press. ISBN 978-0-521-88831-8. Zbl 1187.94001.
- Berstel, Jean; Reutenauer, Christophe (2011). *Non-commutative rational series with applications*. Encyclopedia of Mathematics and Its Applications. **137**. Cambridge: Cambridge University Press. ISBN 978-0-521-19022-0. Zbl 1250.68007.
- Knuth, Donald E. (December 2003). “Robert W Floyd, In Memoriam”. *SIGACT News*. **34** (4): 3–13. doi:10.1145/954092.954488.
- Rodeh, M. (1982). “A fast test for unique decipherability based on suffix trees (Corresp.)”. *IEEE Transactions on Information Theory*. **28** (4): 648–651. doi:10.1109/TIT.1982.1056535..
- Apostolico, A.; Giancarlo, R. (1984). “Pattern matching machine implementation of a fast test for unique decipherability”. *Information Processing Letters*. **18** (3): 155–158. doi:10.1016/0020-0190(84)90020-6..
- Rytter, Wojciech (1986). “The space complexity of the unique decipherability problem”. *Information Processing Letters*. **23** (1): 1–3. doi:10.1016/0020-0190(86)90121-3. MR 853618..
- Salomaa, Arto (1981). *Jewels of Formal Language Theory*. Pitman Publishing. ISBN 0-273-08522-0. Zbl 0487.68064.
- Sardinas, August Albert; Patterson, George W. (1953), “A necessary and sufficient condition for the unique decomposition of coded messages”, *Convention Record of the I.R.E., 1953 National Convention, Part 8: Information Theory*, pp. 104–108.

#### Further reading

- Robert G. Gallager: *Information Theory and Reliable Communication*. Wiley, 1968



# Chapter 95

## Second Johnson bound

In applied mathematics, the **Johnson bound** (named after **Selmer Martin Johnson**) is a limit on the size of **error-correcting codes**, as used in **coding theory** for **data transmission** or **communications**.

$$A_q(n, d) \leq \frac{q^n}{\sum_{i=0}^t \binom{n}{i} (q-1)^i + \frac{\binom{n}{t+1} (q-1)^{t+1}}{A_q(n, d, t+1)}}.$$

**Theorem 2 (Johnson bound for  $A_q(n, d, w)$ ):**

(i) If  $d > 2w$ ,

$$A_q(n, d, w) = 1.$$

(ii) If  $d \leq 2w$ , then define the variable  $e$  as follows. If  $d$  is even, then define  $e$  through the relation  $d = 2e$ ; if  $d$  is odd, define  $e$  through the relation  $d = 2e - 1$ . Let  $q^* = q - 1$ . Then,

$$A_q(n, d, w) \leq \left\lfloor \frac{nq^*}{w} \left\lfloor \frac{(n-1)q^*}{w-1} \left\lfloor \dots \left\lfloor \frac{(n-w+e)q^*}{e} \right\rfloor \dots \right\rfloor \right\rfloor \right\rfloor$$

where  $\lfloor \cdot \rfloor$  is the **floor function**.

**Remark:** Plugging the bound of Theorem 2 into the bound of Theorem 1 produces a numerical upper bound on  $A_q(n, d)$ .

### 95.1 Definition

Let  $C$  be a  $q$ -ary **code** of length  $n$ , i.e. a subset of  $\mathbb{F}_q^n$ . Let  $d$  be the minimum distance of  $C$ , i.e.

$$d = \min_{x, y \in C, x \neq y} d(x, y),$$

where  $d(x, y)$  is the **Hamming distance** between  $x$  and  $y$ .

Let  $C_q(n, d)$  be the set of all  $q$ -ary codes with length  $n$  and minimum distance  $d$  and let  $C_q(n, d, w)$  denote the set of codes in  $C_q(n, d)$  such that every element has exactly  $w$  nonzero entries.

Denote by  $|C|$  the number of elements in  $C$ . Then, we define  $A_q(n, d)$  to be the largest size of a code with length  $n$  and minimum distance  $d$ :

$$A_q(n, d) = \max_{C \in C_q(n, d)} |C|.$$

Similarly, we define  $A_q(n, d, w)$  to be the largest size of a code in  $C_q(n, d, w)$ :

$$A_q(n, d, w) = \max_{C \in C_q(n, d, w)} |C|.$$

**Theorem 1 (Johnson bound for  $A_q(n, d)$ ):**

If  $d = 2t + 1$ ,

$$A_q(n, d) \leq \frac{q^n}{\sum_{i=0}^t \binom{n}{i} (q-1)^i + \frac{\binom{n}{t+1} (q-1)^{t+1} - \binom{d}{t+1} A_q(n, d, t+1)}{A_q(n, d, t+1)}}.$$

If  $d = 2t$ ,

### 95.2 See also

- Singleton bound
- Hamming bound
- Plotkin bound
- Elias Bassalygo bound
- Gilbert–Varshamov bound
- Griesmer bound

### 95.3 References

- Johnson, Selmer Martin (April 1962). "A new upper bound for error-correcting codes". *IRE Transactions on Information Theory*: 203–207.



- Huffman, William Cary; Pless, Vera S. (2003). *Fundamentals of Error-Correcting Codes*. Cambridge University Press. ISBN 978-0-521-78280-7.

## Chapter 96

# Shannon's source coding theorem

This article is about the theory of source coding in data compression. For the term in computer programming, see [Source code](#).

In [information theory](#), **Shannon's source coding theorem** (or **noiseless coding theorem**) establishes the limits to possible [data compression](#), and the operational meaning of the [Shannon entropy](#).

The **source coding theorem** shows that (in the limit, as the length of a stream of [independent and identically-distributed random variable](#) (i.i.d.) data tends to infinity) it is impossible to compress the data such that the code rate (average number of bits per symbol) is less than the Shannon entropy of the source, without it being virtually certain that information will be lost. However it is possible to get the code rate arbitrarily close to the Shannon entropy, with negligible probability of loss.

The **source coding theorem for symbol codes** places an upper and a lower bound on the minimal possible expected length of codewords as a function of the [entropy](#) of the input word (which is viewed as a [random variable](#)) and of the size of the target alphabet.

## 96.1 Statements

*Source coding* is a mapping from (a sequence of) symbols from an [information source](#) to a sequence of alphabet symbols (usually bits) such that the source symbols can be exactly recovered from the binary bits (lossless source coding) or recovered within some distortion (lossy source coding). This is the concept behind [data compression](#).

### 96.1.1 Source coding theorem

In information theory, the **source coding theorem** (Shannon 1948)<sup>[1]</sup> informally states that (MacKay 2003, pg. 81,<sup>[2]</sup> Cover:Chapter 5<sup>[3]</sup>):

N i.i.d. random variables each with entropy  $H(X)$  can be compressed into more than  $N H(X)$  bits with negligible risk of information

loss, as  $N \rightarrow \infty$ ; but conversely, if they are compressed into fewer than  $N H(X)$  bits it is virtually certain that information will be lost.

### 96.1.2 Source coding theorem for symbol codes

Let  $\Sigma_1, \Sigma_2$  denote two finite alphabets and let  $\Sigma^*$  1 and  $\Sigma^*$

2 denote the set of all finite words from those alphabets (respectively).

Suppose that  $X$  is a random variable taking values in  $\Sigma_1$  and let  $f$  be a [uniquely decodable](#) code from  $\Sigma^*$

1 to  $\Sigma^*$

2 where  $|\Sigma_2| = a$ . Let  $S$  denote the random variable given by the word length  $f(X)$ .

If  $f$  is optimal in the sense that it has the minimal expected word length for  $X$ , then (Shannon 1948):

$$\frac{H(X)}{\log_2 a} \leq \mathbb{E}S < \frac{H(X)}{\log_2 a} + 1$$

## 96.2 Proof: Source coding theorem

Given  $X$  is an i.i.d. source, its [time series](#)  $X_1, \dots, X_n$  is i.i.d. with [entropy](#)  $H(X)$  in the discrete-valued case and [differential entropy](#) in the continuous-valued case. The Source coding theorem states that for any  $\varepsilon > 0$  for any [rate](#) larger than the [entropy](#) of the source, there is large enough  $n$  and an encoder that takes  $n$  i.i.d. repetition of the source,  $X^{1:n}$ , and maps it to  $n(H(X) + \varepsilon)$  binary bits such that the source symbols  $X^{1:n}$  are recoverable from the binary bits with probability at least  $1 - \varepsilon$ .

**Proof of Achievability.** Fix some  $\varepsilon > 0$ , and let

$$p(x_1, \dots, x_n) = \Pr[X_1 = x_1, \dots, X_n = x_n].$$

The typical set,  $A_\varepsilon$   
 $n$ , is defined as follows:

$$A_n^\varepsilon = \left\{ (x_1, \dots, x_n) : \left| -\frac{1}{n} \log p(x_1, \dots, x_n) - H_n(X) \right| \leq \varepsilon \right\}$$

The **Asymptotic Equipartition Property** (AEP) shows that for large enough  $n$ , the probability that a sequence generated by the source lies in the typical set,  $A_n^\varepsilon$ , as defined approaches one. In particular, for sufficiently large  $n$ ,  $P((X_1, X_2, \dots, X_n) \in A_n^\varepsilon)$  can be made arbitrarily close to 1, and specifically, greater than  $1 - \varepsilon$  (See **AEP** for a proof).

The definition of typical sets implies that those sequences that lie in the typical set satisfy:

$$2^{-n(H(X)+\varepsilon)} \leq p(x_1, \dots, x_n) \leq 2^{-n(H(X)-\varepsilon)}$$

Note that:

- The probability of a sequence  $(X_1, X_2, \dots, X_n)$  being drawn from  $A_n^\varepsilon$  is greater than  $1 - \varepsilon$ .
- $|A_n^\varepsilon| \leq 2^{n(H(X)+\varepsilon)}$ , which follows from the left hand side (lower bound) for  $p(x_1, x_2, \dots, x_n)$ .
- $|A_n^\varepsilon| \geq (1-\varepsilon)2^{n(H(X)-\varepsilon)}$ , which follows from upper bound for  $p(x_1, x_2, \dots, x_n)$  and the lower bound on the total probability of the whole set  $A_n^\varepsilon$ .

Since  $|A_n^\varepsilon| \leq 2^{n(H(X)+\varepsilon)}$ ,  $n(H(X)+\varepsilon)$  bits are enough to point to any string in this set.

The encoding algorithm: The encoder checks if the input sequence lies within the typical set; if yes, it outputs the index of the input sequence within the typical set; if not, the encoder outputs an arbitrary  $n(H(X) + \varepsilon)$  digit number. As long as the input sequence lies within the typical set (with probability at least  $1 - \varepsilon$ ), the encoder doesn't make any error. So, the probability of error of the encoder is bounded above by  $\varepsilon$ .

**Proof of Converse.** The converse is proved by showing that any set of size smaller than  $A_n^\varepsilon$  ( $n$  in the sense of exponent) would cover a set of probability bounded away from 1.

### 96.3 Proof: Source coding theorem for symbol codes

For  $1 \leq i \leq n$  let  $s_i$  denote the word length of each possible  $x_i$ . Define  $q_i = a^{-s_i}/C$ , where  $C$  is chosen so that  $q_1 + \dots + q_n = 1$ . Then

$$\begin{aligned} & \leq -\sum_{i=1}^n p_i \log_2 q_i \\ & = -\sum_{i=1}^n p_i \log_2 a^{-s_i} + \sum_{i=1}^n p_i \log_2 C \\ & = -\sum_{i=1}^n p_i \log_2 a^{-s_i} + \log_2 C \\ & \leq -\sum_{i=1}^n -s_i p_i \log_2 a \\ & \leq \mathbb{E}S \log_2 a \end{aligned}$$

where the second line follows from **Gibbs' inequality** and the fifth line follows from **Kraft's inequality**:

$$C = \sum_{i=1}^n a^{-s_i} \leq 1$$

so  $\log C \leq 0$ .

For the second inequality we may set

$$s_i = \lceil -\log_a p_i \rceil$$

so that

$$-\log_a p_i \leq s_i < -\log_a p_i + 1$$

and so

$$a^{-s_i} \leq p_i$$

and

$$\sum a^{-s_i} \leq \sum p_i = 1$$

and so by **Kraft's inequality** there exists a prefix-free code having those word lengths. Thus the minimal  $S$  satisfies

$$\begin{aligned} \mathbb{E}S & = \sum p_i s_i \\ & < \sum p_i (-\log_a p_i + 1) \\ & = \sum -p_i \frac{\log_2 p_i}{\log_2 a} + 1 \\ & = \frac{H(X)}{\log_2 a} + 1 \end{aligned}$$

## 96.4 Extension to non-stationary independent sources

### 96.4.1 Fixed Rate lossless source coding for discrete time non-stationary independent sources

Define typical set  $A_\varepsilon$   
 $n$  as:

$$A_\varepsilon^n = \left\{ x_1^n : \left| -\frac{1}{n} \log p(X_1, \dots, X_n) - \overline{H}_n(X) \right| < \varepsilon \right\}.$$

Then, for given  $\delta > 0$ , for  $n$  large enough,  $\Pr(A_\varepsilon^n) > 1 - \delta$ . Now we just encode the sequences in the typical set, and usual methods in source coding show that the cardinality of this set is smaller than  $2^{n(\overline{H}_n(X) + \varepsilon)}$ . Thus, on an average,  $Hn(X) + \varepsilon$  bits suffice for encoding with probability greater than  $1 - \delta$ , where  $\varepsilon$  and  $\delta$  can be made arbitrarily small, by making  $n$  larger.

## 96.5 See also

- Channel coding
- Noisy Channel Coding Theorem
- Error exponent
- Asymptotic Equipartition Property (AEP)

## 96.6 References

- [1] C.E. Shannon, "A Mathematical Theory of Communication", *Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, July, October, 1948
- [2] David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms* Cambridge: Cambridge University Press, 2003. ISBN 0-521-64298-1
- [3] Cover, Thomas M. (2006). "Chapter 5: Data Compression". *Elements of Information Theory*. John Wiley & Sons. ISBN 0-471-24195-4.

## Chapter 97

# Singleton bound

In **coding theory**, the **Singleton bound**, named after Richard Collom Singleton, is a relatively crude upper bound on the size of an arbitrary **block code**  $C$  with block length  $n$ , size  $M$  and minimum distance  $d$ .

### 97.1 Statement of the bound

The minimum distance of a set  $C$  of codewords of length  $n$  is defined as

$$d = \min_{\{x, y \in C: x \neq y\}} d(x, y)$$

where  $d(x, y)$  is the **Hamming distance** between  $x$  and  $y$ . The expression  $A_q(n, d)$  represents the maximum number of possible codewords in a  $q$ -ary block code of length  $n$  and minimum distance  $d$ .

Then the Singleton bound states that

$$A_q(n, d) \leq q^{n-d+1}.$$

### 97.2 Proof

First observe that the number of  $q$ -ary words of length  $n$  is  $q^n$ , since each letter in such a word may take one of  $q$  different values, independently of the remaining letters.

Now let  $C$  be an arbitrary  $q$ -ary block code of minimum distance  $d$ . Clearly, all codewords  $c \in C$  are distinct. If we **puncture** the code by deleting the first  $d - 1$  letters of each codeword, then all resulting codewords must still be pairwise different, since all of the original codewords in  $C$  have **Hamming distance** at least  $d$  from each other. Thus the size of the altered code is the same as the original code.

The newly obtained codewords each have length

$$n - (d - 1) = n - d + 1$$

and thus, there can be at most  $q^{n-d+1}$  of them. Since  $C$  was arbitrary, this bound must hold for the largest possible code with these parameters, thus:<sup>[1]</sup>

$$|C| \leq A_q(n, d) \leq q^{n-d+1}.$$

### 97.3 Linear codes

If  $C$  is a **linear code** with block length  $n$ , dimension  $k$  and minimum distance  $d$  over the **finite field** with  $q$  elements, then the maximum number of codewords is  $q^k$  and the Singleton bound implies:

$$q^k \leq q^{n-d+1}$$

so that

$$k \leq n - d + 1$$

which is usually written as<sup>[2]</sup>

$$d \leq n - k + 1$$

In the linear code case a different proof of the Singleton bound can be obtained by observing that rank of the **parity check matrix** is  $n - k$ .<sup>[3]</sup> Another simple proof follows from observing that the rows of any generator matrix in standard form have weight at most  $n - k + 1$ .

### 97.4 History

The usual citation given for this result is Singleton (1964), but according to Welsh (1988, p. 72) the result can be found in a 1953 paper of Komamiya.<sup>[4]</sup>

### 97.5 MDS codes

Linear block codes that achieve equality in the Singleton bound are called **MDS (maximum distance separable)**

**codes.** Examples of such codes include codes that have only two codewords (the all-zero word and the all-one word, having thus minimum distance  $n$ ), codes that use the whole of  $(\mathbb{F}_q)^n$  (minimum distance 1), codes with a single parity symbol (minimum distance 2) and their **dual codes**. These are often called *trivial* MDS codes.

In the case of binary alphabets, only trivial MDS codes exist.<sup>[5][6]</sup>

Examples of non-trivial MDS codes include **Reed-Solomon codes** and their extended versions.<sup>[7][8]</sup>

MDS codes are an important class of block codes since, for a fixed  $n$  and  $k$ , they have the greatest error correcting and detecting capabilities. There are several ways to characterize MDS codes:<sup>[9]</sup>

*Theorem:* Let  $C$  be a linear  $[n, k, d]$  code over  $\mathbb{F}_q$ . The following are equivalent:

- $C$  is an MDS code.
- Any  $k$  columns of a **generator matrix** for  $C$  are **linearly independent**.
- Any  $n - k$  columns of a **parity check matrix** for  $C$  are linearly independent.
- $C^\perp$  is an MDS code.
- If  $G = (I|A)$  is a generator matrix for  $C$  in standard form, then every square submatrix of  $A$  is **nonsingular**.
- Given any  $d$  coordinate positions, there is a (minimum weight) codeword whose **support** is precisely these positions.

The last of these characterizations permits, by using the **MacWilliams identities**, an explicit formula for the complete weight distribution of an MDS code.<sup>[10]</sup>

*Theorem:* Let  $C$  be a linear  $[n, k, d]$  MDS code over  $\mathbb{F}_q$ . If  $A_w$  denotes the number of codewords in  $C$  of weight  $w$ , then

$$A_w = \binom{n}{w} \sum_{j=0}^{w-d} (-1)^j \binom{w}{j} (q^{w-d+1-j} - 1) = \binom{n}{w} (q-1) \sum_{j=0}^{w-d} (-1)^j \binom{w-1}{j} q^{w-d-j}.$$

### 97.5.1 Arcs in projective geometry

The linear independence of the columns of a generator matrix of an MDS code permits a construction of MDS codes from objects in **finite projective geometry**. Let  $PG(N, q)$  be the finite **projective space** of (geometric) dimension  $N$  over the finite field  $\mathbb{F}_q$ . Let  $K = \{P_1, P_2, \dots, P_m\}$  be a set of points in this projective space represented with **homogeneous coordinates**. Form the  $(N+1) \times m$  matrix  $G$  whose columns are the homogeneous coordinates of these points. Then,<sup>[11]</sup>

*Theorem:*  $K$  is a (spatial)  $m$ -arc if and only if  $G$  is the generator matrix of an  $[m, N+1, m-N]$  MDS code over  $\mathbb{F}_q$ .

## 97.6 See also

- **Gilbert–Varshamov bound**
- **Plotkin bound**
- **Hamming bound**
- **Johnson bound**
- **Griesmer bound**

## 97.7 Notes

- [1] Ling & Xing 2004, p. 93
- [2] Roman 1992, p. 175
- [3] Pless 1998, p. 26
- [4] Komamiya, Y. (1953), “Application of logical mathematics to information theory”, *Proc. 3rd Japan. Nat. Cong. Appl. Math.*: 437
- [5] Verma 1996, Proposition 9.2
- [6] Ling & Xing 2004, p. 94 Remark 5.4.7
- [7] MacWilliams & Sloane 1977, Ch. 11
- [8] Ling & Xing 2004, p. 94
- [9] Roman 1992, p. 237, Theorem 5.3.7
- [10] Roman 1992, p. 240
- [11] Bruen, A.A.; Thas, J.A.; Blokhuis, A. (1988), “On M.D.S. codes, arcs in  $PG(n, q)$ , with  $q$  even, and a solution of three fundamental problems of B. Segre”, *Invent. Math.*, **92**: 441–459, doi:10.1007/bf01393742

## 97.8 References

- Ling, San; Xing, Chaoping (2004), *Coding Theory / A First Course*, Cambridge University Press, ISBN 0-521-52923-9
- MacWilliams, F.J.; Sloane, N.J.A. (1977), *The Theory of Error-Correcting Codes*, North-Holland, pp. 33, 37, ISBN 0-444-85193-3
- Pless, Vera (1998), *Introduction to the Theory of Error-Correcting Codes* (3rd ed.), Wiley Interscience, ISBN 0-471-19047-0
- Roman, Steven (1992), *Coding and Information Theory*, GTM, **134**, Springer-Verlag, ISBN 0-387-97812-7

- Singleton, R.C. (1964), “Maximum distance q-nary codes”, *IEEE Trans. Inf. Theory*, **10** (2): 116–118, doi:10.1109/TIT.1964.1053661
- Vermani, L. R. (1996), *Elements of algebraic coding theory*, Chapman & Hall
- Welsh, Dominic (1988), *Codes and Cryptography*, Oxford University Press, ISBN 0-19-853287-3

## 97.9 Further reading

- J.H. van Lint (1992). *Introduction to Coding Theory*. GTM. **86** (2nd ed.). Springer-Verlag. p. 61. ISBN 3-540-54894-7.
- Niederreiter, Harald; Xing, Chaoping (2001). “6. Applications to algebraic coding theory”. *Rational points on curves over finite fields. Theory and Applications*. London Mathematical Society Lecture Note Series. **285**. Cambridge: Cambridge University Press. ISBN 0-521-66543-4. Zbl 0971.11033.



## Chapter 98

# Soliton distribution

A **soliton distribution** is a type of **discrete probability distribution** that arises in the theory of **erasure correcting codes**. A paper by Luby<sup>[1]</sup> introduced two forms of such distributions, the **ideal soliton distribution** and the **robust soliton distribution**.

### 98.1 Ideal distribution

The **ideal soliton distribution** is a probability distribution on the integers from 1 to  $N$ , where  $N$  is the single parameter of the distribution. The **probability mass function** is given by<sup>[2]</sup>

$$p(1) = \frac{1}{N},$$
$$p(k) = \frac{1}{k(k-1)} \quad (k = 2, 3, \dots, N).$$

### 98.2 Robust distribution

The **robust** form of distribution is defined by adding an extra set of values to the elements of mass function of the ideal soliton distribution and then standardising so that the values add up to 1. The extra set of values,  $t$ , are defined in terms of an additional real-valued parameter  $\delta$  (which is interpreted as a failure probability) and an integer parameter  $M$  ( $M < N$ ). Define  $R$  as  $R=N/M$ . Then the values added to  $p(i)$ , before the final standardisation, are<sup>[2]</sup>


$$t(i) = \frac{1}{iM}, \quad (i = 1, 2, \dots, M-1),$$
$$t(i) = \frac{\ln(R/\delta)}{M}, \quad (i = M),$$
$$t(i) = 0, \quad (i = M+1, \dots, N).$$

While the ideal soliton distribution has a **mode** (or spike) at 1, the effect of the extra component in the robust distribution is to add an additional spike at the value  $M$ .

### 98.3 See also

- **Luby transform code**

### 98.4 References

- [1] Luby, M. (2002). *LT Codes*. The 43rd Annual IEEE Symposium on Foundations of Computer Science.
- [2] Tirronen, Tuomas (2005). "Optimal Degree Distributions for LT Codes in Small Cases". Helsinki University of Technology. [CiteSeerX 10.1.1.140.8104](https://arxiv.org/abs/10.1.1.140.8104).

## Chapter 99

# Spherical code

In geometry and coding theory, a **spherical code** with parameters  $(n, N, t)$  is a set of  $N$  points on the unit hypersphere in  $n$  dimensions for which the dot product of unit vectors from the origin to any two points is less than or equal to  $t$ . The kissing number problem may be stated as the problem of finding the maximal  $N$  for a given  $n$  for which a spherical code with parameters  $(n, N, 1/2)$  exists. The Tammes problem may be stated as the problem of finding a spherical code with minimal  $t$  for given  $n$  and  $N$ .

### 99.1 External links

- Weisstein, Eric W. “Spherical code”. *MathWorld*.
- A library of putatively optimal spherical codes

# Chapter 100

## Srivastava code

In coding theory, **Srivastava codes**, formulated by Professor J. N. Srivastava, form a class of parameterised error-correcting codes which are a special case of alternant codes.

### 100.1 Definition

The original *Srivastava code* over  $\text{GF}(q)$  of length  $n$  is defined by a parity check matrix  $H$  of alternant form

$$\begin{bmatrix} \frac{\alpha_1^\mu}{\alpha_1 - w_1} & \cdots & \frac{\alpha_n^\mu}{\alpha_n - w_1} \\ \vdots & \ddots & \vdots \\ \frac{\alpha_1^\mu}{\alpha_1 - w_s} & \cdots & \frac{\alpha_n^\mu}{\alpha_n - w_s} \end{bmatrix}$$

where the  $\alpha_i$  and  $w_i$  are elements of  $\text{GF}(q^m)$

### 100.2 Properties

The parameters of this code are length  $n$ , dimension  $\geq n - ms$  and minimum distance  $\geq s + 1$ .

### 100.3 References

- F.J. MacWilliams; N.J.A. Sloane (1977). *The Theory of Error-Correcting Codes*. North-Holland. pp. 357–360. ISBN 0-444-85193-3.

# Chapter 101

## Standard array

In coding theory, a **standard array** (or Slepian array) is a  $q^{n-k}$  by  $q^k$  array that lists all elements of a particular  $\mathbb{F}_q^n$  vector space. Standard arrays are used to decode linear codes; i.e. to find the corresponding codeword for any received vector.

### 101.1 Definition

A standard array for an  $[n,k]$ -code is a  $q^{n-k}$  by  $q^k$  array where:

1. The first row lists all **codewords** (with the 0 codeword on the extreme left)
2. Each row is a **coset** with the **coset leader** in the first column
3. The entry in the  $i$ -th row and  $j$ -th column is the sum of the  $i$ -th coset leader and the  $j$ -th codeword.

For example, the  $[n,k]$ -code  $C_3 = \{0, 01101, 10110, 11011\}$  has a standard array as follows:

Note that the above is only one possibility for the standard array; had 00011 been chosen as the first **coset leader** of weight two, another standard array representing the code would have been constructed.

Note that the first row contains the 0 vector and the codewords of  $C_3$  (0 itself being a codeword). Also, the leftmost column contains the vectors of **minimum weight** enumerating vectors of weight 1 first and then using vectors of weight 2. Note also that each possible vector in the vector space appears exactly once.

### 101.2 Constructing a standard array

Because each possible vector can appear only once in a standard array some care must be taken during construction. A standard array can be created as follows:

1. List the codewords of  $C$ , starting with 0, as the first row

2. Choose any vector of minimum weight not already in the array. Write this as the first entry of the next row. This vector is denoted the '**coset leader**'.

3. Fill out the row by adding the coset leader to the codeword at the top of each column. The sum of the  $i$ -th coset leader and the  $j$ -th codeword becomes the entry in row  $i$ , column  $j$ .

4. Repeat steps 2 and 3 until all rows/cosets are listed and each vector appears exactly once.

Note that adding vectors is done mod  $q$ . For example, binary codes are added mod 2 (which equivalent to bit-wise XOR addition). For example, in  $Z_2$ ,  $11000 + 11011 = 00011$ .

Note also that selecting different coset leaders will create a slightly different but equivalent standard array, and will not affect results when decoding.

#### 101.2.1 Construction example

Let  $C$  be the binary  $[4,2]$ -code. i.e.  $C = \{0000, 1011, 0101, 1110\}$ . To construct the standard array, we first list the codewords in a row.

We then select a vector of minimum weight (in this case, weight 1) that has not been used. This vector becomes the coset leader for the second row.

Following step 3, we complete the row by adding the coset leader to each codeword.

We then repeat steps 2 and 3 until we have completed all rows. We stop when we have reached  $q^{n-k} = 2^{4-2} = 2^2 = 4$  rows.

Note that in this example we could not have chosen the vector 0001 as the coset leader of the final row, even though it meets the criteria of having minimal weight (1), because the vector was already present in the array. We could, however, have chosen it as the first coset leader and constructed a different standard array.

### 101.3 Decoding via standard array

To decode a vector using a standard array, subtract the error vector - or coset leader - from the vector received. The result will be one of the codewords in  $C$ . For example, say we are using the code  $C = \{0000, 1011, 0101, 1110\}$ , and have constructed the corresponding standard array, as shown from the example above. If we receive the vector 0110 as a message, we find that vector in the standard array. We then subtract the vector's coset leader, namely 1000, to get the result 1110. We have received the codeword 1110.

Decoding via a standard array is a form of **nearest neighbour decoding**. In practice, decoding via a standard array requires large amounts of storage - a code with 32 codewords requires a standard array with  $2^{32}$  entries. Other forms of decoding, such as **syndrome decoding**, are more efficient.

Note that decoding via standard array does not guarantee that all vectors are decoded correctly. If we receive the vector 1010, using the standard array above would decode the message as 1110, a codeword distance 1 away. However, 1010 is also distance 1 away from the codeword 1011. In such a case some implementations might ask for the message to be resent, or the ambiguous bit may be marked as an erasure and a following **outer code** may correct it. This ambiguity is another reason that different decoding methods are sometimes used.

### 101.4 See also

- **Linear code**

### 101.5 References

- **Hill, Raymond** (1986). *A First Course in Coding Theory*. Oxford Applied Mathematics and Computing Science series. Oxford University Press. ISBN 978-0-19-853803-5.

# Chapter 102

## Systematic code

In coding theory, a **systematic code** is any error-correcting code in which the input data is embedded in the encoded output. Conversely, in a **non-systematic code** the output does not contain the input symbols.

Systematic codes have the advantage that the parity data can simply be appended to the source block, and receivers do not need to recover the original source symbols if received correctly – this is useful for example if error-correction coding is combined with a hash function for quickly determining the correctness of the received source symbols, or in cases where errors occur in **erasures** and a received symbol is thus always correct. Furthermore, for engineering purposes such as synchronization and monitoring, it is desirable to get reasonable good estimates of the received source symbols without going through the lengthy decoding process which may be carried out at a remote site at a later time.<sup>[1]</sup>

### 102.1 Properties

Every non-systematic linear code can be transformed into a systematic code with essentially the same properties (i.e., minimum distance).<sup>[1][2]</sup> Because of the advantages cited above, **linear** error-correcting codes are therefore generally implemented as systematic codes. However, for certain decoding algorithms such as sequential decoding or maximum-likelihood decoding, a non-systematic structure can increase performance in terms of undetected decoding error probability when the minimum *free* distance of the code is larger.<sup>[1][3]</sup>

For a systematic **linear code**, the **generator matrix**,  $G$ , can always be written as  $G = [I_k | P]$ , where  $I_k$  is the **identity matrix** of size  $k$ .

### 102.2 Examples

- **Checksums** and **hash functions**, combined with the input data, can be viewed as systematic error-detecting codes.
- Linear codes are usually implemented as systematic error-correcting codes (e.g., Reed-Solomon codes in

CDs).

- **Convolutional codes** are implemented as either systematic or non-systematic codes. Non-systematic convolutional codes can provide better performance under maximum-likelihood (**Viterbi**) decoding.
- In **DVB-H**, for additional error protection and power efficiency for mobile receivers, a systematic **Reed-Solomon code** is employed as an erasure code over packets within a **data burst**, where each packet is protected with a **CRC**: data in verified packets count as correctly received symbols, and if all are received correctly, evaluation of the additional parity data can be omitted, and receiver devices can switch off reception until the start of the next burst.
- **Fountain codes** may be either systematic or non-systematic: as they do not exhibit a fixed **code rate**, the set of source symbols is diminishing among the possible output set.

### 102.3 Notes

- [1] James L. Massey, Daniel J. Costello, Jr. (1971). "Nonsystematic convolutional codes for sequential decoding in space applications". *IEEE Transactions on Communication Technology*. **19** (5). doi:10.1109/TCOM.1971.1090720.
- [2] Richard E. Blahut (2003). *Algebraic codes for data transmission* (2nd ed.). Cambridge Univ. Press. pp. 53–54. ISBN 978-0-521-55374-2.
- [3] Shu Lin; Daniel J. Costello, Jr. (1983). *Error Control Coding: Fundamentals and Applications*. Prentice Hall. pp. 278–280. ISBN 0-13-283796-X.

### 102.4 References

- Shu Lin; Daniel J. Costello, Jr. (1983). *Error Control Coding: Fundamentals and Applications*. Prentice Hall. pp. 278–280. ISBN 0-13-283796-X.

## Chapter 103

# Tanner graph

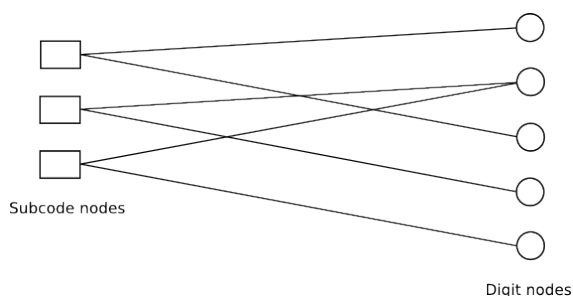
In coding theory, a **Tanner graph**, named after Michael Tanner, is a bipartite graph used to state constraints or equations which specify error correcting codes. In coding theory, Tanner graphs are used to construct longer codes from smaller ones. Both encoders and decoders employ these graphs extensively.

### 103.1 Origins

Tanner graphs were proposed by Michael Tanner<sup>[1]</sup> as a means to create larger error correcting codes from smaller ones using recursive techniques. He generalized the techniques of Elias for product codes.

Tanner discussed lower bounds on the codes obtained from these graphs irrespective of the specific characteristics of the codes which were being used to construct larger codes.

### 103.2 Tanner graphs for linear block codes



*Tanner graph with subcode and digit nodes*

Tanner graphs are partitioned into subcode nodes and digit nodes. For linear block codes, the subcode nodes denote rows of the parity-check matrix  $H$ . The digit nodes represent the columns of the matrix  $H$ . An edge connects a subcode node to a digit node if a nonzero entry exists in the intersection of the corresponding row and column.

### 103.3 Bounds proved by Tanner

Tanner proved the following bounds

Let  $R$  be the rate of the resulting linear code, let the degree of the digit nodes be  $m$  and the degree of the subcode nodes be  $n$ . If each subcode node is associated with a linear code  $(n, k)$  with rate  $r = k/n$ , then the rate of the code is bounded by

$$R \geq 1 - (1 - r)m$$

### 103.4 Computational complexity of Tanner graph based methods

The advantage of these recursive techniques is that they are computationally tractable. The coding algorithm for Tanner graphs is extremely efficient in practice, although it is not guaranteed to converge except for cycle-free graphs, which are known not to admit asymptotically good codes.<sup>[2]</sup>

### 103.5 Applications of Tanner graph

Zemor's decoding algorithm, which is a recursive low-complexity approach to code construction, is based on Tanner graphs.

### 103.6 Notes

- [1] R. Michael Tanner Professor of Computer Science, School of Engineering University of California, Santa Cruz Testimony before Representatives of the United States Copyright Office February 10, 1999
- [2] T. Etzion, A. Trachtenberg, and A. Vardy, Which Codes have Cycle-Free Tanner Graphs?, IEEE Trans. Inf. Theory, 45:6.



- Michael Tanner's Original paper
- Michael Tanner's page

# Chapter 104

## Ternary Golay code

In coding theory, the **ternary Golay codes** are two closely related error-correcting codes. The code generally known simply as the **ternary Golay code** is an  $[11, 6, 5]_3$ -code, that is, it is a linear code over a ternary alphabet; the relative distance of the code is as large as it possibly can be for a ternary code, and hence, the ternary Golay code is a perfect code. The **extended ternary Golay code** is a  $[12, 6, 6]$  linear code obtained by adding a zero-sum check digit to the  $[11, 6, 5]$  code. In finite group theory, the extended ternary Golay code is sometimes referred to as the ternary Golay code.

### 104.1 Properties

#### 104.1.1 Ternary Golay code

The ternary Golay code consists of  $3^6 = 729$  codewords. Its parity check matrix is

$$\begin{bmatrix} 1 & 1 & 1 & 2 & 2 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 2 & 1 & 0 & 2 & 0 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 1 & 2 & 0 & 0 & 1 & 0 & 0 \\ 1 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 2 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Any two different codewords differ in at least 5 positions. Every ternary word of length 11 has a Hamming distance of at most 2 from exactly one codeword. The code can also be constructed as the quadratic residue code of length 11 over the finite field  $\mathbb{F}_3$ .

Used in a football pool with 11 games, the ternary Golay code corresponds to 729 bets and guarantees exactly one bet with at most 2 wrong outcomes.

The set of codewords with Hamming weight 5 is a 3-(11,5,4) design.

#### 104.1.2 Extended ternary Golay code

The complete weight enumerator of the extended ternary Golay code is

$$x^{12} + y^{12} + z^{12} + 22(x^6 y^6 + y^6 z^6 + z^6 x^6) + 220(x^6 y^3 z^3 + y^6 z^3 x^3 + z^6 x^3 y^3).$$

The automorphism group of the extended ternary Golay code is  $2.M_{12}$ , where  $M_{12}$  is the Mathieu group M12.

The extended ternary Golay code can be constructed as the span of the rows of a Hadamard matrix of order 12 over the field  $\mathbb{F}_3$ .

Consider all codewords of the extended code which have just six nonzero digits. The sets of positions at which these nonzero digits occur form the Steiner system  $S(5, 6, 12)$ .

### 104.2 History

The ternary Golay code was discovered by Golay (1949). It was independently discovered two years earlier by the Finnish football pool enthusiast Juhani Virtakallio, who published it in 1947 in issues 27, 28 and 33 of the football magazine *Veikkaaja*. (Barg 1993, p.25)

### 104.3 See also

- Binary Golay code

### 104.4 References

- Barg, Alexander (1993), "At the dawn of the theory of codes", *The Mathematical Intelligencer*, **15** (1): 20–26, doi:10.1007/BF03025254, ISSN 0343-6993, MR 1199273
- M.J.E. Golay, Notes on digital coding, *Proceedings of the I.R.E.* 37 (1949) 657
- I.F. Blake (ed.), *Algebraic Coding Theory: History and Development*, Dowden, Hutchinson & Ross, Stroudsburg 1973

- J. H. Conway and N. J. A. Sloane, *Sphere Packings, Lattices and Groups*, Springer, New York, Berlin, Heidelberg, 1988.
- Robert L. Griess, *Twelve Sporadic Groups*, Springer, 1998.
- G. Cohen, I. Honkala, S. Litsyn, A. Lobstein, *Covering Codes*, Elsevier (1997) ISBN 0-444-82511-8
- Th. M. Thompson, *From Error Correcting Codes through Sphere Packings to Simple Groups*, The Mathematical Association of America 1983, ISBN 0-88385-037-0

# Chapter 105

## Tornado code

In computer science, **Tornado codes** are a class of **erasure codes** that support **error correction**. Tornado codes require a constant  $C$  more redundant blocks than the more data-efficient **Reed–Solomon erasure codes**, but are much faster to generate and can fix erasures faster. Software-based implementations of tornado codes are about 100 times faster on small lengths and about 10,000 times faster on larger lengths than Reed–Solomon erasure codes.<sup>[1]</sup> Since the introduction of Tornado codes, many other similar erasure codes have emerged, most notably **Online codes**, **LT codes** and **Raptor codes**.

Tornado codes use a layered approach. All layers except the last use an **LDPC** error correction code, which is fast but has a chance of failure. The final layer uses a Reed–Solomon correction code, which is slower but is optimal in terms of failure recovery. Tornado codes dictates how many levels, how many recovery blocks in each level, and the distribution used to generate blocks for the non-final layers.

### 105.1 Overview

The input data is divided into blocks. Blocks are sequences of bits that are all the same size. Recovery data uses the same block size as the input data. The erasure of a block (input or recovery) is detected by some other means. (For example, a block from disk does not pass a CRC check or a network packet with a given sequence number never arrived.)

The number of recovery blocks is given by the user. Then the number of levels is determined along with the number of blocks in each level. The number in each level is determined by a factor  $B$  which is less than one. If there are  $N$  input blocks, the first recovery level has  $B \cdot N$  blocks, the second has  $B \cdot B \cdot N$ , the third has  $B \cdot B \cdot B \cdot N$ , and so on.

All levels of recovery except the final one use an LDPC, which works by xor (exclusive-or). Xor operates on binary values, 1s and 0s.  $A \text{ xor } B$  is 1 if  $A$  and  $B$  have different values and 0 if  $A$  and  $B$  have the same values. If you are given  $(A \text{ xor } B)$  and  $A$ , you can determine the value for  $B$ . ( $A \text{ xor } B \text{ xor } A = B$ ) Similarly, if you are

given  $(A \text{ xor } B)$  and  $B$ , you can determine the value for  $A$ . This extends to multiple values, so given  $(A \text{ xor } B \text{ xor } C \text{ xor } D)$  and any 3 of the values, the missing value can be recovered.

So the recovery blocks in level one are just the xor of some set of input blocks. Similarly, the recovery blocks in level two are each the xor of some set of blocks in level one. The blocks used in the xor are chosen randomly, without repetition. However, the *number* of blocks xor'ed to make a recovery block is chosen from a very specific distribution for each level.

Since xor is a fast operation and the recovery blocks are an xor of only a subset of the blocks in the input (or at a lower recovery level), the recovery blocks can be generated quickly.

The final level is a Reed–Solomon code. Reed–Solomon codes are optimal in terms of recovering from failures, but slow to generate and recover. Since each level has fewer blocks than the one before, the Reed–Solomon code has a small number of recovery blocks to generate and to use in recovery. So, even though Reed–Solomon is slow, it only has a small amount of data to handle.

During recovery, the Reed–Solomon code is recovered first. This is guaranteed to work if the number of missing blocks in the next-to-final level is less than the present blocks in the final level.

Going lower, the LDPC (xor) recovery level can be used to recover the level beneath it *with high probability* if all the recovery blocks are present and the level beneath is missing at most  $C'$  fewer blocks than the recovery level. The algorithm for recovery is to find some recovery block that has only one of its generating set missing from the lower level. Then the xor of the recovery block with all of the blocks that are present is equal to the missing block.

### 105.2 Patent issues

Tornado codes are patented inside the United States of America.<sup>[2]</sup> Patents US6163870 A (filed Nov 6, 1997) and US 6081909 A (filed Nov 6, 1997) describe Tornado codes. Patents US6307487 B1 (filed Feb 5, 1999) and US6320520 B1 (filed Sep 17, 1999) also mention Tor-

nado codes. Patents are currently valid until 20 years after filing.

### 105.3 Citations

Michael Luby created the Tornado codes.<sup>[3][4]</sup>

### 105.4 External links

A readable description from CMU (PostScript) and another from Luby at the International Computer Science Institute (PostScript) .

### 105.5 See also

Raptor code

Erasure code

### 105.6 Notes

- [1] A digital fountain approach to reliable distribution of bulk data. <http://portal.acm.org/citation.cfm?id=285243.285258>
- [2] (Mitzenmacher 2004)
- [3] (Luby 1997)
- [4] (Luby 1998)

### 105.7 References

- M. Mitzenmacher (2004). “Digital Fountains: A Survey and Look Forward”. *Proc. 2004 IEEE Information Theory Workshop (ITW)*.
- M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman , V. Stemann (1997). “Practical Loss-Resilient Codes”. *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing (STOC)*: 150–159.
- M. Luby, M. Mitzenmacher, A. Shokrollahi (1998). “Analysis of Random Processes via And-Or Tree Evaluation”. *Proc.of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*: 364–373.

# Chapter 106

## Triangular network coding

In coding theory, **triangular network coding (TNC)** is a **network coding** based packet coding scheme introduced by Qureshi, Foh & Cai (2012).<sup>[1]</sup> Previously, packet coding for network coding was done using linear network coding (LNC). The drawback of LNC over large finite field is that it resulted in high encoding and decoding computational complexity. While linear encoding and decoding over  $GF(2)$  alleviates the concern of high computational complexity, coding over  $GF(2)$  comes at the tradeoff cost of degrading throughput performance.

Triangular network coding therefore essentially addresses the high encoding and decoding computational complexity without degrading the throughput performance, with **code rate** comparable to that of linear network coding.

$O(n^2)$  for each bit location.

### 106.2 References

- [1] Qureshi, Jalaluddin; Foh, Chuan Heng; Cai, Jianfei (2012), "Optimal Solution for the Index Coding Problem Using Network Coding over  $GF(2)$ ", *IEEE SECON*: 134–142, doi:10.1109/SECON.2012.6275780.
- [2] J. B. Fraleigh, and R. A. Beauregard, Linear Algebra. Chapter 10, Addison-Wesley Publishing Company, 1995.

### 106.1 Coding and decoding

$b_{1,1}$	$b_{2,1}$	$b_{3,1}$	$b_{4,1}$	...	...	$b_{B,1}$	0	0	0
0	$b_{1,2}$	$b_{2,2}$	$b_{3,2}$	$b_{4,2}$	...	...	$b_{B,2}$	0	0
0	0	$b_{1,3}$	$b_{2,3}$	$b_{3,3}$	$b_{4,3}$	...	...	$b_{B,3}$	0
0	0	0	$b_{1,4}$	$b_{2,4}$	$b_{3,4}$	$b_{4,4}$	...	...	$b_{B,4}$

An example of coding four packets using TNC. Bit  $b_{i,k} \in \{0,1\}$  is the  $i^{\text{th}}$  bit of the  $k^{\text{th}}$  packet. Each packet has original length of  $B$  bits. The resulting coded packet has length  $B + 3$  bits. Information about the number of redundant '0' bits added at the head of each packet is included in the coded packet's header.

In TNC, coding is performed in two stages. First redundant "0" bits are selectively added at the head and tail of each packet such that all packets are of uniform bit length. Then the packets are **XOR coded**, bit-by-bit. The "0" bits are added in such a way that these redundant "0" bits added to each packet generate a **triangular pattern**.

In essence, the TNC decoding process, like the LNC decoding process involves **Gaussian elimination**. However, since the packets in TNC have been coded in such a manner that the resulting coded packets are in triangular pattern, the computational process of *triangularization*,<sup>[2]</sup> with complexity of  $O(n^3)$ , where  $n$  is the number of packets, can be bypassed. The receiver now only needs to perform *back-substitution*,<sup>[2]</sup> with complexity given as

# Chapter 107

## Unary coding

**Unary coding**, sometimes called **thermometer code**, is an **entropy encoding** that represents a **natural number**,  $n$ , with  $n$  ones followed by a zero (if *natural number* is understood as *non-negative integer*) or with  $n - 1$  ones followed by a zero (if *natural number* is understood as *strictly positive integer*). For example 5 is represented as 11110 or 11110. Some representations use  $n$  or  $n - 1$  zeros followed by a one. The ones and zeros are interchangeable without loss of generality. Unary coding is both a prefix-free code and a self-synchronizing code.

Unary coding is an optimally efficient encoding for the following discrete **probability distribution**

$$P(n) = 2^{-n}$$

for  $n = 1, 2, 3, \dots$

In symbol-by-symbol coding, it is optimal for any **geometric distribution**

$$P(n) = (k - 1)k^{-n}$$

for which  $k \geq \varphi = 1.61803398879\dots$ , the **golden ratio**, or, more generally, for any discrete distribution for which

$$P(n) \geq P(n + 1) + P(n + 2)$$

for  $n = 1, 2, 3, \dots$ . Although it is the optimal symbol-by-symbol coding for such probability distributions, **Golomb coding** achieves better compression capability for the geometric distribution because it does not consider input symbols independently, but rather implicitly groups the inputs. For the same reason, **arithmetic encoding** performs better for general probability distributions, as in the last case above.

### 107.1 Unary code in use today

Examples of unary code uses include:

- In **Golomb Rice code**, unary encoding is used to encode the quotient part of the Golomb code word.

- In **UTF-8**, unary encoding is used in the leading byte of a multi-byte sequence to indicate the number of bytes in the sequence, so that the length of the sequence can be determined without examining the continuation bytes.
- **Instantaneously trained neural networks** use unary coding for efficient data representation.

### 107.2 Unary coding in biological networks

New research has shown that unary coding is used in the neural circuits responsible for **birdsong** production.<sup>[1][2]</sup> The nucleus in the brain of the songbirds that plays a part in both the learning and the production of bird song is the HVC (high vocal center). This coding works as space coding which is an efficient strategy for biological circuits due to its inherent simplicity and robustness.

### 107.3 Generalized unary coding

A generalized version of unary coding is able to represent numbers much more efficiently than standard unary coding.<sup>[3]</sup> Here's an example of generalized unary coding for integers from 1 through 15 that requires only 7 bits (where three bits are arbitrarily chosen in place of a single one in standard unary to show the number). Note that the representation is cyclic where one uses markers to represent higher integers in higher cycles.

Generalized unary coding requires that the range of numbers to be represented be pre-specified because this range determines the number of bits that are needed.

### 107.4 See also

- **Unary numeral system**



## 107.5 References

- [1] Fiete, I.R. and H.S. Seung, Neural network models of birdsong production, learning, and coding. *New Encyclopedia of Neuroscience*. Eds. L. Squire, T. Albright, F. Bloom, F. Gage, and N. Spitzer. Elsevier, 2007.
- [2] Moore, J.M.; et al. (2011). “Motor pathway convergence predicts syllable repertoire size in oscine birds”. *Proc. Natl. Acad. Sci. USA*. **108**: 16440–16445. doi:10.1073/pnas.1102077108.
- [3] Kak, S (2015). “Generalized unary coding”. *Circuits, Systems and Signal Processing*. **35**: 1419–1426. doi:10.1007/s00034-015-0120-7.

# Chapter 108

## Variable-length code

This article is about the transmission of data across noisy channels. For the storage of text in computers, see [Variable-width encoding](#).

In [coding theory](#) a **variable-length code** is a [code](#) which maps source symbols to a *variable* number of bits.

Variable-length codes can allow sources to be [compressed](#) and decompressed with *zero* error ([lossless data compression](#)) and still be read back symbol by symbol. With the right coding strategy an [independent and identically-distributed source](#) may be compressed almost arbitrarily close to its [entropy](#). This is in contrast to fixed length coding methods, for which data compression is only possible for large blocks of data, and any compression beyond the logarithm of the total number of possibilities comes with a finite (though perhaps arbitrarily small) probability of failure.

Some examples of well-known variable-length coding strategies are [Huffman coding](#), [Lempel–Ziv coding](#) and [arithmetic coding](#).

### 108.1 Codes and their extensions

The extension of a code is the mapping of finite length source sequences to finite length bit strings, that is obtained by concatenating for each symbol of the source sequence the corresponding codeword produced by the original code.

Using terms from [formal language theory](#), the precise mathematical definition is as follows: Let  $S$  and  $T$  be two finite sets, called the source and target alphabets, respectively. A **code**  $C : S \rightarrow T^*$  is a total function mapping each symbol from  $S$  to a [sequence](#) of symbols over  $T$ , and the extension of  $C$  to a [homomorphism](#) of  $S^*$  into  $T^*$ , which naturally maps each sequence of source symbols to a sequence of target symbols, is referred to as its **extension**.

### 108.2 Classes of variable-length codes

Variable-length codes can be strictly nested in order of decreasing generality as non-singular codes, uniquely decodable codes and prefix codes. Prefix codes are always uniquely decodable, and these in turn are always non-singular:

#### 108.2.1 Non-singular codes

A code is **non-singular** if each source symbol is mapped to a different non-empty bit string, i.e. the mapping from source symbols to bit strings is [injective](#).

- For example the mapping  $M_1 = \{a \mapsto 0, b \mapsto 0, c \mapsto 1\}$  is **not** non-singular because both “a” and “b” map to the same bit string “0”; any extension of this mapping will generate a lossy (non-lossless) coding. Such singular coding may still be useful when some loss of information is acceptable (for example when such code is used in audio or video compression, where a lossy coding becomes equivalent to source [quantization](#)).
- However, the mapping  $M_2 = \{a \mapsto 1, b \mapsto 011, c \mapsto 01110, d \mapsto 1110, e \mapsto 10011, f \mapsto 0\}$  is non-singular; its extension will generate a lossless coding, which will be useful for general data transmission (but this feature is not always required). Note that it is not necessary for the non-singular code to be more compact than the source (and in many applications, a larger code is useful, for example as a way to detect and/or recover from encoding or transmission errors, or in security applications to protect a source from undetectable tampering).

#### 108.2.2 Uniquely decodable codes

A code is **uniquely decodable** if its extension is non-singular (see above). Whether a given code is uniquely decodable can be decided with the [Sardinas–Patterson algorithm](#).

- The mapping  $M_3 = \{a \mapsto 0, b \mapsto 01, c \mapsto 011\}$  is uniquely decodable (this can be demonstrated by looking at the *follow-set* after each target bit string in the map, because each bitstring is terminated as soon as we see a 0 bit which cannot follow any existing code to create a longer valid code in the map, but unambiguously starts a new code).
- Consider again the code  $M_2$  from the previous section. This code, which is based on an example found in,<sup>[1]</sup> is **not** uniquely decodable, since the string *011101110011* can be interpreted as the sequence of codewords *01110–1110–011*, but also as the sequence of codewords *011–1–011–10011*. Two possible decodings of this encoded string are thus given by *cdb* and *babe*. However, such a code is useful when the set of all possible source symbols is completely known and finite, or when there are restrictions (for example a formal syntax) that determine if source elements of this extension are acceptable. Such restrictions permit the decoding of the original message by checking which of the possible source symbols mapped to the same symbol are valid under those restrictions.

### 108.2.3 Prefix codes

Main article: [Prefix code](#)

A code is a **prefix code** if no target bit string in the mapping is a prefix of the target bit string of a different source symbol in the same mapping. This means that symbols can be decoded instantaneously after their entire codeword is received. Other commonly used names for this concept are **prefix-free code**, **instantaneous code**, or **context-free code**.

- The example mapping  $M_3$  in the previous paragraph is **not** a prefix code because we don't know after reading the bit string "0" if it encodes an "a" source symbol, or if it is the prefix of the encodings of the "b" or "c" symbols.
- An example of a prefix code is shown below.

Example of encoding and decoding:

```
aabacdab      →
0010011011010 →
1010110101101110101 →
aabacdab
```

A special case of prefix codes are **block codes**. Here all codewords must have the same length. The latter are not very useful in the context of **source coding**, but often serve as **error correcting codes** in the context of **channel coding**.

Another special case of prefix codes are **variable-length quantity codes**, which encode arbitrarily large integers as a sequence of octets -- i.e., every codeword is a multiple of 8 bits.

## 108.3 Advantages

The advantage of a variable-length code is that unlikely source symbols can be assigned longer codewords and likely source symbols can be assigned shorter codewords, thus giving a low *expected* codeword length. For the above example, if the probabilities of (a, b, c, d) were  $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8})$ , the expected number of bits used to represent a source symbol using the code above would be:

$$1 \times \frac{1}{2} + 2 \times \frac{1}{4} + 3 \times \frac{1}{8} + 3 \times \frac{1}{8} = \frac{7}{4}$$

As the entropy of this source is 1.7500 bits per symbol, this code compresses the source as much as possible so that the source can be recovered with *zero* error.

## 108.4 Notes

[1] Berstel et al. (2009), Example 2.3.1, p. 63

## 108.5 References

- Berstel, Jean; Perrin, Dominique; Reutenauer, Christophe (2010). *Codes and automata*. Encyclopedia of Mathematics and its Applications. **129**. Cambridge: Cambridge University Press. ISBN 978-0-521-88831-8. Zbl 1187.94001. Draft available online

# Chapter 109

## Z-channel (information theory)

A **Z-channel** is a communications channel used in coding theory and information theory to model the behaviour of some data storage systems.

### 109.1 Definition

A *Z-channel* (or a *binary asymmetric channel*) is a channel with binary input and binary output where the crossover  $1 \rightarrow 0$  occurs with nonnegative probability  $p$ , whereas the crossover  $0 \rightarrow 1$  never occurs. In other words, if  $X$  and  $Y$  are the random variables describing the probability distributions of the input and the output of the channel, respectively, then the crossovers of the channel are characterized by the conditional probabilities

$$\begin{aligned}\text{Prob}\{Y = 0 \mid X = 0\} &= 1 \\ \text{Prob}\{Y = 0 \mid X = 1\} &= p \\ \text{Prob}\{Y = 1 \mid X = 0\} &= 0 \\ \text{Prob}\{Y = 1 \mid X = 1\} &= 1-p\end{aligned}$$

### 109.2 Capacity

The capacity  $\text{cap}(\mathbb{Z})$  of the Z-channel  $\mathbb{Z}$  with the crossover  $1 \rightarrow 0$  probability  $p$ , when the input random variable  $X$  is distributed according to the Bernoulli distribution with probability  $\alpha$  for the occurrence of 0, is calculated as follows.

$$\begin{aligned}\text{cap}(\mathbb{Z}) &= \max_{\alpha} \{H(Y) - H(Y \mid X)\} = \\ &= \max_{\alpha} \left\{ H(Y) - \sum_{x \in \{0,1\}} H(Y \mid X = x) \text{Prob}\{X = x\} \right\} \\ &= \max_{\alpha} \{H((1-\alpha)(1-p)) - H(Y \mid X = 1) \text{Prob}\{X = 1\}\} \\ &= \max_{\alpha} \{H((1-\alpha)(1-p)) - (1-\alpha)H(p)\},\end{aligned}$$

where  $H(\cdot)$  is the binary entropy function.

The maximum is attained for

$$\alpha = 1 - \frac{1}{(1-p)(1 + 2^{H(p)/(1-p)})},$$

yielding the following value of  $\text{cap}(\mathbb{Z})$  as a function of  $p$

$$\text{cap}(\mathbb{Z}) = H\left(\frac{1}{1 + 2^{s(p)}}\right) - \frac{s(p)}{1 + 2^{s(p)}} = \log_2(1 + 2^{-s(p)}) = \log_2\left(1 + (1 - \dots)\right)$$

For small  $p$ , the capacity is approximated by

$$\text{cap}(\mathbb{Z}) \approx 1 - 0.5H(p)$$

as compared to the capacity  $1 - H(p)$  of the binary symmetric channel with crossover probability  $p$ .

### 109.3 Bounds on the size of an asymmetric-error-correcting code

Define the following distance function  $d_A(\mathbf{x}, \mathbf{y})$  on the words  $\mathbf{x}, \mathbf{y} \in \{0, 1\}^n$  of length  $n$  transmitted via a Z-channel

$$d_A(\mathbf{x}, \mathbf{y}) \triangleq \max \left\{ |\{i \mid x_i = 0, y_i = 1\}|, |\{i \mid x_i = 1, y_i = 0\}| \right\}.$$

Define the sphere  $V_t(\mathbf{x})$  of radius  $t$  around a word  $\mathbf{x} \in \{0, 1\}^n$  of length  $n$  as the set of all the words at distance  $t$  or less from  $\mathbf{x}$ , in other words,

$$V_t(\mathbf{x}) = \{\mathbf{y} \in \{0, 1\}^n \mid d_A(\mathbf{x}, \mathbf{y}) \leq t\}.$$

A code  $\mathcal{C}$  of length  $n$  is said to be  $t$ -asymmetric-error-correcting if for any two codewords  $\mathbf{c} \neq \mathbf{c}' \in \{0, 1\}^n$ , one has  $V_t(\mathbf{c}) \cap V_t(\mathbf{c}') = \emptyset$ . Denote by  $M(n, t)$  the maximum number of codewords in a  $t$ -asymmetric-error-correcting code of length  $n$ .

**The Varshamov bound.** For  $n \geq 1$  and  $t \geq 1$ ,

$$M(n, t) \leq \frac{2^{n+1}}{\sum_{j=0}^t \left( \binom{\lfloor n/2 \rfloor}{j} + \binom{\lceil n/2 \rceil}{j} \right)}.$$

**The constant-weight code bound.** For  $n > 2t \geq 2$ , let the sequence  $B_0, B_1, \dots, B_{n-2t-1}$  be defined as

$$B_0 = 2, \quad B_i = \min_{0 \leq j < i} \{B_j + A(n+t+i-j-1, 2t+2, t+i)\} \text{ for } i > 0.$$

Then  $M(n, t) \leq B_{n-2t-1}$ .

## 109.4 References

- T. Kløve, Error correcting codes for the asymmetric channel, *Technical Report 18-09-07-81*, Department of Informatics, University of Bergen, Norway, 1981.
- L.G. Tallini, S. Al-Bassam, B. Bose, On the capacity and codes for the Z-channel, *Proceedings of the IEEE International Symposium on Information Theory*, Lausanne, Switzerland, 2002, p. 422.

# Chapter 110

## Zemor's decoding algorithm

In coding theory, **Zemor's algorithm**, designed and developed by Gilles Zemor,<sup>[1]</sup> is a recursive low-complexity approach to code construction. It is an improvement over the algorithm of Sipser and Spielman.

Zemor considered a typical class of Sipser–Spielman construction of **expander codes**, where the underlying graph is **bipartite graph**. Sipser and Spielman introduced a constructive family of asymptotically good linear-error codes together with a simple parallel algorithm that will always remove a constant fraction of errors. The article is based on Dr. Venkatesan Guruswami's course notes<sup>[2]</sup>

every vertex of  $E$  is adjacent to exactly 2 vertices of  $V$ . It means that  $V$  and  $E$  make up, respectively, the vertex set and edge set of  $d$  regular graph  $G$ .

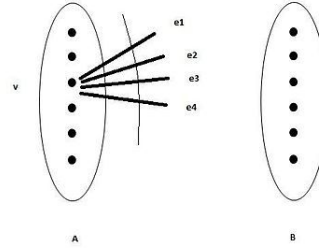
Let us call the code  $C$  constructed in this way as  $(G, C_o)$  code. For a given graph  $G$  and a given code  $C_o$ , there are several  $(G, C_o)$  codes as there are different ways of ordering edges incident to a given vertex  $v$ , i.e.,  $v(1), v(2), \dots, v(d)$ . In fact our code  $C$  consist of all codewords such that  $x_v \in C_o$  for all  $v \in A, B$ . The code  $C$  is linear  $[N, K, D]$  in  $\mathbb{F}$  as it is generated from a subcode  $C_o$ , which is linear. The code  $C$  is defined as  $C = \{c \in \mathbb{F}^N : (c)_v \in C_o\}$  for every  $v \in V$ .

### 110.1 Code construction

Zemor's algorithm is based on a type of **expander graphs** called **Tanner graph**. The construction of code was first proposed by Tanner.<sup>[3]</sup> The codes are based on **double cover**  $d$ , regular expander  $G$ , which is a bipartite graph.  $G = (V, E)$ , where  $V$  is the set of vertices and  $E$  is the set of edges and  $V = A \cup B$  and  $A \cap B = \emptyset$ , where  $A$  and  $B$  denotes the set of 2 vertices. Let  $n$  be the number of vertices in each group, i.e.,  $|A| = |B| = n$ . The edge set  $E$  be of size  $N = nd$  and every edge in  $E$  has one endpoint in both  $A$  and  $B$ .  $E(v)$  denotes the set of edges containing  $v$ .

Assume an ordering on  $V$ , therefore ordering will be done on every edges of  $E(v)$  for every  $v \in V$ . Let **finite field**  $\mathbb{F} = GF(2)$ , and for a word  $x = (x_e), e \in E$  in  $\mathbb{F}^N$ , let the subword of the word will be indexed by  $E(v)$ . Let that word be denoted by  $(x)_v$ . The subset of vertices  $A$  and  $B$  induces every word  $x \in \mathbb{F}^N$  a partition into  $n$  non-overlapping sub-words  $(x)_v \in \mathbb{F}^d$ , where  $v$  ranges over the elements of  $A$ . For constructing a code  $C$ , consider a linear subcode  $C_o$ , which is a  $[d, r_o d, \delta]$  code, where  $q$ , the size of the alphabet is 2. For any vertex  $v \in V$ , let  $v(1), v(2), \dots, v(d)$  be some ordering of the  $d$  vertices of  $E$  adjacent to  $v$ . In this code, each bit  $x_e$  is linked with an edge  $e$  of  $E$ .

We can define the code  $C$  to be the set of binary vectors  $x = (x_1, x_2, \dots, x_N)$  of  $\{0, 1\}^N$  such that, for every vertex  $v$  of  $V$ ,  $(x_{v(1)}, x_{v(2)}, \dots, x_{v(d)})$  is a code word of  $C_o$ . In this case, we can consider a special case when



Graph  $G$  and code  $C$

In this figure,  $(x)_v = (x_{e1}, x_{e2}, x_{e3}, x_{e4}) \in C_o$ . It shows the graph  $G$  and code  $C$ .

In matrix  $G$ , let  $\lambda$  is equal to the second largest **eigen value** of **adjacency matrix** of  $G$ . Here the largest eigen value is  $d$ . Two important claims are made:

#### 110.1.1 Claim 1

$\left(\frac{K}{N}\right) \geq 2r_o - 1$ . Let  $R$  be the rate of a linear code constructed from a bipartite graph whose digit nodes have degree  $m$  and whose subcode nodes have degree  $n$ . If a single linear code with parameters  $(n, k)$  and rate  $r = \left(\frac{k}{n}\right)$  is associated with each of the subcode nodes, then  $k \geq 1 - (1 - r)m$ .

**Proof**

Let  $R$  be the rate of the linear code, which is equal to  $K/N$ . Let there are  $S$  subcode nodes in the graph. If the degree of the subcode is  $n$ , then the code must have  $\binom{n}{m} S$  digits, as each digit node is connected to  $m$  of the  $(n) S$  edges in the graph. Each subcode node contributes  $(n - k)$  equations to parity check matrix for a total of  $(n - k) S$ . These equations may not be linearly independent.

Therefore,  $\left(\frac{K}{N}\right) \geq \frac{\left(\frac{n}{m}\right) S - (n - k) S}{\binom{n}{m} S}$   
 $\geq 1 - m \left(\frac{n - k}{n}\right) \geq 1 - m(1 - r)$ , Since the value of  $m$ , i.e., the digit node of this bipartite graph is 2 and here  $r = r_o$ , we can write as:

$$\left(\frac{K}{N}\right) \geq 2r_o - 1$$

**110.1.2 Claim 2**

$$D \geq N \left( \frac{\left(\delta - \left(\frac{\lambda}{d}\right)\right)}{\left(1 - \left(\frac{\lambda}{d}\right)\right)} \right)^2$$

$$= N \left( \delta^2 - O\left(\frac{\lambda}{d}\right) \right) \rightarrow (1)$$

If  $S$  is linear code of rate  $r$ , block code length  $d$ , and minimum relative distance  $\delta$ , and if  $B$  is the edge vertex incidence graph of a  $d$ -regular graph with second largest eigen value  $\lambda$ , then the code  $C(B, S)$  has rate at least  $2r_o - 1$  and minimum relative distance at least

$$\left( \frac{\left(\delta - \left(\frac{\lambda}{d}\right)\right)}{\left(1 - \left(\frac{\lambda}{d}\right)\right)} \right)^2.$$

**Proof**

Let  $B$  be derived from the  $d$  regular graph  $G$ . So, the number of variables of  $C(B, S)$  is  $\binom{dn}{2}$  and the number of constraints is  $n$ . According to Alon - Chung,<sup>[4]</sup> if  $X$  is a subset of vertices of  $G$  of size  $\gamma n$ , then the number of edges contained in the subgraph is induced by  $X$  in  $G$  is at most  $\binom{dn}{2} \left( \gamma^2 + \left(\frac{\lambda}{d}\right) \gamma (1 - \gamma) \right)$ .

As a result, any set of  $\binom{dn}{2} \left( \gamma^2 + \left(\frac{\lambda}{d}\right) \gamma (1 - \gamma) \right)$  variables will be having at least  $\gamma n$  constraints as neighbours. So the average number of variables per

$$\text{constraint is : } \left( \frac{\binom{2nd}{2} \left( \gamma^2 + \left(\frac{\lambda}{d}\right) \gamma (1 - \gamma) \right)}{\gamma n} \right) =$$

$$d \left( \gamma + \left(\frac{\lambda}{d}\right) (1 - \gamma) \right) \rightarrow (2)$$

So if  $d \left( \gamma + \left(\frac{\lambda}{d}\right) (1 - \gamma) \right) < \gamma d$ , then a word of relative weight  $\left( \gamma^2 + \left(\frac{\lambda}{d}\right) \gamma (1 - \gamma) \right)$ , cannot be a codeword of  $C(B, S)$ . The inequality (2) is satisfied for  $\gamma < \left( \frac{1 - \left(\frac{\lambda}{d}\right)}{\delta - \left(\frac{\lambda}{d}\right)} \right)$ . Therefore,  $C(B, S)$  cannot have

a non zero codeword of relative weight  $\left( \frac{\delta - \left(\frac{\lambda}{d}\right)}{1 - \left(\frac{\lambda}{d}\right)} \right)^2$  or less.

In matrix  $G$ , we can assume that  $\lambda/d$  is bounded away from 1. For those values of  $d$  in which  $d - 1$  is odd prime, there are explicit constructions of sequences of  $d$ -regular bipartite graphs with arbitrarily large number of vertices such that each graph  $G$  in the sequence is a **Ramanujan graph**. It is called Ramanujan graph as it satisfies the inequality  $\lambda(G) \leq 2\sqrt{d - 1}$ . Certain expansion properties are visible in graph  $G$  as the separation between the eigen values  $d$  and  $\lambda$ . If the graph  $G$  is Ramanujan graph, then that expression (1) will become 0 eventually as  $d$  becomes large.

**110.2 Zemor's algorithm**

The iterative decoding algorithm written below alternates between the vertices  $A$  and  $B$  in  $G$  and corrects the codeword of  $C_o$  in  $A$  and then it switches to correct the codeword  $C_o$  in  $B$ . Here edges associated with a vertex on one side of a graph are not incident to other vertex on that side. In fact, it doesn't matter in which order, the set of nodes  $A$  and  $B$  are processed. The vertex processing can also be done in parallel.

The decoder  $\mathbb{D} : \mathbb{F}^d \rightarrow C_o$  stands for a decoder for  $C_o$  that recovers correctly with any codewords with less than  $\binom{d}{2}$  errors.

**110.2.1 Decoder algorithm**

Received word :  $w = (w_e), e \in E$

$z \leftarrow w$  For  $t \leftarrow 1$  to  $m$  do //  $m$  is the number of iterations  
 { if ( $t$  is odd) // Here the algorithm will alternate between its two vertex sets.

$X \leftarrow A$  else  $X \leftarrow B$

Iteration  $t$  : For every  $v \in X$ , let  $(z)_v \leftarrow \mathbb{D}((z)_v)$  // Decoding  $z_v$  to its nearest codeword.

}  
 Output:  $z$



### 110.2.2 Explanation of the algorithm

Since  $G$  is bipartite, the set  $A$  of vertices induces the partition of the edge set  $E = \cup_{v \in A} E_v$ . The set  $B$  induces another partition,  $E = \cup_{v \in B} E_v$ .

Let  $w \in \{0, 1\}^N$  be the received vector, and recall that  $N = dn$ . The first iteration of the algorithm consists of applying the complete decoding for the code induced by  $E_v$  for every  $v \in A$ . This means that for replacing, for every  $v \in A$ , the vector  $(w_{v(1)}, w_{v(2)}, \dots, w_{v(d)})$  by one of the closest codewords of  $C_o$ . Since the subsets of edges  $E_v$  are disjoint for  $v \in A$ , the decoding of these  $n$  subvectors of  $w$  may be done in parallel.

The iteration will yield a new vector  $z$ . The next iteration consists of applying the preceding procedure to  $z$  but with  $A$  replaced by  $B$ . In other words, it consists of decoding all the subvectors induced by the vertices of  $B$ . The coming iterations repeat those two steps alternately applying parallel decoding to the subvectors induced by the vertices of  $A$  and to the subvectors induced by the vertices of  $B$ .

**Note:** [If  $d = n$  and  $G$  is the complete bipartite graph, then  $C$  is a product code of  $C_o$  with itself and the above algorithm reduces to the natural hard iterative decoding of product codes].

Here, the number of iterations,  $m$  is  $\left( \frac{(\log n)}{\log(2 - \alpha)} \right)$ . In general, the above algorithm can correct a code word whose Hamming weight is no more than  $\left( \frac{1}{2} \right) \cdot \alpha N \delta \left( \left( \frac{\delta}{2} \right) - \left( \frac{\lambda}{d} \right) \right) = \left( \frac{1}{4} \right) \cdot \alpha N (\delta^2 - O(\frac{\lambda}{d}))$  for values of  $\alpha < 1$ . Here, the decoding algorithm is implemented as a circuit of size  $O(N \log N)$  and depth  $O(\log N)$  that returns the codeword given that error vector has weight less than  $\alpha N \delta^2 (1 - \epsilon)/4$ .

### 110.2.3 Theorem

If  $G$  is a Ramanujan graph of sufficiently high degree, for any  $\alpha < 1$ , the decoding algorithm can correct  $(\frac{\alpha \delta^2}{4})(1 - \epsilon)N$  errors, in  $O(\log n)$  rounds (where the big- $O$  notation hides a dependence on  $\alpha$ ). This can be implemented in linear time on a single processor; on  $n$  processors each round can be implemented in constant time.

#### Proof

Since the decoding algorithm is insensitive to the value of the edges and by linearity, we can assume that the transmitted codeword is the all zeros - vector. Let the received codeword be  $w$ . The set of edges which has an incorrect value while decoding is considered. Here by incorrect value, we mean 1 in any of the bits. Let  $w = w^0$  be the initial value of the codeword,  $w^1, w^2, \dots, w^t$

be the values after first, second . . .  $t$  stages of decoding. Here,  $X^i = \{e \in E \mid x_e^i = 1\}$ , and  $S^i = \{v \in V \mid E_v \cap X^{i+1} \neq \emptyset\}$ . Here  $S^i$  corresponds to those set of vertices that was not able to successfully decode their codeword in the  $i^{th}$  round. From the above algorithm  $S^1 < S^0$  as number of unsuccessful vertices will be corrected in every iteration. We can prove that  $S^0 > S^1 > S^2 > \dots$  is a decreasing sequence. In fact,  $|S_{i+1}| \leq (\frac{1}{2 - \alpha})|S_i|$ . As we are assuming,  $\alpha < 1$ , the above equation is in a geometric decreasing sequence. So, when  $|S_i| < n$ , more than  $\log_{2-\alpha} n$  rounds are necessary. Furthermore,  $\sum |S_i| = n \sum (\frac{1}{(2 - \alpha)^i}) = O(n)$ , and if we implement the  $i^{th}$  round in  $O(|S_i|)$  time, then the total sequential running time will be linear.

### 110.3 Drawbacks of Zemor's algorithm

1. It is lengthy process as the number of iterations  $m$  in decoder algorithm takes is  $\lceil (\log n) / (\log(2 - \alpha)) \rceil$
2. Zemor's decoding algorithm finds it difficult to decode erasures. A detailed way of how we can improve the algorithm is

given in.[5]

### 110.4 See also

- Expander codes
- Tanner graph
- Linear time encoding and decoding of error-correcting codes

### 110.5 References

- [1] Gilles Zemor
- [2] <http://www.cs.washington.edu/education/courses/cse590vg/03wi/scribes/1-27.ps>
- [3] <http://www.cs.washington.edu/education/courses/cse533/06au/lecnotes/lecture14.pdf>
- [4] <http://math.ucsd.edu/~fan/mypaps/fanpap/93tolerant.pdf>
- [5] "Archived copy". Archived from the original on September 14, 2004. Retrieved May 1, 2012.

# Chapter 111

## Zigzag code

Not to be confused with [zigzag cipher](#).

In coding theory, a **zigzag code** is a type of linear error-correcting code introduced by Ping, Huang & Phamdo (2001).<sup>[1]</sup> They are defined by partitioning the input data into segments of fixed size, and adding sequence of check bits to the data, where each check bit is the [exclusive or](#) of the bits in a single segment and of the previous check bit in the sequence.

The [code rate](#) is high:  $J/(J + 1)$  where J is the number of bits per segment. Its worst-case ability to correct transmission errors is very limited: in the worst case it can only detect a single bit error and cannot correct any errors. However, it works better in the [soft-decision model of decoding](#): its regular structure allows the task of finding a [maximum-likelihood decoding](#) or a posteriori probability decoding to be performed in constant time per input bit.

### 111.1 References

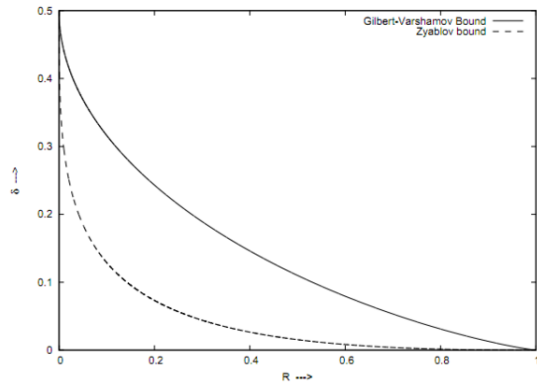
- [1] Ping, Li; Huang, Xiaoling; Phamdo, Nam (2001), “Zigzag codes and concatenated zigzag codes”, *IEEE Transactions on Information Theory*, **47** (2): 800–807, doi:10.1109/18.910590, MR 1820492.

# Chapter 112

## Zyablov bound

In coding theory, the **Zyablov bound** is a lower bound on the rate  $R$  and relative distance  $\delta$  of concatenated codes.

### 112.1 Statement of the bound



The Zyablov bound. For comparison, the GV bound is also plotted.

Let  $R$  be the rate of the outer code  $C_{out}$  and  $\delta$  be the relative distance, then the rate of the concatenated codes satisfies the following bound.

$$R \geq \max_{0 \leq r \leq 1 - H_q(\delta + \varepsilon)} r \left( 1 - \frac{\delta}{H_q^{-1}(1 - r) - \varepsilon} \right)$$

where  $r$  is the rate of the inner code  $C_{in}$ .

### 112.2 Description

Let  $C_{out}$  be the outer code,  $C_{in}$  be the inner code.

Consider  $C_{out}$  meets the **Singleton bound** with rate of  $R$ , i.e.  $C_{out}$  has relative distance  $\delta > 1 - R$ . In order for  $C_{out} \circ C_{in}$  to be an asymptotically good code,  $C_{in}$  also needs to be an asymptotically good code which means,  $C_{in}$  needs to have rate  $r > 0$  and relative distance  $\delta_{in} > 0$ .

Suppose  $C_{in}$  meets the **Gilbert-Varshamov bound** with rate of  $r$  and thus with relative distance

$$\delta_{in} \geq H_q^{-1}(1 - r) - \varepsilon, \quad \varepsilon > 0,$$

then  $C_{out} \circ C_{in}$  has rate of  $rR$  and  $\delta = (1 - R)(H_q^{-1}(1 - r) - \varepsilon)$ .

Expressing  $R$  as a function of  $\delta, r$

$$R = 1 - \frac{\delta}{H_q^{-1}(1 - r) - \varepsilon}$$

Then optimizing over the choice of  $r$ , we get that rate of the **Concatenated error correction code** satisfies,

$$R \geq \max_{0 \leq r \leq 1 - H_q(\delta + \varepsilon)} r \left( 1 - \frac{\delta}{H_q^{-1}(1 - r) - \varepsilon} \right)$$

This lower bound is called Zyablov bound (the bound of  $r < 1 - H_q(\delta + \varepsilon)$  is necessary to ensure  $R > 0$ ). See Figure 2 for a plot of this bound.

Note that the Zyablov bound implies that for every  $\delta > 0$ , there exists a (concatenated) code with rate  $R > 0$ .

### 112.3 Remarks

We can construct a code that achieves the Zyablov bound in polynomial time. In particular, we can construct explicit asymptotically good code (over some alphabets) in polynomial time.

Linear Codes will help us complete the proof of the above statement since linear codes have polynomial representation. Let  $C_{out}$  be an  $[N, K]_Q$  **Reed-Solomon error correction code** where  $N = Q - 1$  (evaluation points being  $\mathbb{F}_Q^*$  with  $Q = q^k$ , then  $k = \theta(\log N)$ ).

We need to construct the Inner code that lies on **Gilbert-Varshamov bound**. This can be done in two ways

1. To perform an exhaustive search on all generator matrices until the required property is satisfied for  $C_{in}$ . This is because Varshamovs bound states

that there exists a linear code that lies on Gilbert-Varshamov bound which will take  $q^{O(kn)}$  time. Using  $k = rn$  we get  $q^{O(kn)} = q^{O(k^2)} = N^{O(\log N)}$ , which is upper bounded by  $nN^{O(\log nN)}$ , a quasi-polynomial time bound.

1. To construct  $C_{in}$  in  $q^{O(n)}$  time and use  $(nN)^{O(1)}$  time overall. This can be achieved by using the method of conditional expectation on the proof that random linear code lies on the bound with high probability.

Thus we can construct a code that achieves the Zyablov bound in polynomial time.

## 112.4 See also

- Singleton bound
- Gilbert-Varshamov bound

## 112.5 References and External Links

- MIT Lecture Notes on Essential Coding Theory – Dr. Madhu Sudan
- University at Buffalo Lecture Notes on Coding Theory – Dr. Atri Rudra
- University of Washington Lecture Notes on Coding Theory- Dr. Venkatesan Guruswami

## 112.6 Text and image sources, contributors, and licenses

### 112.6.1 Text

- **Fountain code** *Source:* [https://en.wikipedia.org/wiki/Fountain\\_code?oldid=748060781](https://en.wikipedia.org/wiki/Fountain_code?oldid=748060781) *Contributors:* Chris~enwiki, Haakon, Phr, Edcolins, Scirus, Linas, BD2412, Quuxplusone, Van der Hoorn, Crystallina, Bluebot, Oli Filth, Mark@digitalfountain.com, Djcmackay, JoeBot, Requestion, Ntsimp, Tdunning, Headbomb, Max Hyre, Abortz, Yobot, Citation bot, Thore Husfeldt, Nageh, The mike luby, Willie.zeng, Solarra, Dcirovic, H3llBot, BG19bot, Camlf, Mark viking, Shorty66, Comp.arch, Cocph, Suaybarslan and Anonymous: 24
- **Forward error correction** *Source:* [https://en.wikipedia.org/wiki/Forward\\_error\\_correction?oldid=761704125](https://en.wikipedia.org/wiki/Forward_error_correction?oldid=761704125) *Contributors:* The Anome, Imran, Michael Hardy, Kku, Ixfd64, Technopilgrim, Denelson83, Mjscud, DocWatson42, DavidCary, Karn, Frau Holle, Abdull, Discospinster, Rich Farmbrough, ESkog, ZeroOne, West London Dweller, MaxHund, A-Day, Jumbuck, Guy Harris, Culix, Linas, Eyreland, BD2412, Raymond Hill, Cirao, Rjwilmsi, Brucelee, Planetneutral, The Rambling Man, YurikBot, Gaius Cornelius, Bovineone, Aryanae, Robertvan1, SmackBot, Unyoyega, Skizzik, Bluebot, RDBrown, Oli Filth, Nbarth, Frap, Ghiraddje, Chrylis, DylanW, A5b, Ligulembot, Novangelis, Kvng, JoeBot, Chetvorno, Ylloh, Requestion, Nilfanion, Widefox, Em3ryguy, .anacondabot, Upholder, First Harmonic, Edu-rant, Bugtrio, CliffC, Mange01, Warut, Xavier Giró, Davecrosby uk, 28bytes, Spongecat, CanOfWorms, Cuddlyable3, Jamelan, Wykyp-dya, Synthebot, Yc Zheng, Snarkosis, Mkjo, PixelBot, Jakarr, Callinus, Subversive.sound, Addbot, Hunting dog, Arsenalboi22, Happyclay-ton, Legobot, Yobot, TaBOT~zerem, AnomieBOT, Xqbot, Agasta, TechBot, Isheden, Omnipaedista, Nageh, Caprisan, Itusg15q4user, UteFan16, BenzolBot, I dream of horses, Full-date unlinking bot, EmausBot, John of Reading, Pugliavi, Dewritech, Dcirovic, Fred Gandt, Quondum, BillHart93, Petr, ClueBot NG, JordoCo, Curb Chain, Hartlojl, Adamroyce, Camlf, Informationist1, Alarbus, BattyBot, Mabokham, ChrisGualtieri, Aymankamelwiki, Ajraymond, SlmedioP, Snookerr, James timberly, Monkbob, Samyelygendy89, Wurtemberg and Anonymous: 95
- **List of algebraic coding theory topics** *Source:* [https://en.wikipedia.org/wiki/List\\_of\\_algebraic\\_coding\\_theory\\_topics?oldid=668401286](https://en.wikipedia.org/wiki/List_of_algebraic_coding_theory_topics?oldid=668401286) *Contributors:* Michael Hardy, Charles Matthews, ZeroOne, Davidgothberg, Guy Harris, Cburnett, Oleg Alexandrov, Mathbot, Ospalh, Fplay, Syrcatbot, Cydebot, The Transhumanist, R'n'B, DavidCBryant, Niceguyedc, Robert Skyhawk, Worldbruce, Nageh, Abductive, BG19bot, W96 and Anonymous: 2
- **Coding theory** *Source:* [https://en.wikipedia.org/wiki/Coding\\_theory?oldid=760759904](https://en.wikipedia.org/wiki/Coding_theory?oldid=760759904) *Contributors:* Andre Engels, Drahflow, Michael Hardy, Raven in Orbit, Charles Matthews, Xiaodai~enwiki, Phil Boswell, Kaol, Altenmann, Giftlite, DavidCary, Sam Hocevar, D6, Rich Farmbrough, LindsayH, Bender235, CanisRufus, Bobo192, Photonique, Pouya, Cburnett, Jheald, Xiaoyangu, Oleg Alexandrov, GregorB, BD2412, Salix alba, Mathbot, Jpkotta, YurikBot, Gene.arboit, Gaius Cornelius, Aaron Brenneman, Kevmo, Einsteinsatheesh, Ott2, SmackBot, Incnis Msi, InverseHypercube, Relaxing, RDBrown, SEIBasaurus, Jlpayton, Drewnoakes, Leiting, Dicklyon, Kvng, CRGreathouse, CBM, Jesse Viviano, Phatom87, Icazzi, Nearfar, Thijs!bot, Epbr123, Andyjsmith, Bobblehead, Gdickeson, Careless hx, Applrpn, STBot, Chiswick Chap, NewEnglandYankee, Dhaluza, DavidCBryant, Selinger, Magmi, Picojeff, Spur, Jamessungjin.kim, JackSchmidt, OK-Bot, Alexkrules1234, Martarius, Justin W Smith, Copyeditor42, SteveJothan, Addbot, RPHv, Vavlap, Uncia, Ramses68, Teles, Zorrobot, Yobot, AnomieBOT, Keithbob, Meriscute, Isheden, SchnitzelMannGreek, Solarium2, FrescoBot, Mmiller, Nageh, Sławomir Bialy, Zero Thrust, RedBot, Lotje, Yatescr, Zink Dawg, John of Reading, Primefac, GoingBatty, Dcirovic, Bethnim, ZéroBot, Ahughes6, Devanshuh-pandey, L Kensington, Bomazi, ChuispastonBot, ClueBot NG, Wcherowi, Rezabot, Helpful Pixie Bot, Sleeping is fun, Neutral current, drift chambers, CitationCleanerBot, BattyBot, ChrisGualtieri, Madhurasuresh, Harikine, Pintoeh, Brirush, Trapmoth, Joeinwiki, Pham-hatkhanh, 314Username, Ahasn, Mark-W-T, Comp.arch, Gmk7, Anurag Bishnoi, Monkbob, Zppix, BlueFenixReborn, KasparBot, Ashkan hdd, Bender the Bot and Anonymous: 71
- **Alternant code** *Source:* [https://en.wikipedia.org/wiki/Alternant\\_code?oldid=601546555](https://en.wikipedia.org/wiki/Alternant_code?oldid=601546555) *Contributors:* Michael Hardy, Vanish2, Addbot, Trappist the monk, RjwilmsiBot, Luizpuodzius, Qetuth and Anonymous: 1
- **Arbitrarily varying channel** *Source:* [https://en.wikipedia.org/wiki/Arbitrarily\\_varying\\_channel?oldid=760670029](https://en.wikipedia.org/wiki/Arbitrarily_varying_channel?oldid=760670029) *Contributors:* Michael Hardy, Kku, Magioladitis, Edratzer, MalcolmX15, Boing! said Zebedee, Auntof6, LilHelpa, Survivor1126, Helpful Pixie Bot, SmittysmithIII and Anonymous: 1
- **Bar product** *Source:* [https://en.wikipedia.org/wiki/Bar\\_product?oldid=662272750](https://en.wikipedia.org/wiki/Bar_product?oldid=662272750) *Contributors:* SimonP, Michael Hardy, Kzollman, Reetep, Jpkotta, Bluebot, Vanish2, DavidCBryant, Yobot, Trappist the monk, Gorobay and Anonymous: 2
- **BCH code** *Source:* [https://en.wikipedia.org/wiki/BCH\\_code?oldid=760621155](https://en.wikipedia.org/wiki/BCH_code?oldid=760621155) *Contributors:* AxelBoldt, Arvindn, Michael Hardy, Mdupont, AugPi, Charles Matthews, Timwi, Dysprosia, Xiaodai~enwiki, Sander123, RedWolf, Naddy, Giftlite, Bob.v.R, Rick Burns, Abdull, Qutezuze, Photonique, Cburnett, Culix, Bookandcoffee, Oleg Alexandrov, Justinlebar, Marudubshinki, Rjwilmsi, Jasoneth, Mathbot, Bmicomp, Reetep, Gene.arboit, Gaius Cornelius, Canadadane, SmackBot, Oli Filth, Zom-B, LouScheffer, J. Finkelstein, Ylloh, Smarantonio, Arun ks, Boemanneke, Thijs!bot, Epbr123, Hermel, Sterrys, Vanish2, Hpfister, Glrx, Tikiwont, Selinger, Uurtamo, Cr-williamsuk, Timeqiu, AlphaPyro, Flyer22 Reborn, Mild Bill Hiccup, Dblagg, Ykh Wong, Sun Creator, Addbot, Mortense, Rcgldr, MrOllie, Legobot, Lucas~bot, Yobot, Sivanov87, AnomieBOT, Motfellow, Obersachsebot, Omnipaedista, Ergosys, Raphael082, Chsivic, Bobmath, EmausBot, Ajraddatz, GoingBatty, JMtB03, 1typesetter, BattyBot, Cyberbot II, Awoldar, ChrisGualtieri, YFdyh~bot, Hippo.69, Snookerr, Lenacore, Douira100, Cesarchaves1, Latex-yow, GreenC bot, Tim Row and Anonymous: 78
- **Belief propagation** *Source:* [https://en.wikipedia.org/wiki/Belief\\_propagation?oldid=759708673](https://en.wikipedia.org/wiki/Belief_propagation?oldid=759708673) *Contributors:* Heron, Michael Hardy, CesarB, A5, Francs2000, SimonMayer, Giftlite, PeR, Andreas Kaufmann, Rich Farmbrough, Mecanismo, Emin63, YUL89YYZ, Xezbeth, 3mta3, Soultaco, Linas, BD2412, Kbdank71, Rjwilmsi, ElKevbo, Mathbot, VSION, Adoniscik, Pi Delport, Ikcotyck, CharlesHBennett, Lserni, SmackBot, Mcl, KYN, Gilliam, Averisk, Cantalamessa, Lage~enwiki, Synergy, Tomixdf, Dougher, AndreasWittenstein, Ph.eyes, David Eppstein, EssRon, Edratzer, Wikip rhyme, Maghnus, Jamelan, Quietbritishjim, Trumpsternator, Iknowyourider, Melcombe, Kvihill, Spacetimewave, Simon04, Zxcv2000, D.scain.farenzena, CohesionBot, Addbot, Arefindk, Yobot, Gdewilde, Yangtseyangtse, AnomieBOT, Citation bot, Hwymeers, Tokidokix, Achalldave, Citation bot 1, Trappist the monk, Jamesmcoughlan, Obankston, Tapirtrust, EmausBot, John of Reading, Victolunik, EdoBot, FreePeter3000, Helpful Pixie Bot, BG19bot, Xuxing716, Tuonawa, Max Libbrecht, Marcocapelle, Intervallc, ChrisGualtieri, Jfwk, Dexbot, Primal dual, Snookerr, Monkbob, Chenglongjiang, Mpkuse, Zppix, StefanOllinger, Mrauto56, Ocli5568, Flt entendre, Cqql and Anonymous: 71
- **Berger code** *Source:* [https://en.wikipedia.org/wiki/Berger\\_code?oldid=740658904](https://en.wikipedia.org/wiki/Berger_code?oldid=740658904) *Contributors:* Pol098, Bgwhite, Mitchan, Bmearns, CmdrObot, Reywas92, Cobi and Anonymous: 9
- **Berlekamp–Welch algorithm** *Source:* [https://en.wikipedia.org/wiki/Berlekamp%E2%80%93Welch\\_algorithm?oldid=766993644](https://en.wikipedia.org/wiki/Berlekamp%E2%80%93Welch_algorithm?oldid=766993644) *Contributors:* Michael Hardy, Phil Boswell, Qwertyus, Rjwilmsi, Ground Zero, Spacepotato, Glrx, Wilhelmina Will, Addbot, Rcgldr, Yobot, Lil-Helpa, FrescoBot, WikitanvirBot, ClueBot NG, Vbutterin, Helpful Pixie Bot, Karthik.sk2001, Rohitram 88, BattyBot, Pintoeh, A.coder20, Latex-yow, IrishWizzard and Anonymous: 4

- **Binary erasure channel** *Source:* [https://en.wikipedia.org/wiki/Binary\\_erasure\\_channel?oldid=748790551](https://en.wikipedia.org/wiki/Binary_erasure_channel?oldid=748790551) *Contributors:* Damian Yerrick, Giftlite, Grubber, EAdherhold, Closedmouth, Bluebot, Thijs!bot, A3nm, Edratzer, Panfakes, Addbot, Yobot, Nageh, Survivor1126, SBaker43, Nerdcorenet and Anonymous: 12
- **Binary Goppa code** *Source:* [https://en.wikipedia.org/wiki/Binary\\_Goppa\\_code?oldid=696748605](https://en.wikipedia.org/wiki/Binary_Goppa_code?oldid=696748605) *Contributors:* Michael Hardy, Hiiiiiii-iiiiiiiiiiiiii, Myasuda, Brycehughes, ClueBot NG, ChrisGualtieri, Nouniquenames, Exaexa and Anonymous: 5
- **Binary symmetric channel** *Source:* [https://en.wikipedia.org/wiki/Binary\\_symmetric\\_channel?oldid=737972923](https://en.wikipedia.org/wiki/Binary_symmetric_channel?oldid=737972923) *Contributors:* Derek Ross, Enchanter, Nd12345, Bdesham, Michael Hardy, Kku, Rholtan, Giftlite, Klemen Kocjancic, Oleg Alexandrov, MarSch, Gurch, Tomer Ish Shalom, Reetep, YurikBot, Gene.arboit, Grubber, Chris the speller, Wafulz, David Eppstein, Edratzer, Signalhead, Vandalscript, X0mp, Addbot, Samsam.yh, AnomieBOT, Rubinbot, LilHelpa, FrescoBot, LucienBOT, Sankha25, John of Reading, AvicAWB, Survivor1126, SBaker43, Snotbot, BG19bot, Samblack956, Latex-yow and Anonymous: 19
- **Blackwell channel** *Source:* [https://en.wikipedia.org/wiki/Blackwell\\_channel?oldid=737131497](https://en.wikipedia.org/wiki/Blackwell_channel?oldid=737131497) *Contributors:* BD2412, Rjwilmsi, Phantomsteve, Grubber, EAdherhold, Khamar, Headbomb, DavidCBryant, RjwilmsiBot and Mogism
- **Blahut–Arimoto algorithm** *Source:* [https://en.wikipedia.org/wiki/Blahut%E2%80%93Arimoto\\_algorithm?oldid=748809416](https://en.wikipedia.org/wiki/Blahut%E2%80%93Arimoto_algorithm?oldid=748809416) *Contributors:* The Anome, Michael Hardy, Phil Boswell, Lastras, Yobot, Jonesey95, AManWithNoPlan, Pintoch and Anonymous: 4
- **Block code** *Source:* [https://en.wikipedia.org/wiki/Block\\_code?oldid=751858122](https://en.wikipedia.org/wiki/Block_code?oldid=751858122) *Contributors:* Michael Hardy, Ciphergoth, Charles Matthews, Tea2min, Enochlau, Bob.v.R, ArnoldReinhold, Xezbeth, Rgdboer, Syd1435, Kocio, Linas, BD2412, Rjwilmsi, Maxal, Salvatore Ingala, Bgwhite, Zwobot, Nuwankumara, SmackBot, RDBury, Dicklyon, Igoldste, Ylloh, CRGreathouse, Thijs!bot, Arch dude, Vanish2, Upholder, Glrx, Mange01, Camrn86, Synthebot, SieBot, Plastikspork, Mild Bill Hiccup, He7d3r, Bender2k14, C. Iorenz, Addbot, OlEnglish, Yobot, C5813, AnomieBOT, Li3939108, Citation bot, Xqbot, Alex.kshevetskiy, Nageh, Mhadi.afraasiabi, Pinethicket, RedBot, Trappist the monk, Jonkerz, EmausBot, Xeransis, ClueBot NG, Snotbot, Helpful Pixie Bot, BG19bot, Meatsgains, ChrisGualtieri, Mogism, Ramshyam1111, A4b3c2d1eOf, Gmk7, Ginsulof, Lanpingu, Frank Klemm, Justinzilla, Acayl, Mahmoudmnsor, Bender the Bot, Asamov and Anonymous: 35
- **Burst error-correcting code** *Source:* [https://en.wikipedia.org/wiki/Burst\\_error-correcting\\_code?oldid=748913076](https://en.wikipedia.org/wiki/Burst_error-correcting_code?oldid=748913076) *Contributors:* Michael Hardy, Bgwhite, Script3r, Yobot, LilHelpa, FrescoBot, Klbrain, BG19bot, Mabokham, Reatlas, Mark viking, Adityakapre, InternetArchiveBot, Latex-yow and Anonymous: 6
- **Canonical Huffman code** *Source:* [https://en.wikipedia.org/wiki/Canonical\\_Huffman\\_code?oldid=761085587](https://en.wikipedia.org/wiki/Canonical_Huffman_code?oldid=761085587) *Contributors:* Michael Hardy, Furrykef, Klox, Sladen, Justinlebar, Ketiltrout, Lockley, Tinku99, Josh Triplett, SmackBot, Chris the speller, Bluebot, Oli Filth, MT-Decerbo, Fan-1967, Snorkelman, Magioladitis, David Eppstein, MathematicalOrchid, R'n'B, Tautrimas, Addbot, Marcelobusico, Luckasbot, Yobot, John737, LilHelpa, 3family6, Erik9bot, PigFlu Oink, AvicBot, MonoAV, Wazimahammed, Eyesnore, Jon.creighton and Anonymous: 17
- **Coding gain** *Source:* [https://en.wikipedia.org/wiki/Coding\\_gain?oldid=630173324](https://en.wikipedia.org/wiki/Coding_gain?oldid=630173324) *Contributors:* Michael Hardy, Photonique, RadioKirk, Msmolnikar, Synergy, Ilmarinen swe, Wdwd, Niceguyedc, Addbot, Yobot, Isheden, FrescoBot, Quondum, Toccata quarta, Maddymuffin, Guanxs and Anonymous: 5
- **Comma code** *Source:* [https://en.wikipedia.org/wiki/Comma\\_code?oldid=744289792](https://en.wikipedia.org/wiki/Comma_code?oldid=744289792) *Contributors:* Bearcat, Hooperbloob, MindlessXD and Bender the Bot
- **Comma-free code** *Source:* [https://en.wikipedia.org/wiki/Comma-free\\_code?oldid=747444966](https://en.wikipedia.org/wiki/Comma-free_code?oldid=747444966) *Contributors:* PierreAbbat, JTN, Incnis Mersi, MindlessXD, Jaan Vajakas, Bender the Bot and Anonymous: 4
- **Computationally bounded adversary** *Source:* [https://en.wikipedia.org/wiki/Computationally\\_bounded\\_adversary?oldid=729252360](https://en.wikipedia.org/wiki/Computationally_bounded_adversary?oldid=729252360) *Contributors:* Edward, Michael Hardy, Smarteralec, Chris the speller, Dekart, AnomieBOT, I dream of horses, Dcirovic, Jim dobler, Rfs23 and Anonymous: 1
- **Concatenated error correction code** *Source:* [https://en.wikipedia.org/wiki/Concatenated\\_error\\_correction\\_code?oldid=722583957](https://en.wikipedia.org/wiki/Concatenated_error_correction_code?oldid=722583957) *Contributors:* Michael Hardy, Asparagus, Giftlite, Beamishboy, Guy Harris, Jheald, Eyreland, Vegaswikian, Tony1, SmackBot, Amux, Iridescent, Ylloh, Alaibot, Magioladitis, Vanish2, Ohms law, Brenont, Finetooth, ClueBot, EoGuy, Isheden, FrescoBot, Nageh, Chaotic-neuron~enwiki, RjwilmsiBot, Dcirovic, JavaDfan, Karthik.sk2001, BattyBot, Monkbob and Anonymous: 5
- **Coset leader** *Source:* [https://en.wikipedia.org/wiki/Coset\\_leader?oldid=610923556](https://en.wikipedia.org/wiki/Coset_leader?oldid=610923556) *Contributors:* Bhny, Deskana, SmackBot, JackSchmidt, KathrynLybarger, Pcap, GlvsOP, Marcelst, Helpful Pixie Bot, Qetuth and Codename Lisa
- **Covering code** *Source:* [https://en.wikipedia.org/wiki/Covering\\_code?oldid=699203491](https://en.wikipedia.org/wiki/Covering_code?oldid=699203491) *Contributors:* Michael Hardy, Michael Slone, Rwalker, Iridescent, YK Times, David Eppstein, Rumping, SchreiberBike, Incraton, Ramses68, Primaler, GoingBatty, Klbrain, Wcherowi and Anonymous: 3
- **Cyclic code** *Source:* [https://en.wikipedia.org/wiki/Cyclic\\_code?oldid=751513576](https://en.wikipedia.org/wiki/Cyclic_code?oldid=751513576) *Contributors:* Michael Hardy, Charles Matthews, Sander123, Giftlite, DavidCary, Bob.v.R, Culix, Linas, Rjwilmsi, MarSch, SmackBot, RDBury, Chris the speller, CorbinSimpson, Kvng, RJChapman, Dugwiki, Gioto, Hermel, Vanish2, David Eppstein, Glrx, STBotD, Selinger, Tomaxer, Dlrohrer2003, Addbot, Tassede-the, Samsam.yh, C5813, AnomieBOT, Motfellow, LilHelpa, GrouchoBot, Nageh, Alexander Chervov, RedBot, 777sms, M1arvin, Proz, Wcherowi, Snotbot, Helpful Pixie Bot, Mark Arsten, Trombonechamp, Randomguess, Lanpingu, Asamov and Anonymous: 28
- **Decoding methods** *Source:* [https://en.wikipedia.org/wiki/Decoding\\_methods?oldid=762027609](https://en.wikipedia.org/wiki/Decoding_methods?oldid=762027609) *Contributors:* Michael Hardy, Kku, CALR, Rgdboer, Culix, BD2412, Reetep, Me and, Grubber, SmackBot, RDBury, Eskimbot, J. Finkelstein, Perfectblue97, CmdrObot, Codetiger, Dougher, Vanish2, AV-2, Mayur82, Chiswick Chap, ClueBot, B2smith, Addbot, SpBot, Jim1138, Hessamnia, RjwilmsiBot, Grv1007, ClueBot NG, Jack Greenmaven, Elizabeth4494, Snake of Intelligent Ignorance, BattyBot, Sharma4prasad, Sandeep.ps4, Monkbob, CAPTAIN RAJU and Anonymous: 26
- **Deletion channel** *Source:* [https://en.wikipedia.org/wiki/Deletion\\_channel?oldid=731430763](https://en.wikipedia.org/wiki/Deletion_channel?oldid=731430763) *Contributors:* A3nm, David Eppstein, Addbot, Yobot, Timeroot, AnomieBOT, SBaker43, Hypergraph, Tongxinli and Anonymous: 3
- **Distributed source coding** *Source:* [https://en.wikipedia.org/wiki/Distributed\\_source\\_coding?oldid=762597637](https://en.wikipedia.org/wiki/Distributed_source_coding?oldid=762597637) *Contributors:* Michael Hardy, Dcoetzee, Gaius Cornelius, SmackBot, Chris the speller, Dicklyon, GiantSnowman, Dougher, Sigmundpetersen, Phsamuel, Hailangsea~enwiki, Addbot, Mortense, Incraton, Yobot, AnomieBOT, Shadowjams, FrescoBot, Rc3002, Fangyong, Ulatekh, Ubcodeingtheory, ClueBot NG, Timtb, InternetArchiveBot, GreenC bot and Anonymous: 10



- **Dual code** *Source:* [https://en.wikipedia.org/wiki/Dual\\_code?oldid=724475539](https://en.wikipedia.org/wiki/Dual_code?oldid=724475539) *Contributors:* Revolver, Charles Matthews, Fropuff, TenOfAllTrades, Culix, MacRusgail, Reetep, Chris the speller, Wandrer2, Thijs!bot, Vanish2, Wpolly, Niceguyedc, WikHead, Addbot, Citation bot, FrescoBot, BattyBot, Primetimer, Nmayerho~enwiki, Monkbob and Anonymous: 4
- **Elias Bassalygo bound** *Source:* [https://en.wikipedia.org/wiki/Elias\\_Bassalygo\\_bound?oldid=737029625](https://en.wikipedia.org/wiki/Elias_Bassalygo_bound?oldid=737029625) *Contributors:* Michael Hardy, Rjwilmsi, Wikid77, Hermel, WereSpielChequers, Non-dropframe, Yobot, FrescoBot, Xeransis, Latex-yow and Anonymous: 2
- **Enumerator polynomial** *Source:* [https://en.wikipedia.org/wiki/Enumerator\\_polynomial?oldid=750554053](https://en.wikipedia.org/wiki/Enumerator_polynomial?oldid=750554053) *Contributors:* Charles Matthews, Giftlite, Kbdank71, DagErlingSmørgrav, Vanish2, Sharov, Addbot, Citation bot, BattyBot and Anonymous: 4
- **Erasure code** *Source:* [https://en.wikipedia.org/wiki/Erasure\\_code?oldid=762029114](https://en.wikipedia.org/wiki/Erasure_code?oldid=762029114) *Contributors:* Michael Hardy, Charles Matthews, Phr, Phil Boswell, BenFrantzDale, Everyking, Mboverload, Matt Crypto, Neilc, Agl~enwiki, Nroets, Linas, Mirror Vax, Ysangkok, SmackBot, Nbarth, Radagast83, SeanAhern, Derek farn, Green caterpillar, Kozuch, Stangaa, Magioladitis, Osndok, Rogerdpack, Uvaraj6, Svick, Randallbsmith, Nnemo, John.sokol, Alexbot, Dachary, Dsimic, Addbot, Cunchem, Nageh, OgreBot, Akim Demaille, Woodlot, Jimw338, Mchen.ja, Pintoeh, Mtlindow, LCS check, Aks4751, LaLoo and Anonymous: 24
- **Even code** *Source:* [https://en.wikipedia.org/wiki/Even\\_code?oldid=633318163](https://en.wikipedia.org/wiki/Even_code?oldid=633318163) *Contributors:* Topbanana, Fropuff, MarSch, Malcolm, SmackBot, Melchoir, Chris the speller, CorbinSimpson, Cydebot, Alaiobot, Katharineamy, 777sms, Vieque and Anonymous: 2
- **Expander code** *Source:* [https://en.wikipedia.org/wiki/Expander\\_code?oldid=674915304](https://en.wikipedia.org/wiki/Expander_code?oldid=674915304) *Contributors:* Michael Hardy, Icairns, Photonique, RDBrown, Ylooh, David Eppstein, R'n'B, Niceguyedc, AnomieBOT, John of Reading, Dpadgett, Meatsgains, Harikine, Lcode007, Srednuas Lenoroc and Anonymous: 1
- **Factorization of polynomials over finite fields** *Source:* [https://en.wikipedia.org/wiki/Factorization\\_of\\_polynomials\\_over\\_finite\\_fields?oldid=754661023](https://en.wikipedia.org/wiki/Factorization_of_polynomials_over_finite_fields?oldid=754661023) *Contributors:* Michael Hardy, Bearcat, Tea2min, MathKnight, Hydrox, Pontus, Sperril, Walkie, Efnar, MvH, Petr Matas, Headbomb, Asmeurer, JoergenB, R'n'B, Demokratickid, JL-Bot, Jan1nad, Mild Bill Hiccup, Arjayay, Rcampbell1, Marc van Leeuwen, Jncraton, Yobot, DutchCanadian, AnomieBOT, LilHelpa, Stbuehler, FrescoBot, TLange, Sławomir Biały, Masahiro Sakai, John of Reading, D.Lazard, SilentBobby2, Wcherowi, Crypto2010, AALProject2010, Amdeberhan, Joechem77, Helpful Pixie Bot, Zieglerk, Regeadowdy, Teddyktchan, Some Gadget Geek and Anonymous: 31
- **First Johnson bound** *Source:* [https://en.wikipedia.org/wiki/Johnson\\_bound?oldid=747143446](https://en.wikipedia.org/wiki/Johnson_bound?oldid=747143446) *Contributors:* Michael Hardy, Giftlite, Pierremenard, SmackBot, Synergy, Wikid77, Magioladitis, STBot, Yobot, Omnipaedista, ZéroBot, Matthiaspaul, KLBOT2 and Anonymous: 1
- **Folded Reed–Solomon code** *Source:* [https://en.wikipedia.org/wiki/Folded\\_Reed%E2%80%93Solomon\\_code?oldid=741382271](https://en.wikipedia.org/wiki/Folded_Reed%E2%80%93Solomon_code?oldid=741382271) *Contributors:* Michael Hardy, Eyreland, Glrx, Mild Bill Hiccup, Yobot, Bobmath, John of Reading, Dewritech, Frietjes, BG19bot, McZusatz, Madhurasuresh, Latex-yow and Anonymous: 2
- **Forney algorithm** *Source:* [https://en.wikipedia.org/wiki/Forney\\_algorithm?oldid=668913433](https://en.wikipedia.org/wiki/Forney_algorithm?oldid=668913433) *Contributors:* Michael Hardy, Conquerist, Glrx, Sun Creator, Addbot, Luckas-bot, Bobmath, BG19bot, Lrq3000, BattyBot, Mark viking and Hippo.69
- **Fuzzy extractor** *Source:* [https://en.wikipedia.org/wiki/Fuzzy\\_extractor?oldid=684432696](https://en.wikipedia.org/wiki/Fuzzy_extractor?oldid=684432696) *Contributors:* Edward, Michael Hardy, Bearcat, Wavelength, Stormbay, Nick Number, Heracleus, AnomieBOT, LilHelpa, John of Reading, Biernat2, Bjr244 and Anonymous: 11
- **Generalized minimum-distance decoding** *Source:* [https://en.wikipedia.org/wiki/Generalized\\_minimum-distance\\_decoding?oldid=737820594](https://en.wikipedia.org/wiki/Generalized_minimum-distance_decoding?oldid=737820594) *Contributors:* Michael Hardy, Sidjaggi, Mandarax, Tony1, SmackBot, Katharineamy, Lambtron, FrescoBot, Nageh, Tom.Reding, JamietwBot, Yeonju, ChrisGualtieri, Latex-yow and Anonymous: 1
- **Generator matrix** *Source:* [https://en.wikipedia.org/wiki/Generator\\_matrix?oldid=750422475](https://en.wikipedia.org/wiki/Generator_matrix?oldid=750422475) *Contributors:* Billymac00, Culix, RussBot, Gareth Jones, SmackBot, Bluebot, MaxSem, MrZap, P.L.A.R., JAnDbot, LordAnubisBOT, Peskydan, Gamall Wednesday Ida, Addbot, Luckas-bot, Cunchem, ArthurBot, John of Reading, Wcherowi, Frietjes, Theoneandonlysigma, Mark viking and Anonymous: 12
- **Gibbs' inequality** *Source:* [https://en.wikipedia.org/wiki/Gibbs%27\\_inequality?oldid=762795940](https://en.wikipedia.org/wiki/Gibbs%27_inequality?oldid=762795940) *Contributors:* Michael Hardy, Giftlite, Burn, Jheald, Nigosh, Srleffler, Reetep, SmackBot, Maksim-e~enwiki, MclD, Bluebot, Chungc, Thermochap, D.H, ReviewDude, Mild Bill Hiccup, Addbot, MARijlaarsdam, SpBot, Lightbot, RVS, Erik9bot, JimbobHolton, Vieque and Anonymous: 18
- **Gilbert–Varshamov bound** *Source:* [https://en.wikipedia.org/wiki/Gilbert%E2%80%93Varshamov\\_bound?oldid=737027189](https://en.wikipedia.org/wiki/Gilbert%E2%80%93Varshamov_bound?oldid=737027189) *Contributors:* Michael Hardy, Fredrik, Giftlite, Gene Nygaard, Pierremenard, Rjwilmsi, Reetep, SmackBot, Wikid77, Kiwanda, David Eppstein, Sharov, Ulamgamer, Addbot, Uri.laserson, Omnipedian, Luckas-bot, DirlBot, Omnipaedista, Erik9bot, FrescoBot, Darij, AvicAWB, Widr, JavaDfan, SteenthWbot, Overcapata, Some1Redirects4You, Latex-yow and Anonymous: 13
- **Goppa code** *Source:* [https://en.wikipedia.org/wiki/Goppa\\_code?oldid=744601432](https://en.wikipedia.org/wiki/Goppa_code?oldid=744601432) *Contributors:* The Anome, Charles Matthews, Xiaodai~enwiki, Giftlite, Gene.arboit, SmackBot, G716, CBM, Rudolf Schürer, He7d3r, Addbot, InsufficientData, Jayarathina, Cryptohoolligans4life, I dream of horses, Onoes, D.Lazard, ClueBot NG, Exaexa, Latex-yow and Anonymous: 11
- **Grammar-based code** *Source:* [https://en.wikipedia.org/wiki/Grammar-based\\_code?oldid=762028206](https://en.wikipedia.org/wiki/Grammar-based_code?oldid=762028206) *Contributors:* Edward, Michael Hardy, Giftlite, Bender235, TheParanoidOne, RHaworth, Galwhaa, Rjwilmsi, Robertvan1, SmackBot, Dake.he, Frap, CmdrObot, Amanbhatia, Tinucherian, A3nm, Wdflake, JohnicholasHines, BenTels, Jamelan, DragonBot, Addbot, Yobot, Citation bot 1, Rjwilmsi-Bot, Seninp, Pomax, John of Reading, George Ponderevo, Grammar-master, Jochen Burghardt and Anonymous: 8
- **Gray isometry** *Source:* [https://en.wikipedia.org/wiki/Gray\\_code?oldid=765473948](https://en.wikipedia.org/wiki/Gray_code?oldid=765473948) *Contributors:* XJaM, Matusz, PierreAbbat, Heron, Tomo, RTC, Michael Hardy, Shellreef, Wapcaplet, Ahoerstermeier, Cyp, Ronz, Bogdangiusca, BAXelrod, Charles Matthews, Dcoetzee, RayKiddy, Pakaran, David.Monnaux, Denelson83, Phil Boswell, Cholling, Nick Pizarro, Jr., Dina, Giftlite, DavidCary, MSGJ, Wwoods, Elias, Mellum, Leonard G., Noe, MarkSweep, Wikkrockiana, Rich Farmbrough, Bender235, ZeroOne, Plugwash, El C, AJP, Smalljim, RobH (2004 account), Pacifier, Haham hanuka, Guy Harris, Ricky81682, Demi, Snowolf, Wtshymanski, Emvee~enwiki, Cburnett, Su-ruena, Pgimeno~enwiki, Egg, Ahseaton, Linas, TheNightFly, MattGiuca, Acerperi, Gerbrant, BD2412, Rjwilmsi, Seraphimblade, Salix alba, Nneonneo, Isaiah Schwartz, Bubba73, Maxal, GünniX, Fresheneesz, YurikBot, RussBot, Pi Delpot, Iamfscked, Yahya Abdal-Aziz, Hv, Plamka, Mike1024, JLaTondre, Vanka5, SmackBot, Slashme, Bggoldie~enwiki, McGeddon, Zeycus, Isaac Dupree, MooMan1, Chris the speller, Droll, Sciyoshi~enwiki, Hgrosner, Johnmorgan, Jjbeard~enwiki, Tamfang, DMacks, Yoshigev, Lambiam, Sjock, Ocatecir, Kjk-java, Jankratochvil, Dicklyon, Iridescent, Tawkerbot2, Sakurambo, PuerExMachina, Requestion, Penbat, Cydebot, A876, Tawkerbot4, Thijs!bot, Wikid77, Plesner, Qwyrxian, ChrisEich, Headbomb, Djfeldman, Bobblehead, JAnDbot, MER-C, CosineKitty, BrotherE, Nyq, Vanish2, Tedickey, Wikipodium, Nikevich, Jos.koot, David Eppstein, Gwern, Edratzer, Glrx, Kateshortforbob, Jaiguru42, Dispenser, KylieTastic, Daxx wp, STBotD, Jturkia, SuneJ~enwiki, Pleasantville, AlnoktaBOT, Elphion, Jeff 113, BotKung, Inductiveload, Andy Dingley, SieBot, Maxalot, Ronald S. Davis, Winzurf, Flyer22 Reborn, Myrddine, Animagi1981, Svick, Hariva, Kanesue, ClueBot, The Thing



That Should Not Be, Max711, Mild Bill Hiccup, Watchduck, Sun Creator, Johnuniq, Seantellis, XLinkBot, Ohiosoil, Addbot, DOI bot, TutterMouse, MrOllie, Gnorthup, Lightbot, Luckas-bot, Pbotgourou, Jason Recliner, Esq., Kilom691, AnomieBOT, Jim1138, Law, MaterialsScientist, Citation bot, LilHelpa, Xqbot, Jan olieslagers, RibotBOT, Charvest, Oashii, Machine Elf 1735, Citation bot 1, Bunyk, Skyerise, Dinamik-bot, Nevin.williams, RjwilmsiBot, EmausBot, AileTheAlien, Deacs33, GoingBatty, Klbrain, Dr0b3rts, Westley Turner, H3llBot, Fcdesign, ClueBot NG, Iiii I I I, Matthiaspaul, Frietjes, Helpful Pixie Bot, Prashantgonarkar, SciCompTeacher, Toploftical, ChrisGualtieri, Pintoch, Andyhowlett, Lbaralgeen, Tomcrux, VoluminousComputer, Mythas11, Cousteau, AKS.9955, Jackal 1586, Some1Redirects4You, KSFT, Catchiaro, Bender the Bot, 00ff00 and Anonymous: 182

- **Griesmer bound** *Source:* [https://en.wikipedia.org/wiki/Griesmer\\_bound?oldid=737001574](https://en.wikipedia.org/wiki/Griesmer_bound?oldid=737001574) *Contributors:* Michael Hardy, Giftlite, Texas-Android, SmackBot, Crecy99, Omnipaedista, VanceIII, Darij, Latex-yow and Anonymous: 4
- **Group code** *Source:* [https://en.wikipedia.org/wiki/Group\\_code?oldid=678509836](https://en.wikipedia.org/wiki/Group_code?oldid=678509836) *Contributors:* Netpilot43556, DRE, Photonique, Linas, Grubber, LouScheffer, NewWikiMan, Addbot, AnomieBOT, Quondum, ClueBot NG, Nlives1, Dexbot and Anonymous: 4
- **Guruswami–Sudan list decoding algorithm** *Source:* [https://en.wikipedia.org/wiki/Guruswami%E2%80%93Sudan\\_list\\_decoding\\_algorithm?oldid=753936518](https://en.wikipedia.org/wiki/Guruswami%E2%80%93Sudan_list_decoding_algorithm?oldid=753936518) *Contributors:* Michael Hardy, Bearcat, Stuartyeates, GregorB, Salix alba, Malcolma, David Eppstein, Bbukh, Yobot, FrescoBot, Jamiw, John of Reading, Dewritech, ArticlesForCreationBot, Stausifr, Hiteshjain87, Kiran040486, Rhtovo and Anonymous: 8
- **GV-linear-code** *Source:* <https://en.wikipedia.org/wiki/GV-linear-code?oldid=742324085> *Contributors:* Michael Hardy, Andrewman327, Bearcat, Brianhe, SmackBot, Gr8tmir~enwiki, Katharineamy, Carriearchdale, OIEnglish, Yobot, Jjauhien, AnomieBOT, Dan6hell66, FrescoBot, JamiwBot, JavaDfan, BG19bot, BattyBot, Sequoia 42, Overcapata, Latex-yow and Anonymous: 1
- **Hadamard code** *Source:* [https://en.wikipedia.org/wiki/Hadamard\\_code?oldid=749301530](https://en.wikipedia.org/wiki/Hadamard_code?oldid=749301530) *Contributors:* Awaterl, Michael Hardy, Phil Boswell, Sander123, Tea2min, Giftlite, Will Orrick, Eyreland, Grubber, RDBrown, Oli Filth, Ylooh, Chrisahn, Alaibot, Defective, Horacelamb, Infrangible, Hpfister, R'n'B, CommonsDelinker, Marcosaedro, EverGreg, Amanda Breckenridge, Mizst, PipepBot, Watchduck, Addbot, Lightbot, Yobot, AnomieBOT, Andrewrp, LilHelpa, J04n, Cpflienger, HROestBot, RjwilmsiBot, Chris.c.keller, Cleo, BG19bot, Pintoch, Stormmilk, Comp.arch, OccultZone, Frank Klemm, TheSawTooth, Mahmoudmnsor and Anonymous: 22
- **Hamming bound** *Source:* [https://en.wikipedia.org/wiki/Hamming\\_bound?oldid=713575236](https://en.wikipedia.org/wiki/Hamming_bound?oldid=713575236) *Contributors:* The Anome, Michael Hardy, Ix6d64, Charles Matthews, Bearcat, Giftlite, Mpeisenbr, Diego Moya, Culix, Pierremenard, BD2412, Reetep, Malcolma, SmackBot, Oli Filth, CmdrObot, Thijs!bot, Hermel, Drizzd~enwiki, Vanish2, David Eppstein, Jamelan, Sharov, Wdwd, Mild Bill Hiccup, Addbot, Luckas-bot, Kilom691, Piano non troppo, Citation bot, VanceIII, Citation bot 1, GoingBatty, Wcherowi, Rezabot, Schjora, Monkbob and Anonymous: 16
- **Hamming code** *Source:* [https://en.wikipedia.org/wiki/Hamming\\_code?oldid=760494065](https://en.wikipedia.org/wiki/Hamming_code?oldid=760494065) *Contributors:* Ap, Malcolm Farmer, PierreAbbat, Maury Markowitz, Michael Hardy, JakeVortex, Chinju, Karada, Ffx, GregRobson, Timwi, Dcoetzee, Magnus.de, Dysprosia, Colin Marquardt, Populus, Bcorr, Garo, Sander123, Naddy, Lowellian, Bkell, Diberri, Tea2min, David Gerard, Giftlite, DavidCary, Dratman, Galaxy07, Gracefool, OverlordQ, Bob.v.R, Marc Mongenet, Sam Hocevar, Pjacobi, LindsayH, Antaeus Feldspar, ZeroOne, El C, Indil, Spoon!, Gatta, A-Day, Dmanning, Gerweck, Kmill, Wtmitchell, Cburnett, Kusma, KTC, Kasuga~enwiki, Robert K S, Sbrorlongan, Eyreland, Bilbo1507, BD2412, Liq, Kbdank71, Sjö, Rjwilmsi, Bubba73, JanSuchy, StuartBrady, Mathbot, Tas50, Bgwhite, YurikBot, TeeEmCee, Grubber, Hawkeye7, Vlad, ArielGold, Rwww, Attilios, SmackBot, Timrb, Mitchan, David.Mestel, Gilliam, Lozen130, Bluebot, Jjalexand, Oli Filth, Zelphar, DHN-bot~enwiki, Royboycrashfan, Can't sleep, clown will eat me, ElementFire, Minna Sora no Shita, Loadmaster, Tasc, Aboeing, Hu12, Majora4, Ylooh, Dokaspar, 345Kai, Blaisorblade, Thijs!bot, Barticus88, LachlanA, Widefox, Dylan Lake, Harish victory, Time3000, GreatEgret, Magioladitis, First Harmonic, DerHexer, ZekeG4, Jackson Peebles, Edratzer, Mange01, Mathuranathan, Joewin, Plounder, Gmoose1, TXiKiBot, Oshwah, Park70, Seraphim, Gekritzl, Brian Helsinki, McM.bot, Ray andrew, Lonioumonk, Df747jet, Daveofthenewcity, Michael Frind, Theoneintraining, SieBot, Glennecooper, Mac389, Bomot113, Svick, ClueBot, GreekHouse, Wwheaton, Anysquid, Doprendek, Kubek15, Crowsnest, Dsmic, Addbot, Cwitt, LeMonje, Tide rolls, Lightbot, Zorrobot, Margin1522, Legobot, Luckas-bot, Yobot, Pasmargo, Nallimbob, AnomieBOT, Iexel, MaterialsScientist, ArthurBot, Kliikakis, MagnusOre, Beekeepingschool, Jesse1013000, Killian441, AstaBOT15, Thinking of England, Cculpepper, EmausBot, RA0808, White Trillium, Ebrambot, Quondum, Omer.qadir, Demonkoryu, Donner60, Carmichael, Mikhail Ryazanov, ClueBot NG, Wcherowi, Millermk, Fferen, Padishar124, Trombonechamp, Marcogomex, Cimorcus, JYBot, Dexbot, SFK2, Alireza270, Babitaarora, Gmk7, Monkbob, Yi-Jun Chen, Shamiulovi, Kinetic37, Ecianne, Ne roll 10, Bisteca23, TheFredicator and Anonymous: 234
- **Hamming distance** *Source:* [https://en.wikipedia.org/wiki/Hamming\\_distance?oldid=766689831](https://en.wikipedia.org/wiki/Hamming_distance?oldid=766689831) *Contributors:* Damian Yerrick, Ap, Pit~enwiki, Kku, Axlrosen, Kevin Baas, Poor Yorick, Dcoetzee, Silvonen, David Shay, Altenmann, Wile E. Heresiarch, Tosha, Giftlite, Seabhean, BenFrantzDale, Markus Kuhn, CryptoDerk, Beland, Gene s, Mindspillage, Leibniz, Zaslav, Danakil, Aaronbrick, Blotwell, Flammiifer, 3mta3, Aoldsoldier, Obradovic Goran, ABCD, Pouya, Cburnett, Wsloand, Joepzander, Linas, Kasuga~enwiki, Ruud Koot, Zelse81, Qwertyus, Rjwilmsi, Tizio, Pentolith, Mathbot, Margosbot~enwiki, Quuxplusone, Bgwhite, YurikBot, Personman, Michael Slone, Armistej, Archelon, Alcides, Ttam, Zwobot, Attilios, SmackBot, BiT, Bluebot, DHN-bot~enwiki, Scray, Frap, Decltype, DM-Palmer, Slach~enwiki, SashatoBot, Lambiam, Shir Khan, Loadmaster, ThePacker, DagErlingSmørgrav, ChetTheGray, Eastlaw, CR-Greathouse, Krauss, Thijs!bot, Headbomb, Wainson, Fulkkari~enwiki, Adma84, JAnDbot, Sterrys, JPG-GR, David Eppstein, JMyrleFuller, ANONYMOUS COWARD0xCODE, Ksero, Glrx, DorganBot, JohnBlackburne, Boxofbox, Lixo2, Sue Rangel, Svick, Hhbruun, Thegeneralguy, TSylvester, DragonBot, Alexbot, SchreiberBike, Muro Bot, Cerireid, Oğuz Ergin, MystBot, Addbot, LaaknorBot, Gnorthup, Ramses68, Lightbot, Math Champion, Luckas-bot, Pbotgourou, AnomieBOT, Joule36e5, MaterialsScientist, RibotBOT, Citation bot 1, I dream of horses, Kiefer.Wolfowitz, Hardwigg, Compish, Ripchip Bot, Valyt, EmausBot, Olof nord, Juanfal, Froch514, Jnaranjo86, Wcherowi, Sumanah, Jcarrete, Zorro1024, Jcherman, Vival007b, Wkschwartz, Tiagofrepereira, Rubenaodom, GreyNanook, ScrapIronIV, Some1Redirects4You, Baba Arouj, SebastianStiles, Fmadd and Anonymous: 94
- **Hamming scheme** *Source:* [https://en.wikipedia.org/wiki/Hamming\\_scheme?oldid=602969365](https://en.wikipedia.org/wiki/Hamming_scheme?oldid=602969365) *Contributors:* Michael Hardy, Jitse Niesen, CardinalDan, Yobot, FrescoBot, Krptnht and Luismanu
- **Hamming space** *Source:* [https://en.wikipedia.org/wiki/Hamming\\_space?oldid=743784024](https://en.wikipedia.org/wiki/Hamming_space?oldid=743784024) *Contributors:* The Anome, HappyCamper, Maustraiser, Phantomsteve, David Eppstein, Some1Redirects4You, Bender the Bot and Anonymous: 3
- **Hamming weight** *Source:* [https://en.wikipedia.org/wiki/Hamming\\_weight?oldid=751828858](https://en.wikipedia.org/wiki/Hamming_weight?oldid=751828858) *Contributors:* Dcoetzee, Giftlite, Markus Kuhn, CryptoDerk, Andreas Kaufmann, RossPatterson, Jonathancamp, Aperculum, Bender235, Spoon!, Remuel, Blotwell, Oleg Alexandrov, Kasuga~enwiki, GregorB, BD2412, Qwertyus, Jpkotta, RussBot, Amakuha, Cedar101, That Guy, From That Show!, SmackBot, Mmernex, BiT, Elliott Drabek, Bluebot, Racklever, Iammisc, JonHarder, Allansteel, Chrylis, Lambiam, Shir Khan, ChadyWady, Walling, Fireice, Sterrys, Sheda, David Eppstein, Gwern, Jrodor, VolkovBot, Jbradfor, Anakin101, MichaelWattam, Avenged Eightfold, Justin W

- Smith, Niceguyedc, IMneme, Cerireid, XLinkBot, KindDragon33, Dekart, Wyatt915, Addbot, Ramses68, Yobot, FUZxxl, Kstueve, FrescoBot, DrilBot, Pinethicket, Pcg01, Txt.file, Sbmeirow, Wcherowi, Haitising, Diegohavenstein, Nigelito, Fruitinspace, ESAD-Hooker and Anonymous: 40
- **Hamming(7,4)** Source: [https://en.wikipedia.org/wiki/Hamming\(7%2C4\)?oldid=763647514](https://en.wikipedia.org/wiki/Hamming(7%2C4)?oldid=763647514) Contributors: Nerd65536, ZeroOne, Cburnett, Suruena, Bubba73, Bgwhite, Grubber, Hmains, Oli Filth, NeoDeGenero, Ylloh, CmdrObot, Thijs!bot, Adhityan, Darklilac, Swpb, JSSX, Camrn86, Cassandra B~enwiki, WIFF-FM, GreekHouse, CalculIntegral, Addbot, Minkythecat, Djllloy1, Wiso, Yobot, Wikipedian Penguin, Nummify, AnomieBOT, Galoubet, GrouchoBot, Beekeepingschool, Alexander Chervov, Dixtosa, ClueBot NG, Rimesime, AVC1380, Dexbot, Babitaarora, Bryanrutherford0 and Anonymous: 43
  - **Hexacode** Source: <https://en.wikipedia.org/wiki/Hexacode?oldid=618843789> Contributors: Fropuff, SmackBot, RJChapman, T@nn, AlexNewArtBot, Jason.galazidis, Mark viking, Melonkelon, Monkbob and Anonymous: 2
  - **Homomorphic signatures for network coding** Source: [https://en.wikipedia.org/wiki/Homomorphic\\_signatures\\_for\\_network\\_coding?oldid=553018437](https://en.wikipedia.org/wiki/Homomorphic_signatures_for_network_coding?oldid=553018437) Contributors: Michael Hardy, Mild Bill Hiccup, Jovianeye, AnomieBOT, BG19bot, Karthik.sk2001 and Anonymous: 2
  - **Johnson bound** Source: [https://en.wikipedia.org/wiki/Johnson\\_bound?oldid=747143446](https://en.wikipedia.org/wiki/Johnson_bound?oldid=747143446) Contributors: Michael Hardy, Giftlite, Pierre-menard, SmackBot, Synergy, Wikid77, Magioladitis, STBot, Yobot, Omnipaedista, ZéroBot, Matthiaspaul, KLBot2 and Anonymous: 1
  - **Justesen code** Source: [https://en.wikipedia.org/wiki/Justesen\\_code?oldid=737337522](https://en.wikipedia.org/wiki/Justesen_code?oldid=737337522) Contributors: Michael Hardy, Giftlite, RHaworth, Eyreland, Qwertyus, Ylloh, Headbomb, Magioladitis, Vanish2, WereSpielChequers, Bender2k14, Addbot, DOI bot, Yobot, Citation bot, GrouchoBot, Omnipaedista, FrescoBot, Citation bot 1, Darij, Ebe123, JavaDfan, Qetuth, Monkbob, Latex-yow and Anonymous: 1
  - **Kraft-McMillan inequality** Source: [https://en.wikipedia.org/wiki/Kraft%E2%80%93McMillan\\_inequality?oldid=753378184](https://en.wikipedia.org/wiki/Kraft%E2%80%93McMillan_inequality?oldid=753378184) Contributors: Michael Hardy, Ixfd64, Charles Matthews, Dina, Giftlite, Roo72, Spoon!, Pearle, Jheald, Xiaoyangu, Oleg Alexandrov, Milez, Arunkumar, Mikm, Reetep, YurikBot, Danthexox, David Pal, Javalenok, FilipeS, ElPoojmar, Thijs!bot, Heysan, David Eppstein, Twotonkatrucks, Camrn86, Akshatsinghal, Vitz-RS, Leena34, Gamall Wednesday Ida, Mko3okm, Mjoachimiak, Addbot, Yobot, Li3939108, Citation bot, Xqbot, Mr.gondolier, D'ohBot, Darij, MaxDel, MondalorBot, Trappist the monk, Patmorin, RjwilmsiBot, ChuispastonBot, Ahmahran and Anonymous: 21
  - **Lee distance** Source: [https://en.wikipedia.org/wiki/Lee\\_distance?oldid=732133418](https://en.wikipedia.org/wiki/Lee_distance?oldid=732133418) Contributors: Michael Hardy, Giftlite, Urhixidur, Rich Farmbrough, YK Times, David Eppstein, Glrx, Hailangsea~enwiki, Justin W Smith, Addbot, Ramses68, AnomieBOT, Fortdj33, DivineAlpha, Helpful Pixie Bot, Frank Klemm, Some1Redirects4You and Anonymous: 5
  - **Linear code** Source: [https://en.wikipedia.org/wiki/Linear\\_code?oldid=762026869](https://en.wikipedia.org/wiki/Linear_code?oldid=762026869) Contributors: Michael Hardy, Charles Matthews, Sander123, Giftlite, Fropuff, Bob.v.R, Abdull, Paul August, Zaslav, V79, Oleg Alexandrov, Linas, Shreevatsa, Will Orrick, BD2412, Qwertyus, Rjwilmsi, Maxal, Reetep, Gene.arboit, AeOnflx, EAdherhold, Heathhunnicut, CorbinSimpson, Allansteel, Stalkerster, Ylloh, Chrisahn, Dougher, P.L.A.R., Edratzer, DavidCBryant, Selinger, TXiKiBoT, Trachten, Uurtamo, Jamelan, ClueBot, Arjayay, Ljbell1, Addbot, Lucas-bot, Yobot, Sivanov87, AnomieBOT, Citation bot, Isheden, FrescoBot, Nageh, Mhadi.afraasiabi, I dream of horses, Alfaisanomega, Xeransis, Wcherowi, Frietjes, Helpful Pixie Bot, Some1Redirects4You, Bender the Bot, Asamov and Anonymous: 33
  - **List decoding** Source: [https://en.wikipedia.org/wiki/List\\_decoding?oldid=741224889](https://en.wikipedia.org/wiki/List_decoding?oldid=741224889) Contributors: Michael Hardy, Andreas Kaufmann, D6, Jayjg, BD2412, SmackBot, Dicklyon, Yuide, Wizard191, Ylloh, Blaisorblade, Arch dude, David Eppstein, Signalhead, Emeritusl, The Thing That Should Not Be, C. A. Russell, Addbot, LaaknorBot, LilHelpa, Anonash, Locobot, KrisRetroVirus, Snotbot, BG19bot, Madhurasuresh, Srednuas Lenoroc, Latex-yow and Anonymous: 8
  - **Long code (mathematics)** Source: [https://en.wikipedia.org/wiki/Long\\_code\\_\(mathematics\)?oldid=672015415](https://en.wikipedia.org/wiki/Long_code_(mathematics)?oldid=672015415) Contributors: Bearcat, Gilliam, Ylloh, David Eppstein, Tcshasaposse, Qetuth, Baracuda911, Voidmainup and Anonymous: 2
  - **Low-density parity-check code** Source: [https://en.wikipedia.org/wiki/Low-density\\_parity-check\\_code?oldid=755804219](https://en.wikipedia.org/wiki/Low-density_parity-check_code?oldid=755804219) Contributors: The Anome, Michael Hardy, Stevenj, Charles Matthews, Reina riemann, Mrand, Giftlite, DavidCary, Joconnor, Jpp, Tagishsimon, Edcolins, Nerd65536, Emin63, Dmeranda, ZeroOne, Kop, Photonique, Rd232, Emvee~enwiki, Alvis, Linas, Eyreland, Vegaswikian, Quuxplu-sone, Grubber, Gaius Cornelius, Nahaj, Zvika, SmackBot, Chris the speller, Alistair9210, Oli Filth, OrphanBot, FilippoSidoti, Joey-das-WBF, Dicklyon, Kvng, Pfagerburg~enwiki, Jesse Viviano, BrettOliver, M.Marangio, Bposert, Longhai, Greg L, Esromneb, DekuDekuplex, Em3ryguy, Arch dude, Magioladitis, First Harmonic, Mayur82, Edratzer, Rettetast, Try0yrt, Ohms law, Xavier Giró, MptMan, Yonch, AlleborgoBot, SieBot, TJRC, Vlsergey, Lightmouse, Trojancowboy, Mild Bill Hiccup, Arjayay, HumphreyW, Rror, Addbot, Esteban-valles, Dfhayes, Proton donor, MurphysLaw58, Waveletrules, Cjerrells, Lucas-bot, Yobot, Galoubet, Cunchem, Drilnoth, DataWraith, GrouchoBot, Aminrahimian, Nageh, W Nowicki, Advancedtelcotv, Itusgl5q4user, UteFan16, Btlm, Fcy, John of Reading, Ubcodingtheory, Pyecello, ClueBot NG, Bigeatable, Dpadgett, A\*-search, YFdyh-bot, Hmainsbot1, Ajraymond, Wilhelm-Conrad, Snookerr, Melcous, Mrccodeguy, Abhay.s29, JosVan, Asamov and Anonymous: 84
  - **Luby transform code** Source: [https://en.wikipedia.org/wiki/Luby\\_transform\\_code?oldid=607161317](https://en.wikipedia.org/wiki/Luby_transform_code?oldid=607161317) Contributors: Michael Hardy, Samw, Ehn, Linas, Daira Hopwood, Ntsimp, DavidCBryant, Melcombe, Yobot, Cunchem, Reiskopf, DrilBot, The mike luby, 28bot and Anonymous: 14
  - **Minimum weight** Source: [https://en.wikipedia.org/wiki/Minimum\\_weight?oldid=733399250](https://en.wikipedia.org/wiki/Minimum_weight?oldid=733399250) Contributors: Michael Hardy, Srleffler, AnOddName, Jj137, Misarxist, Deor, Cerireid and Anonymous: 3
  - **Multiple description coding** Source: [https://en.wikipedia.org/wiki/Multiple\\_description\\_coding?oldid=711269190](https://en.wikipedia.org/wiki/Multiple_description_coding?oldid=711269190) Contributors: Edward, Michael Hardy, Andrewman327, Marudubshinki, Malcolma, Dv82matt, Stuinzuri, SmackBot, SB Johnny, OrphanBot, CmdrObot, Amalas, LachlanA, Gwern, Addbot, Yobot, Erik9bot, Rushtofire, BG19bot, Eustrat Zhupa, Ali fa1360 and Anonymous: 7
  - **Linear network coding** Source: [https://en.wikipedia.org/wiki/Linear\\_network\\_coding?oldid=762696259](https://en.wikipedia.org/wiki/Linear_network_coding?oldid=762696259) Contributors: Bryan Derksen, The Anome, Edward, Michael Hardy, Charles Matthews, Nv8200pa, Rich Farmbrough, Photonique, SteinbDJ, Joriki, Kelly Martin, ^demon, Rjwilmsi, MarSch, Who, Grubber, Howcheng, Ott2, SmackBot, OrphanBot, JDipierro, JonHarder, A5b, Harryboyles, Swellesley, Ckngai, Cbuckley, CmdrObot, Lark ascending, Jackiechen01, Bluto29, Ckhung, David Eppstein, Conquerist, Doctahdrey, Maybenot-mart, TXiKiBoT, Skydoor, EoGuy, Dpmuk, Mhfirooz, DnetSvg, Duleorlovic, Time2zone, H.Marxen, DumZiBoT, Addbot, Grayfell, DOI bot, Solrex, Ettrig, Lucas-bot, Yobot, Jalal0, AnomieBOT, Piano non troppo, Xizhi.zhu, Pixelpapst, Alex.kshevetskiy, Green Cardamom, Venkatac, Citation bot 1, HRoestBot, Trappist the monk, Nordbad, FrankieMoran, ClueBot NG, BG19bot, Karthik.sk2001, ChrisGualtieri, Jqureshi786, Namnatulco, Cocph, Ali fa1360, Vikifor57, Hamlinb, Ishan1992 and Anonymous: 59
  - **Noisy text** Source: [https://en.wikipedia.org/wiki/Noisy\\_text?oldid=678752793](https://en.wikipedia.org/wiki/Noisy_text?oldid=678752793) Contributors: Michael Hardy, Kku, Malcolma, SmackBot, Alaibot, Leolaursen, Lvsubram, Fabrictramp, Yobot, Fortdj33 and Rayna Jaymes

- **Noisy-channel coding theorem** *Source:* [https://en.wikipedia.org/wiki/Noisy-channel\\_coding\\_theorem?oldid=753809655](https://en.wikipedia.org/wiki/Noisy-channel_coding_theorem?oldid=753809655) *Contributors:* The Anome, Michael Hardy, Kku, Phr, Bkell, Giftlite, Spoon!, Smalljim, Photonique, LutzL, Jheald, Linas, John Baez, Bornhj, Welsh, Deodar~enwiki, BertK, Voidxor, SmackBot, C.Fred, Calbaer, DRLB, Chungc, Mental Blank, Dicklyon, CRGreathouse, Trellis, Pulkitgrover, Cydebot, NotQuiteEXPComplete, WikiIT, Iseetho, Hai Dang Quang~enwiki, WikiC~enwiki, The prophet wizard of the crayon cake, Wullj, Mange01, Bahram.zahir, VolkovBot, LokiClock, Alinja, Saibod, Geometry guy, AlleborgoBot, Jean-Christophe BENOIST, Galoiserdos, COBot, Melcombe, ClueBot, Humanenrg, SilvononBot, Addbot, Zotzilaha, Softy, Yobot, Ptbotgourou, Looby45, Materials scientist, Vididsense, Marcelo.jimenez, Quondum, Snotbot, BG19bot, AvocadoBot, Manoguru, Î½, Mimblerouser, InfoTheorist and Anonymous: 44
- **Online codes** *Source:* [https://en.wikipedia.org/wiki/Online\\_codes?oldid=648509901](https://en.wikipedia.org/wiki/Online_codes?oldid=648509901) *Contributors:* Charles Matthews, Nikita Borisov~enwiki, MichaelLeonhard, Agl~enwiki, Linas, Bluebot, CmdrObot, Richardguk, Explicit, Erel Segal, Mark viking, Comp.arch and Anonymous: 13
- **Package-merge algorithm** *Source:* [https://en.wikipedia.org/wiki/Package-merge\\_algorithm?oldid=733225584](https://en.wikipedia.org/wiki/Package-merge_algorithm?oldid=733225584) *Contributors:* Enochlau, Rjwilmsi, SmackBot, Od Mishehu, Calbaer, Dicklyon, Skapur, Serpent's Choice, Oravec, MwGamera, Vegasprof, DavidCBryant, Yobot, Matthiaspaul and Anonymous: 4
- **Packet erasure channel** *Source:* [https://en.wikipedia.org/wiki/Packet\\_erasure\\_channel?oldid=625745681](https://en.wikipedia.org/wiki/Packet_erasure_channel?oldid=625745681) *Contributors:* Edratzer, Mange01, Helpful Pixie Bot and Mark viking
- **Parity-check matrix** *Source:* [https://en.wikipedia.org/wiki/Parity-check\\_matrix?oldid=727612992](https://en.wikipedia.org/wiki/Parity-check_matrix?oldid=727612992) *Contributors:* Zundark, Bkell, Culix, RussBot, Grubber, PoorLeno, Ma8thew, Thijs!bot, Phosphoricx, Vanish2, David Eppstein, R'n'B, Trachten, IForTheMoney, Addbot, Samsam.yh, Loftwyr, GrouchoBot, Rc3002, Tom.Reding, John of Reading, Quondum, MSDousti, Wcherowi, BattyBot, Trismahesh and Anonymous: 14
- **Parvaresh-Vardy code** *Source:* [https://en.wikipedia.org/wiki/Parvaresh%E2%80%93Vardy\\_code?oldid=725143864](https://en.wikipedia.org/wiki/Parvaresh%E2%80%93Vardy_code?oldid=725143864) *Contributors:* Michael Hardy, Bearcat, J. Finkelstein, David Eppstein, Biscuitin, Yobot, Tom.Reding and Monkbot
- **Plotkin bound** *Source:* [https://en.wikipedia.org/wiki/Plotkin\\_bound?oldid=747144167](https://en.wikipedia.org/wiki/Plotkin_bound?oldid=747144167) *Contributors:* Michael Hardy, Skysmith, David Gerard, Giftlite, Dethron, Culix, Pierremenard, TigerShark, RussBot, SmackBot, Alaibot, Wikid77, Hermel, Magioladitis, David Eppstein, PipepBot, Addbot, VanceIII, Artem M. Pelenitsyn, Rc3002 and Anonymous: 14
- **Polar code (coding theory)** *Source:* [https://en.wikipedia.org/wiki/Polar\\_code\\_\(coding\\_theory\)?oldid=764086032](https://en.wikipedia.org/wiki/Polar_code_(coding_theory)?oldid=764086032) *Contributors:* Michael Hardy, Cheraghchi, DVdm, RDBrown, Harish victory, Raucanum, Kasunprabash, Gsarkis, Yobot, Gilgamesh4, Daveburstein, KLBot2, Ajraymond, DiscantX, The Channel of Random, Hmahdavifar and Anonymous: 11
- **Polynomial code** *Source:* [https://en.wikipedia.org/wiki/Polynomial\\_code?oldid=762693928](https://en.wikipedia.org/wiki/Polynomial_code?oldid=762693928) *Contributors:* Michael Hardy, Dcoetzee, Dolphinling, Alaibot, Selinger, Non-dropframe, Cunchem, Nageh, Sławomir Biały, Alexander Chervov, Spectral sequence, Paul R. Roth, Ariasfxavier and Anonymous: 5
- **Prefix code** *Source:* [https://en.wikipedia.org/wiki/Prefix\\_code?oldid=753119566](https://en.wikipedia.org/wiki/Prefix_code?oldid=753119566) *Contributors:* PierreAbbat, Michael Hardy, Deljr, Timwi, AnonMoos, DavidCary, WiseWoman, Tagishsimon, Edcolins, Mormegil, JTN, MeltBanana, Antaeus Feldspar, Limbo socrates, RadetzkyVonRadetz, Richi, Celada, Pearle, TheParanoidOne, Jheald, Oleg Alexandrov, BD2412, Drpaul, Margosbot~enwiki, Quuxplusone, Corington, Ihope127, NawlinWiki, DrAtouk, SmackBot, Faisal.akeel, Bluebot, Bpg1968, Nbarth, Henning Makhholm, Euchiasmus, JoeBot, Eastlaw, CmdrObot, CBM, Chris83, Hezink8, Verdy p, Thijs!bot, Esowteric, LachlanA, Dougher, Magioladitis, David Eppstein, Mythealias, Peterwshor, DavidCBryant, Jamelan, OsamaBinLogin, Classicaecon, PipepBot, WalterGR, SchreiberBike, Johnuniq, XLinkBot, Jaan Vajakas, Addbot, Scientus, Luckas~bot, Yobot, Ptbotgourou, Infvwl, Citation bot 1, RjwilmsiBot, Matthiaspaul, J.Dong820, Cosmicide, ChrisGualtieri, Deltahedron, Mutley1989, Bender the Bot and Anonymous: 48
- **Preparata code** *Source:* [https://en.wikipedia.org/wiki/Preparata\\_code?oldid=727568627](https://en.wikipedia.org/wiki/Preparata_code?oldid=727568627) *Contributors:* Michael Hardy, Vanish2, DOI bot, Citation bot, Citation bot 1, Tom.Reding, Qetuth, SomeIRedirects4You and Anonymous: 2
- **Puncturing** *Source:* <https://en.wikipedia.org/wiki/Puncturing?oldid=747621483> *Contributors:* Ronz, Bearcat, DavidCary, R6144, Xezbeth, RobertG, SmackBot, Alaibot, Chemako0606, Addbot, Erik9bot, Dewritech, ClueBot NG, Ksodayaji3, Deltahedron, Bender the Bot and Anonymous: 10
- **Quadratic residue code** *Source:* [https://en.wikipedia.org/wiki/Quadratic\\_residue\\_code?oldid=752140891](https://en.wikipedia.org/wiki/Quadratic_residue_code?oldid=752140891) *Contributors:* Rsrikanth05, Jmlk17, RJChapman, Headbomb, David Eppstein, Hpfister, 71755, Wcherowi, ChrisGualtieri, Olaaruinow and Anonymous: 5
- **Rank error-correcting code** *Source:* [https://en.wikipedia.org/wiki/Rank\\_error-correcting\\_code?oldid=730305813](https://en.wikipedia.org/wiki/Rank_error-correcting_code?oldid=730305813) *Contributors:* Michael Hardy, Rjwilmsi, Addbot, Yobot, LilHelpa, Alex.kshevetskiy, Thehelpfulbot, Helpful Pixie Bot, BattyBot and Anonymous: 3
- **Raptor code** *Source:* [https://en.wikipedia.org/wiki/Raptor\\_code?oldid=682672446](https://en.wikipedia.org/wiki/Raptor_code?oldid=682672446) *Contributors:* Chris~enwiki, Grin, Linas, Rjwilmsi, Gaius Cornelius, Fmccown, Bluebot, Oli Filth, Cantalamessa, Mark@digitalfountain.com, Requestion, Askmath, Scarian, KC109, Cunchem, Nageh, The mike luby, Berkovskyy and Anonymous: 20
- **Recursive indexing** *Source:* [https://en.wikipedia.org/wiki/Recursive\\_indexing?oldid=608622135](https://en.wikipedia.org/wiki/Recursive_indexing?oldid=608622135) *Contributors:* Photonique, Jpbowen, SmackBot, Bluebot, Kvng, MarshBot, R'n'B, SJP, DavidCBryant, Yobot, AnomieBOT and Anonymous: 3
- **Reed-Muller code** *Source:* [https://en.wikipedia.org/wiki/Reed%E2%80%93Muller\\_code?oldid=762909591](https://en.wikipedia.org/wiki/Reed%E2%80%93Muller_code?oldid=762909591) *Contributors:* Michael Hardy, Komap, Sander123, DavidCary, Bob.v.R, RetiredUser2, Bender235, Cmdrjameson, Photonique, Eyreland, Florian Huber, Reetep, Jpkotta, YurikBot, Gene.arboit, SmackBot, Oli Filth, Beetstra, Dicklyon, Iridescent, Ylloh, Jsrobin, Hermel, Vanish2, Ohms law, DavidCBryant, Jamelan, Trogsworth, Beastinwith, Mild Bill Hiccup, Sun Creator, Cerireid, Addbot, Yobot, Isheden, Ablmf, Darij, Ralfreiner-mueller, Kedwar5, Nordbad, Mmeijeri, Alexbhandari, Oddbodz, Helpful Pixie Bot, Cyberbot II, ChrisGualtieri, Poppopsun, Monkbot, Fliptel, Ebadihamid, Norm16wiki and Anonymous: 32
- **Reed-Solomon error correction** *Source:* [https://en.wikipedia.org/wiki/Reed%E2%80%93Solomon\\_error\\_correction?oldid=766555870](https://en.wikipedia.org/wiki/Reed%E2%80%93Solomon_error_correction?oldid=766555870) *Contributors:* WojPob, Arvindn, PierreAbbat, Roadrunner, Maury Markowitz, TomCerule, Michael Hardy, Ejrh, Ffx, Samw, Technopilgrim, Mulad, Charles Matthews, Chatool, Zoicon5, Reina riemann, Oaktree b, AnonMoos, Fredrik, Altenmann, Gobeirne, BenFrantzDale, Karn, Nayuki, Ojl, Naleks, Sam Hocevar, Yonkeltron, Mindspillage, Qutezuca, ArnoldReinhold, ZeroOne, Rgdboer, Dolphinling, A-Day, Giraffedata, Waltervulej, Photonique, Shawn K. Quinn, Msa11usec, Guy Harris, Pjkundert, Kocio, Lee S. Svoboda, Scottvw, Cburnett, Guthrie, Eyreland, Marudubshinki, Harmonica~enwiki, Graham87, Rjwilmsi, Joe Decker, JMCOREY, TedPostol, Bubba73, Quuxplusone, Chobot, DVdm, Bgwhite, RobotE, Gene.arboit, Hydrargyrum, Grubber, Blutfink, PhilipO, Ke6jjj, Fmccown, Geoffrey.landis, Poculum, RupertMillard, SmackBot, Mitchan, Gigs, Msjacoby23, C.Fred, KelleyCook, Vescudero, Slaniel, Oli Filth, Jeysaba, OrphanBot, Mhym,



MichaelBillington, Lpgeffen, Pandar~enwiki, Theme97, J. Finkelstein, Mr Stephen, Dicklyon, Metao, Kvng, Hu12, Iridescent, Ylloh, CR-Greathouse, Karenjc, Kansas Sam, Thijs!bot, Andyjsmith, Headbomb, Stefankroon, Medconn, .anacondabot, David Eppstein, Edurant, Connor Behan, Gwern, Conquerist, Glrx, Gfox88, Erlichya~enwiki, Selinger, Hailangsea~enwiki, Mathuranathan, Banjodog, Jcea, TXiKiBoT, Marcosaedro, Atrirudra, Rogerdpack, SieBot, Jimplank, Svick, ClueBot, TableManners, Plastikspork, EoGuy, Izab1, Niceguyedc, Andreagold, Estirabot, Qwfp, LozK, WikHead, Jaco.versfeld, RealWorldExperience, Addbot, Rcglr, Happyhappy2008, David0811, Luckas~bot, Yobot, Nallimbot, AnomieBOT, Motfellow, Cunchem, Sz-iwbot, Xqbot, DataWraith, RibotBOT, FrescoBot, Nageh, Spm-RmvBot, ImageTagBot, MastiBot, Trappist the monk, Bobmath, NoNewsToday, John of Reading, Ahughes6, Quondum, Casascius, ClueBot NG, H2.818, Helpful Pixie Bot, Rohitram 88, Chmarkine, BattyBot, Justincheng12345~bot, Toploftical, Madhurasuresh, Biernat2, Leijurv, Mark viking, Snookerr, Zvord, Djoka.panama, Leegrc, Ing. Tomáš Pajda, CCNA, Riceissa, PerryKunderd, CAPTAIN RAJU, Leonardoaraujo.santos, Fitindia, Latex-yow, Brian-armstrong and Anonymous: 175

- **Sardinas–Patterson algorithm** *Source:* [https://en.wikipedia.org/wiki/Sardinas%E2%80%93Patterson\\_algorithm?oldid=736206753](https://en.wikipedia.org/wiki/Sardinas%E2%80%93Patterson_algorithm?oldid=736206753) *Contributors:* Tea2min, Bender235, Rjwilmsi, Dicklyon, PKT, Hermel, David Eppstein, Rosiestep, Yobot, Deltahedron, Danielhales and Anonymous: 4
- **Second Johnson bound** *Source:* [https://en.wikipedia.org/wiki/Johnson\\_bound?oldid=747143446](https://en.wikipedia.org/wiki/Johnson_bound?oldid=747143446) *Contributors:* Michael Hardy, Giftlite, Pierremenard, SmackBot, Synergy, Wikid77, Magioladitis, STBot, Yobot, Omnipaedista, ZéroBot, Matthiaspaul, KLBot2 and Anonymous: 1
- **Shannon's source coding theorem** *Source:* [https://en.wikipedia.org/wiki/Shannon's\\_source\\_coding\\_theorem?oldid=710013832](https://en.wikipedia.org/wiki/Shannon's_source_coding_theorem?oldid=710013832) *Contributors:* Dcoetzee, Vamos, Sander123, Giftlite, Rick Sidwell, Jheald, Reetep, Closedmouth, SmackBot, Bluebot, Calbaer, Robofish, CBM, FilipeS, Pulkitgrover, Hpalaiya, Hezink8, Verdy p, Thijs!bot, Egriffin, WikiIT, Mojo Hand, WikiC~enwiki, The prophet wizard of the crayon cake, Alexei Kopylov, Mange01, VolkovBot, Geometry guy, AlleborgoBot, SieBot, Johnuniq, Addbot, Luckas~bot, Obersachsebot, TankMiche, Bruno.loff, EmausBot, Bozicmilvoj, BG19bot, Brad7777 and Anonymous: 23
- **Singleton bound** *Source:* [https://en.wikipedia.org/wiki/Singleton\\_bound?oldid=741652168](https://en.wikipedia.org/wiki/Singleton_bound?oldid=741652168) *Contributors:* Michael Hardy, Charles Matthews, Adam Bishop, DJ Clayworth, Giftlite, Pierremenard, Shreevatsa, Ruud Koot, Rjwilmsi, Reetep, SmackBot, Frap, Thijs!bot, Hermel, Vanish2, David Eppstein, Addbot, Luckas~bot, Citation bot, Xqbot, Omnipaedista, Sonofnob, VanceIII, Citation bot 1, Orenburg1, Zxl.gzhu, Wcherowi, BG19bot, Boriaj, Deltahedron, ZeeMurph and Anonymous: 19
- **Soliton distribution** *Source:* [https://en.wikipedia.org/wiki/Soliton\\_distribution?oldid=757880942](https://en.wikipedia.org/wiki/Soliton_distribution?oldid=757880942) *Contributors:* Michael Hardy, Phil Boswell, Benwing, Shreevatsa, Melcombe, Qwfp, Marcocapelle, Pintoeh, BeyondNormality and Anonymous: 1
- **Spherical code** *Source:* [https://en.wikipedia.org/wiki/Spherical\\_code?oldid=603806570](https://en.wikipedia.org/wiki/Spherical_code?oldid=603806570) *Contributors:* Joriki, Malcolma, SmackBot, Mark viking and Anonymous: 1
- **Srivastava code** *Source:* [https://en.wikipedia.org/wiki/Srivastava\\_code?oldid=618272444](https://en.wikipedia.org/wiki/Srivastava_code?oldid=618272444) *Contributors:* Malaiya, Vanish2, Addbot, RjwilmsiBot, Luizpuodzius, Qetuth and Monkbot
- **Standard array** *Source:* [https://en.wikipedia.org/wiki/Standard\\_array?oldid=750077850](https://en.wikipedia.org/wiki/Standard_array?oldid=750077850) *Contributors:* Michael Hardy, Culix, RussBot, Deskana, Magioladitis, Niceguyedc, HexaChord, Alfaisanomega, Helpful Pixie Bot and Anonymous: 5
- **Systematic code** *Source:* [https://en.wikipedia.org/wiki/Systematic\\_code?oldid=723919740](https://en.wikipedia.org/wiki/Systematic_code?oldid=723919740) *Contributors:* Abdull, Wikinaut, Shabble, Eric.weigle, Oli Filth, DavidCBryant, Niceguyedc, Addbot, Xqbot, Nageh, RjwilmsiBot, Dcirovic, ChuispastonBot, Helpful Pixie Bot, EleGu79 and Anonymous: 2
- **Tanner graph** *Source:* [https://en.wikipedia.org/wiki/Tanner\\_graph?oldid=675349728](https://en.wikipedia.org/wiki/Tanner_graph?oldid=675349728) *Contributors:* Michael Hardy, Netpilot43556, Stesmo, Oleg Alexandrov, MZMcBride, Grubber, Tawkerbot2, Flamencorex, David Eppstein, Reedy Bot, Trachten, Raymondwin, Beast-inwith, Yobot, Sujith17889 and Anonymous: 8
- **Ternary Golay code** *Source:* [https://en.wikipedia.org/wiki/Ternary\\_Golay\\_code?oldid=742473775](https://en.wikipedia.org/wiki/Ternary_Golay_code?oldid=742473775) *Contributors:* Michael Hardy, Charles Matthews, Giftlite, 4pq1injok, Linas, R.e.b., RJChapman, Ylloh, Tkynerd, SchreiberBike, Ramses68, Lightbot, AnomieBOT, LittleWink, Trappist the monk, BertSeghers, Awoldar and Anonymous: 9
- **Tornado code** *Source:* [https://en.wikipedia.org/wiki/Tornado\\_code?oldid=723732199](https://en.wikipedia.org/wiki/Tornado_code?oldid=723732199) *Contributors:* The Anome, CesarB, Tea2min, Utcursch, Gazpacho, Andrewpmk, RJFJR, Linas, Dogcow, SmackBot, Oli Filth, A3nm, Waterpie, DavidCBryant, Rogerdpack, Mdnahas, RjwilmsiBot, MarkusWanner, Astronautguo, Camlf and Anonymous: 11
- **Triangular network coding** *Source:* [https://en.wikipedia.org/wiki/Triangular\\_network\\_coding?oldid=540180654](https://en.wikipedia.org/wiki/Triangular_network_coding?oldid=540180654) *Contributors:* Michael Hardy, Missvain, RjwilmsiBot, Jqureshi786 and Anonymous: 1
- **Unary coding** *Source:* [https://en.wikipedia.org/wiki/Unary\\_coding?oldid=747366925](https://en.wikipedia.org/wiki/Unary_coding?oldid=747366925) *Contributors:* Poor Yorick, Clausen, Furrykef, Inkling, Nsaa, Rjwilmsi, Wavelength, WouterBolsterlee, Kjetil1001, Calbaer, Yms, Headbomb, David Eppstein, STBotD, DavidCBryant, VolkovBot, TXiKiBoT, Classicalecon, Arkanosis, DumZiBoT, Addbot, Yobot, Ego White Tray, Frietjes, Arlene47, Jabawock02 and Anonymous: 17
- **Variable-length code** *Source:* [https://en.wikipedia.org/wiki/Variable-length\\_code?oldid=767092294](https://en.wikipedia.org/wiki/Variable-length_code?oldid=767092294) *Contributors:* Michael Hardy, DavidCary, Jheald, Johnpseudo, SmackBot, Hashbrowncipher, Calbaer, Cybercobra, Mwtoews, CmdrObot, CBM, Neelix, Hermel, DavidCBryant, JackSchmidt, Svick, Niceguyedc, Addbot, AnomieBOT, AdjustShift, DX-MON, 6Sixx, Rc3002, John of Reading, K6ka, ZéroBot, Helpful Pixie Bot, Cup o' Java, Deltahedron, Michael Lee Baker and Anonymous: 15
- **Z-channel (information theory)** *Source:* [https://en.wikipedia.org/wiki/Z-channel\\_\(information\\_theory\)?oldid=635825539](https://en.wikipedia.org/wiki/Z-channel_(information_theory)?oldid=635825539) *Contributors:* Michael Hardy, Vamos, Oleg Alexandrov, Tomer Ish Shalom, Sodin, Pegship, Pulkitgrover, Alaiobot, Jeepday, DavidCBryant, InfoTheory, Sharov, AnomieBOT, Erik9bot, Ywhuang and Anonymous: 5
- **Zemor's decoding algorithm** *Source:* [https://en.wikipedia.org/wiki/Zemor's\\_decoding\\_algorithm?oldid=746178421](https://en.wikipedia.org/wiki/Zemor's_decoding_algorithm?oldid=746178421) *Contributors:* Michael Hardy, Bearcat, Anthony Appleyard, Denniss, Drbreznjev, Torchiest, Picojeff, Mild Bill Hiccup, Sun Creator, Yobot, Citation bot, Fixer88, Sujith17889, Lcode007, Rybec, Flat Out, Jed2347, InternetArchiveBot, GreenC bot and Anonymous: 1
- **Zigzag code** *Source:* [https://en.wikipedia.org/wiki/Zigzag\\_code?oldid=695757834](https://en.wikipedia.org/wiki/Zigzag_code?oldid=695757834) *Contributors:* Michael Hardy, SmackBot, OrphanBot, KurtRaschke, John254, David Eppstein, DimaG, AlphaPyro, IForTheMoney, Barabenka and U2fanboi
- **Zyablov bound** *Source:* [https://en.wikipedia.org/wiki/Zyablov\\_bound?oldid=737815304](https://en.wikipedia.org/wiki/Zyablov_bound?oldid=737815304) *Contributors:* Sidjaggi, Magioladitis, Dalit Llama, Vishwasr dec20, Freemrain, Latex-yow and Anonymous: 1

## 112.6.2 Images

- **File:16QAM\_Gray\_Coded.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/1/1e/16QAM\\_Gray\\_Coded.svg](https://upload.wikimedia.org/wikipedia/commons/1/1e/16QAM_Gray_Coded.svg) *License:* CC-BY-SA-3.0 *Contributors:* Own work *Original artist:* Splash
- **File:AVLtreef.svg** *Source:* <https://upload.wikimedia.org/wikipedia/commons/0/06/AVLtreef.svg> *License:* Public domain *Contributors:* Own work *Original artist:* User:Mikm
- **File:Ambox\_important.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/b/b4/Ambox\\_important.svg](https://upload.wikimedia.org/wikipedia/commons/b/b4/Ambox_important.svg) *License:* Public domain *Contributors:* Own work, based off of Image:Ambox scales.svg *Original artist:* Dsmurat (talk · contribs)
- **File:Ambox\_wikify.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/e/e1/Ambox\\_wikify.svg](https://upload.wikimedia.org/wikipedia/commons/e/e1/Ambox_wikify.svg) *License:* Public domain *Contributors:* Own work *Original artist:* penubag
- **File:Binary-reflected\_Gray\_code\_construction.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/c/c1/Binary-reflected\\_Gray\\_code\\_construction.svg](https://upload.wikimedia.org/wikipedia/commons/c/c1/Binary-reflected_Gray_code_construction.svg) *License:* Public domain *Contributors:* Own work *Original artist:* Inductiveload
- **File:Binary\_symmetric\_channel\_(en).svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/8/8e/Binary\\_symmetric\\_channel\\_%28en%29.svg](https://upload.wikimedia.org/wikipedia/commons/8/8e/Binary_symmetric_channel_%28en%29.svg) *License:* Public domain *Contributors:* Based upon a nearly-identical version created by en>User:gene.arboit. This rendering done by bdesham in Inkscape. *Original artist:* Benjamin D. Esham (bdesham)
- **File:Binary\_tree.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/f/f7/Binary\\_tree.svg](https://upload.wikimedia.org/wikipedia/commons/f/f7/Binary_tree.svg) *License:* Public domain *Contributors:* Own work *Original artist:* Derrick Coetzee
- **File:Binaryerasurechannel.png** *Source:* <https://upload.wikimedia.org/wikipedia/commons/2/23/Binaryerasurechannel.png> *License:* Public domain *Contributors:* Transferred from en.wikipedia to Commons by Wdwd using CommonsHelper. *Original artist:* Edratzer at English Wikipedia
- **File:Block,Interleaver,Example.jpg** *Source:* <https://upload.wikimedia.org/wikipedia/commons/7/7c/Block%2CInterleaver%2CExample.jpg> *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* script3r
- **File:Burst\_Error\_Length\_5\_Description.jpg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/0/04/Burst\\_Error%2CLength\\_5%2CDescription.jpg](https://upload.wikimedia.org/wikipedia/commons/0/04/Burst_Error%2CLength_5%2CDescription.jpg) *License:* CC BY-SA 3.0 *Contributors:* Paint *Original artist:* Script3r
- **File:Butterfly\_network.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/3/33/Butterfly\\_network.svg](https://upload.wikimedia.org/wikipedia/commons/3/33/Butterfly_network.svg) *License:* Public domain *Contributors:* Transferred from en.wikipedia to Commons. Transfer was stated to be made by User:Storkk. *Original artist:* The original uploader was DnetSvg at English Wikipedia
- **File:Code\_d'effacement\_optimal\_1.gif** *Source:* [https://upload.wikimedia.org/wikipedia/commons/b/b4/Code\\_d%27effacement\\_optimal\\_1.gif](https://upload.wikimedia.org/wikipedia/commons/b/b4/Code_d%27effacement_optimal_1.gif) *License:* Public domain *Contributors:* I (Nic Roets) am the author and hereby place the image in the public domain. *Original artist:* Nroets at English Wikipedia
- **File:Code\_d'effacement\_optimal\_2.gif** *Source:* [https://upload.wikimedia.org/wikipedia/commons/9/92/Code\\_d%27effacement\\_optimal\\_2.gif](https://upload.wikimedia.org/wikipedia/commons/9/92/Code_d%27effacement_optimal_2.gif) *License:* Public domain *Contributors:* I (Nic Roets) am the author and hereby place the image in the public domain. *Original artist:* Nroets at English Wikipedia
- **File:Codinh\_theory.png** *Source:* [https://upload.wikimedia.org/wikipedia/commons/8/8f/Codinh\\_theory.png](https://upload.wikimedia.org/wikipedia/commons/8/8f/Codinh_theory.png) *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Madhurasuresh
- **File:Comm\_Channel.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/4/48/Comm\\_Channel.svg](https://upload.wikimedia.org/wikipedia/commons/4/48/Comm_Channel.svg) *License:* Public domain *Contributors:* en wikipedia *Original artist:* Dicklyon
- **File:Concatenated\_codes\_diagram.png** *Source:* [https://upload.wikimedia.org/wikipedia/en/f/f2/Concatenated\\_codes\\_diagram.png](https://upload.wikimedia.org/wikipedia/en/f/f2/Concatenated_codes_diagram.png) *License:* PD *Contributors:* Own work *Original artist:* Beamishboy (talk) (Uploads)
- **File:Concatenation\_of\_Reed-Solomon\_code\_with\_Hadamard\_code.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/8/8c/Concatenation\\_of\\_Reed%E2%80%93Solomon\\_code\\_with\\_Hadamard\\_code.svg](https://upload.wikimedia.org/wikipedia/commons/8/8c/Concatenation_of_Reed%E2%80%93Solomon_code_with_Hadamard_code.svg) *License:* CC0 *Contributors:* Own work *Original artist:* Ylloh
- **File:Convolutional,Interleaver,Example.jpg** *Source:* <https://upload.wikimedia.org/wikipedia/commons/3/31/Convolutional%2CInterleaver%2CExample.jpg> *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* script3r
- **File:Decoding\_a\_Folded\_Reed-Solomon\_Code.png** *Source:* [https://upload.wikimedia.org/wikipedia/commons/d/d3/Decoding\\_a\\_Folded\\_Reed-Solomon\\_Code.png](https://upload.wikimedia.org/wikipedia/commons/d/d3/Decoding_a_Folded_Reed-Solomon_Code.png) *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Madhurasuresh
- **File:Deinterleaver,Example.jpg** *Source:* <https://upload.wikimedia.org/wikipedia/commons/c/c5/Deinterleaver%2CExample.jpg> *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Script3r
- **File:E-to-the-i-pi.svg** *Source:* <https://upload.wikimedia.org/wikipedia/commons/3/35/E-to-the-i-pi.svg> *License:* CC BY 2.5 *Contributors:* No machine-readable source provided. Own work assumed (based on copyright claims). *Original artist:* No machine-readable author provided. Dermeister assumed (based on copyright claims).
- **File>Edit-clear.svg** *Source:* <https://upload.wikimedia.org/wikipedia/en/f/f2/Edit-clear.svg> *License:* Public domain *Contributors:* The Tango! Desktop Project. *Original artist:* The people from the Tango! project. And according to the meta-data in the file, specifically: “Andreas Nilsson, and Jakub Steiner (although minimally).”
- **File:Emoji\_u1f510.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/3/35/Emoji\\_u1f510.svg](https://upload.wikimedia.org/wikipedia/commons/3/35/Emoji_u1f510.svg) *License:* Apache License 2.0 *Contributors:* <https://code.google.com/p/noto/> *Original artist:* Google
- **File:Encoder\_Disc\_(3-Bit).svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/9/9b/Encoder\\_Disc\\_%283-Bit%29.svg](https://upload.wikimedia.org/wikipedia/commons/9/9b/Encoder_Disc_%283-Bit%29.svg) *License:* Public domain *Contributors:* Replacement for Image:Encoder\_disc.png *Original artist:* jjbeard
- **File:Enkelspoors-Graycode.svg** *Source:* <https://upload.wikimedia.org/wikipedia/commons/5/5e/Enkelspoors-Graycode.svg> *License:* CC BY-SA 2.5 nl *Contributors:* Own work *Original artist:* Gerbrant

- **File:Folding\_of\_Reed-Solomon\_Code\_with\_folding\_parameter\_m=3.png** Source: [https://upload.wikimedia.org/wikipedia/commons/5/5b/Folding\\_of\\_Reed-Solomon\\_Code\\_with\\_folding\\_parameter\\_m%3D3.png](https://upload.wikimedia.org/wikipedia/commons/5/5b/Folding_of_Reed-Solomon_Code_with_folding_parameter_m%3D3.png) License: CC BY-SA 3.0 Contributors: Own work Original artist: Madhurasuresh
- **File:FuzzyExtGen.png** Source: <https://upload.wikimedia.org/wikipedia/commons/b/b3/FuzzyExtGen.png> License: CC BY-SA 3.0 Contributors: Own work Original artist: Bjr244
- **File:Gray\_code\_permutation\_matrix\_16.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/6/6a/Gray\\_code\\_permutation\\_matrix\\_16.svg](https://upload.wikimedia.org/wikipedia/commons/6/6a/Gray_code_permutation_matrix_16.svg) License: Public domain Contributors: Own work Original artist: [a href="//commons.wikimedia.org/wiki/File:Watchduck.svg"](https://commons.wikimedia.org/wiki/File:Watchduck.svg) `<img alt="Watchduck.svg" class="image"></a>` Watchduck (a.k.a. Tilman Piesk)
- **File:Gray\_code\_rotary\_encoder\_13-track\_opened.jpg** Source: [https://upload.wikimedia.org/wikipedia/commons/a/a8/Gray\\_code\\_rotary\\_encoder\\_13-track\\_opened.jpg](https://upload.wikimedia.org/wikipedia/commons/a/a8/Gray_code_rotary_encoder_13-track_opened.jpg) License: Public domain Contributors: Mike1024 Original artist: Mike1024
- **File:Hadamard-Code.svg** Source: <https://upload.wikimedia.org/wikipedia/commons/3/37/Hadamard-Code.svg> License: Public domain Contributors: Own work Original artist: [a href="//commons.wikimedia.org/wiki/File:Watchduck.svg"](https://commons.wikimedia.org/wiki/File:Watchduck.svg) `<img alt="Watchduck.svg" class="image"></a>` Watchduck (a.k.a. Tilman Piesk)
- **File:Hamming(7,4).svg** Source: <https://upload.wikimedia.org/wikipedia/commons/b/b0/Hamming%287%2C4%29.svg> License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett
- **File:Hamming(7,4)\_as\_bits.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/0/00/Hamming%287%2C4%29\\_as\\_bits.svg](https://upload.wikimedia.org/wikipedia/commons/0/00/Hamming%287%2C4%29_as_bits.svg) License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett
- **File:Hamming(7,4)\_example\_0000.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/5/5b/Hamming%287%2C4%29\\_example\\_0000.svg](https://upload.wikimedia.org/wikipedia/commons/5/5b/Hamming%287%2C4%29_example_0000.svg) License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett
- **File:Hamming(7,4)\_example\_0000\_with\_extra\_parity.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/f/ff/Hamming%287%2C4%29\\_example\\_0000\\_with\\_extra\\_parity.svg](https://upload.wikimedia.org/wikipedia/commons/f/ff/Hamming%287%2C4%29_example_0000_with_extra_parity.svg) License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett
- **File:Hamming(7,4)\_example\_0001.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/d/d3/Hamming%287%2C4%29\\_example\\_0001.svg](https://upload.wikimedia.org/wikipedia/commons/d/d3/Hamming%287%2C4%29_example_0001.svg) License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett
- **File:Hamming(7,4)\_example\_0001\_with\_extra\_parity.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/f/f7/Hamming%287%2C4%29\\_example\\_0001\\_with\\_extra\\_parity.svg](https://upload.wikimedia.org/wikipedia/commons/f/f7/Hamming%287%2C4%29_example_0001_with_extra_parity.svg) License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett
- **File:Hamming(7,4)\_example\_0010.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/2/2b/Hamming%287%2C4%29\\_example\\_0010.svg](https://upload.wikimedia.org/wikipedia/commons/2/2b/Hamming%287%2C4%29_example_0010.svg) License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett
- **File:Hamming(7,4)\_example\_0010\_with\_extra\_parity.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/a/ac/Hamming%287%2C4%29\\_example\\_0010\\_with\\_extra\\_parity.svg](https://upload.wikimedia.org/wikipedia/commons/a/ac/Hamming%287%2C4%29_example_0010_with_extra_parity.svg) License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett
- **File:Hamming(7,4)\_example\_0011.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/5/5b/Hamming%287%2C4%29\\_example\\_0011.svg](https://upload.wikimedia.org/wikipedia/commons/5/5b/Hamming%287%2C4%29_example_0011.svg) License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett
- **File:Hamming(7,4)\_example\_0011\_with\_extra\_parity.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/a/a5/Hamming%287%2C4%29\\_example\\_0011\\_with\\_extra\\_parity.svg](https://upload.wikimedia.org/wikipedia/commons/a/a5/Hamming%287%2C4%29_example_0011_with_extra_parity.svg) License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett
- **File:Hamming(7,4)\_example\_0100.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/d/da/Hamming%287%2C4%29\\_example\\_0100.svg](https://upload.wikimedia.org/wikipedia/commons/d/da/Hamming%287%2C4%29_example_0100.svg) License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett
- **File:Hamming(7,4)\_example\_0100\_with\_extra\_parity.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/4/45/Hamming%287%2C4%29\\_example\\_0100\\_with\\_extra\\_parity.svg](https://upload.wikimedia.org/wikipedia/commons/4/45/Hamming%287%2C4%29_example_0100_with_extra_parity.svg) License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett
- **File:Hamming(7,4)\_example\_0101.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/a/ae/Hamming%287%2C4%29\\_example\\_0101.svg](https://upload.wikimedia.org/wikipedia/commons/a/ae/Hamming%287%2C4%29_example_0101.svg) License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett
- **File:Hamming(7,4)\_example\_0101\_with\_extra\_parity.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/e/e7/Hamming%287%2C4%29\\_example\\_0101\\_with\\_extra\\_parity.svg](https://upload.wikimedia.org/wikipedia/commons/e/e7/Hamming%287%2C4%29_example_0101_with_extra_parity.svg) License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett
- **File:Hamming(7,4)\_example\_0110.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/b/bf/Hamming%287%2C4%29\\_example\\_0110.svg](https://upload.wikimedia.org/wikipedia/commons/b/bf/Hamming%287%2C4%29_example_0110.svg) License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett
- **File:Hamming(7,4)\_example\_0110\_with\_extra\_parity.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/a/a6/Hamming%287%2C4%29\\_example\\_0110\\_with\\_extra\\_parity.svg](https://upload.wikimedia.org/wikipedia/commons/a/a6/Hamming%287%2C4%29_example_0110_with_extra_parity.svg) License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett
- **File:Hamming(7,4)\_example\_0111.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/0/0a/Hamming%287%2C4%29\\_example\\_0111.svg](https://upload.wikimedia.org/wikipedia/commons/0/0a/Hamming%287%2C4%29_example_0111.svg) License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett
- **File:Hamming(7,4)\_example\_0111\_with\_extra\_parity.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/4/48/Hamming%287%2C4%29\\_example\\_0111\\_with\\_extra\\_parity.svg](https://upload.wikimedia.org/wikipedia/commons/4/48/Hamming%287%2C4%29_example_0111_with_extra_parity.svg) License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett
- **File:Hamming(7,4)\_example\_1000.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/3/37/Hamming%287%2C4%29\\_example\\_1000.svg](https://upload.wikimedia.org/wikipedia/commons/3/37/Hamming%287%2C4%29_example_1000.svg) License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett



- **File:Hamming(7,4)\_example\_1000\_with\_extra\_parity.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/f/f8/Hamming%287%2C4%29\\_example\\_1000\\_with\\_extra\\_parity.svg](https://upload.wikimedia.org/wikipedia/commons/f/f8/Hamming%287%2C4%29_example_1000_with_extra_parity.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming(7,4)\_example\_1001.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/1/1b/Hamming%287%2C4%29\\_example\\_1001.svg](https://upload.wikimedia.org/wikipedia/commons/1/1b/Hamming%287%2C4%29_example_1001.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming(7,4)\_example\_1001\_with\_extra\_parity.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/b/b0/Hamming%287%2C4%29\\_example\\_1001\\_with\\_extra\\_parity.svg](https://upload.wikimedia.org/wikipedia/commons/b/b0/Hamming%287%2C4%29_example_1001_with_extra_parity.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming(7,4)\_example\_1010.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/5/59/Hamming%287%2C4%29\\_example\\_1010.svg](https://upload.wikimedia.org/wikipedia/commons/5/59/Hamming%287%2C4%29_example_1010.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming(7,4)\_example\_1010\_with\_extra\_parity.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/8/87/Hamming%287%2C4%29\\_example\\_1010\\_with\\_extra\\_parity.svg](https://upload.wikimedia.org/wikipedia/commons/8/87/Hamming%287%2C4%29_example_1010_with_extra_parity.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming(7,4)\_example\_1011.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/6/63/Hamming%287%2C4%29\\_example\\_1011.svg](https://upload.wikimedia.org/wikipedia/commons/6/63/Hamming%287%2C4%29_example_1011.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming(7,4)\_example\_1011\_bit\_5\_error.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/5/58/Hamming%287%2C4%29\\_example\\_1011\\_bit\\_5\\_error.svg](https://upload.wikimedia.org/wikipedia/commons/5/58/Hamming%287%2C4%29_example_1011_bit_5_error.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming(7,4)\_example\_1011\_bits\_4\_&\_5\_error.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/6/67/Hamming%287%2C4%29\\_example\\_1011\\_bits\\_4\\_%26\\_5\\_error.svg](https://upload.wikimedia.org/wikipedia/commons/6/67/Hamming%287%2C4%29_example_1011_bits_4_%26_5_error.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming(7,4)\_example\_1011\_with\_extra\_parity.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/c/cf/Hamming%287%2C4%29\\_example\\_1011\\_with\\_extra\\_parity.svg](https://upload.wikimedia.org/wikipedia/commons/c/cf/Hamming%287%2C4%29_example_1011_with_extra_parity.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming(7,4)\_example\_1100.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/7/7a/Hamming%287%2C4%29\\_example\\_1100.svg](https://upload.wikimedia.org/wikipedia/commons/7/7a/Hamming%287%2C4%29_example_1100.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming(7,4)\_example\_1100\_with\_extra\_parity.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/0/01/Hamming%287%2C4%29\\_example\\_1100\\_with\\_extra\\_parity.svg](https://upload.wikimedia.org/wikipedia/commons/0/01/Hamming%287%2C4%29_example_1100_with_extra_parity.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming(7,4)\_example\_1101.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/d/da/Hamming%287%2C4%29\\_example\\_1101.svg](https://upload.wikimedia.org/wikipedia/commons/d/da/Hamming%287%2C4%29_example_1101.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming(7,4)\_example\_1101\_with\_extra\_parity.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/6/68/Hamming%287%2C4%29\\_example\\_1101\\_with\\_extra\\_parity.svg](https://upload.wikimedia.org/wikipedia/commons/6/68/Hamming%287%2C4%29_example_1101_with_extra_parity.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming(7,4)\_example\_1110.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/4/44/Hamming%287%2C4%29\\_example\\_1110.svg](https://upload.wikimedia.org/wikipedia/commons/4/44/Hamming%287%2C4%29_example_1110.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming(7,4)\_example\_1110\_with\_extra\_parity.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/8/8e/Hamming%287%2C4%29\\_example\\_1110\\_with\\_extra\\_parity.svg](https://upload.wikimedia.org/wikipedia/commons/8/8e/Hamming%287%2C4%29_example_1110_with_extra_parity.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming(7,4)\_example\_1111.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/b/b6/Hamming%287%2C4%29\\_example\\_1111.svg](https://upload.wikimedia.org/wikipedia/commons/b/b6/Hamming%287%2C4%29_example_1111.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming(7,4)\_example\_1111\_with\_extra\_parity.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/0/05/Hamming%287%2C4%29\\_example\\_1111\\_with\\_extra\\_parity.svg](https://upload.wikimedia.org/wikipedia/commons/0/05/Hamming%287%2C4%29_example_1111_with_extra_parity.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming(8,4).svg** *Source:* <https://upload.wikimedia.org/wikipedia/commons/1/13/Hamming%288%2C4%29.svg> *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming.jpg** *Source:* <https://upload.wikimedia.org/wikipedia/commons/3/39/Hamming.jpg> *License:* Public domain *Contributors:* Own work *Original artist:* Josiedraus
- **File:HammingLimit.png** *Source:* <https://upload.wikimedia.org/wikipedia/commons/c/c2/HammingLimit.png> *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Frank Klemm
- **File:Hamming\_distance\_3\_bit\_binary.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/b/b4/Hamming\\_distance\\_3\\_bit\\_binary.svg](https://upload.wikimedia.org/wikipedia/commons/b/b4/Hamming_distance_3_bit_binary.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming\_distance\_3\_bit\_binary\_example.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/6/6e/Hamming\\_distance\\_3\\_bit\\_binary\\_example.svg](https://upload.wikimedia.org/wikipedia/commons/6/6e/Hamming_distance_3_bit_binary_example.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming\_distance\_4\_bit\_binary.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/b/bf/Hamming\\_distance\\_4\\_bit\\_binary.svg](https://upload.wikimedia.org/wikipedia/commons/b/bf/Hamming_distance_4_bit_binary.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Hamming\_distance\_4\_bit\_binary\_example.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/b/b4/Hamming\\_distance\\_4\\_bit\\_binary\\_example.svg](https://upload.wikimedia.org/wikipedia/commons/b/b4/Hamming_distance_4_bit_binary_example.svg) *License:* CC-BY-SA-3.0 *Contributors:* This vector image was created with Inkscape. *Original artist:* en>User:Cburnett
- **File:Illustration\_of\_Proof\_for\_Computationally\_Bounded\_Adversary.png** *Source:* [https://upload.wikimedia.org/wikipedia/commons/d/d4/Illustration\\_of\\_Proof\\_for\\_Computationally\\_Bounded\\_Adversary.png](https://upload.wikimedia.org/wikipedia/commons/d/d4/Illustration_of_Proof_for_Computationally_Bounded_Adversary.png) *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Jim dobler



- **File:Internet\_map\_1024.jpg** Source: [https://upload.wikimedia.org/wikipedia/commons/d/d2/Internet\\_map\\_1024.jpg](https://upload.wikimedia.org/wikipedia/commons/d/d2/Internet_map_1024.jpg) License: CC BY 2.5 Contributors: Originally from the English Wikipedia; description page is/was here. Original artist: The Opte Project
- **File:Josiah\_Willard\_Gibbs\_-from\_MMS-.jpg** Source: [https://upload.wikimedia.org/wikipedia/commons/c/c7/Josiah\\_Willard\\_Gibbs\\_-from\\_MMS-.jpg](https://upload.wikimedia.org/wikipedia/commons/c/c7/Josiah_Willard_Gibbs_-from_MMS-.jpg) License: Public domain Contributors: Frontispiece of *The Scientific Papers of J. Willard Gibbs*, in two volumes, eds. H. A. Bumstead and R. G. Van Name, (London and New York: Longmans, Green, and Co., 1906) Original artist: Unknown. Uploaded by Serge Lachinov (обработка для wiki)
- **File:Kraft\_inequality\_example.png** Source: [https://upload.wikimedia.org/wikipedia/commons/0/04/Kraft\\_inequality\\_example.png](https://upload.wikimedia.org/wikipedia/commons/0/04/Kraft_inequality_example.png) License: Public domain Contributors: Own work Original artist: Mjoachimiaik
- **File:LDPC\_encoder\_Figure.png** Source: [https://upload.wikimedia.org/wikipedia/commons/4/48/LDPC\\_encoder\\_Figure.png](https://upload.wikimedia.org/wikipedia/commons/4/48/LDPC_encoder_Figure.png) License: CC BY-SA 3.0 Contributors: Word Original artist: Mrcodeguy
- **File:LampFlowchart.svg** Source: <https://upload.wikimedia.org/wikipedia/commons/9/91/LampFlowchart.svg> License: CC-BY-SA-3.0 Contributors: vector version of Image:LampFlowchart.png Original artist: svg by Booyabazooka
- **File:Ldpc\_code\_fragment\_factor\_graph.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/7/77/Ldpc\\_code\\_fragment\\_factor\\_graph.svg](https://upload.wikimedia.org/wikipedia/commons/7/77/Ldpc_code_fragment_factor_graph.svg) License: Public domain Contributors: en-wiki Original artist: The raster image was created by Emin Martinian and placed into the public domain, vector version by
- **File:Ldpc\_code\_fragment\_factor\_graph\_w\_erasures\_decode\_step\_2.svg** Source: [https://upload.wikimedia.org/wikipedia/en/a/ad/Ldpc\\_code\\_fragment\\_factor\\_graph\\_w\\_erasures\\_decode\\_step\\_2.svg](https://upload.wikimedia.org/wikipedia/en/a/ad/Ldpc_code_fragment_factor_graph_w_erasures_decode_step_2.svg) License: PD Contributors: ? Original artist: ?
- **File:Linear\_Binary\_Block\_Codes\_and\_their\_needed\_Check\_Symbols.png** Source: [https://upload.wikimedia.org/wikipedia/commons/4/4b/Linear\\_Binary\\_Block\\_Codes\\_and\\_their\\_needed\\_Check\\_Symbols.png](https://upload.wikimedia.org/wikipedia/commons/4/4b/Linear_Binary_Block_Codes_and_their_needed_Check_Symbols.png) License: CC BY-SA 4.0 Contributors: Own work Original artist: Frank Klemm
- **File:Lock-green.svg** Source: <https://upload.wikimedia.org/wikipedia/commons/6/65/Lock-green.svg> License: CC0 Contributors: en:File:Free-to-read\_lock\_75.svg Original artist: User:Trappist the monk
- **File:Mergefrom.svg** Source: <https://upload.wikimedia.org/wikipedia/commons/0/0f/Mergefrom.svg> License: Public domain Contributors: ? Original artist: ?
- **File:Multigrade\_operator\_XOR.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/1/1e/Multigrade\\_operator\\_XOR.svg](https://upload.wikimedia.org/wikipedia/commons/1/1e/Multigrade_operator_XOR.svg) License: Public domain Contributors: Own work Original artist: <a href="//commons.wikimedia.org/wiki/File:Watchduck.svg" class="image"></a> Watchduck (a.k.a. Tilman Piesk)
- **File:Nuvola\_apps\_edu\_mathematics\_blue-p.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/3/3e/Nuvola\\_apps\\_edu\\_mathematics\\_blue-p.svg](https://upload.wikimedia.org/wikipedia/commons/3/3e/Nuvola_apps_edu_mathematics_blue-p.svg) License: GPL Contributors: Derivative work from Image:Nuvola apps edu mathematics.png and Image:Nuvola apps edu mathematics-p.svg Original artist: David Vignoni (original icon); Flamurai (SVG conversion); bayo (color)
- **File:Online\_codes\_highlevel.png** Source: [https://upload.wikimedia.org/wikipedia/en/2/2c/Online\\_codes\\_highlevel.png](https://upload.wikimedia.org/wikipedia/en/2/2c/Online_codes_highlevel.png) License: PD Contributors: Own work Original artist: Agl
- **File:Online\_codes\_time\_against\_blocks\_graph.png** Source: [https://upload.wikimedia.org/wikipedia/en/d/d4/Online\\_codes\\_time\\_against\\_blocks\\_graph.png](https://upload.wikimedia.org/wikipedia/en/d/d4/Online_codes_time_against_blocks_graph.png) License: PD Contributors: Own work Original artist: Agl
- **File:POV-Ray-Dodecahedron.svg** Source: <https://upload.wikimedia.org/wikipedia/commons/a/a4/Dodecahedron.svg> License: CC-BY-SA-3.0 Contributors: Vectorisation of Image:Dodecahedron.jpg Original artist: User:DTR
- **File:Phone\_icon\_rotated.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/d/df/Phone\\_icon\\_rotated.svg](https://upload.wikimedia.org/wikipedia/commons/d/df/Phone_icon_rotated.svg) License: Public domain Contributors: Originally uploaded on en.wikipedia Original artist: Originally uploaded by Beao (Transferred by varnent)
- **File:Question\_book-new.svg** Source: [https://upload.wikimedia.org/wikipedia/en/9/99/Question\\_book-new.svg](https://upload.wikimedia.org/wikipedia/en/9/99/Question_book-new.svg) License: Cc-by-sa-3.0 Contributors: Created from scratch in Adobe Illustrator. Based on Image:Question book.png created by User:Equazcion Original artist: Tkgd2007
- **File:Question\_dropshade.png** Source: [https://upload.wikimedia.org/wikipedia/commons/d/dd/Question\\_dropshade.png](https://upload.wikimedia.org/wikipedia/commons/d/dd/Question_dropshade.png) License: Public domain Contributors: Image created by JRM Original artist: JRM
- **File:Recovering\_P(x)\_with\_an\_Error\_Locator\_Polynomial.png** Source: [https://upload.wikimedia.org/wikipedia/commons/e/e8/Recovering\\_P%28x%29\\_with\\_an\\_Error\\_Locator\\_Polynomial.png](https://upload.wikimedia.org/wikipedia/commons/e/e8/Recovering_P%28x%29_with_an_Error_Locator_Polynomial.png) License: CC0 Contributors: Drawn with Matplotlib and Inkscape Original artist: Vbutterin
- **File:Reflected\_binary\_Gray\_2632058.png** Source: [https://upload.wikimedia.org/wikipedia/commons/c/c4/Reflected\\_binary\\_Gray\\_2632058.png](https://upload.wikimedia.org/wikipedia/commons/c/c4/Reflected_binary_Gray_2632058.png) License: Public domain Contributors: from Frank Gray's U.S. Patent 2,632,058 (TIFF from USPTO) Original artist: Frank Gray
- **File:Rep.png** Source: <https://upload.wikimedia.org/wikipedia/commons/1/1c/Rep.png> License: CC BY-SA 3.0 Contributors: Own work Original artist: Bjr244
- **File:Rotate\_right.svg** Source: [https://upload.wikimedia.org/wikipedia/commons/3/37/Rotate\\_right.svg](https://upload.wikimedia.org/wikipedia/commons/3/37/Rotate_right.svg) License: CC-BY-SA-3.0 Contributors: This vector image was created with Inkscape. Original artist: en:User:Cburnett

- **File:Rubik'{}s\_cube\_v3.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/b/b6/Rubik%27s\\_cube\\_v3.svg](https://upload.wikimedia.org/wikipedia/commons/b/b6/Rubik%27s_cube_v3.svg) *License:* CC-BY-SA-3.0 *Contributors:* Image:Rubik'{}s\_cube\_v2.svg *Original artist:* User:Booyabazooka, User:Meph666 modified by User:Niabot
- **File:Secure\_Sketch\_Constructions.PNG** *Source:* [https://upload.wikimedia.org/wikipedia/commons/d/d3/Secure\\_Sketch\\_Constructions.PNG](https://upload.wikimedia.org/wikipedia/commons/d/d3/Secure_Sketch_Constructions.PNG) *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Biernat2
- **File:Symbol\_template\_class.svg** *Source:* [https://upload.wikimedia.org/wikipedia/en/5/5c/Symbol\\_template\\_class.svg](https://upload.wikimedia.org/wikipedia/en/5/5c/Symbol_template_class.svg) *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:TNC\_coding\_4\_packets\_together.PNG** *Source:* [https://upload.wikimedia.org/wikipedia/commons/1/15/TNC%2C\\_coding\\_4\\_packets\\_together.PNG](https://upload.wikimedia.org/wikipedia/commons/1/15/TNC%2C_coding_4_packets_together.PNG) *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Jqureshi786
- **File:Tanner\_graph\_example.PNG** *Source:* [https://upload.wikimedia.org/wikipedia/commons/6/6b/Tanner\\_graph\\_example.PNG](https://upload.wikimedia.org/wikipedia/commons/6/6b/Tanner_graph_example.PNG) *License:* BSD *Contributors:* ? *Original artist:* ?
- **File:Telecom-icon.svg** *Source:* <https://upload.wikimedia.org/wikipedia/commons/4/4e/Telecom-icon.svg> *License:* Public domain *Contributors:* Vectorized by User:Booyabazooka from original small PD raster image File:Telecom-icon.jpg *Original artist:* Vectorized by User:Booyabazooka from original small PD raster image File:Telecom-icon.jpg
- **File:Text\_document\_with\_red\_question\_mark.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/a/a4/Text\\_document\\_with\\_red\\_question\\_mark.svg](https://upload.wikimedia.org/wikipedia/commons/a/a4/Text_document_with_red_question_mark.svg) *License:* Public domain *Contributors:* Created by bdesham with Inkscape; based upon Text-x-generic.svg from the Tango project. *Original artist:* Benjamin D. Esham (bdesham)
- **File:US02632058\_Gray.png** *Source:* [https://upload.wikimedia.org/wikipedia/commons/1/14/US02632058\\_Gray.png](https://upload.wikimedia.org/wikipedia/commons/1/14/US02632058_Gray.png) *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Wiki\_letter\_w.svg** *Source:* [https://upload.wikimedia.org/wikipedia/en/6/6c/Wiki\\_letter\\_w.svg](https://upload.wikimedia.org/wikipedia/en/6/6c/Wiki_letter_w.svg) *License:* Cc-by-sa-3.0 *Contributors:* ? *Original artist:* ?
- **File:Wiki\_letter\_w\_cropped.svg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/1/1c/Wiki\\_letter\\_w\\_cropped.svg](https://upload.wikimedia.org/wikipedia/commons/1/1c/Wiki_letter_w_cropped.svg) *License:* CC-BY-SA-3.0 *Contributors:* This file was derived from Wiki letter w.svg: `<a href="//commons.wikimedia.org/wiki/File:Wiki_letter_w.svg" class="image"></a>` *Original artist:* Derivative work by Thumperward
- **File:Zbound.png** *Source:* <https://upload.wikimedia.org/wikipedia/commons/f/fc/Zbound.png> *License:* FAL *Contributors:* <http://www.cse.buffalo.edu/~{ }atri/courses/coding-theory/> <http://www.cse.buffalo.edu/~{ }atri/courses/coding-theory/lectures/lect24.pdf> *Original artist:* Atri Rudra
- **File:Zemor\_Decoding.jpg** *Source:* [https://upload.wikimedia.org/wikipedia/commons/e/ed/Zemor\\_Decoding.jpg](https://upload.wikimedia.org/wikipedia/commons/e/ed/Zemor_Decoding.jpg) *License:* CC BY-SA 3.0 *Contributors:* I had created the figure for wikipedia content Zemor Decoding Algorithm *Original artist:* Sujith17889

### 112.6.3 Content license

- Creative Commons Attribution-Share Alike 3.0