



HAL
open science

Algorithms for the Sparse Random 3XOR Problem

Charles Bouillaguet, Claire Delaplace

► **To cite this version:**

Charles Bouillaguet, Claire Delaplace. Algorithms for the Sparse Random 3XOR Problem. 2020.
hal-02306917v2

HAL Id: hal-02306917

<https://hal.science/hal-02306917v2>

Preprint submitted on 12 May 2020 (v2), last revised 2 Oct 2021 (v4)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Algorithms for the Sparse Random 3XOR Problem

Charles Bouillaguet 

University of Lille, France

charles.bouillaguet@univ-lille.fr

Claire Delaplace

Ruhr University Bochum, Germany

claire.delaplace@rub.de

Abstract

We present two new algorithms for a variant of the 3XOR problem with lists consisting of n -bit vectors whose coefficients are drawn randomly according to a Bernoulli distribution of parameter $p < 1/2$. We show that in this particular context the problem can be solved much more efficiently than in the general setting. We first present a simple adaptation of the folklore quadratic algorithm that discards heavy vectors in a preprocessing step. This leads to a linear algorithm with overwhelming success probability for $p < 1/11$, and is sub-quadratic for all $p < 1/2$. We also describe a variant of this method which succeeds deterministically, which is also linear for $p < 1/47$ and always sub-quadratic.

We finally propose a randomized algorithm with a sub-quadratic time complexity when the lists consists of vector of fixed Hamming weight, and discuss possible further improvements.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography; Theory of computation

Keywords and phrases Algorithms, 3-xor problem, random sparse 3-xor

1 Introduction

Given three lists A , B and C of n -bit vectors, the 3XOR problem consists in deciding the existence of (or even finding) a triplet $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in A \times B \times C$ such that $\mathbf{x} \oplus \mathbf{y} \oplus \mathbf{z}$ is equal to a given target, often assumed to be zero (here the \oplus symbol represent the exclusive-OR or XOR).

This problem can be seen as a variant of the celebrated 3SUM problem, where this time the input list items are seen as integers and we must have $x + y + z = 0$. Many geometric problems can be reduced to 3SUM in sub-quadratic time, and those problem are said to be 3SUM hard [6]. Although the 3XOR problem has enjoyed less interest in the complexity theory field, there exists a few such reductions. For instance, it is a fact that any $\mathcal{O}(N^{2-\epsilon})$ algorithm for the 3XOR problem with input lists of size N would imply faster-than-expected algorithms for listing triangles in a graph [14, 7]. Another result due to [4] show that an algorithm solving the 3XOR problem in time $\Omega(n^{2-o(1)})$ also reduces the time complexity of the offline SETDISJOINTNESS and SETINTERSECTION problems.

The 3XOR problem also has some cryptographic applications, in which the input lists consist of uniformly random vectors (the cryptographic community makes this assumption “by default”). In particular, we can mention Nandi’s attack [11] against the COPA mode of authenticated encryption, or the more recent attack against the two-round single-key Even-Mansour cipher by Leurent and Sibleyras [9]. May and Both have been considering a variant of the 3XOR problem, the *approximate 3-list birthday problem*: given three lists of N uniformly random elements of $\{0, 1\}^n$ the goal consist in finding triplets $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ in the list such that the hamming weight of $\mathbf{x} \oplus \mathbf{y} \oplus \mathbf{z}$ is small [2].

The simplest possible algorithm to solve the 3XOR problem is the quadratic algorithm, which consists in taking all $\mathbf{x} \oplus \mathbf{y} \in A \times B$ and checking whether they belong to C . Using an optimal static dictionary [5] to hold C , this results in a time complexity of $\mathcal{O}(|A| \cdot |B| + |C|)$.

46 In the particular case where $|A| = |B| = |C| = N$ this algorithm runs in time $\mathcal{O}(N^2)$. In
 47 the following, this simple algorithm will be referred to as QUADRATICALGORITHM.

48 When the input lists are made of random vectors, the decisional variant of the problem
 49 may be trivial: if the input lists are too long (resp. too short), then the existence (resp.
 50 absence) of a “3XOR triplet” ($\mathbf{x} \oplus \mathbf{y} \oplus \mathbf{z} = 0$) in the input may be asserted with high
 51 probability without even observing the input. In this setting, a computational variant of the
 52 problem, namely actually finding a *single* 3XOR triplet given the input lists makes more
 53 sense.

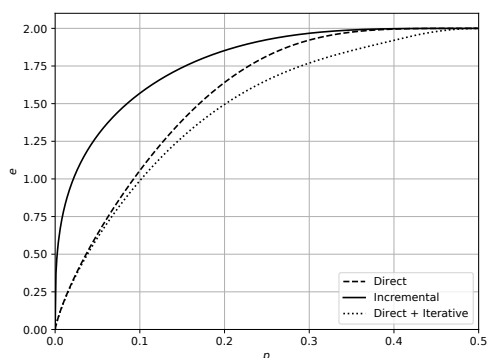
54 We believe that with random vectors, the hardest case occurs when the size of the input
 55 lists N is chosen such that they contain one (and only one) solution with high probability. In
 56 any case, if the input lists were longer, they could always be truncated to this size. In the
 57 case where the vectors are drawn uniformly at random in $\{0, 1\}^n$, this means that $N \approx 2^{n/3}$.
 58 In this particular case, the quadratic algorithm is mostly the only option to recover the
 59 solution. Some improvements of this method exist [3, 4], however these improvements allows
 60 only to gain a polynomial factor in n . It is not clear today whether it is possible to find an
 61 algorithm for this problem with complexity below $N^{2-o(1)}$.

62 **Contributions.** In this paper, we focus on a variant of the 3XOR problem where the elements
 63 of the lists are random and *sparse*. More precisely, each input bit is drawn independently
 64 at random according to a Bernoulli distribution of parameter $0 < p < 1/2$ — the “dense”
 65 random case corresponds to $p = 1/2$. The sparse variant of the problem is quite different
 66 from its dense counterpart.

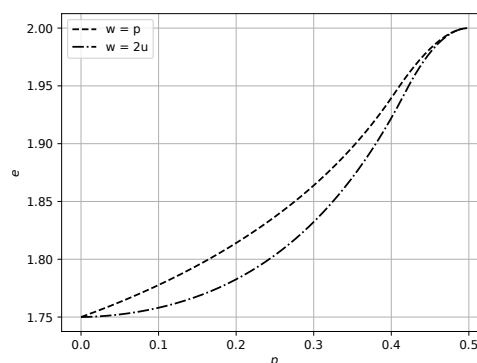
67 We first give the probability that the input actually contains a 3XOR triplet for given N
 68 and p . To the best of our knowledge, this result was not readily available from the existing
 69 literature, not even in the simple case where $p = 1/2$. We then describe three algorithms to
 70 solve the random sparse 3XOR problem. The simplest possible one (Section 3) works by
 71 discarding useless input vectors (whose hamming weight is above a well-chosen threshold),
 72 then searches a solution using the quadratic algorithm. This first algorithm returns the
 73 solution with overwhelming probability. We also propose an incremental version of this
 74 algorithm (Section 4) which deterministically returns a solution if there is one in the input.
 75 These algorithms have a time complexity of $\mathcal{O}(N + N^e)$, for some parameter $e < 2$ when
 76 $p < 1/2$. The evolution of this parameter e in function of p is shown in Figure 1 (left diagram).
 77 In particular, both algorithms are shown to be *linear* when p is small enough — this stands
 78 in strong contrast with the dense case.

79 In the rest of the paper we propose alternatives to the quadratic algorithms to deal with
 80 instances (A_i, B_j, C_k) where the elements of the lists have fixed Hamming weight, respectively
 81 i, j, k . In this context, useless heavy vectors have already been discarded and the sparsity of
 82 the input vectors has to be exploited differently. We use techniques inspired from decoding
 83 algorithms.

84 This is an interesting case as all instances of the sparse 3XOR problem can be converted
 85 to several independent sub-instance of this type. In the first of these algorithms (Section 5)
 86 we select randomly a subset J of the indices and “guess” that a 3XOR triplet has only
 87 zeroes on columns in J . From here, we consider the sublists A', B' and C' of A_i, B_j , and
 88 C_k , consisting only of vectors whose coefficients indexed by $j \in J$ are zeroes. We solve this
 89 smaller instance with the quadratic algorithm. If no solution is found, we try again with a
 90 different J . For a well chosen size of J , this algorithm is at least as fast as the quadratic
 91 algorithm. In the particular case where $i = j = k$, we show that the complexity of the
 92 algorithm is between $N^{7/4}$ (when p is close to zero) and N^2 (when p is close to $1/2$), where



(a) The DIRECT algorithm of section 3 and the INCREMENTAL algorithm of section 4 run in time $\mathcal{O}(N + N^e)$ where e is shown here.



(b) The ITERATIVE algorithm of section 5 runs in time $\tilde{\mathcal{O}}(N^e)$ where e is shown here.

Figure 1 Exponents in the time complexity in function of p when the input lists contain one 3XOR triplet with high probability.

93 N is the size of the input lists.

94 Finally, we discuss possible improvement of this method in section 6, which basically
 95 consists in re-iterating the filtering steps a constant number of times instead of solving the
 96 sub-instance directly with the quadratic algorithm. This borrows the main technique of the
 97 “nearest neighbors” algorithm of May and Ozerov [10] (which is used in a decoding algorithm).
 98 Given a parameter t , we split the indices in t slices. We select randomly a subset J_1 of the
 99 indices belonging to the first slice and guess that the solution is zero over the columns in
 100 J_1 . We then build the sublists $A_i^{(1)}, B_j^{(1)}, C_k^{(1)}$ of the vectors whose coefficients indexed by
 101 $\ell \in J_1$ are zero. After that we select a random subsets J_2 of the indices belonging to the
 102 second slice and re-iterate until we obtain the lists $A_i^{(t)}, B_j^{(t)}, C_k^{(t)}$, which we process with the
 103 quadratic algorithm. The trick is that, if one of our guess J_ℓ was wrong, we do not have to
 104 restart the whole process, but only starting from J_ℓ . Although we did not fully investigate
 105 the time complexity of this algorithm, we believe that this method should be more efficient
 106 in practice than the previous one.

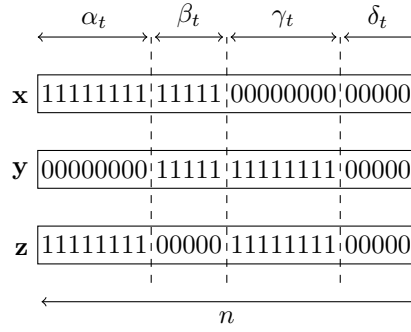
107 **Motivation** The algorithms described in this paper have no concrete application that we
 108 know of, and we don’t really care. However they can be used to obtain the —non-trivial and
 109 “interesting”— result shown in Appendix C.

110 2 Preliminaries

111 2.1 Notations, Definition and Useful Properties

112 Let $\mathbf{x} = x_0x_1 \dots x_{n-1}$ be an n -bit string (we use “bit string” and “vector” as synonyms).
 113 We denote by $\text{wt}(\mathbf{x})$ its Hamming weight. We denote by $\sim \mathbf{x}$ the negation of \mathbf{x} (XORing 1
 114 to each bit) and by $\mathbf{x} \& \mathbf{y}$ the bit-wise AND of \mathbf{x} and \mathbf{y} . We denote by $\mathbf{x}_{\setminus j}$ the bit-string
 115 $x_0 \dots x_{j-1}x_{j+1} \dots x_{n-1}$; more generally, if J is a subset of $\{0, \dots, n-1\}$, we denote by $\mathbf{x}_{\setminus J}$
 116 the sub-string of \mathbf{x} , where all x_j for $j \in J$ have been discarded. Let A be a list ; we denote
 117 by $|A|$ the number of elements in A . Let $A[i]$ be the i -th element of A . We denote by A_i the
 118 sublist of A such that $A_i = \{\mathbf{x} \in A \mid \text{wt}(\mathbf{x}) = i\}$.

119 **Definition 1** (3XOR triplet). *Let \mathbf{x}, \mathbf{y} and \mathbf{z} be three n -bit strings. We say that $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is*
 120 *a 3XOR triplet if $\mathbf{x} \oplus \mathbf{y} \oplus \mathbf{z} = 0$.*



■ **Figure 2** Shape of a sparse random 3XOR triplet $t = (\mathbf{x}, \mathbf{y}, \mathbf{z})$, up to column permutation.

121 For any triplet $t = (\mathbf{x}, \mathbf{y}, \mathbf{z})$ of n -bit strings and any index $0 \leq j < n$, we denote by \mathbf{t}_j the 3-
 122 bit string $x_j y_j z_j$ and we say that it is the *type* of column j in t . In a 3XOR triplet, the possible
 123 column types are $\{000, 011, 101, 110\}$. Therefore, up to columns permutations, the shape of a
 124 3XOR triplet can be described by Figure 2. Given again a triplet $t = (\mathbf{x}, \mathbf{y}, \mathbf{z})$ of n -bit strings,
 125 we consider the following functions $\alpha_t = wt(\mathbf{x} \& \mathbf{z})$, $\beta_t = wt(\mathbf{x} \& \mathbf{y})$, $\gamma_t = wt(\mathbf{y} \& \mathbf{z})$ and
 126 $\delta_t = wt(\sim \mathbf{x} \& \sim \mathbf{y} \& \sim \mathbf{z})$. In any case, if $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is a 3XOR triplet we have $\alpha_t + \beta_t + \gamma_t + \delta_t = n$.

127

128 ► **Definition 2** (3XOR problem with distribution). Let \mathcal{D} be a probability distribution over
 129 $\{0, 1\}^n$. Let A, B and C be three lists of elements drawn independently at random according
 130 to \mathcal{D} . A solution to the instance of the 3XOR problem given by (A, B, C) is a 3XOR triplet
 131 $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in A \times B \times C$. A random 3XOR triplet is a triplet $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ chosen according to \mathcal{D}
 132 conditioned to $\mathbf{x} \oplus \mathbf{y} \oplus \mathbf{z} = 0$.

133 Because of randomness of the input, the question of the existence of a 3XOR triplet in
 134 (A, B, C) may be easy to decide with low probability of error without even observing A, B
 135 and C , depending on n, \mathcal{D} and the size of the lists. Therefore, our main focus is on the search
 136 problem (actually producing a solution, not merely deciding its existence).

137 **Bounds for Binomial Distributions.** Let $\mathcal{B}(n, p)$ denote the binomial distribution with
 138 parameters n, p . We denote by \log the logarithm in basis 2 and by H the binary entropy
 139 function, meaning that $H(x) = -x \log(x) - (1-x) \log(1-x)$, for all $0 < x < 1$. The following
 140 standard bounds for the binomial coefficient can be derived from Stirling's formula:

$$141 \quad \frac{2^{nH(x)}}{\sqrt{8nx(1-x)}} \leq \binom{n}{xn} \leq \frac{2^{nH(x)}}{\sqrt{2\pi nx(1-x)}}, \quad (0 < x < 1/2) \quad (1)$$

142 We make heavy use of tail bounds for binomial distributions, notably the Chernoff
 143 bound (2) and the tighter classical inequality (3), a proof of which can be found in [1]
 144 amongst others.

$$145 \quad \Pr(X \leq k) \leq \exp\left(-\frac{1}{2p} \frac{(np - k)^2}{n}\right), \quad \text{if } \frac{k}{n} < p \quad (2)$$

$$146 \quad \Pr(X \leq an) \leq \exp -nD(a, p) \quad \text{if } a < p, \quad (3)$$

$$147 \quad \Pr(X \geq an) \leq \exp -nD(a, p) \quad \text{if } a > p,$$

149 where $D(a, p) = a \ln \frac{a}{p} + (1-a) \ln \frac{1-a}{1-p}$ is the Kullback-Leibler divergence between an a -coin
 150 and a p -coin.

151 **Computational Model.** We consider a transdichotomous word Random Access Machine
 152 (word-RAM) model. In this model, we have access to a machine in which each “memory
 153 cell” contains a n -bit word. We assume that the usual arithmetic and bit-wise operations on
 154 n -bit words, as well as the comparison of two n -bit integers and memory access with n -bit
 155 addresses can be done in constant time. In other terms, we assume that the machine is large
 156 enough to accommodate the instance of the problem at hand.

157 2.2 Properties of Random Sparse 3XOR Triplets

158 Let $0 < p < 1/2$ be fixed. We denote by Ber_p the Bernoulli distribution of parameter p (if
 159 $x \stackrel{\$}{\leftarrow} Ber_p$, then $\Pr(x = 1) = p$). Let \mathcal{D} the distribution over $\{0, 1\}^n$, where each bit is drawn
 160 independently from Ber_p . This paper focus on the 3XOR problem with input distribution \mathcal{D} .

161 **Existence of a 3XOR triplet.** Let a, b and c be random bits drawn according to Ber_p ; the
 162 probability that they XOR to zero is $(1 - p)(1 - 2p + 4p^2)$. It follows that if \mathbf{x}, \mathbf{y} and \mathbf{z} are
 163 random bit strings drawn according to \mathcal{D} , then $\Pr(\mathbf{x} \oplus \mathbf{y} \oplus \mathbf{z} = 0) = (1 - p)^n(1 - 2p + 4p^2)^n$.
 164 Given three lists A, B and C of random bit strings drawn according to \mathcal{D} , each of size N ,
 165 the number of 3XOR triplets in (A, B, C) is a random variable, denoted by Y below, that
 166 follows a binomial distribution. We have the following result.

167 **► Theorem 3.** Let $q_0 = (1 - p)(1 - 2p + 4p^2)$, $q_1 = (1 - p)(1 - 4p + 8p^2 - 4p^3)$ and
 168 $q_2 = (1 - p)(1 - 3p + 4p^2)$. Then:

- 169 *i)* $\mathbb{E}Y = N^3 q_0^n$
 170 *ii)* $\Pr(Y = 0) \leq 3N^{-1} (q_1/q_0^2)^n + 3N^{-2} (q_2/q_0^2)^n + N^{-3}(1/q_0)^n$.
 171 *iii)* $\Pr(Y = 0) \leq \frac{3}{(\mathbb{E}Y)^{1/3}} + \frac{3}{(\mathbb{E}Y)^{2/3}} + \frac{1}{\mathbb{E}Y}$.

172 The proof of this theorem can be found in Appendix A.1. The inequality *ii)* is tighter
 173 than *iii)*, but *iii)* is often more practical. A first difference between the sparse case and the
 174 dense case is that in (exponentially) smaller input lists are sufficient to ensure the existence
 175 of a 3XOR triplet with high probability.

176 For instance, with $p = 1/16$ and $n = 512$, there is *one expected* 3XOR triplet in the input
 177 with $N = 2^{44.41}$ (compare this to $N = 2^{170.7}$ with $p = 1/2$). Point *ii)* of theorem 3 states
 178 that if we want this solution to be there with probability 99%, we need $N \geq 2^{46.70}$ — the
 179 input lists must be 6.5 times larger.

180 **Expected Density of a 3XOR Triplet.** Let us now consider a sparse random 3XOR triplet
 181 $t = (\mathbf{x}, \mathbf{y}, \mathbf{z})$; for a given column j , we find that

$$182 \quad u := \Pr(\mathbf{t}_j = 110) = \Pr(\mathbf{t}_j = 101) = \Pr(\mathbf{t}_j = 011) = p^2/(1 - 2p + 4p^2),$$

$$183 \quad v := \Pr(\mathbf{t}_j = 000) = (1 - p)^2/(1 - 2p + 4p^2).$$

184
 185 Note that $3u + v = 1$. This means that $(\alpha_t, \beta_t, \gamma_t, \delta_t)$ is a vector of random variables that
 186 follows a multinomial distribution of parameters n and (u, u, u, v) . Therefore, α_t, β_t and γ_t
 187 individually follow the binomial distribution $\mathcal{B}(n, u)$ and it follows that the expected “density”
 188 of \mathbf{x}, \mathbf{y} and \mathbf{z} is $2u$. This is always smaller than p when $0 < p < 1/2$. In other terms: *random*
 189 *triplets drawn from \mathcal{D} have density p , but random 3XOR triplets drawn from \mathcal{D} have smaller*
 190 *density*. The algorithms described in this paper take advantage of this fact.

191 **2.3 Methodology**

192 We decided to focus on the case where $N = \text{poly}(n) \cdot q_0^{-n/3}$. In that case we can expect
 193 to have a constant number of 3XOR triplets in $A \times B \times C$ with high probability thanks to
 194 theorem 3. This decision can be justified in two ways.

- 195 ■ First, if the input lists are too long, then we may simply look only at the first $\text{poly}(n) \cdot$
 196 $q_0^{-n/3}$ entries. Theorem 3 then tells us that we can still expect to find a solution with
 197 high probability.
- 198 ■ Second, if the input lists contain “too many” elements, there will be trivial and unin-
 199 teresting solutions. Indeed, if $N \geq [1/(1-p)]^n$, we can expect the string $000\dots 0$ to
 200 be present in all three lists. It follows that “return $(0, 0, 0)$ ” would be a constant-time
 201 algorithm for the sparse random 3XOR problem with a high success probability.

202 The algorithms we present below would also work for $N < q_0^{-n/3}$, but since the lists
 203 consists of random elements, it is unlikely that a solution exists in this case. We can however
 204 imagine the following scenario, when a solution following the distribution \mathcal{D} is created and
 205 injected inside of smaller lists. In this case, our algorithms will find it with the claimed
 206 probability.

207 Consistently with this assumption on the size of the input, we assume that there is a
 208 3XOR triplet $t^* = (\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ in the input, and we study the performance of our algorithms
 209 in returning this triplet t^* .

210 **3 A Very Simple, “Direct”, Algorithm**

211 In this section, we present the simplest possible algorithm that solves the sparse 3XOR
 212 problem with interesting theoretical guarantees when p is small. Its most striking feature is
 213 that it succeeds with overwhelming probability in linear time when p is small enough.

214 It follows from our preliminary observations that random sparse 3XOR triplets are *very*
 215 sparse. This can be exploited in the following simple way: choose a *threshold weight* w ;
 216 remove from the input all vectors of hamming weight greater than or equal to w and run the
 217 quadratic algorithm on the filtered lists.

218 Choosing a small value of w leads to smaller “filtered” instances and thus to a smaller
 219 running time for the actual computation using the quadratic algorithm. However, if w is too
 220 small, then a potential solution present in the input might be discarded by the filtering step.
 221 To avoid this, we choose the value of w in the range $]2nu, np[$ — above the expected density
 222 of random 3XOR triplets so that we do not discard them, and below the density of the input
 223 lists in order to actually discard input vectors that are too heavy.

224 Any value of w in this “admissible range” ensures an exponentially small failure probability.
 225 Indeed, the solution present in the original input is discarded by the filtering step if and
 226 only if the weight of either $\mathbf{x}^*, \mathbf{y}^*$ or \mathbf{z}^* is greater than or equal to w . Define ϵ with
 227 $2un(1 + \epsilon) = w$; it follows that ϵ is strictly positive. We know that the expected weight
 228 of $\mathbf{x}^*, \mathbf{y}^*$ and \mathbf{z}^* is $2un$, therefore the Chernoff bound (2) shows that either has weight
 229 greater than w with probability less than $\exp(-nue^2)$. A union bound then ensures that the
 230 solution is discarded with probability less than $3\exp(-nue^2)$, so the algorithm succeeds with
 231 overwhelming probability as long as the original input contains at least one solution.

232 We now discuss the complexity of the algorithm. Filtering the input lists takes time $\mathcal{O}(N)$.
 233 Let $X \sim \mathcal{B}(n, p)$ be a (binomial) random variable modeling the weight of an input vector
 234 of density p . Such a vector is kept by the filtering step if its weight is less than w , and
 235 this happens with probability $s := \Pr(X < w)$. Let A', B', C' denote the filtered lists; their

236 sizes are stochastically independent random variables following binomial distributions of
 237 parameters N, s and their expected size is Ns . The expected running time of the quadratic
 238 algorithm on the filtered instance is therefore $E(|A'| + |B'| + |C'|) = N^2 s^2 + Ns$.

239 How to choose w ? We could target the middle of the admissible range $]2nu; np[$, however
 240 a much better choice is $w = np^3 + (1 - p^2)2nu = 2nu(1 + \epsilon)$ with $\epsilon = \frac{p(1-2p)^2}{2}$. The idea is
 241 that the filtering threshold weight drifts closer to $2nu$ when p gets closer to zero. This leads
 242 to the DIRECT algorithm shown below.

Algorithm 1 A “direct” algorithm for the sparse 3XOR problem.

```

1: function FILTER( $L, w$ )
2:   return  $\{\mathbf{x} \in L \mid \text{wt}(\mathbf{x}) < w\}$ 

3: function DIRECT( $A, B, C$ )
4:   Set  $w \leftarrow n(p^3 + (1 - p^2)2u)$ .
5:   Set  $A' \leftarrow \text{FILTER}(A, w)$ ,  $B' \leftarrow \text{FILTER}(B, w)$  and  $C' \leftarrow \text{FILTER}(C, w)$ .
6:   return QUADRATICALGORITHM( $A', B', C'$ ).

```

243 Using the binomial tail bound (3), we see that the DIRECT algorithm has an expected
 244 running-time of:

$$245 \quad ET = N + N^2 \exp(-\lambda n) + o(N) \quad \text{with} \quad \lambda = 2D \left(\frac{(2 - 3p + 2p^2)(2p + 1)p^2}{1 - 2p + 4p^2}, p \right)$$

246 Let $e = \log_N(ET - N)$, so that the algorithm runs in time $\mathcal{O}(N + N^e)$. A quick calculation
 247 shows that:

$$248 \quad e = 2 + 6 \frac{D \left(\frac{(2 - 3p + 2p^2)(2p + 1)p^2}{1 - 2p + 4p^2}, p \right)}{\ln(1 - p)(1 - 2p + 4p^2)}.$$

249 The graph of e is shown in Fig. 1a. When p goes to zero, the exponent e reaches a limit
 250 of zero. When p goes to $\frac{1}{2}$, then $e = 2$. In between, e is increasing. Using the bisection
 251 algorithm, we find that $e \leq 1$ when $p \leq 0.0925$. It follows that the DIRECT algorithm is
 252 linear when $p \leq \frac{1}{11}$. Looking at Fig. 1a, we conjecture that $e \leq 2p(1 - 2 \ln p)$. Establishing
 253 this is left for future work.

254 It is worth noting that the unwieldy expression of e would be greatly simplified from
 255 using the simpler Chernoff bound (2) instead of (3). However, the resulting upper-bound on
 256 the complexity of the algorithm is much looser for small p : it results in $\lim e = 1$ when p
 257 goes to zero.

258 **4 An Incremental Version**

259 The DIRECT algorithm uses an *a priori* threshold on the density of solution to reduce the
 260 size of the instance. This can be improved by starting with a low threshold weight and
 261 progressively increasing it while no solution is found. The resulting INCREMENTAL algorithm
 262 deterministically reveals the 3XOR triplet present in the input, and it also does so in linear
 263 time for small values of p .

264 The main idea is the following: when $w \geq np$, then the filtering step does not reduce
 265 significantly the size of the lists. Thus, all iterations after the np -th cost essentially $\Omega(N^2)$.
 266 However, the w -th iteration (with filtering weight w) is done if and only if any of $\mathbf{x}^*, \mathbf{y}^*$

Algorithm 2 An improved, “incremental” algorithm for the sparse 3XOR problem.

```

1: function INCREMENTAL( $A, B, C$ )
2:   for  $w = 0, 1, 2, \dots, n$  do
3:     Set  $A' \leftarrow \text{FILTER}(A, w)$ ,  $B' \leftarrow \text{FILTER}(B, w)$  and  $C' \leftarrow \text{FILTER}(C, w)$ .
4:      $S \leftarrow \text{QUADRATICALGORITHM}(A', B', C')$ .
5:     if  $S \neq \perp$  then
6:       return  $S$ 

```

267 and \mathbf{z}^* has Hamming weight greater than or equal to w . Their expected weight is $2un$, and
 268 therefore the probability that the most expensive iterations take place is exponentially small.

269 ► **Theorem 4.** *The INCREMENTAL algorithm returns a solution if it exists in time $\mathcal{O}(N + N^e)$,*
 270 *where*

$$271 \quad e = 2 + 3 \frac{2D \left(\frac{1}{1 + \sqrt[3]{(1-\frac{1}{p})^2(1-\frac{1}{2u})}}, p \right) + D \left(\frac{1}{1 + \sqrt[3]{(1-\frac{1}{p})^2(1-\frac{1}{2u})}}, \frac{2p^2}{1-2p+4p^2} \right)}{\ln(1-p)(1-2p+4p^2)}.$$

272 The proof of this Theorem is given in Appendix A.2. The graph of e is also shown in
 273 Fig. 1a. We find again that e is increasing, $\lim_{p \rightarrow 0} e = 0$, $\lim_{p \rightarrow \frac{1}{2}} e = 2$ and $e \leq 1$ when
 274 $p \leq 0.02155$. Thus, this unfailling algorithm is linear when $p \leq \frac{1}{47}$. We conjecture that
 275 $e \leq 3\sqrt[3]{2p} - \frac{4}{5}2p - \frac{1}{5}(2p)^2$ but we leave for future work to prove it. It is worthwhile noting that
 276 the complexity of this algorithm is significantly higher than that of the DIRECT algorithm
 277 of section 3. The difference comes from the difference in success probability: this one is
 278 unfailling.

279 In practice, in order to avoid repeating the same work several time, we propose to dispatch
 280 the entries of the lists according to their Hamming weight (e.g. the list A_i is the sub-list
 281 of A containing only elements of Hamming weight equal to i), and treat independently all
 282 instances (A_i, B_j, C_k) such that the values i, j, k make it possible to find a solution. More
 283 details about how we should proceed is given in Appendix B.

284 **5 A Better Algorithm for the Base Case**

285 The previous algorithms (DIRECT and INCREMENTAL) all exploit the sparsity of the 3XOR
 286 triplets by filtering the input lists in order to reduce their size, and they use the quadratic
 287 algorithm as the “last resort” solution. In this section, we give a better algorithm for the base
 288 case where the lists have already been reduced to low-density vectors. The same filtering
 289 ideas are therefore not directly applicable, and we instead use a different technique, inspired
 290 by the Information Set Decoding algorithm of Lee-Brickell [8].

291 We consider the setting where we are given three lists A_i, B_j and C_k composed of vectors
 292 of hamming weight exactly i, j and k respectively, of possibly different sizes. Up to renaming,
 293 we assume without loss of generality that $|A_i| \leq |B_j| \leq |C_k|$. Our goal is find a 3XOR triplet
 294 with high probability if there is one in the input lists, or return \perp if there is none.

295 The restriction to input vectors of fixed hamming weight is not really stringent ; indeed, if
 296 the input lists were made of vectors of arbitrary hamming weight, then the following strategy
 297 could be implemented: partition each input list in n parts according to the Hamming weight ;
 298 solve the original problem by searching the $\binom{n+3}{3}$ sub-instances $A_{\mu+\nu} \times B_{\nu+\lambda} \times C_{\mu+\lambda}$ subject
 299 to $\mu + \nu + \lambda \leq n$. The point is that any potential 3XOR triplet t^* present in the input is
 300 contained in the sub-instance where μ, ν and λ denote the number of columns of type 110,
 301 101 and 011 of t^* , respectively.

302 The main algorithmic idea is the following: if there is a 3XOR triplet t^* in the input lists,
 303 then it necessarily has $\delta = n - (i + j + k)/2$ columns of “type 000”. We randomly guess a
 304 subset J of these columns and keep only the vectors from the input lists that are zero on
 305 all columns of J . This produces a smaller sub-instance and we solve it using the quadratic
 306 algorithm. If no solution is found, we try again.

307 As opposed to the two previous algorithms, here the generated sub-instances are not
 308 sparser but instead denser than the original input. Correctly choosing the number s of
 309 columns that are clamped to zero is critical for performance. This leads to the following
 310 ITERATIVE algorithm.

Algorithm 3 A sub-quadratic algorithm for the base case.

```

1: function CLAMP( $L, J$ )
2:   // Return the sublist made of vectors which are 0 on the columns in  $J$ .
3:   return  $\{\mathbf{x}_{\setminus J} \mid (\mathbf{x} \in L) \wedge (\forall j \in J, x_j = 0)\}$ 

4: function ITERATIVE( $A_i, B_j, C_k$ )
5:   Let  $\delta \leftarrow n - (i + j + k)/2$ .
6:   Set  $s$  to the integer  $0 \leq s \leq \delta$  that minimizes  $\frac{\binom{n-i}{s}\binom{n-j}{s}}{\binom{n}{s}\binom{\ell}{s}}|A_i| \cdot |B_j| + \frac{\binom{n}{s}}{\binom{\ell}{s}}|C_k|$ .
7:   Let  $I \leftarrow H(s/n) - (\delta/n)H(s/\delta)$ 
8:   for  $t = 0, 1, \dots, n2^{nI}$  do
9:      $J \leftarrow$  uniformly random subset of  $\{1, 2, \dots, n\}$  of size  $s$ .
10:     $A' \leftarrow$  CLAMP( $A_i, J$ ),  $B' \leftarrow$  CLAMP( $B_j, J$ ),  $C' \leftarrow$  CLAMP( $C_k, J$ ).
11:     $S \leftarrow$  QUADRATICALGORITHM( $A', B', C'$ ).
12:    if  $S \neq \perp$  then return  $S$ 
13:   return  $\perp$ 

```

311 To discuss the properties of the algorithm, we assume as usual that there is a 3XOR
 312 triplet $t^* = (\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ in the input. The following lemma establishes the success probability
 313 and justifies the choice of s .

314 **► Lemma 5.** *The ITERATIVE algorithm fails to returns t^* with probability e^{-n} and its*
 315 *expected running time is less than $\frac{\binom{n-i}{s}\binom{n-j}{s}}{\binom{n}{s}\binom{\ell}{s}}|A_i| \cdot |B_j| + \frac{\binom{n}{s}}{\binom{\ell}{s}}|C_k|$.*

316 **Proof.** When the “golden triplet” t^* belongs to $A' \times B' \times C'$, then the loop stops and the
 317 algorithm succeeds. Let r denote the probability (over the random choice of J) that a
 318 given triplet in the input is discarded by the “clamping” step; we have $r = \binom{\delta}{s} / \binom{n}{s}$. The
 319 expected number of iterations therefore follows a geometric distribution of parameter r , and
 320 its expectation is therefore $1/r$. Thanks to the bounds on binomial coefficients (1), we obtain:

$$321 \quad \mathbb{E} [\# \text{ iterations}] = \frac{1}{r} \leq \frac{2^{nH(s/n)}}{\sqrt{2\pi s(1-s/n)}} \Bigg/ \frac{2^{\delta H(s/\delta)}}{\sqrt{8s(1-s/\delta)}} \leq cst \times 2^{nI}$$

323 The probability that n/r iterations occur without success is $(1-r)^{n/r}$ and by concavity
 324 of $\ln 1-x$ we find that this is less than e^{-n} . Therefore, the algorithm fails only with
 325 exponentially small probability after having done n times the expected number of iterations.

326 The total time spent in CLAMP is dominated by $|C_k|/r$. After clamping, the expected
 327 size of the sub-instances is $|A'| = |A_i| \binom{n-i}{s} / \binom{n}{s}$. The same goes for B_j and C_k . Therefore,
 328 the expected cost of solving a single sub-instance using the quadratic algorithm is:

$$329 \quad \frac{\binom{n-i}{s}\binom{n-j}{s}}{\binom{n}{s}\binom{\ell}{s}}|A_i||B_j| + \frac{\binom{n-k}{s}}{\binom{n}{s}}|C_k|.$$

330 The lemma follows, because the component in $|C_k|$ of the cost of solving subproblems is
 331 dominated by the cost of filtering. ◀

332 It is not straightforward to state anything meaningful about the complexity of the
 333 algorithm in general. However, when the input contains a 3XOR triplet, then the ITERATIVE
 334 algorithm always returns it faster than the quadratic algorithm. Indeed, assume that the
 335 input contains a 3XOR triplet ; had we chosen $s = 0$ then no clamping would take place
 336 and the original input lists would have been fed to the quadratic algorithm. The whole
 337 procedure would then stop and succeed during the first iteration, with a running time equal
 338 to that of the quadratic algorithm (up to negligible terms). The choice of s guarantees a
 339 *better* expected running time. This value can be found by exhaustive search over n elements.
 340 It follows that the algorithm is at most quadratic.

341 **Equidistributed Inputs.** To progress in the analysis, we therefore restrict ourselves to a
 342 simpler setting: following again our methodology discussed in section 2.3, we start from
 343 three input lists A, B and C of size $\text{poly}(n)q_0^{-n/3}$ — therefore containing one 3XOR triplet
 344 with high probability. We choose a constant w and we build the sublists A_{wn}, B_{wn} and C_{wn}
 345 by keeping only the vectors of weight wn from A, B and C . In this setting, the expected
 346 size of the input lists is $N_w = \mathbb{E}|A_{wn}| = \mathbb{E}|B_{wn}| = \mathbb{E}|C_{wn}| = N \binom{n}{wn} p^{wn} (1-p)^{n(1-w)}$. The
 347 running time of the ITERATIVE algorithm on input (A_{wn}, B_{wn}, C_{wn}) is then a function of n
 348 and p only.

349 If the input contains a 3XOR triplet, then it has $\delta = n(1 - \frac{3}{2}w)$ columns of “type 000”.
 350 The value of the s parameter can be provided manually and/or decided automatically. Define:

$$\begin{aligned}
 351 \quad L &= -\frac{1}{3} \log q_0 + H(w) + w \log p + (1-w) \log(1-p) \\
 352 \quad I &= H\left(\frac{s}{n}\right) - \frac{\delta}{n} H\left(\frac{s}{\delta}\right) \\
 353 \quad R &= H\left(\frac{s}{n}\right) - (1-w) H\left(\frac{s}{n(1-w)}\right) \\
 354
 \end{aligned}$$

355 With these notations, the size of the input lists is $N_w = \text{poly}(n)2^{nL}$, the number of iterations
 356 of the loop is $n2^{nI}$ and the expected size of the clamped sub-lists is $2^{n(L-R)}$. It follows that
 357 the total time spent clamping the lists is $2^{n(I+L)}$ and the total time spend in the quadratic
 358 algorithm is $2^{n(I+2(L-R))}$. Therefore, in this setting, the ITERATIVE algorithm runs in time
 359 $\tilde{O}(N_w^e)$ where the exponent is $e = \frac{I}{L} + \max(1, 2 - 2\frac{R}{L})$.

360 Given a choice of w (and a value of s), this allows the exponent to be computed as
 361 function of p — as shown in Fig. 1b for $w = p$ and $w = 2u$. The figure strongly suggests that
 362 the exponent e reaches a limit of $\frac{7}{4}$ when p goes to zero. This would make the ITERATIVE
 363 algorithm asymptotically better than the quadratic algorithm. It is indeed provably the
 364 case, at least for some values of w . We consider two interesting cases : $w = p$ and $w = 2u$.
 365 With $w = p$, the “filtering” targets the average density of the input vectors, yielding only
 366 a polynomial reduction in size. with $w = 2u$, the filtering targets the expected density of
 367 3XOR triplets.

368 ▶ **Theorem 6.** *For $w \in \{2u, p\}$, the exponent e reaches a limit of $\frac{7}{4}$ when p goes to zero.*

369 The proof amounts to choosing $s/\delta = 1 - e^{-1/2}$ when $w = p$ and $s/\delta = 1 - \sqrt[4]{2u}$ when
 370 $w = 2u$, then compute the limits. More details can be found in Appendix A.3.

371 **Application to the Direct Algorithm.** This algorithm for the base case can be used to
 372 improve the DIRECT algorithm of section 3. After discarding the vectors of weight greater
 373 than w from the input, the DIRECT algorithm feeds the filtered lists to the quadratic
 374 algorithm. The proposed modification consists in partitioning the filtered lists by hamming
 375 weight, and solving the $\mathcal{O}(n^3)$ sub-instances using the ITERATIVE algorithm.

376 The complexity of this procedure is dominated by the running time of the ITERATIVE
 377 algorithm on the heaviest balanced sub-instance (A_w, B_w, C_w) , with $w/n = p^3 + 2u(1 - p^2)$.
 378 Given a value of p , we can compute w ; from there, we can compute the exponent e of the
 379 iterative algorithm (by choosing the optimal value of s/δ numerically); we then know that
 380 the sub-instance can be dealt with in time $(Ns)^e$, where s is the quantity defined in section 3
 381 ; this in turn allows us to compute the exponent of the combination of the DIRECT algorithm
 382 with the ITERATIVE algorithm, numerically, for a given value of p . The result is again shown
 383 in Fig. 1a.

384 Proving an upper-bound on the exponent of the combination is left as an interesting open
 385 problem.

386 6 Possible Improvement and Discussion

387 We discuss a possible improvement of the previous algorithm. After the clamping step, the
 388 sub-instances may not be fully dense. Therefore it would make sense to use the ITERATIVE
 389 algorithm recursively instead of using the quadratic algorithm. More precisely, in order to
 390 make this idea work, we propose to rely on a technique due to May and Ozerov to improve
 391 Information Set Decoding [10].

Algorithm 4 Sparse Random 3XOR with Improved Clamping

```

1: function PERMUTED( $A_i, B_j, C_k$ )
2:   // Returns a 3XOR triplet in  $A_i \times B_j \times C_k$  w.h.p. or  $\perp$  if none exists
3:   for  $\kappa = 1, \dots, \text{Poly}(n)$  do  $\triangleright \deg \text{Poly}(n) > (t-1)/2$ 
4:      $\delta \leftarrow n - (i + j + k)/2$ 
5:      $q \leftarrow$  random permutation of  $\{0, \dots, n-1\}$ 
6:      $A_i \leftarrow q(A_i), B_j \leftarrow q(B_j), C_k \leftarrow q(C_k)$   $\triangleright$  Permute the columns of the lists
7:     Set  $s$  to the value that minimize the runtime
8:      $S \leftarrow \text{RECURSIVE}(A_i, B_j, C_k, s, 1)$ 
9:     return  $S$ 
10:  return  $\perp$ 

11: function RECURSIVE( $A_i, B_j, C_k, s, \ell$ )
12:  if  $\ell = t + 1$  then
13:    return QUADRATICALGORITHM( $A_i, B_j, C_k$ )
14:  else
15:     $I \leftarrow H(s/n) - (\delta/n)H(s/\delta)$ 
16:    for  $0 \leq \mu < n2^{nh_\ell I}$  do
17:       $J \leftarrow$  random subset of  $h_\ell s$  columns in  $[(h_1 + \dots + h_{\ell-1})n : (h_1 + \dots + h_\ell)n]$ 
18:       $A' \leftarrow \text{CLAMP}(A_i, J), B' \leftarrow \text{CLAMP}(B_j, J), C' \leftarrow \text{CLAMP}(C_k, J)$ 
19:       $S \leftarrow \text{RECURSIVE}(A', B', C', \ell + 1)$ 
20:      if  $S \neq \perp$  then
21:        return  $S$ 
22:  return  $\perp$ 

```

392 Once again, we are in the setting where the lists contains entries of fixed Hamming
 393 weight, respectively i, j and k . Let us fix a constant t and positive reals h_1, \dots, h_t such that
 394 $h_1 + \dots + h_t = 1$. We know that if there is a 3XOR triplet t^* in the input, then there are at
 395 $\delta = n - (i + j + k)/2$ columns of “type 000” in t^* .

396 Here is the main idea of the method. We choose a total number s of columns to clamp,
 397 as in the ITERATIVE algorithm. However, we do the clamping in t stages:

- 398 1. Choose a random subset J_1 of $h_1 s$ columns in the interval $[0 : h_1 n[$, and assume that the
 399 type of all columns in J_1 is 000. Compute the sub-lists $A_i^{(1)}, B_j^{(1)}, C_k^{(1)}$ by clamping the
 400 original input lists over J_1 .
- 401 2. Choose a random subset J_2 of $h_2 s$ columns in the interval $[h_1 n : (h_1 + h_2)n[$. Compute
 402 sublists $A_i^{(2)}, B_j^{(2)}, C_k^{(2)}$ by clamping the output of the previous stage over J_2
- 403 3. Continue for t steps until we came up with small sublists $A_i^{(t)}, B_j^{(t)}, C_k^{(t)}$ and clamping
 404 has been done on s columns.
- 405 4. Solve the resulting instance $(A_i^{(t)}, B_j^{(t)}, C_k^{(t)})$ using the quadratic algorithm.

406 In the ITERATIVE algorithm, if the choice of J is wrong, then all the clamping that has
 407 been done must be discarded. Here, only a part of it will be done in vain, but the choices
 408 that have been made *before* the bad one can still stand. We assume that the columns of
 409 type 000 are uniformly distributed in $[0, n - 1]$. If not, we can randomize the instance by
 410 randomly permuting the columns.

411 ► **Lemma 7.** *The PERMUTED Algorithm finds a solution present in the input with over-*
 412 *whelming probability.*

413 The proof is given in Appendix A.4. Due to the number of parameters that have to be
 414 taken into account, the complexity of this algorithm is not easy to analyze, and we leave it
 415 as an open problem. We claim however, that, in order to reach the best time complexity, we
 416 need to fix the values of the parameters h_1, \dots, h_t such that the time spent clamping the
 417 lists is mostly the same in each level.

418 — References —

- 419 1 R. Arratia and L. Gordon. Tutorial on large deviations for the binomial distribu-
 420 tion. *Bulletin of Mathematical Biology*, 51(1):125 – 131, 1989. URL: <http://www.sciencedirect.com/science/article/pii/S0092824089800527>, doi:[https://doi.org/10.1016/S0092-8240\(89\)80052-7](https://doi.org/10.1016/S0092-8240(89)80052-7).
- 423 2 Leif Both and Alexander May. The approximate k-list problem. *IACR Transactions on*
 424 *Symmetric Cryptology*, 2017(1):380–397, 2017.
- 425 3 Charles Bouillaguet, Claire Delaplace, and Pierre-Alain Fouque. Revisiting and improving
 426 algorithms for the 3xor problem. *IACR Transactions on Symmetric Cryptology*, 2018(1):254–
 427 276, 2018.
- 428 4 Martin Dietzfelbinger, Philipp Schlag, and Stefan Walzer. A subquadratic algorithm for 3xor.
 429 *arXiv preprint arXiv:1804.11086*, 2018.
- 430 5 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $o(1)$
 431 worst case access time. *J. ACM*, 31(3):538–544, June 1984. URL: <http://doi.acm.org/10.1145/828.1884>, doi:10.1145/828.1884.
- 433 6 Anka Gajentaan and Mark Overmars. On a class of $\mathcal{O}(n^2)$ problems in computational geometry.
 434 *Computational geometry*, 5(3):165–185, 1995.
- 435 7 Zahra Jafargholi and Emanuele Viola. 3sum, 3xor, triangles. *CoRR*, abs/1305.3827, 2013.
 436 URL: <http://arxiv.org/abs/1305.3827>, arXiv:1305.3827.

- 437 **8** Pil Joong Lee and Ernest Brickell. An observation on the security of McEliece’s public-key
 438 cryptosystem. In *Workshop on the Theory and Application of Cryptographic Techniques*,
 439 pages 275–280. Springer, 1988.
- 440 **9** Gaëtan Leurent and Ferdinand Sibleyras. Low-memory attacks against two-round even-
 441 mansour using the 3-xor problem. In Alexandra Boldyreva and Daniele Micciancio, editors,
 442 *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference*,
 443 *Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*, volume 11693 of *Lecture*
 444 *Notes in Computer Science*, pages 210–235. Springer, 2019. URL: [https://doi.org/10.1007/](https://doi.org/10.1007/978-3-030-26951-7_8)
 445 [978-3-030-26951-7_8](https://doi.org/10.1007/978-3-030-26951-7_8), doi:10.1007/978-3-030-26951-7_8.
- 446 **10** Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding
 447 of binary linear codes. In *EUROCRYPT*, pages 203–228, 2015.
- 448 **11** Mridul Nandi. Revisiting Security Claims of XLS and COPA. *IACR Cryptology ePrint Archive*,
 449 2015:444, 2015.
- 450 **12** S.M. Ross. *Probability Models for Computer Science*. Elsevier Science, 2002. URL: <https://books.google.fr/books?id=fG3iEZ8f3CcC>.
- 451 **13** Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logist-*
 452 *ics Quarterly*, 3(1-2):59–66, 1956. URL: [https://onlinelibrary.wiley.com/doi/abs/10.](https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800030106)
 453 [1002/nav.3800030106](https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800030106), arXiv:[https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.](https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.3800030106)
 454 [3800030106](https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.3800030106), doi:10.1002/nav.3800030106.
- 455 **14** Emanuele Viola. Reducing 3xor to listing triangles, an exposition. Technical report, North-
 456 eastern University, College of Computer and Information Science, May 2012. Available at
 457 <http://www.ccs.neu.edu/home/viola/papers/xxx.pdf>.
 458

459 **A** Missing Proofs

460 **A.1** Proof of Theorem: Existence of a Solution

461 Recall that Y is the random variable that counts the number of 3XOR triplets in $A \times B \times C$
 462 (taken over the random choice of A, B and C). To establish the theorem, we want to estimate
 463 $\Pr[Y > 0]$. The expected value of Y is bounded away from zero, so a concentration inequality
 464 would bring the result. We cannot use a Chernoff-type bound because the N^3 triplets in
 465 $A \times B \times C$ are identically distributed, but not independent of each other.

466 Let $X(i, j, k)$ denote the binary random variable that takes the value 1 if and only if
 467 $A[i] \oplus B[j] \oplus C[k] = 0$, and let $Y = \sum X(i, j, k)$. Unless mentioned otherwise, in this section
 468 all sums are taken over $0 \leq i, j, k < N$; we omit the indices to alleviate notations.

469 **Proof of theorem 3.** The expected value of Y is easy to determine. Because the elements
 470 of the lists are identically distributed, $\Pr(A[i] \oplus B[j] \oplus C[k] = 0)$ is independent of i, j and k ,
 471 and we get:

$$472 \mathbb{E}(Y) = \mathbb{E}\left(\sum X(i, j, k)\right) = \sum \mathbb{E}(X(i, j, k)) = \sum \Pr(A[i] \oplus B[j] \oplus C[k] = 0) = \mathbb{E}(Y) = N^3 q_0^n.$$

473 Because Y is the sum of binary random variables, we are entitled to use Ross’s conditional
 474 expectation inequality [12]:

$$475 \Pr(Y > 0) \geq \frac{\mathbb{E}(X(i, j, k))}{\mathbb{E}(Y \mid X(i, j, k) = 1)}.$$

476 As argued above, the value of the term under the sum is independent of i, j and k , so this
 477 boils down to: $\Pr(Y > 0) \geq \mathbb{E}(Y)/\mathbb{E}(Y \mid X(0, 0, 0) = 1)$. It remains to compute the number

478 of solutions knowing that there is at least one.

$$\begin{aligned}
 479 \quad E(Y \mid X(0,0,0) = 1) &= \sum \Pr(A[i] \oplus B[j] \oplus C[k] = 0 \mid A[0] \oplus B[0] \oplus C[0] = 0) \\
 480 &= q_0^{-n} \sum \Pr(A[i] \oplus B[j] \oplus C[k] = 0 \wedge A[0] \oplus B[0] \oplus C[0] = 0) \\
 481
 \end{aligned}$$

482 To study this joint distribution, we simply distinguish 8 cases by considering separately the
 483 situation where $i = 0, j = 0$ and $k = 0$ (resp $\neq 0$ for each index). We introduce the shorthand
 484 $p_{ijk} = \Pr(A[i] \oplus B[j] \oplus C[k] = 0 \wedge A[0] \oplus B[0] \oplus C[0] = 0)$ and assume that $i, j, k > 0$.
 485 Then the two joined events are in fact independent and $p_{ijk} = q_0^{2n}$. But when at least one
 486 indice is zero, this is no longer the case ; the extreme situation is $p_{000} = q_0^n$. By symmetry
 487 between the three input lists, we find that $p_{ij0} = p_{i0k} = p_{0jk}$ and $p_{i00} = p_{0j0} = p_{00k}$.

488 Let us compute p_{0jk} . What happens here depends mostly on the hamming weight of
 489 $A[0]$. Indeed $B[j]$ and $C[k]$ have to sum to one where $A[0]$ is one, and to sum to zero where
 490 $A[0]$ is zero. Two bits drawn according to Ber_p sum to one with probability $2p(1-p)$, and
 491 they sum to zero with probability $p^2 + (1-p)^2$. Therefore we get (using the definition of the
 492 binomial distribution and the binomial theorem):

$$\begin{aligned}
 493 \quad p_{0jk} &= \sum_{w=0}^n \Pr(\text{wt}(A[0]) = w) \Pr(B[j] \oplus C[k] = A[0] \wedge B[0] \oplus C[0] = A[0] \mid \text{wt}(A[0]) = w) \\
 494 &= \sum_{w=0}^n \binom{n}{w} p^w (1-p)^{n-w} \left([2p(1-p)]^w [p^2 + (1-p)^2]^{n-w} \right)^2 \\
 495 &= \sum_{w=0}^n \binom{n}{w} [4p^3(1-p)^2]^w \left[(1-p)(1-2p+2p^2) \right]^{n-w} \\
 496 &= \left(4p^3(1-p)^2 + (1-p)(1-2p+2p^2) \right)^n \\
 497 &= q_1^n
 \end{aligned}$$

499 Next, we determine p_{00k} . Here, $C[k]$ has to be equal to $A[0] \oplus B[0]$. This then mostly
 500 depends on the hamming weight of $A[0] \oplus B[0]$, and the situation is somewhat symmetrical:

$$\begin{aligned}
 501 \quad p_{00k} &= \sum_{w=0}^n \Pr(\text{wt}(A[0] \oplus B[0]) = w) \Pr(C[k] = A[0] \oplus B[0] \wedge C[0] = A[0] \oplus B[0] \mid \text{wt}(A[0] \oplus B[0]) = w) \\
 502 &= \sum_{w=0}^n \binom{n}{w} [2p(1-p)]^w [p^2 + (1-p)^2]^{n-w} (p^w(1-p)^{n-w})^2 \\
 503 &= \sum_{w=0}^n \binom{n}{w} [2p^3(1-p)]^w [(1-p)^2(1-2p+2p^2)]^{n-w} \\
 504 &= (2p^3(1-p) + (1-p)^2(1-2p+2p^2))^n \\
 505 &= q_2^n
 \end{aligned}$$

507 We can now write:

$$\begin{aligned}
 508 \quad E(Y \mid X(0,0,0) = 1) &= q_0^{-n} [(N-1)^3 q_0^{2n} + 3(N-1)^2 q_1^n + 3(N-1) q_2^n + q_0^n] \\
 509 &= N^3 q_0^n + 3N^2 (q_1/q_0)^n + 3N (q_2/q_0)^n + 1 - \Delta \quad \text{with:} \\
 510 \quad \Delta &= (3N^2 - 3N + 1) q_0^n + 3(2N - 1) (q_1/q_0)^n + 3 (q_2/q_0)^n
 \end{aligned}$$

512 Where the “error term” Δ is always positive for $N \geq 1$. Going back to the beginning, we

513 finally have (using the convexity of $1/(1+x)$ in the last step):

$$\begin{aligned}
514 \quad \Pr(Y > 0) &\geq \mathbb{E}(Y)/\mathbb{E}(Y \mid X(0,0,0) = 1) \\
515 &\geq \frac{N^3 q_0^n}{N^3 q_0^n + 3N^2 (q_1/q_0)^n + 3N (q_2/q_0)^n + 1 - \Delta} \\
516 &\geq \frac{1}{1 + 3N^{-1} (q_1/q_0^2)^n + 3N^{-2} (q_2/q_0^2)^n + 1/\mathbb{E}(Y)} \\
517 \quad \Pr(Y = 0) &\leq 3N^{-1} (q_1/q_0^2)^n + 3N^{-2} (q_2/q_0^2)^n + 1/\mathbb{E}(Y) \\
518
\end{aligned}$$

519 This establishes the second point of the theorem.

520 It is worthwhile noting that the second moment inequality $\Pr(Y > 0) \geq (\mathbb{E}(X))^2/\mathbb{E}(X^2)$
521 gives exactly the same bound; computing the variance $\sigma^2 = \mathbb{E}(X^2) - (\mathbb{E}(X))^2$ and using
522 Chebyshev's inequality also yields the result of the theorem.

523 Next, using $\mathbb{E}(Y) = N^3 q_0^n$, we may rewrite our last inequality in terms of the expectation:

$$524 \quad \Pr(Y = 0) \leq 3 \frac{(q_1^3/q_0^5)^{n/3}}{(\mathbb{E}(Y))^{-1/3}} + 3 \frac{(q_2^3/q_0^4)^{n/3}}{(\mathbb{E}(Y))^{-2/3}} + \frac{1}{\mathbb{E}(Y)}.$$

526 We claim that q_1^3/q_0^5 and q_2^3/q_0^4 are both smaller than one over $(0, 1/2)$. This follows from
527 the facts that: *i*) they are both equal to 1 when $p = 0$ and *ii*) they are decreasing functions of p
528 over this interval. The latter claim can be established by actually computing their derivatives
529 and checking that they are negative (all factors are easily seen to take only positive values):

$$\begin{aligned}
530 \quad \frac{\partial}{\partial p} \left(\frac{q_1^3}{q_0^5} \right) &= - \frac{6p(4p^3 - 8p^2 + 4p - 1)^2(2p^2 - 6p + 3)(2p - 1)^2}{(4p^2 - 2p + 1)^6(1 - p)^3}, \\
531 \quad \frac{\partial}{\partial p} \left(\frac{q_2^3}{q_0^4} \right) &= - \frac{6p(4p^2 - 3p + 1)^2(4p^2 - 6p + 3)(1 - 2p)}{(4p^2 - 2p + 1)^5(p - 1)^2}.
\end{aligned}$$

533 This establishes the third point of the theorem. ◀

534 A.2 Proof of the Complexity of the Incremental Algorithm

535 **Proof of Theorem 4.** Let T_i denote the running time of the i -th iteration and S the number
536 of iterations done when the algorithm stops. The total running time is then given by
537 $T = \sum_{i=0}^n [S \geq i] T_i$. The two random variables $[S \geq i]$ and T_i are not independent, however
538 we claim that $\mathbb{E}([S \geq i] T_i) \leq (\mathbb{E}[S \geq i]) (\mathbb{E} T_i)$ — *i.e.* they are negatively correlated. The
539 point is that is that the fact that input lists contain a “large weight” triplet t^* can only
540 reduce the expected size of filtered lists by at most one, and therefore reduce the expected
541 running time of processing them. It follows that $\mathbb{E} T \leq \sum_{i=0}^n \Pr(S \geq i) (\mathbb{E} T_i)$.

542 Next, let us consider two random variables following binomial distributions $X_{2u} \sim \mathcal{B}(n, 2u)$
543 and $X_p \sim \mathcal{B}(n, p)$. From the previous section, we know that $\mathbb{E} T_i = N^2 s^2 + N s$, where
544 $s = \Pr(X_p \leq i)$. In addition, following the same reasoning as in section 3, we have
545 $\Pr(S \geq i) \leq 3 \Pr(X_{2u} \geq i)$. This gives:

$$546 \quad \mathbb{E} T \leq 3N^2 \sum_{i=0}^n \Pr(X_p \leq i)^2 \Pr(X_{2u} \geq i)$$

548 Set $u_i = \Pr(X_p \leq i)^2 \Pr(X_{2u} \geq i)$. Our goal is to upper-bound the sum of the u_i 's. To
549 this end, we split the sum in three parts:

$$550 \quad \sum_{i=0}^n u_i \leq \sum_{i=0}^{2nu} u_i + \sum_{i=2nu}^{np} u_i + \sum_{i=np}^n u_i$$

551 First, we note that $\Pr(X_p \leq i)$ is increasing when $i \leq 2nu$, because $2u < p$. Therefore
 552 we have $u_i \leq \Pr(X_p \leq 2nu)^2$, and because $\Pr(X_{2u} \geq i)$ is greater than $\frac{1}{2}$ when $i \leq 2nu$, we
 553 find that $u_i \leq 2u_{2nu}$ when $0 \leq i \leq 2nu$. A symmetric argument shows that $2u_{np} \geq u_i$ for all
 554 $np \leq i \leq n$. Let M denote the largest u_i for $2nu \leq i \leq np$; then the sum of all the u_i 's is
 555 less than $2nM$. We use (3) to get an upper-bound on M . Set $f(x) = 2D(x, p) + D(x, 2u)$, so
 556 that $u_i \leq e^{-nf(\frac{i}{n})}$. We next seek the maximum of f , and for this we compute its derivative:

$$557 \quad f'(x) = 2 \log \frac{x}{p} - 2 \log \frac{x-1}{p-1} + \log \frac{x}{2u} - \log \frac{x-1}{2u-1}$$

558 Solving $f'(x_0) = 0$ reveals only one possible real solution, that satisfies:

$$559 \quad 1 - \frac{1}{x_0} = \sqrt[3]{\left(1 - \frac{1}{p}\right)^2 \left(1 - \frac{1}{2u}\right)}$$

560 It appears that $1 - 1/x_0$ is the geometric mean of $1 - 1/p$, $1 - 1/p$ and $1 - 1/(2u)$; therefore
 561 we find that $2u \leq x_0 \leq p$ (in other terms, the largest u_i actually has an index in the range
 562 $[2nu; np]$). It follows that the total expected cost of the algorithm is upper-bounded by:

$$563 \quad \mathbb{E}T \leq N + 6nN^2 e^{-n[2D(x_0, p) + D(x_0, 2u)]}$$

564 Setting again $e = \log_N(\mathbb{E}T - N)$, the INCREMENTAL algorithm runs in time $\mathcal{O}(N + N^e)$,
 565 with:

$$566 \quad e = 2 + 3 \frac{2D\left(\frac{1}{1 + \sqrt[3]{\left(1 - \frac{1}{p}\right)^2 \left(1 - \frac{1}{2u}\right)}}, p\right) + D\left(\frac{1}{1 + \sqrt[3]{\left(1 - \frac{1}{p}\right)^2 \left(1 - \frac{1}{2u}\right)}, \frac{2p^2}{1 - 2p + 4p^2}\right)}{\ln(1 - p)(1 - 2p + 4p^2)}.$$

567 ◀

568 A.3 Proofs for the Analysis of the Iterative Algorithm

569 **Proof of theorem 6.** First of all, let us start by considering the case where $w = np$ — we
 570 “filter” the input list by keeping the most common weight. We set the number of clamped
 571 columns to $s/\delta = \left(1 - e^{-\frac{1}{2}}\right)$. This gives

$$572 \quad L = -\frac{1}{3} \log_2 q_0$$

$$573 \quad I = H\left(\left(1 - e^{-\frac{1}{2}}\right)\left(1 - \frac{3}{2}p\right)\right) - \left(1 - \frac{3}{2}p\right) H\left(1 - e^{-\frac{1}{2}}\right)$$

$$574 \quad R = H\left(\left(1 - e^{-\frac{1}{2}}\right)\left(1 - \frac{3}{2}p\right)\right) - (1 - p) H\left(\left(1 - e^{-\frac{1}{2}}\right)\left(1 - \frac{3}{2}p\right) \frac{1}{1 - p}\right)$$

576 This allows the exponent e to be computed, at least numerically. It is not well-defined at
 577 $p = 0$, but the limit can be computed: we find that $\lim_{p \rightarrow 0} \frac{I}{L} = \frac{3}{4}$ while $\lim_{p \rightarrow 0} \frac{R}{L} = \frac{1}{2}$ (these
 578 limits can be computed automatically by the Maple computer algebra system; unfortunately
 579 the open-source SageMath system fails). This means that the exponent reaches a limit of $\frac{7}{4}$
 580 when p goes to zero.

581 Let us next consider another interesting case, namely $w = 2u$ — we target the expected
 582 density of 3XOR triplets. This time, we set the number of clamped columns so that

583 $s/\delta = 1 - \sqrt[4]{2u}$. This gives:

$$584 \quad L = -\frac{1}{3} \log q_0 + H(2u) + 2u \log p + (1 - 2u) \log(1 - p)$$

$$585 \quad I = H\left(\left(1 - \sqrt[4]{2u}\right)(1 - 3u)\right) - (1 - 3u)H\left(1 - \sqrt[4]{2u}\right)$$

$$586 \quad R = H\left(\left(1 - \sqrt[4]{2u}\right)(1 - 3u)\right) - (1 - 2u)H\left(\left(1 - \sqrt[4]{2u}\right)\frac{1 - 3u}{1 - 2u}\right)$$

587
588 We again find that $\lim_{p \rightarrow 0} \frac{I}{L} = \frac{3}{4}$ while $\lim_{p \rightarrow 0} \frac{R}{L} = \frac{1}{2}$. Therefore, the exponent also reaches a
589 limit of $\frac{7}{4}$ when p goes to zero. \blacktriangleleft

590 A.4 Proof regarding the Permuted Algorithm

591 **Proof of Lemma 7.** We denote by $t^* = (\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ the solution we aim to recover.

592 Let $A_i^{(\ell)}, B_j^{(\ell)}, C_k^{(\ell)}$ denote the lists that are taken as input by the RECURSIVE algorithm
593 alongside with the index ℓ . Let \mathbf{x}_ℓ^* denote the vector $\mathbf{x}_{I_\ell}^*$, where $I_\ell = J_1 \cup \dots \cup J_{\ell-1}$, for $\ell > 1$
594 and $I_1 = \emptyset$. We define \mathbf{y}_ℓ^* and \mathbf{z}_ℓ^* accordingly.

595 Let us denote by π_ℓ the probability that the triplet $(\mathbf{x}_{\ell+1}^*, \mathbf{y}_{\ell+1}^*, \mathbf{z}_{\ell+1}^*)$ is in $A_i^{(\ell+1)} \times$
596 $B_j^{(\ell+1)} \times C_k^{(\ell+1)}$, knowing that $(\mathbf{x}_\ell^*, \mathbf{y}_\ell^*, \mathbf{z}_\ell^*)$ is in $A_i \times B_j \times C_k$. Assuming that there is a
597 fraction h_ℓ of the δ the columns of “type 000” in the interval of size $h_\ell n$ inside which we
598 choose J_ℓ , we have (using the bounds on binomial coefficients (1)):

$$599 \quad \pi_\ell \geq \frac{\binom{h_\ell \delta}{h_\ell s}}{\binom{h_\ell n}{h_\ell s}} \geq cst \cdot 2^{h_\ell(\delta H(s/\delta) - nH(s/n))} = cst \cdot 2^{-nh_\ell I}.$$

601 At each step ℓ , if the solution is not found, we restart the procedure up to $\approx n/\pi_\ell$
602 times. Assuming that $(\mathbf{x}_\ell^*, \mathbf{y}_\ell^*, \mathbf{z}_\ell^*)$ is in $A_i^{(\ell)}, B_j^{(\ell)}, C_k^{(\ell)}$, the probability that we do not find
603 the solution after this many iterations is

$$604 \quad \Pr[\text{fail at step } \ell] \approx (1 - \pi_\ell)^{\frac{n}{\pi_\ell}} \leq e^{-n},$$

605 In particular, this is true for $i = 1$. This means that the algorithm will return the solution
606 with overwhelming probability, as long as the 000 columns are uniformly distributed (i.e.
607 there are $h_1 \delta$ columns of type 000 in each slice of size $h_1 n$).

608 It remains to show that there exist a permutation Q of the columns of the lists, such that
609 the columns of type 000 in the solution are uniformly distributed. We claim that such a
610 permutation can be found in roughly $n^{(t-1)/2}$ iterations of the PERMUTED Algorithm.

611 Let Q be a random permutation of the columns of the lists. Let δ^* be the exact number
612 of columns of type 000 in the solution. We say that Q is “good enough” if after applying Q
613 to A, B and C , the columns of type 000 are uniformly distributed. In other words, in each
614 slice of $h_\ell n$ columns we would like to have about $s_\ell \delta^*$ columns of type 000. The probability
615 that Q satisfies this condition is given by

$$616 \quad \Pr[Q \text{ good enough}] = \frac{\binom{h_1 n}{h_1 \delta^*} \dots \binom{h_t n}{h_t \delta^*}}{\binom{n}{\delta^*}}$$

$$617 \quad \geq cst \cdot \frac{\sqrt{n} 2^{h_1 n H(\delta^*/n)} \dots 2^{h_t n H(\delta^*/n)}}{\sqrt{h_1 n \dots h_t n} 2^{n H(\delta^*/n)}}$$

$$618 \quad \geq cst \cdot \frac{\sqrt{n} 2^{n H(\delta^*/n)}}{\sqrt{(h_1 \dots h_t) n^t} 2^{n H(\delta^*/n)}}$$

$$619 \quad \geq cst \cdot n^{(1-t)/2}.$$

621 It follows that the expected number of iteration of the PERMUTED Algorithm is $\mathcal{O}(n^{(t-1)/2})$.
 622 Making at least $\text{Poly}(n)$ iterations of the PERMUTED Algorithm where the degree of the
 623 polynomial is greater than $(t-1)/2$ should ensure that a good enough permutation is
 624 found. ◀

625 **B** “Practical” Considerations for the Incremental Algorithm

626 The iterative algorithm outlined in Section 4 may terminate earlier than its direct counterpart
 627 of section 3, but it may also do more work (this happens in particular when the chosen filtering
 628 threshold weight is “just right” and all previous iterations have been wasted). However, it
 629 can be modified to do less work in all circumstances.

630 Let A_i denote the sub-list formed by all vectors of A of hamming weight exactly i ; B_i
 631 and C_i are defined accordingly. This partitions each input list in n parts. The original
 632 problem could then be solved by searching the n^3 sub-instances $A_i \times B_j \times C_k$ for all i, j, k .
 633 In fact, some of these sub-instance cannot contain a 3XOR triplet. This is obvious with
 634 $i = 1, j = 1$ and $k = 5$ for instance: A and B are too sparse to cancel the heavier C . A 3XOR
 635 triplet t necessarily belongs to $A_{\mu+\nu} \times B_{\nu+\lambda} \times C_{\mu+\lambda}$, where μ, ν and λ denote the number
 636 of columns of type 110, 101 and 011 of t , respectively. This is subject to the constraint that
 637 $\mu + \nu + \lambda \leq n$, and there are $\binom{n+3}{3}$ such possibilities.

638 All these “admissible” sub-instances are not equally likely to contain a solution, and they
 639 require a variable amount of time to search. Finding the order in which to process them
 640 to minimize the expected running time is a classical scheduling problem, namely that of
 641 minimizing the weighted sum of completion times on a single machine ($1 \parallel \sum w_i C_i$ in the usual
 642 nomenclature). It can be solved optimally in polynomial time using Smith’s ratio rule [13]:
 643 process the sub-instances by decreasing order of cost-efficiency (probability of success divided
 644 by time required). It turns out that the cost-efficiency of searching $A_{\mu+\nu} \times B_{\nu+\lambda} \times C_{\mu+\lambda}$
 645 is very well correlated to $\mu + \nu + \lambda$, which counts the number of non-000 columns of the
 646 solution. This yields the following “practical” algorithm.

Algorithm 5 A refined, more “practical” iterative algorithm.

```

1: function PRACTICAL( $A, B, C, w$ )
2:   Partition  $A$  (resp.  $B$  and  $C$ ) by hamming weight into  $A_0, \dots, A_n$  (resp.  $B_i, C_i$ )
3:   for  $m = 0, \dots, n$  do
4:     for each  $(\mu, \nu, \lambda)$  such that  $\mu + \nu + \lambda = m$  do
5:       if  $\mu + \nu < w, \nu + \lambda < w$  and  $\mu + \lambda < w$  then
6:         Search  $A_{\mu+\nu} \times B_{\nu+\lambda} \times C_{\mu+\lambda}$  using the quadratic algorithm.
7:         If a solution has been found, report it and stop.
8:   return  $\perp$ 

```

647 With the “if” statement of line 4, the PRACTICAL algorithm succeeds in reporting a
 648 solution $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ if all the three components have hamming weight less than w ; therefore
 649 it succeeds at the same conditions than the DIRECT algorithm of section 3 with the same
 650 threshold w , but faster. Without the “if” statement (or with $w = +\infty$), it always succeeds,
 651 but faster than the “iterative” algorithm outlined in section 4. In all cases, the speedup is at
 652 most polynomial.

653 **C** Computation of a sparse 3XOR

```

#!/usr/bin/env python3
from hashlib import sha512

```

```

# Des. Codes Cryptogr. 33(2)
a = "Mark Goresky and Andrew Klapper: Periodicity and Correlation Properties of d-FCSR Sequences. (2004)"

# TCC (2) 2015
b = "Ran Canetti, Yael Tauman Kalai and Omer Paneth: On Obfuscation with Random Oracles. (2015)"

# Cryptologia 10(1)
c = "David Kahn: Secrets of the Codebreakers. (1986)"

# CT-RSA 2007
d = "Mario Lamberger, Norbert Pramstaller, Christian Rechberger and Vincent Rijmen: Second Preimages for SMASH. (2007)"

# EUROCRYPT 2013
e = "Patrick Derbez, Pierre-Alain Fouque and J r my Jean: Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. (2013)"

# Financial Cryptography 2003
f = "Javier Herranz and Germ n S ez: Verifiable Secret Sharing for General Access Structures, with Application to Fully Distributed Proxy Signatures. (2003)"

# CCSI 2019
g = "Yongge Wang and Qutaibah M. Malluhi: Reusable Garbled Turing Machines Without FHE. (2019)"

# CRYPTO 2000
h = "Masayuki Abe and Tatsuaki Okamoto: Provably Secure Partially Blind Signatures. (2000)"

# Cryptologia 38(2)
i = "Chris Christensen: The National Cash Register Company Additive Recovery Machine. (2014)"

# ICISC 2003
j = "Jonathan Katz: Binary Tree Encryption: Constructions and Applications. (2003)"

# CRYPTO 1994
k = "Olivier Delos and Jean-Jacques Quisquater: An Identity-Based Signature Scheme with Bounded Life-Span. (1994)"

# J. Mathematical Cryptology 9(2)
l = "Shlomi Dolev, Juan A. Garay, Niv Gilboa, Vladimir Kolesnikov and Yelena Yuditsky: Towards efficient private distributed computation on unbounded input streams. (2015)"

def H(s):
    """Apply SHA-512 to a string and converts the hash to an integer"""
    return int.from_bytes(sha512(s.encode('utf8')).digest(), byteorder='big')

# An unexpected relationship through SHA-512... with ``Secrets of the Codebreakers'' ???
assert (H(a) & H(b) & H(c) & H(d)) ^ (H(e) & H(f) & H(g) & H(h)) ^ (H(i) & H(j) & H(k) & H(l)) == 0

```