Faster Algorithms for the Sparse Random 3XOR Problem

Charles Bouillaguet, Claire Delaplace

Faster 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 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. In particular, we present two new algorithms. The first one has a time complexity which is both $\mathcal{O}\left(N^{1+2.583p}\right)$ and $\mathcal{O}\left(N^{2-(1-2p)^{2.1}}\right)$. The second one has a time complexity which is almost linear in N for small values of p $p \leq 0.15$ and has a time complexity of $\tilde{\mathcal{O}}\left(N^{2-1.97(1-2p)^{2.37}}\right)$ for $p > 0.13$. The analysis of these algorithms reveal a "phase change" for a certain threshold p.

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

Funding *Charles Bouillaguet*: author-specific funding acknowledgements

1 Introduction

Given three lists A, B and C of n-bit vectors, the 3XOR problem consists in 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).

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 [5]. 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 $\mathcal{O}\left(N^{2-\epsilon}\right)$ algorithm for the 3XOR problem with input lists of size N would imply faster-than-expected algorithms for listing triangles in a graph [11, 6]. Another result due to [3] show that an algorithm solving the 3XOR problem in time $\Omega\left(n^{2-o(1)}\right)$ also reduces the time complexity of the offline SetDisjointness and SetIntersection.

The 3XOR problem also has some cryptographic applications, in which the input lists consists of uniformly random vectors (the cryptographic community makes this assumption "by default"). In particular, we can mention Nandi's attack [10] 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 [8].

May and Both have been considering a variant of the 3XOR problem, the *approximate 3-list birthday problem* where giving three lists of 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 [1].

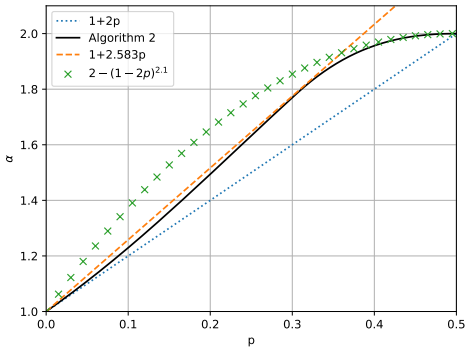
The simplest possible algorithm to solve the 3XOR problem is the quadratic algorithm, which consists in taking all xors $\mathbf{x} \oplus \mathbf{y} \in A \times B$ and checking whether this belongs to the last list C. Using an optimal static dictionnary [4] to hold C, this results in a time complexity of $\mathcal{O}\left(|A||B| + |C|\right)$. In the particular case where $|A| = |B| = |C| = N$, so that only one solution exists (with high probability) this algorithm runs in time $\mathcal{O}\left(N^2\right)$.

We focus on the case where N is such that there is one and only one solution with large probability. In the case where the vectors are drawn uniformly at random in $\{0,1\}^n$, this means that $N = 2^{n/3}$. In this particular case, the quadratic algorithm is mostly the only option to recover the solution. Some improvements of this method exist [2, 3], however these improvements allows only to gain a polynomial factor in n compared to the quadratic algorithm. It is not clear today whether it is possible to find an algorithm for this problem with complexity below $N^{2-o(1)}$.

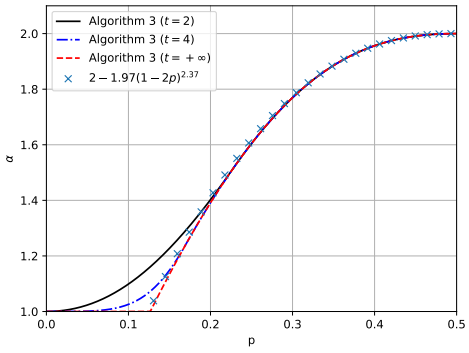
To some extent, the problem we consider in this paper is dual to the "approximate 3-list birthday problem" [1]: starting from (dense) random lists A, B, C, this asks for *approximate* 3XOR triplets $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ — triplets that approximately sum to zero, (*i.e.* such that the hamming weight of $\mathbf{x} \oplus \mathbf{y} \oplus \mathbf{z}$ is small). Here, we start from random input lists with vectors of small hamming weight, and we want to find an exact match — an actual 3XOR triplet.

Contributions. In this paper, we focus on a variant of the 3XOR problem, where the elements of the lists are sparse and random. More precisely, each bit of each vector is drawn at random according to a Bernoulli distribution of parameter $p < 1/2$ (the "dense" random case corresponds to $p = 1/2$). A first consequence is that (exponentially) smaller input lists are sufficient to ensure the existence of a 3XOR triplet with high probability.

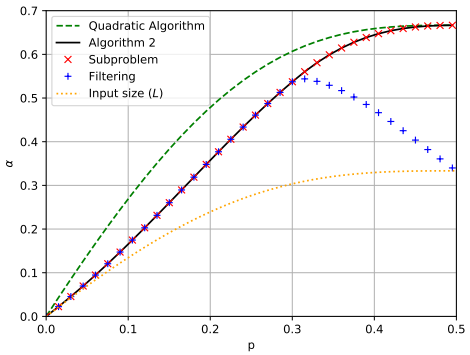
As a second consequence, we show that the 3XOR problem can be solved much faster than $\mathcal{O}\left(N^2\right)$. We take advantage of the fact that the proportion of indices j such that our



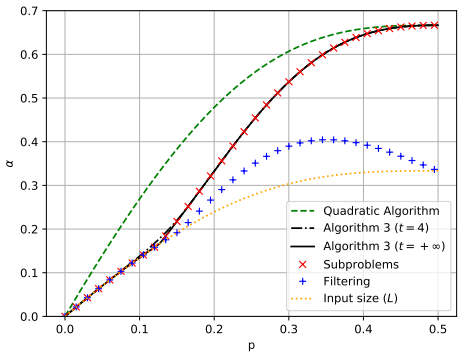
(a) Algorithm 2 (the complexity is N^α).



(b) Algorithm 3 (the complexity is N^α).



(c) Algorithm 2 (the complexity is $2^{\alpha n}$).



(d) Algorithm 3 (the complexity is $2^{\alpha n}$).

Figure 1 Complexities of our Algorithms.

solution $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ satisfies $\mathbf{x}_j = \mathbf{y}_j = \mathbf{z}_j = 0$ is greater than $1/4$ in expectation to come up with two new algorithms described in sections 3 and 3.

In the first one, we select randomly a subset J of the indices, guess that the solution satisfies $\mathbf{x}_j = \mathbf{y}_j = \mathbf{z}_j = 0$ for all $j \in J$. From here, we consider the sublists A', B' and C' of A, B, and C, consisting only of vectors whose coefficients indexed by $j \in J$ are zero. We solve this smaller instance with the quadratic algorithm. If no solution is found, we restart.

In our second algorithm, we borrow the main technique of the "nearest neighbors" algorithm of May and Ozerov [9] (which is an algorithm to decode linear codes). Given a parameter t, we split the indices in t slices. We select randomly a subset J_1 of the indices belonging to the first slice and guess that the solution satisfies $\mathbf{x}_j = \mathbf{y}_j = \mathbf{z}_j = 0$ for all $j \in J_1$. We then build the sublists A_1, B_1, C_1 of the vectors whose coefficients indexed by $j \in J_1$ are zero. After that we select a random subsets J_2 of the indices belonging to the second slice and build the sublists A_2, B_2, C_2 of A_1, B_1, C_1 whose coefficients indexed by $j \in J_2$ are zero and so on an so forth, until we obtain the lists A_t, B_t, C_t, which we process with the quadratic algorithm. The trick is that, if one of our guess J_k was wrong, we do not have to restart the whole guess, but only starting from J_k.

These algorithms are polynomial in N (the size of the lists) and exponential in n (the number of bits of the input vectors). In both case, we focus on the exponent in the complexity, and we disregard all lower-order terms. The complexities of the two algorithms given in this paper are shown in fig. 1. Taking $p = 1/8$, for instance, three lists of $2^{0.165n}$ should contain

at least one solution. Finding it with the quadratic algorithm would take time $2^{0.33n}$. Our first algorithm finds it in time $2^{0.213n}$, while our second algorithm finds it in time $2^{0.17n}$.

Organisation We recall the computational model as well as important properties regarding the distribution of the zeroes and ones in the triplets in Section 2. Then, we present our first new algorithm and study its complexity in Section 3. Finally we present the second algorithm and its complexity in Section 4.

2 Preliminaries

2.1 Notations, Definition and Useful Properties

Let $\mathbf{x}$ be an n-bit string. We have $\mathbf{x} = x_0 x_1 \dots x_{n-1}$. Let A be a list we denote by $|A|$ the size of A, that is the number of elements in A. For any triplet $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \{0,1\}^n$, and any index $0 \leq j < n$, we call *type* of the column j the 3-bit string $x_j y_j z_j$. For any $\mathbf{x} = x_0 \dots x_{n-1}$, we denote by $\mathbf{x}_{\setminus j}$ the bit-string $x_0 \dots x_{j-1} x_{j+1} \dots x_{n-1}$. More generally, if J is a subset of $[0:n]$, we denote by $\mathbf{x}_{\setminus J}$ the sub-string of $\mathbf{x}$, where all x_j for $j \in J$ have been discarded.

Let us now define formally our version of the 3XOR problem.

▶ **Definition 1** (3XOR triplet). *Let $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \{0,1\}^n$, we say that the triplet $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is a* 3XOR triplet *if $\mathbf{x} \oplus \mathbf{y} \oplus \mathbf{z} = 0$.*

▶ **Definition 2** (3XOR problem with distribution $\mathcal{D}$). *Let $\mathcal{D}$ be a distribution over $\{0,1\}^n$. Let A, B and C be three lists of elements drawn independently in $\{0,1\}^n$, according to $\mathcal{D}$. A* solution *to the* 3XOR problem *is a* 3XOR triplet *$(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in A \times B \times C$.*

We say that n is the *dimension* of the problem.

We denote by log the logarithm in basis 2, by Ber_p the Bernoulli distribution of parameter p, and by H the binary entropy function, meaning that $H(x) = -x\log(x) - (1-x)\log(1-x)$, for all $0 < x < 1$. The following standard approximation for the binomial coefficient can be derived from Stirling's formula:

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

We denote by the notation *cst* any constant. We also use the following result.

▶ **Lemma 3** (Chernoff's lower bound). *Let $X_1 \dots X_n$ be independent random variables taking values in $\{0,1\}$, and let $X = \sum_{i=1}^{n}$, then for every $0 < \epsilon < 1$,*

$$\mathbb{P}[X \leq (1-\epsilon)\mathbb{E}[X]] \leq e^{-\frac{\epsilon^2 \mathbb{E}[X]}{2}}.$$

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

The Quadratic Algorithm for 3XOR For the sake of completeness, we recall the Quadratic Algorithm for 3XOR in Algorithm 1.

Algorithm 1 QuadraticAlgorithm

```
1: function QuadraticAlgorithm(A,B,C)
2:     // Returns a 3XOR triplet (x, y, z) ∈ A × B × C or ⊥ if none exist.
3:     Initialize a static dictionnary C with the content of C.
4:     for all x, y ∈ A × B do
5:         if (x ⊕ y) ∈ C then return (x, y, v)
6:     return ⊥
```

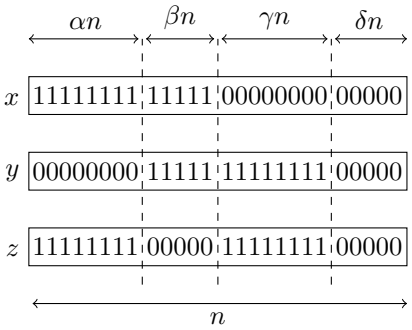


Figure 2 Shape of a sparse random 3XOR triplet, up to column permutation.

2.2 Structural Properties of Sparse 3XOR Triplets

Let $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ be a 3XOR triplet. For each column j, the type $\mathbf{t}_j$ of the j-th column belongs to $\{000, 011, 101, 110\}$. Let $\alpha, \beta, \gamma, \delta \in [0, 1]$ be four parameters satisfying $\alpha + \beta + \gamma + \delta = 1$ and such that αn columns are of type 101, βn columns are of type 110, γn are of type 000. Modulo columns permutations, the shape of a 3XOR triplet can be described by Figure 2. We claim that if $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ are all drawn from the uniform distribution over $\{0, 1\}^n$, then $\mathbb{E}[\alpha] = \mathbb{E}[\beta] = \mathbb{E}[\gamma] = \mathbb{E}[\delta] = \frac{1}{4}$.

Let $\mathcal{D}$ the distribution over $\{0, 1\}^n$, where each bit is drawn independently from Ber_p, with $p < \frac{1}{2}$. Let $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ be a triplet of n-bit vectors drawn from $\mathcal{D}$ (i.e. $\mathbb{P}[x_j = 1] = \mathbb{P}[y_j = 1] = \mathbb{P}[z_j = 1] = p$ for all j). Then, for a given column j of type $\mathbf{t}_j$, we have

$$\begin{aligned}\mathbb{P}[x_j \oplus y_j \oplus z_j = 0] &= \mathbb{P}[\mathbf{t}_j = 000] + \mathbb{P}[\mathbf{t}_j = 011] + \mathbb{P}[\mathbf{t}_j = 101] + \mathbb{P}[\mathbf{t}_j = 110] \\ &= (1-p)(1-2p+4p^2)\end{aligned}$$

It follows that $\mathbb{P}[\mathbf{x} \oplus \mathbf{y} \oplus \mathbf{z} = 0] = (1-p)^n(1-2p+4p^2)^n$. Now let us consider three lists A, B and C, each consisting of N n-bit vectors whose coordinates are drawn independently from Ber_p. The expected number S of 3XOR triplets in $A \times B \times C$ is

$$\mathbb{E}[S] = \frac{N^3}{(1-p)^n(1-2p+4p^2)^n}.$$

As such, as soon as $N \geq (1-p)^{-n/3}(1-2p+4p^2)^{-n/3}$, there will be a 3XOR triplet in $A \times B \times C$ with high probability. Let us therefore define $L = -\frac{1}{3}\log(1-p)(1-2p+4p^2)$ and $N = 2^{nL}$. The algorithms we describe take as input three lists A, B and C, each of N elements drawn independently from $\mathcal{D}$. Their goal is to find a 3XOR triplet $(x, y, z) \in A \times B \times C$.

Let us now consider a sparse random 3XOR triplet $(\mathbf{x}, \mathbf{y}, \mathbf{z})$. For a given column j, we would like to estimate the probability that j is of a certain type. The only possibilities are $000, 011, 101, 110$. Furthermore

$$\mathbb{P}[\mathbf{t}_j = 110 |\ \text{3-xor}] = \mathbb{P}[\mathbf{t}_j = 101 |\ \text{3-xor}] = \mathbb{P}[011 |\ \text{3-xor}] = p^2/(1 - 2p + 4p^2),$$
$$\mathbb{P}[\mathbf{t}_j = 000 |\ \text{3-xor}] = (1-p)^2/(1 - 2p + 4p^2).$$

As $0 < p < \frac{1}{2}$, it follows that the most common column type in a sparse random 3XOR triplet is 000. For the remaining of this paper, we denote by δ the quantity $(1-p)^2/(1-2p+4p^2)$: its the expected proportion of 000 columns in a random sparse 3XOR triplet.

We take advantage of this column repartition to propose several new algorithms to solve the following problem.

3 A Simple Sparse 3-XOR Algorithm

For the sake of simplicity, let us assume that such a 3XOR triplet exists in the input lists and denote it by $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$. Let δ^* denote the proportion of 000 columns in this triplet. Because of the randomness of the input lists, we assume that $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ is a random 3XOR triplet. As such, we expect to find $\mathbb{E}[\delta^*] = \delta$.

The algorithms described in this section exploit the same underlying idea exposed in section 2.2: because the most frequent "column type" in a 3XOR triplet is 000, we try to reduce the size of the instance by guessing that the 3XOR triplet contained in the input is 000 on some columns.

Let δ_0 be a parameter chosen such that with overwhelming probability, at least $\delta_0 n$ columns are of type 000 in $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$. Using the Chernoff bound, we may pick $\delta_0 = \left(1 - n^{-1/3}\right)\delta$.

Algorithm 2 A Simple Sparse Random 3XOR Algortihm.

```
function Filter(L, J)
    // Return the sublist made of vectors which are 0 on the columns in J.
    return {x_{\J} | (x ∈ L) ∧ (∀j ∈ J, x_j = 0)}
function Cashew(A, B, C)
    // Returns a 3XOR triplet w.h.p. if there is one in A × B × C.
    repeat
        J ← uniformly random subset of {1, 2, ..., n} of size κδ_0 n.
        A' ← Filter(A, J), B' ← Filter(B, J), C' ← Filter(C, J).
        S ← QuadraticAlgorithm(A', B', C').
    until S ≠ ⊥
    return S
```

Algorithm 2 is inspired by information set decoding techniques (notably by the Lee-Brickell [7] algorithm). It takes a parameter $0 \leq \kappa \leq 1$, whose value is discussed below.

▶ **Theorem 4.** *The expected time complexity of* Cashew *is*

i) $\Omega\left(N^{1+2p}\right)$ *and* $\mathcal{O}\left(N^{1+2.583p}\right)$

ii) $\mathcal{O}\left(N^{2-(1-2p)^{2.1}}\right)$

The rest of this section is devoted to proving theorem 4. Let us denote:

$$I = H(\kappa\delta_0) - \delta^* H(\kappa\delta_0/\delta^*)$$
$$R = -\kappa\delta_0 \log_2(1-p)$$

▶ **Lemma 5.** *i) The expected total time spent in* Filter *is* $\mathcal{O}\left(2^{n(I+L)}\right)$.
ii) The expected total time spent in QuadraticAlgorithm *is* $\mathcal{O}\left(2^{n(I+2(L-R))}\right)$.
iii) The expected complexity of Cashew *is* $\mathcal{O}\left(2^{n(I+L)} + 2^{n(I+2(L-R))}\right)$.

Proof. We first observe that Filter is linear in the size of its input. The probability that the choice of J is compatible with $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ is:

$$\mathbb{P}\left[x^*_j = y^*_j = z^*_j = 0 \text{ for all } j \in J\right] = \binom{\delta^* n}{\kappa\delta_0 n} \Big/ \binom{n}{\kappa\delta_0 n}.$$

The expected number of iterations of Cashew is the inverse of this probability, and thanks to eq. (1), it is upper-bounded by:

$$\begin{aligned}\mathbb{E}\left[\#\text{ iterations}\right] &\leq \frac{2^{nH(\kappa\delta_0)}}{\sqrt{2\pi n\kappa\delta_0(1-\kappa\delta_0)}} \Big/ \frac{2^{n\delta^* H(\kappa\delta_0/\delta^*)}}{\sqrt{8n\kappa\delta_0/\delta^*(1-\kappa\delta_0/\delta^*)}} \\ &\leq cst \times 2^{n[H(\kappa\delta_0) - \delta^* H(\kappa\delta_0/\delta^*)]}\end{aligned}$$

This establishes point *i*).

We now estimate the size of the subproblems (A', B', C'). A random sparse vector of density p is zero on the $\kappa\delta_0 n$ columns chosen in J with probability $q = (1-p)^{\kappa\delta_0 n}$. Because A, B and C are made of independent and uniformly random sparse vectors of density p, we find that $\mathbb{E}[|A'|] = q|A|$ (an identical result holds for B' and C'). It follows that the expected complexity of solving a single subproblem is $|A'| \times |B'| = q^2 N^2$. This establishes point *ii*). Point *iii*) is a trivial consequence of *i*) and *ii*). ◀

By choosing the value of κ, we can adjust the expecting number of iterations and the size of the subproblems. A quick examination reveals that $I + L$ is an increasing function of κ, while $I + 2(L - R)$ first decreases to a minimum then increases again when κ varies between 0 and 1. It turns out that there are two distinct situations, depending on p: the best option is either to balance the cost of filtering and solving the subproblems (for small p) or to minimize the total time spent dealing with the subproblems (for large p).

▶ **Lemma 6.** *There exist a constant* $p^* = 0.30534...$ *such that the value of* κ *minimizing the expected running time of* Cashew *is:*

$$\kappa = \begin{cases} -L/(2\delta \log 1 - p) & \text{when } p < p^* \\ 4 + 6/(p-2) & \text{when } p > p^* \end{cases}$$

The threshold p^* *is the single value that equates the two alternatives.*

Proof. Let us consider the expected values of I and R over the random choice of a 3XOR triplet (i.e. assuming that $\delta^* = \delta_0 = \delta$):

$$I' = H(\kappa\delta) - \delta H(\kappa)$$
$$R' = -\kappa\delta \log_2(1-p)$$

It follows from lemma 5 that the optimum expected value of κ is the one that minimizes $\max\{I' + L, I' + 2(L - R')\}$ — this minimizes the exponent in the complexity.

Let κ_1 denote the solution of $I' + L' = I' + 2(L - R')$ (this is the value of κ that balances the cost of filtering and solving the subproblems). Let κ_2 denote the solution of $\partial(I' - 2R')/\partial\kappa = 0$ (this is the value of κ that minimizes the total expected time required to solve the subproblems).

When $\kappa_1 < \kappa_2$, then κ_1 minimizes the expected running time of CASHEW. Indeed, taking κ below κ_1 increases the complexity of solving the subproblems (which is decreasing below κ_2); taking $\kappa > \kappa_1$ increases the complexity of the filtering step (which is always increasing).

When $\kappa_1 > \kappa_2$, then κ_2 is the optimal value: moving beyond κ_2 increases the complexity of both steps. Because the cost of the filtering step is increasing as a function of κ, it is smaller than that of solving the subproblems while $\kappa < \kappa_1$. As such, solving the subproblems dominate the complexity, and the cost of this step is minimum when $\kappa = \kappa_2$.

A somewhat tedious calculation shows that $\kappa_1 = -L/(2\delta \log 1-p)$ and $\kappa_2 = 4+6/(p-2)$. Another tedious calculation shows that $\kappa_1 < \kappa_2 \iff p < p^*$, where p^* is the solution ◀

Theorem 4 can be established by taking the optimal value of κ given by lemma 6, and verifying numerically that the resulting complexity indeed satisfies the announced bounds. This is visible on Fig. 1.

4 Improved Filtering *à la* May and Ozerov

In this section we describe an improved algorithm, which is highly inspired by that of May and Ozerov for Information Set Decoding [9].

Let $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ be a 3XOR triplet in $A \times B \times C$ and let δ^* be the proportion of columns of type 000 in this triplet. Let us fix a constant t and positive reals $s_1, \dots, s_t$ such that $s_1 + \cdots + s_t = 1$. We again consider the parameter δ_0 such that, with overwhelming probability, at least $\delta_0 n$ columns are of type 000 in $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$.

As in section 3, we define $I = H(\delta_0\kappa) - \delta_0 H(\kappa)$ and $R = -\log(1-p)\kappa\delta_0$.

We assume that the columns of type 000 are uniformly distributed in $[0, n-1]$. If not, we can randomize the instance by permutating the columns randomly. After a polynomial number of such permutation, the columns of type 000 should be uniformly distributed. Here is the main idea of the method.

1. Choose a random subset J_1 of $s_1\delta_0 n$ columns among the first $s_1 n$, and assume that for all $j \in J_1$, the type $\mathbf{t}_j$ of the column j is $\mathbf{t}_j = 000$.
2. Compute the sub-lists A_1, B_1, C_1 such that for all $\mathbf{x} \in A_1$, (resp. B_1, C_1) $x_j = 0$.
3. Compute sublists A_2, B_2, C_2 recursively in a similar way, and so on and so forth until we came up with small sublists A_t, B_t, C_t.
4. Solve the instance with A_t, B_t, C_t using the quadratic algorithm.

Now let $1 \le i_0 \le t$ be the first index for which our choice of the set J_{i_0} is wrong (i.e. $\exists j \in J_{i_0}$ such that $\mathbf{t}_j \neq 000$ in the solution). We do not have to restart computing the lists $A_{i_0-1}, B_{i_0-1}, C_{i_0-1}$, but we only have to restart the computation starting from $A_{i_0}, B_{i_0}, C_{i_0}$.

More formally, this yields Algorithm 3. The remainder of this section is devoted to establishing its complexity.

▶ **Theorem 7.** *For any constant t, there exists $s_1, \dots, s_t$ and a polynomial P such that* PISTACHIO *finds a solution in expected time*

$$P(n)\left[2^{n\left(L+I\frac{1-(R/I)}{1-(R/I)^t}\right)} + 2^{n(I+2(L-R))}\right].$$

▶ **Lemma 8.** *Let $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ be a 3XOR triplet in $A \times B \times C$. We assume that this is the only 3XOR triplet in $A \times B \times C$. Let δ^* denote the proportion of columns of type* 000 *in $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$. Assume that these columns are uniformly distributed among all the columns of $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$; then* PEANUT$(A, B, C, 1)$ *returns $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ with overwhelming probability.*

Algorithm 3 Sparse Random 3XOR with Improved Filtering

```
function Pistachio(A, B, C)
    // Returns a 3XOR triplet in A × B × C w.h.p. or ⊥ if none exists
    do
        q ← random permutation of {0, ..., n − 1}
        A ← q(A), B ← q(B), C ← q(C)    ▷ Permute randomly the columns of the lists
        S ← Peanut(A, B, C, 1)
    while S = ⊥
    return S

function Peanut(A, B, C, i)
    if i = t + 1 then
        return QuadraticAlgorithm(A, B, C)
    else
        for 0 ≤ ℓ < n2^{n s_i I} do
            J ← random subset of κδ_0 s_i n columns in [(s_1 + ··· + s_{i−1})n : (s_1 + ··· + s_i)n]
            A' ← Filter(A, J), B' ← Filter(B, J), C' ← Filter(C, J)
            S ← Peanut(A', B', C', i + 1)
            if S ≠ ⊥ then
                return S
    return ⊥
```

Proof. Let A_i, B_i, C_i denote the lists that are taken as input by Peanut alongside with the index i. Let $\mathbf{x}_i^*$ denote the vector $\mathbf{x}^*_{\setminus I_i}$, where $I_i = J_1 \cup \dots J_{i-1}$, for $i > 1$ and $I_1 = \emptyset$. We define $\mathbf{y}_i^*$ and $\mathbf{z}_i^*$ accordingly.

Let us denote by π_i the probability that the triplet $(\mathbf{x}^*_{i+1}, \mathbf{y}^*_{i+1}, \mathbf{z}^*_{i+1})$ is in $A_{i+1} \times B_{i+1} \times C_{i+1}$, knowing that $(\mathbf{x}_i^*, \mathbf{y}_i^*, \mathbf{z}_i^*)$ is in $A_i \times B_i \times C_i$. Assuming a uniform distribution of the columns of type 000, we have (using (1)):

$$\pi_i = \binom{s_i\delta^* n}{\kappa s_i \delta_0 n} \Big/ \binom{s_i n}{\kappa s_i \delta_0 n} \geq \binom{s_i \delta_0 n}{\kappa s_i \delta_0} \Big/ \binom{s_i n}{\kappa s_i \delta_0 n}$$

$$\geq cst \cdot \frac{s_i n 2^{\delta_0 s_i n H(\kappa)}}{\delta_0 s_i n 2^{s_i n H(\kappa\delta_0)}} = cst \cdot \frac{2^{s_i n(\delta_0 H(\kappa) - H(\kappa\delta_0))}}{\delta_0} \approx 2^{s_i n(\delta_0 H(\kappa) - H(\kappa\delta_0))}.$$

At each step i, if the solution is not found, we restart the procedure up to $\approx n/\pi_i$ times. Assuming that (x_i^*, y_i^*, z_i^*) is in A_i, B_i, C_i, the probability that we do not find the solution after this many call to the procedure is

$$\mathbb{P}[\text{fail at step } i] \approx (1 - \pi_i)^{\frac{n}{\pi_i}} \leq e^{-n},$$

In particular, this is true for $i = 1$. This means that the algorithm will return the solution with overwhelming probability, as long as the 000 columns are uniformly distributed. ◀

▶ **Lemma 9.** *The expected size of the input lists A_i, B_i, C_i of* Peanut *at step i is:*

$$N_i = 2^{n\left(L - R\sum_{j=1}^{i-1} s_i\right)}.$$

Proof. The probability that a vector of size greater than $s_i n$, whose coefficients are drawn from Ber_p is 0 on $\kappa\delta_0 s_i n$ arbitrary fixed columns is

$$q_i = (1-p)^{\kappa\delta_0 s_i n} = 2^{\kappa\delta_0 s_i n \log(1-p)} = 2^{-R s_i n}.$$

It follows that the expected size N_{i+1} of the list $A_{i+1}, B_{i+1}, C_{i+1}$, for $i \geq 1$ is given by

$$\mathbb{E}[N_{i+1}] = q_i N_i = N_1 \prod_{j=1}^{i} q_j N_1 = 2^{-n\left(L - R\sum_{j=1}^{i} s_i\right)}$$

◀

▶ **Lemma 10.** *The number of iterations of* Pistachio *is* $\mathcal{O}\left(n^{(t-1)/2}\right)$.

Proof. Let Q be a random permutation of the columns of the lists. We say that Q is "good enough" if after applying Q to A, B and C, the $\delta^* n$ columns of type 000 are uniformly distributed. In other words, in each slice of $s_i n$ columns we would like to have about $s_i \delta^* n$ columns of type 000. The probability that Q satisfies this condition is given by

$$\begin{aligned}
\mathbb{P}[Q \text{ good enough}] &= \binom{s_1 n}{s_1 \delta^* n} \dots \binom{s_t n}{s_t \delta^* n} / \binom{n}{\delta^* n} \\
&\geq cst \cdot \frac{\sqrt{n} 2^{s_1 n H(\delta^*)} \dots 2^{s_t n H(\delta^*)}}{\sqrt{s_1 n \dots s_t n} 2^{n H(\delta^*)}} \\
&\geq cst \cdot \frac{\sqrt{n} 2^{n H(\delta^*)}}{\sqrt{(s_1 \dots s_t) n^t} 2^{n H(\delta^*)}} \\
&\geq cst \cdot n^{(1-t)/2}.
\end{aligned}$$

It follows that the expected number of iteration of Pistachio is $\mathcal{O}\left(n^{(t-1)/2}\right)$. ◀

We are now ready to prove Theorem 7.

Proof. Let us denote by C_i the time complexity of one iteration i of Peanut. In particular, we have

$$\begin{aligned}
C_{t+1} &= \mathcal{O}\left(N_{t+1}^2\right) \\
C_i &= \mathcal{O}\left(2^{n s_i I}(N_i + C_{i+1})\right), \quad \forall i \leq t
\end{aligned} \tag{2}$$

where N_i is the size of the lists at the beginning of Step i. Furthermore, Step i of the procedure has to be restarted at most $2^{nI\sum_{j=1}^{i-1} s_i}$ times. Let us denote by T_i the value $2^{nI\sum_{j=1}^{i-1} C_i}$. From (2), we have

$$T_i = \mathcal{O}\left(2^{nI\sum_{j=1}^{i} s_j} N_i\right) + T_{i+1}.$$

From Lemma 9, this is

$$T_i = \mathcal{O}\left(2^{n\left(L + I s_i + (I-R)\sum_{j=1}^{i-1} s_j\right)}\right) + T_{j+1}.$$

For all $1 \leq i \leq t-1$, we want the following to hold

$$n\left(L + I s_i + (I-R)\sum_{j=1}^{i-1} s_j\right) = n\left(L + I s_{i+1} + (I-R)\sum_{j=1}^{i} s_j\right).$$

This would mean that the time spent filtering the lists is mostly the same in all levels i. After simplification, this gives

$$s_{i+1} = \frac{R}{I} s_i = \frac{R^i}{I^i} s_1.$$

The condition $\sum_i s_i = 1$ ensures that $s_1 = \frac{1-(R/I)}{1-(R/I)^t}$. It follows that for all i,

$$T_i = \mathcal{O}\left(2^{n(L+Is_1)}\right) + T_{i+1} = \mathcal{O}\left(2^{n\left(L+I^t\frac{I-R}{I^t-R^t}\right)}\right) + T_{i+1}.$$

In particular, as t is a constant, this implies that

$$C_1 = T_1 = \mathcal{O}\left(2^{n\left(L+I^t\frac{I-R}{I^t-R^t}\right)}\right) + T_{n+1}$$

and

$$T_{n+1} = \mathcal{O}\left(2^{nI\sum_{i=1}^t s_i} 2^{2n\left(L-R\sum_{i=1}^t s_i\right)}\right) = 2^{n(I+2(L-R))}.$$

It follows that

$$C_1 = \mathcal{O}\left(2^{n\left(L+I^t\frac{I-R}{I^t-R^t}\right)} + 2^{n(I+2(L-R))}\right).$$

We use Lemma 10 to obtain the claimed complexity. ◀

Tuning for Maximum Speed

It remains to choose a value of κ that minimizes the running time. This time, κ depends on p and t. We can essentially replay the analysis in lemma 6, but everything becomes messier because of t.

Let κ_1 be the value that balances the cost of filtering and solving the subproblems in Peanut. It can be seen that this is the solution (in κ) of:

$$\left(1-(R/I)^{t-1}\right) / \left(1-(R/I)^t\right) = 2 - L/R. \tag{3}$$

Barring anything else, this equation can be solved numerically. This allows to evaluate the exponent in the complexity. Let κ_2 denote the value that minimizes the total time spent solving subproblems. As previously, we find that $\kappa_2 = 4 + 6/(p-2)$, and that the best value of κ is the minimum of κ_1 and κ_2.

To make things more concrete, we will consider two settings: a "small" value of t (say, $t = 4$), and a "large" value of t ($t = +\infty$).

For $t = 4$, We find that the threshold where $\kappa_1 = \kappa_2$ is attained for $p^\star = 0.1690...$ (overwise we have $\kappa_1 < \kappa_2 \Leftrightarrow p < p^\star$). In other terms, for $p > p^\star$, the best strategy is to minimize the total time spent in dealing with the subproblems. We find numerically that while $p < 0.169$, algorithm 3 runs in time less than $N^{5/4}$.

We now consider the case $t = \infty$; arguably, this is not a realistic value, but we think that it helps understand the global behavior of the algorithm. We pass to the limit by noting that when $t \to +\infty$, the function $f(x) = \frac{1-x}{1-x^t}$ becomes

$$\hat{f}(x) = \begin{cases} 1-x & \text{when } x \leq 1 \\ 0 & \text{when } x \geq 1 \end{cases}$$

Therefore, balancing the cost of filtering with that of solving the subproblems means finding κ_1 satisfying:

$$L + I\hat{f}\left(\frac{R}{I}\right) = I + 2(L-R)$$

This yields a new threshold $p^\star = 0.1265...$: for $p < p^\star$, the best value of κ is κ_1, while for $p > p^*$ it is again κ_2. For $p < p^\star$, with $\kappa = \kappa_1$, we find that $R/I > 1$, and therefore the complexity of the whole algorithm is exactly N (it is *linear* in its input). In fact, we observe that even for smaller ("reasonable") values of t, the complexity is very close to N when p is close enough t zero. We summarise this in the following corollary.

▶ **Corollary 11.** **1.** *For any $t \geq 2$, for all $\epsilon > 0$, there exist a threshold p_0 such that for $p \leq p_0$, Algorithm 3 runs in time $\mathcal{O}\left(n^{(t-1)/2}N^{1+\epsilon}\right)$*

2. *In particular, for $t = 12$ and $\epsilon = 0.01$, $p_0 \geq 1/8$.*

3. *For $t \geq 10$ and $p \geq 0.130$, Algorithm 3 runs in time $\mathcal{O}\left(n^{(t-1)/2}N^{2-1.97(1-2p)^{2.37}}\right)$.*

Tuning the parameters to minimize the concrete running time on an actual computer to solve a given instance with specified values of n and p is another problem.

To conclude, Algorithm 3 spends less time filtering the lists than Algorithm 2. As such, solving the subproblems using the quadratic algorithm dominates the running time for smaller values of p with an optimal choice of κ. It follows that for $p \geq 0.305...$, both algorithms exhibit essentially the same behavior. Algorithm 3 shines for small values of p (as can be seen on Fig 1).

References

1 Leif Both and Alexander May. The approximate k-list problem. *IACR Transactions on Symmetric Cryptology*, 2017(1):380–397, 2017.

2 Charles Bouillaguet, Claire Delaplace, and Pierre-Alain Fouque. Revisiting and improving algorithms for the 3xor problem. *IACR Transactions on Symmetric Cryptology*, 2018(1):254–276, 2018.

3 Martin Dietzfelbinger, Philipp Schlag, and Stefan Walzer. A subquadratic algorithm for 3xor. *arXiv preprint arXiv:1804.11086*, 2018.

4 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with o(1) 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`.

5 Anka Gajentaan and Mark Overmars. On a class of $\mathcal{O}(n^2)$ problems in computational geometry. *Computational geometry*, 5(3):165–185, 1995.

6 Zahra Jafargholi and Emanuele Viola. 3sum, 3xor, triangles. *CoRR*, abs/1305.3827, 2013. URL: `http://arxiv.org/abs/1305.3827`, `arXiv:1305.3827`.

7 Pil Joong Lee and Ernest Brickell. An observation on the security of McEliece's public-key cryptosystem. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 275–280. Springer, 1988.

8 Gaëtan Leurent and Ferdinand Sibleyras. Low-memory attacks against two-round even-mansour using the 3-xor problem. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 210–235. Springer, 2019. URL: `https://doi.org/10.1007/978-3-030-26951-7_8`, `doi:10.1007/978-3-030-26951-7\_8`.

9 Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *EUROCRYPT*, pages 203–228, 2015.

10 Mridul Nandi. Revisiting Security Claims of XLS and COPA. *IACR Cryptology ePrint Archive*, 2015:444, 2015.

11 Emanuele Viola. Reducing 3xor to listing triangles, an exposition. Technical report, Northeastern University, College of Computer and Information Science, May 2012. Available at `http://www.ccs.neu.edu/home/viola/papers/xxx.pdf`.