

Reducing 3XOR to listing triangles, an exposition

Emanuele Viola

November 4, 2011

The 3SUM problem asks if there are three integers a, b, c summing to 0 in a given set of n integers of magnitude $\text{poly}(n)$. This problem can be easily solved in time $\tilde{O}(n^2)$. (Throughout this note, \tilde{O} and $\tilde{\Omega}$ hide subpolynomial factors $n^{o(1)}$.) It seems natural to believe that this problem also requires time $\tilde{\Omega}(n^2)$, and this has been confirmed in some restricted models.[Eri99, AC05] The importance of this belief was brought to the forefront by Gajentaan and Overmars who show that the belief implies lower bounds for a number of problems in computational geometry;[GO95] and the list of such reductions has grown ever since. Recently, a series of exciting papers by Baran, Demaine, Pătraşcu, Vassilevska, and Williams set the stage for, and establish, reductions from 3SUM to new types of problems which are not defined in terms of summation.[BDP08, VW09, PW10, Păt10] In particular, Pătraşcu reduces 3SUM to the problem of listing triangles of a graph.[Păt10]

In this note we present this reduction by Pătraşcu but for a variant of the 3SUM problem which we call 3XOR. The problem 3XOR is like 3SUM except that integer summation is replaced with bit-wise xor. So one can think of 3XOR as asking if a given $n \times O(\lg n)$ matrix over the field with two elements has a linear combination of length 3. This problem is likely less relevant to computational geometry, but is otherwise quite natural. Similarly to 3SUM, 3XOR can be solved in time $\tilde{O}(n^2)$, and it seems natural to conjecture that 3XOR requires time $\tilde{\Omega}(n^2)$. We now state the reduction we present.

Theorem 1. Suppose that given the adjacency list of a graph with m edges and z triangles (and $O(m)$ nodes) one can list $\min\{z, m\}$ triangles in time $m^{1.3\bar{3}-\epsilon}$ for a constant $\epsilon > 0$. Then one can solve 3XOR on a set of size n in time $n^{2-\epsilon'}$ with error 1% for a constant $\epsilon' > 0$.

To put the above theorem in context it is useful to discuss the difference between listing *all* triangles in a graph and listing just m of them. For listing all triangles, the best we can hope for is time $\tilde{O}(m^{1.5})$, since the maximum number of triangles in graphs with m edges is $\Theta(m^{1.5})$. There are algorithms that achieve time $\tilde{O}(m^{1.5})$. (For example, we can first list the $\leq O(m\sqrt{m})$ triangles going through some node of degree $\leq \sqrt{m}$, and then the $\leq O(m/\sqrt{m})^3 = O(m^{1.5})$ triangles using nodes of degree $> \sqrt{m}$ only.)

However, the assumption of the above Theorem 1 asks to list only $\leq m$ triangles, for which conceivably time $\tilde{O}(m)$ suffices. In fact, to list m triangles Pagh (personal communication 2011) points out an algorithm achieving time $\tilde{O}(m^{1.5-\Omega(1)})$ and, assuming that the exponent of matrix multiplication is 2, time $\tilde{O}(m^{1.4})$.

The proof of Theorem 1 follows the one in [Pät10] for 3SUM, which builds on results in [BDP08]. However the proof of Theorem 1 is a bit simpler. This is because it avoids some steps in [BDP08, Pät10] which are mysterious to us. And because in our context we have at our disposal hash functions that are *linear*, while over the integers one has to work with “almost-linearity,” cf. [BDP08, Pät10].

This note is organized as follows. In §0.1 we cover some preliminaries and prove a hashing lemma by [BDP08] that will be used later. The proof of the reduction in Theorem 1 is broken up in two stages. First, in §0.2 we reduce 3XOR to the problem C3XOR which is a “convolution” version of 3XOR. Then in §0.3 we reduce C3XOR to listing triangles.

0.1 Hashing and preliminaries

We define next the standard hash function we will use.

Definition 2. For input length ℓ and output length r , the hash function h uses r ℓ -bit keys $\bar{a} := (a^1, \dots, a^r)$ and is defined as $h_{\bar{a}}(x) := (\langle a^1, x \rangle, \dots, \langle a^r, x \rangle)$, where $\langle \cdot, \cdot \rangle$ denotes inner product modulo 2.

We note that this hash function is linear: $h_{\bar{a}}(x) + h_{\bar{a}}(y) = h_{\bar{a}}(x + y)$ for any $x \neq y \in \{0, 1\}^\ell$, where addition is bit-wise xor. Also, $h_{\bar{a}}(0) = 0$ for any \bar{a} , and $\Pr_{\bar{a}}[h_{\bar{a}}(x) = h_{\bar{a}}(y)] \leq 2^{-r}$ for any $x \neq y$.

Before discussing the reductions, we make some remarks on the problem 3XOR. First, for simplicity we are going to assume that the input vectors are unique. It is easy to deal separately with solutions involving repeated vectors. Next we argue that for our purposes the length ℓ of the vectors in instances of 3XOR can be assumed to be $(2 - o(1)) \lg n \leq \ell \leq 3 \lg n$. Indeed, if $\ell \leq (2 - \Omega(1)) \lg n$ one can use the fast Walsh-Hadamard transform to solve 3XOR efficiently, just like one can use fast Fourier transform for 3SUM, cf. [CLRS01, Exercise 30.1-7]. For 3XOR one gets a running time of $2^\ell \ell^{O(1)} + \tilde{O}(n\ell)$, where the first term comes from the fast algorithms to compute the transform, see e.g. [MR97, §2.1]. (The second term accounts for preprocessing the input.) When $\ell \leq (2 - \Omega(1)) \lg n$, the running time is $n^{2-\Omega(1)}$, i.e., subquadratic.

Also, the length can be reduced to $3 \lg n$ via hashing. Specifically, an instance $v_1, \dots, v_n \in \{0, 1\}^\ell$ of 3XOR is reduced to $h(v_1), \dots, h(v_n)$ where $h = h_{\bar{a}}$ is the hash function with range of $r = 3 \lg n$ bits for a randomly chosen \bar{a} . Correctness follows because on the one hand if $v_i + v_j + v_k = 0$ then $h(v_i) + h(v_j) + h(v_k) = h(v_i + v_j + v_k) = h(0) = 0$ by linearity of h and the fact that $h(0) = 0$ always; on the other hand if $v_i + v_j + v_k \neq 0$ then $\Pr[h(v_i + v_j + v_k) = 0] = 1/2^r$ since h maps uniformly in $\{0, 1\}^r$ any non-zero input. Hence by a union bound over all $\leq \binom{n}{3}$ choices for vectors such that $v_i + v_j + v_k \neq 0$, the probability of a false positive is $\binom{n}{3}/n^3 < 1/6$.

For the proof we need to bound the number of elements x whose buckets $B_h(x) := \{y \in S : h(x) = h(y)\}$ have unusually large load. If our hash function was 3-wise independent the desired bound would follow from Chebyshev’s inequality. But our hash function is only

pairwise independent, and we do not see a better way than using a hashing lemma from [BDP08] that in fact relies on a weaker property, cf. the discussion in [BDP08].

When hashing n elements to $[R] = \{1, 2, \dots, R\}$, the expected load of each bucket is n/R . The lemma guarantees that the expected number of elements hashing to buckets with a load $\geq 2n/R + k$ is $\leq n/k$.

Lemma 3 ([BDP08]). Let h be a random function $h : U \rightarrow [R]$ such that for any $x \neq y$, $\Pr_h[h(x) = h(y)] \leq 1/R$. Let S be a set of n elements, and denote $B_h(x) = \{y \in S : h(x) = h(y)\}$. We have

$$\Pr_{h,x}[|B_h(x)| \geq 2n/R + k] \leq 1/k.$$

In particular, the expected number of elements from S with $|B_h(x)| \geq 2n/R + k$ is $\leq n/k$.

The proof of the lemma uses the following fact, whose proof is an easy application of the Cauchy-Schwarz inequality.

Fact 4. Let $f : D \rightarrow [R]$ be a function. Pick x, y independently and uniformly in D . Then $\Pr_{x,y}[f(x) = f(y)] \geq 1/R$.

Proof:[of Lemma 3] Pick x, y uniformly and independently in S ($x = y$ possible). For given h , let

$$\begin{aligned} p_h &:= \Pr_x[|B(x)| \geq 2n/R + k], \\ q_h &:= \Pr_{x,y}[h(x) = h(y)]. \end{aligned}$$

Note we aim to bound $E[p_h] \leq 1/k$, while by assumption

$$E[q_h] = \Pr_{h,x,y}[h(x) = h(y)] \leq 1/R + 1/n. \quad (1)$$

Now let $L_h := \{x : |B_h(x)| < 2n/R + k\}$, and note $|L_h| = (1 - p_h)n$. Let us write

$$q_h = \Pr_{x,y}[h(x) = h(y) | x \in L_h] \Pr[x \in L_h] + \Pr_{x,y}[h(x) = h(y) | x \notin L_h] \Pr[x \notin L_h].$$

The latter summand is $\geq ((2n/R + k)/n)p_h = (2/R + k/n)p_h$.

For the first summand, note

$$\Pr_{x,y}[h(x) = h(y) | x \in L_h] \Pr[x \in L_h] = \Pr_{x,y}[h(x) = h(y) | x \wedge y \in L_h] \Pr[x \wedge y \in L_h]$$

because if $y \notin L_h$ then there cannot be a collision with $x \in L_h$. The term $\Pr_{x,y}[h(x) = h(y) | x \wedge y \in L_h]$ is $\geq 1/R$ by Fact 4. The term $\Pr[x \wedge y \in L_h]$ is $(1 - p_h)^2 \geq 1 - 2p_h$.

Overall,

$$q_h \geq \frac{1}{R}(1 - 2p_h) + (2/R + k/n)p_h = p_h k/n + 1/R.$$

Taking expectations and recalling (1),

$$E[p_h]k/n + 1/R \leq 1/R + 1/n,$$

as desired. ■

0.2 Convolution 3XOR

Define the problem *convolution 3XOR*, denoted C3XOR, as: Given array A of n strings of $O(\lg n)$ bits, determine if $\exists i, j \leq n : A[i] + A[j] = A[i + j]$. Again, sum is bit-wise xor.

Lemma 5. If C3XOR can be solved with error 1% in time $t \leq n^{2-\Omega(1)}$, then so can 3XOR.

Intuition. We are given an instance of 3XOR consisting of a set S of n vectors. Suppose for any $x \in S$ we define $A[x] := x$, and untouched elements of $A[x]$ are set randomly so as to never participate in a solution.

Now if $x + y = z$ then $A[x] + A[y] = A[z] = A[x + y]$. Using again $x + y = z$ we get $A[x] + A[y] = A[x + y]$. Hence this solution will be found in C3XOR. Conversely a solution to C3XOR corresponds to a 3XOR solution, since A is filled with elements with S (and random otherwise).

This reduction is correct. But it is too slow because the size of A may be too large.

In our second attempt we try to do as above, but make sure the vector A is small. Suppose we had a hash function $h : S \rightarrow [n]$ that was both 1-1 and linear.

Then we could let again $A[h(x)] := x$.

If $x + y = z$ then $A[h(x)] + A[h(y)] = A[h(z)]$ by definition. And using again $x + y = z$ and linearity, we get $h(x + y) = h(x) + h(y) = h(z)$, and so we get $A[h(x)] + A[h(y)] = A[h(x) + h(y)]$ as desired.

But the problem is that there is no such hash function. (Using linear algebra one sees that there is no hash function that shrinks and is both linear and 1-1.)

The solution is to implement the hash-function based solution, and handle the few collisions separately.

Proof: Use the hash function h from Definition 2 mapping input elements of $\ell = O(\lg n)$ bits to $r := (1 - \alpha) \lg n$ bits, for a constant α to be determined. So the range has size $R = 2^r = n^{1-\alpha}$. By Lemma 3, the expected number of elements falling into buckets with more than $t := 3n/R$ elements is $\leq R$. For each of these elements, we can easily determine in time $\tilde{O}(n)$ if it participates to a solution. The time for this part is $\tilde{O}(Rn)$ with high probability, by a Markov bound.

It remains to look for solutions $x + y + z = 0$ where the three elements all are hashed to not-overloaded buckets. For each $i, j, k \in [R]$, we look for a solution where x, y, z are respectively at positions i, j, k of their buckets. Specifically, fill an array A of size $O(R)$ as follows: put the i th (j th, k th) element x of bucket $h(x)$ at position $A[h(x)01]$ ($A[h(x)10], A[h(x)11]$), where $h(x)01$ denotes the concatenation of the bit-strings $h(x)$ and 01. The untouched elements of A are set to a value large enough that it can be easily shown they cannot participate in a solution. Run the algorithm for C3XOR on A .

If there is a solution $x + y + z = 0$, suppose x, y, z are the i th (j th, k th) elements of their buckets. Then for that choice of i, j, k we have $A[h(x)01] = x, A[h(y)10] = y, A[h(z)11] = z$, and so $A[h(x)01] + A[h(y)10] = A[h(z)11]$. By linearity of h , and the choice of the vectors 01, 10, 11, we get $h(z)11 = h(x)01 + h(y)10$. So this solution will be found.

Conversely, any solution found will be a valid solution for 3XOR, by construction of A .

The time for this part is as follows. We run over $t^3 = O(n^3/R^3)$ choices for the indices. For each choice we run the C3XOR algorithm on an array of size $O(R)$. If the time for the latter is $R^{2-\epsilon}$, we can pick $R = n^{1-\alpha}$ for a small enough α so that the time is $\tilde{O}(n^{3\alpha}n^{(2-\epsilon)(1-\alpha)}) = n^{2-\epsilon'}$. (Here we first amplify the error of the C3XOR algorithm to $1/n^3$ by running it $O(\lg n)$ times and taking majority.)

The first part only takes time $O(Rn) = O(n^{2-\alpha})$, so overall the time is $n^{2-\epsilon''}$. \blacksquare

0.3 Reducing C3XOR to listing triangles

Lemma 6. Suppose that given the adjacency list of a graph with m edges and z triangles (and $O(m)$ nodes) one can list $\min\{z, m\}$ triangles in time $m^{1.33-\epsilon}$ for a constant $\epsilon > 0$. Then one can solve C3XOR on a set of size n with error 1% in time $n^{2-\epsilon'}$ for a constant $\epsilon' > 0$.

In fact, the hard graph instances will have $n = m^{1-\Omega(1)}$ nodes.

Proof: We are given an array A and want to know if $\exists a, b \leq n : A[a] + A[b] = A[a + b]$. It is convenient to work with the equivalent question of the existence of a, b such that $A[a + b_h] + A[a + b_\ell] = A[b]$, where b_h, b_ℓ are each half the $\lg n$ bits of b .

We use again the linear hash function h . To prove Lemma 5 we hashed to $R = n^{1-\epsilon}$ elements. Now we pick $R := \sqrt{n}$. By the paragraph after Definition 2, among the $\leq n^2$ pairs a, b that do not constitute a solution (i.e., $A[a + b_h] + A[a + b_\ell] \neq A[b]$), we expect $\leq n^2/R$ of them to satisfy

$$h(A[a + b_h]) + h(A[a + b_\ell]) = h(A[b]) \quad (\star).$$

By a Markov argument, with constant probability there are $\leq 2n^2/R = 2n^{1.5}$ pairs a, b that do not constitute a solution but satisfy (\star) . The reduction works in that case. (One can amplify the success probability by repetition.)

We set up a graph with $m := 3n^{1.5}$ edges where triangles are in an easily-computable 1 – 1 correspondence with pairs a, b satisfying (\star) . We then run the algorithm for listing triangles. For each triangle in the list, we check if it corresponds to a solution for C3XOR. This works because if the triangle-listing algorithm returns as many as m triangles then, by above, at least one triangle corresponds to a correct solution. Hence, if listing can be done in time $m^{4/3-\epsilon}$ then C3XOR can be solved in time $(3n^{1.5})^{4/3-\epsilon} = n^{2-\epsilon'}$.

We now describe the graph. The graph is tripartite. One part has $\sqrt{n} \times R$ nodes of the form (b_h, x) ; another has $\sqrt{n} \times R$ nodes of the form (b_ℓ, y) ; and the last part has n nodes of the form (a) . Node (a) is connected to (b_h, x) if $h(A[a + b_h]) = x$, and to (b_ℓ, y) if $h(A[a + b_\ell]) = y$. Nodes (b_h, x) and (b_ℓ, y) are connected if, letting $b = b_h + b_\ell$, $h(A[b]) = x + y$.

We now count the number of edges of the graph. Edges of the form $(a) - (b_h, x)$ (or $(a) - (b_\ell, y)$) number $n^{1.5}$, since a, b_h determine x . Edges $(b_h, x) - (b_\ell, y)$ number again $n^{1.5}$, since for each $b = b_h + b_\ell$ and x there is exactly one y yielding an edge.

The aforementioned 1-1 correspondence between solutions to C3XOR and triangles is present by construction. \blacksquare

Acknowledgments. I am very grateful to Rasmus Pagh and Virginia Vassilevska Williams for answering my many questions on finding triangles in graphs. I also thank Siyao Guo for pointing out that a step in a previous proof of Lemma 6 was useless.

References

- [AC05] Nir Ailon and Bernard Chazelle. Lower bounds for linear degeneracy testing. *J. ACM*, 52(2):157–171, 2005.
- [BDP08] Ilya Baran, Erik D. Demaine, and Mihai Pătraşcu. Subquadratic algorithms for 3sum. *Algorithmica*, 50(4):584–596, 2008.
- [CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, Cambridge, MA, second edition, 2001.
- [Eri99] Jeff Erickson. Lower bounds for linear satisfiability problems. *Chicago J. Theoret. Comput. Sci.*, (8), 1999.
- [GO95] Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995.
- [MR97] David K. Maslen and Daniel N. Rockmore. Generalized FFTs—a survey of some recent results. In *Groups and computation, II (New Brunswick, NJ, 1995)*, volume 28 of *DI-MACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 183–237. Amer. Math. Soc., Providence, RI, 1997.
- [Păt10] Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Symposium on the Theory of Computing (STOC)*, pages 603–610, 2010.
- [PW10] Mihai Pătraşcu and Ryan Williams. On the possibility of faster SAT algorithms. In *Symposium on Discrete Algorithms (SODA)*, pages 1065–1075, 2010.
- [VW09] Virginia Vassilevska and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. In *Symposium on the Theory of Computing (STOC)*, pages 455–464, 2009.