Electronic Colloquium on Computational Complexity, Report No. 9 (2013)

3SUM, 3XOR, Triangles

Zahra Jafargholi* Emanuele Viola*

January 3, 2013

Abstract

We show that if one can solve 3SUM on a set of size n in time $n^{1+\epsilon}$ then one can list t triangles in a graph with m edges in time $\tilde{O}(m^{1+\epsilon}t^{1/3+\epsilon'})$ for any $\epsilon' > 0$. This is a reversal of Pătraşcu's reduction from 3SUM to listing triangles (STOC '10).

We then re-execute both Pătraşcu's reduction and our reversal for the variant 3XOR of 3SUM where integer summation is replaced by bit-wise xor. As a corollary we obtain that if 3XOR is solvable in linear time but 3SUM requires quadratic randomized time, or vice versa, then the randomized time complexity of listing m triangles in a graph with m edges is $m^{4/3}$ up to a factor m^α for any $\alpha > 0$.

Our results are obtained building on and extending works by the Paghs (PODS '06) and by Vassilevska and Williams (FOCS '10).

*Supported by NSF grant CCF-0845003. Email: {zahra,viola}@ccs.neu.edu

ISSN 1433-8092

1 Introduction

The 3SUM problem asks if there are three integers a, b, c summing to 0 in a given set of n integers of magnitude poly(n). This problem can be easily solved in time $\tilde{O}(n^2)$. (Throughout, $\tilde{O}$ and $\tilde{\Omega}$ hide logarithmic factors.) 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 shows that if we can list m triangles in a graph with m edges given as adjacency list in time $m^{1.3\bar{3}-\Omega(1)}$ then we can solve 3SUM in time $n^{2-\Omega(1)}$.[Păt10]

To put this outstanding result in context we briefly review the state-of-the-art on triangle detection and listing algorithms. All the graphs in this paper are undirected and simple. Given the adjacency list of a graph with m edges, Alon, Yuster, and Zwick show in [AYZ97] how to determine if it contains a triangle in time $O(m^{2\omega/\omega+1})$ where $\omega < 2.376$ is the exponent of matrix multiplication. If $\omega = 2$ then the bound is $O(m^{1.3\bar{3}})$. For listing *all* triangles in a graph 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 $\le O(m\sqrt{m})$ triangles going through some node of degree $\le \sqrt{m}$, and then the $\le O(m/\sqrt{m})^3 = O(m^{1.5})$ triangles using nodes of degree $> \sqrt{m}$ only.) However, to list only m triangles conceivably time $\tilde{O}(m)$ suffices. In fact, Pagh (personal communication 2011) points out an algorithm for this problem achieving time $\tilde{O}(m^{1.5-\Omega(1)})$ and, assuming that the exponent of matrix multiplication is 2, time $\tilde{O}(m^{1.4})$.

In this work we suggest to study variants of 3SUM over other domains. For concreteness, we focus on the problem which we call 3XOR and which 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 $\tilde{\Omega}(n^2)$. But it is interesting to note that if we ask for *any* number (as opposed to 3) of elements that sums to 0 the difference in domains translates in a significant difference in complexity: SUBSET-SUM is NP-hard, whereas SUBSET-XOR can be solved efficiently via gaussian elimination. On the other hand, SUBSET-XOR remains NP-hard if the number of elements that need to sum to 0 is given as part of the input.[Var97]

In light of this, it would be interesting to relate the complexities of 3SUM and 3XOR. For example, it would be interesting to show that one problem is solvable in time $n^{2-\Omega(1)}$ if and only if the other is. Less ambitiously, the weakest possible link would be to exclude a scenario where, say, 3SUM requires time $\tilde{\Omega}(n^2)$ while 3XOR is solvable in time $\tilde{O}(n)$. We are not even able to exclude this scenario, and we raise it as an open problem.

However we manage to spin a web of reductions around 3SUM, 3XOR, and various problems related to triangles, a web that extends and complements the pre-existing web. One consequence is that the only way in which the aforementioned scenario is possible is that listing m triangles requires exactly $m^{1.3\bar{3}}$ up to lower-order factors.

Corollary 1. Suppose that 3SUM requires randomized time $\tilde{\Omega}(n^2)$ and 3XOR is solvable in time $\tilde{O}(n)$, or vice versa. Then, given the adjacency list of a graph with m edges and z triangles (and $O(m)$ nodes), the randomized time complexity of listing $\min\{z, m\}$ triangles is $m^{1.3\bar{3}}$ up to a factor m^α for any $\alpha > 0$.

We now overview our reductions. First we build on and extend a remarkable reduction [WW10] by Vassilevska and Williams from listing triangles to detecting triangles. Their reduction worked on adjacency matrixes, and a main technical contribution of this work is an extension to adjacency lists which is needed in our subquadratic context.

Lemma 2. Suppose given the adjacency list of a graph with m edges and $n = O(m)$ nodes, one can decide if it is triangle-free in time $m^{1+\epsilon}$ for $\epsilon > 0$.Then, given the adjacency list of a graph G with m edges, $n = O(m)$ nodes and z triangles, a positive integer t and $\epsilon' > 0$ one can list $\min\{t, z\}$ triangles in G, in time $\tilde{O}(m^{1+\epsilon}t^{1/3+\epsilon'})$.

For context, Pagh shows a reduction from finding the set of edges involved in some triangle to listing triangles, see [Amo11, §6].

Next we move to reductions between 3SUM and detecting triangles. The Paghs [PP06, §6] give an algorithm to "compute the join of three relations [...] where any pair has a common attribute not shared by the third relation." One component of their algorithm can be phrased as a randomized reduction from detecting (tripartite, directed) triangles to 3SUM. The same reduction works for 3XOR. Here our main technical contribution is to exhibit a deterministic reduction.

Lemma 3. Suppose that one can solve 3SUM or 3XOR on a set of size n in time $\tilde{O}(n^{1+\epsilon})$ for $\epsilon \geq 0$. Then, given the adjacency list of a graph with m edges, $n = O(m)$ nodes, one can decide if it is triangle-free in time $\tilde{O}(m^{1+\epsilon})$.

In particular, this shows that solving either 3SUM or 3XOR in time $O(m^{2\omega/(\omega+1)-\Omega(1)})$, where ω is the exponent of matrix multiplication, would improve the aforementioned triangle-detection algorithm in [AYZ97].

The combination of the previous two lemmas yields a reversal of Pătraşcu's aforementioned reduction from 3SUM to listing triangles. In particular, this shows that solving 3SUM or 3XOR in time $\tilde{O}(n^{1+\alpha})$ for some $\alpha < 1/15$ would improve the aforementioned Pagh's triangle-listing algorithm. (Recall the latter has complexity $\tilde{O}(m^{1.4})$ assuming $\omega = 2$.)

Corollary 4. [Reverse Pătraşcu] Suppose that one can solve 3SUM or 3XOR on a set of size n in time $n^{1+\epsilon}$ for $\epsilon > 0$. Then, given the adjacency list of a graph G with m edges, $n = O(m)$ nodes and z triangles, a positive integer t and $\epsilon' > 0$, one can list $\min\{t, z\}$ triangles in G in time $\tilde{O}(m^{1+\epsilon}t^{1/3+\epsilon'})$.

Finally, we re-execute Pătraşcu's reduction for 3XOR instead of 3SUM. Our execution also avoids some technical difficulties and so is a bit simpler.

Theorem 5. 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 $\epsilon' > 0$.

1.1 Techniques

We now explain how we reduce listing t triangles in a graph to detecting triangles. First we recall the strategy [WW10] by Vassilevska and Williams that works in the setting of adjacency matrixes, as opposed to lists.

Without loss of generality, we work with a tripartite graph with n nodes per part. Their recursive algorithm proceeds as follows. First, divide each part in two halves of $n/2$ nodes, then recurse on each of the 8 subgraphs consisting of triples of halves. Note edges are duplicated, but triangles are not. One uses the triangle-detection algorithm to avoid recursing on subproblems that do not contain triangles. The most important feature is that one keeps track, throughout the execution of the algorithm, of how many subproblems have been produced, and if this number reaches t one stops introducing new subproblems.

We next explain how to extend this idea to adjacency lists. At a high level we use the same recursive approach based on partitions and keeping track of the total number of subproblems. However in our setting partitioning is more difficult, and we resort to *random partitioning*. Each part of the graph at hand is partitioned in 2 subsets by flipping an unbiased coin for each node. If we start with a graph with m edges, each of the 8 subgraphs (induced by the 8 triples of subsets) expects $m/4$ edges. We would like to guarantee this result with high probability simultaneously for each of the 8 subgraphs. For this goal we would like to show that the size of each of the 8 subgraphs is concentrated around its expectation. Specifically, fix a subgraph and let X_e be the indicator random variable for edge e being in the subgraph. We would like to show

$$\Pr[\sum_e X_e > m/4 + \gamma m] < 1/8 \qquad (\star)$$

for some small γ. By a union bound we can then argue about all the 8 subgraphs simultaneously.

Assuming $(\star)$ we conclude the proof similarly to [WW10] as follows, setting for simplicity $\gamma = 0$. Each recursive step reduces the problem size by a factor 4. Let s_i be the number of subproblems at level i of the recursion. The running time of the algorithm is of the order of

$$\sum_{i \le \lg_4 m} s_i T(m/4^i) < \sum s_i m^{1+\epsilon}/4^i,$$

where $T(x) = x^{1+\epsilon}$ is the time of the triangle detection algorithm. Since we recurse on ≤ 8 subproblems we have $s_i \le 8^i$; since we make sure to never have more than t subproblems we have $s_i \le t$. Picking a breaking-point ℓ we can write the order of the time as

$$m^{1+\epsilon}\left(\sum_{i\le\ell} 8^i/4^i + \sum_{i>\ell}^{\lg_4 m} t/4^\ell\right) = m^{1+\epsilon}\tilde{O}(2^\ell + t/4^\ell)$$

which is minimized to $\tilde{O}(m^{1+\epsilon}t^{1/3})$ for $\ell = \lg t^{1/3}$.

We now discuss how we guarantee $(\star)$. The obstacle is that the variables X_e are not even pairwise independent; consider for example two edges sharing a node. We overcome this obstacle by introducing a first stage in the algorithm in which we list all triangles involving at least one node of high degree ($> m^{1-\epsilon}$), which only costs time $O(m^{1+\epsilon})$. We then remove these high-degree nodes. What we have gained is that now most pairs of variables $X_e, X_{e'}$ are pairwise independent. This lets us carry through an argument like Chebychev's inequality and in turn argue about concentration around the expectation $m/4$.

Organization. In §2 we reduce detecting triangles to 3SUM and 3XOR. In §3 we give the reduction from listing to detecting triangles in a graph. In §4 through §4.3 we give the reduction from 3XOR to listing triangles.

Note that Corollary 1 follows immediately from the combination of: Pătraşcu's reduction from 3SUM to listing triangles [Păt10], our reversal (Corollary 4), and our re-execution for 3XOR (Theorem 5).

2 Reducing detecting triangles to 3SUM and 3XOR

In this section we prove Lemma 3, restated next.

Lemma 3. Suppose that one can solve 3SUM or 3XOR on a set of size n in time $\tilde{O}(n^{1+\epsilon})$ for $\epsilon \ge 0$. Then, given the adjacency list of a graph with m edges, $n = O(m)$ nodes, one can decide if it is triangle-free in time $\tilde{O}(m^{1+\epsilon})$.

Recall that all graphs in this paper are undirected (and simple). Still, we use ordered-pair notation for edges. A triangle is a set of 3 distinct edges where each node appears twice, such as $(a,b),(c,b),(a,c)$.

The deterministic reduction relies on combinatorial designs, family of m subsets of a small universe with small pairwise intersections. Specifically we need the size of the sets to be linear in the universe size, and the bound on the intersection a constant fraction of the set size. Such parameters were achieved in [NW94] but with a construction running in time exponential in m. We use the different construction computable in time $\tilde{O}(m)$ by Gutfreund and Viola [GV04].

Lemma 6. [GV04] For every constant $c > 1$ and large enough m there is a family of m sets $S_i, i = 1, \ldots, m$ such that:
1) $|S_i| = c^2 \lg m$,
2) $|S_i \cap S_j| \le 2c \lg m$ for $i \ne j$,

3) $S_i \subseteq [50 \cdot c^3 \lg m]$,
4) the family may be constructed in time $\tilde{O}(m)$.

Note that by increasing c in Lemma 6 we can have the bound on the intersection size be an arbitrarily small constant fraction of the set size.

Proof: We show how to reduce detecting directed triangles to 3SUM. The same approach reduces detecting un-directed triangles to 3XOR (except that the numbers below would be considered in base 2 instead of 10). To reduce detecting un-directed triangles to 3SUM, we can simply make our graphs directed by repeating each edge with direction swapped.

We first review the randomized reduction, then we make it deterministic. Given the adjacency list of graph $G = (V, E)$, assign an ℓ-bit number, uniformly and independently to each node in the graph G, ℓ to be determined, i.e. $\forall a \in V, \quad X_a \in_U \{0,1\}^\ell$. For each edge $e = (a, b)$ let $Y_{(a,b)} := (X_a - X_b)$. Return the output of 3SUM on the set $Y := \{Y_{(a,b)} | (a, b) \in E\}$. If there is a triangle there are always 3 elements in Y summing to 0. Otherwise, by a union bound the probability that there are such 3 elements is $\leq 1/2$ for $\ell = 3 \lg m$.

To make the reduction deterministic, consider the family S_i of $O(m)$ sets from Lemma 6, with intersection size less than 1/5 of the set size. Assign to node a the number x_a whose decimal representation has 1 in the digits that belong to S_a, and 0 otherwise.

As before, we need to show that if there are 3 numbers $(x_a - x_{a'}), (x_b - x_{b'}), (x_c - x_{c'})$ in Y that sum to 0 then there is a triangle in the graph. Since the graph has no self loops, note that the existence of a triangle is implied by the fact that in the expression $(x_a - x_{a'}) + (x_b - x_{b'}) + (x_c - x_{c'})$ each of x_a, x_b, x_c appears exactly once with each of the two signs. We will show the latter. First, we claim that each of x_a, x_b, x_c appears the same number of times with each sign. Indeed, otherwise write the equation $x_a + x_b + x_c = x_{a'} + x_{b'} + x_{c'}$ and simplify equal terms. We are left with a number on one side of the equation that is non-zero in a set of digits that cannot be covered by the other 5, by the properties of the design. Hence the equation cannot hold. Note that when performing the sums in this equation there is no carry among decimal digits. Finally, we claim that no number can appear twice with the same sign. For else it is easy to see that there would be a self loop. ■

3 Reducing listing to detecting triangles

In this section we prove Lemma 2, restated next.

Lemma 2. Suppose given the adjacency list of a graph with m edges and $n = O(m)$ nodes, one can decide if it is triangle-free in time $m^{1+\epsilon}$ for $\epsilon > 0$.Then, given the adjacency list of a graph G with m edges, $n = O(m)$ nodes and z triangles, a positive integer t and $\epsilon' > 0$ one can list $\min\{t, z\}$ triangles in G, in time $\tilde{O}(m^{1+\epsilon} t^{1/3+\epsilon'})$.

Proof: Call A the triangle-detection algorithm. The triangle-listing algorithm is called B and has three stages. In stage one, we list triangles in G involving at least one high degree node. In this part we do not use algorithm A. In the second stage, we create a new graph G'

which is tripartite, and has the property that for each triangle in G there uniquely correspond in G' 6. In the final stage we run a recursive algorithm on G' and list $\min\{6t, 6z\}$ triangles in G' which would correspond to $\min\{t, z\}$ triangles in G. This recursive algorithm will make use of algorithm A.

Stage One. We consider a node to be high degree if its degree is $> \delta m$, for a parameter δ to be set later. We can list triangles involving a high degree node, if any exists, in time $\tilde{O}(m/\delta)$. To see this, note that we can sort the adjacency list and also make a list of high degree nodes in time $\tilde{O}(m)$. Also note that the number of nodes with high degree is $O(1/\delta)$, because the sum of all degrees is $2m$. For any high degree node h, for each edge $(a, b) \in E$ we search for two edges (a, h) and (b, h) in the adjacency list. Since the adjacency list is sorted, the search for each edge will take $\tilde{O}(\lg m)$ and for each high degree nodes we do this search $2m$ times so the running time of Stage One is $T_{B_1}(m) = \tilde{O}(m/\delta)$. Obviously at any point of this process, if the number of listed triangles reaches t we stop. If not, we remove the high degree nodes from G and move to the next stage.

Stage Two. We convert G into a tripartite graph $G' = (V', E')$ where $V' := I_1 \cup I_2 \cup I_3$ and each part of V' is a copy of V. For each edge (a, b) in E place in E' edge (a_i, b_j) for any $i, j \in \{1, 2, 3\}, i \neq j$.

A triangle in G yields 6 in G' by any choice of the indices i and j. A triangle in G' yields one in G by removing the indices, using that the graph is simple. This stage takes time $T_{B_2}(m) = \tilde{O}(m)$. In the next stage we look for $t' = 6t$ triangles in G'. Note that $|E'| = 6|E|$.

Stage Three. We partition each of I_1, I_2 and I_3 of V' randomly into two subsets, in a way specified below. Now we have 8 subgraphs, where each subgraph is obtained by choosing one subsets from each of I_1, I_2 and I_3. For each of the subgraphs, we use A to check if the subgraph contains a triangle. If it does, we recurse on the subgraph. We recurse till the number of edges in the subgraph is smaller than a constant C, at which point by brute force in time $\tilde{O}(C^3)$ we return all the triangles in the subgraph. Note that each triangle reported is unique since it only appears in one subproblem. We only need to list t' triangles in the graph, so when the number of subproblems that are detected to have at least one triangle reaches t', we do not need to introduce more.

To bound the running time, we need to bound the size of the input for each subproblem. If the random partition above is selected by deciding uniformly and independently for each node which subset it would be in, the expected number of edges in each subgraph is $m/4$. We introduce another parameter γ and consider the probability that all the 8 subproblems are of size smaller than $m/4 + m\gamma$. We call these subproblems roughly balanced.

To make the reduction deterministic we choose the partition by an almost 4-wise independent space [NN93, AGHP92].

Lemma 7 ([NN93, AGHP92])**.** There is an algorithm that maps a seed of $O(\lg \lg n + k + \lg 1/\alpha)$ bits into n bits in time $\tilde{O}(n)$ such that the induced distribution on any k bits is α-close to uniform in statistical distance.

Claim 8. Let $0 < \gamma < 1/4$. There are δ and α such that for all sufficiently large m, if we partitioning each of I_1, I_2 and I_3 into two subsets using an α-almost 4-wise independent generator, with probability > 0 all the 8 subgraphs induced by triples of subsets have less than $m(1/4 + \gamma)$ edges.

We later give the proof of this claim. To make sure that all the subproblems generated during the execution of the entire algorithm are roughly balanced, each time we partition we enumerate all seeds for the almost 4-wise independent generator, and pick the first yielding the conclusion of Claim 8. This only costs $\tilde{O}(m)$ time.

To analyze the running time of Stage Three, let s_i denote the number of subproblems at level i of the recursion. At the ith level, we run algorithm A s_i times on an input of size $\leq 6m\,(1/4+\gamma)^i$, so the running time of the recursive algorithm at level i is $\tilde{O}\left(s_i \cdot T_A\left(6m\,(1/4+\gamma)^i\right)\right)$, where $T_A(m) = m^{1+\epsilon}$ is the running time of algorithm A.

Note that $s_i \leq 8^i$ by definition and $s_i \leq t'$ because at any level we keep at most t' subproblems.

Pick $\ell := \lg t^{1/3}$. The running time of this stage is

$$\begin{aligned}
T_{B_3}(6m, 6t) &= \tilde{O}\left(\sum_{i=0}^{\ell-1} 8^i T_A\left(6m\,(1/4+\gamma)^i\right) + \sum_{i=\ell}^{\lg 6m} 6t \cdot T_A\left(6m\,(1/4+\gamma)^i\right) + 6tC^3\right) \\
&= \tilde{O}\left(\sum_{i=0}^{\ell-1} 8^i \left(m\,(1/4+\gamma)^i\right)^{1+\epsilon} + \sum_{i=\ell}^{\lg 6m} t \cdot \left(m\,(1/4+\gamma)^i\right)^{1+\epsilon}\right) \\
&= \tilde{O}\left(m^{1+\epsilon} \cdot (1/4+\gamma)^{\ell(1+\epsilon)} \cdot (8^\ell + t)\right) \\
&= \tilde{O}\left(m^{1+\epsilon} \cdot t^{(1/3)\lg_2(1/4+\gamma)} \cdot t\right) \\
&= \tilde{O}\left(m^{1+\epsilon} t^{1/3+\epsilon'}\right),
\end{aligned}$$

for a sufficiently small $\gamma > 0$. (Note $(1/3)\lg_2(1/4+\gamma) \to_{\gamma\to 0} -2/3$.)

In the end the running time of algorithm B is

$$T_B(m,t) = T_{B_1}(m) + T_{B_2}(m) + T_{B_3}(6m, 6t) = \tilde{O}\left(m^{1+\epsilon} t^{1/3+\epsilon'}\right).$$

$\blacksquare$

Claim 8. Let $0 < \gamma < 1/4$. There are δ and α such that for all sufficiently large m, if we partitioning each of I_1, I_2 and I_3 into two subsets using an α-almost 4-wise independent generator, with probability > 0 all the 8 subgraphs induced by triples of subsets have less than $m(1/4 + \gamma)$ edges.

Proof:[of claim 8] Let us fix one of the subgraphs and call it S and define the following random variables,

$$\forall 0 \leq i \leq m, \quad X_i = \begin{cases} 1 & \text{if } e_i \in S, \\ 0 & \text{if } e_i \notin S. \end{cases}$$

We have $|E[X_i] - 1/4| \leq \alpha$ and $|E[\sum_i X_i] - m/4| \leq \alpha m$. To prove the claim, we show that the probability that S has more than $m(1/4+\gamma)$ edges is less than $1/16$; and by a union bound we conclude. In other words we need to show:

$$P_S := \Pr\left[\sum_i X_i - m/4 \geq m\gamma\right] \leq 1/16.$$

By a Markov bound we have,

$$P_S \leq \Pr\left[\left(\sum_i X_i - m/4\right)^2 \geq (m\gamma)^2\right] \leq E\left[\left(\sum_i X_i - m/4\right)^2\right] / (m\gamma)^2 .$$

Later we bound $E\left[(\sum_i X_i - m/4)^2\right] = O((\alpha+\delta)m^2)$ from which the claim follows.

Now we get the bound for $E\left[(\sum_i X_i - m/4)^2\right]$.

$$\begin{aligned}
E\left[\left(\sum_i X_i - m/4\right)^2\right] &= E\left[\left(\sum_i X_i\right)^2 + (m/4)^2 - \left(m\sum_i X_i\right)/2\right] \\
&\leq E\left[\sum_{i\neq j} X_iX_j\right] + E\left[\sum_i X_i^2\right] + \frac{m^2}{16} - \frac{m}{2}m\left(\frac{1}{4}-\alpha\right) \\
&\leq E\left[\sum_{i\neq j} X_iX_j\right] + \frac{m}{4} + \alpha m - \frac{m^2}{16} + m^2\alpha/2 \\
&= E\left[\sum_{i\neq j} X_iX_j\right] + O(\alpha m^2) - \frac{m^2}{16}.
\end{aligned}$$

$E[X_iX_j]$ is the probability that two edges e_i and e_j are both in S. If our distribution were uniform the probability would be $1/16$ for the pairs of edges that do not share a node, and $1/8$ for the pairs of edges that do share a node. Let ρ be the number of unordered pairs of edges that share a node. We have:

$$\begin{aligned}
E\left[\sum_{i\neq j} X_iX_j\right] = \sum_{i\neq j} E[X_iX_j] &\leq 2\rho(1/8+\alpha) + 2\left(\binom{m}{2} - \rho\right)(1/16+\alpha) \\
&\leq m^2/16 + \rho/8 + 4\rho\alpha + \alpha m^2 \leq m^2/16 + \rho/8 + O(\alpha m^2).
\end{aligned}$$

Note

$$\rho = \sum_{a\in V}\binom{\text{degree}(a)}{2} \leq \sum_{a\in V}\text{degree}(a)^2/2 \leq \delta m \sum_{a\in V}\text{degree}(a)/2 \leq \delta m^2,$$

since after stage one of the algorithm there are no nodes with degree more than δm.

Hence we obtain

$$E\left[\left(\sum_i X_i - m/4\right)^2\right] \leq \frac{m}{4} + O(\alpha+\delta)m^2 = O(\alpha+\delta)m^2,$$

as desired. ■

4 Reducing 3XOR to listing triangles

In this section we prove theorem 5.

Theorem 5. 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 $\epsilon' > 0$.

The proof of Theorem 5 follows the one in [Păt10] for 3SUM, which builds on results in [BDP08]. However the proof of Theorem 5 is a bit simpler. This is because it avoids some steps in [BDP08, Păt10] 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, Păt10].

The remainder of this section is organized as follows. In §4.1 we cover some preliminaries and prove a hashing lemma by [BDP08] that will be used later.[1] The proof of the reduction in Theorem 5 is broken up in two stages. First, in §4.2 we reduce 3XOR to the problem C3XOR which is a "convolution" version of 3XOR. Then in §4.3 we reduce C3XOR to listing triangles.

4.1 Hashing and preliminaries

We define next the standard hash function we will use.

Definition 9. For input length ℓ and output length r, the hash function h uses r ℓ-bit keys $\bar{a} := (a^1, \ldots, a^r)$ and is defined as $h_{\bar{a}}(x) := (\langle a^1, x\rangle, \ldots, \langle a^r, x\rangle)$, where $\langle ., . \rangle$ 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

[1] In [BDP08, Păt10] they appear to use this lemma with a hash function that is not known to satisfy the hypothesis of the lemma. However probably one can use instead similar hash functions such as one in [Die96] that does satisfy the hypothesis. We thank Martin Dietzfelbinger for a discussion on hash functions.

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 $v_1,\ldots,v_n \in \{0,1\}^\ell$ of 3XOR is reduced to $h(v_1),\ldots,h(v_n)$ where $h = h_a$ 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 $[R] = \{1,2,\ldots,R\}$, 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 10 ([BDP08])**.** Let h be a random function $h : U \to [R]$ 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 11. Let $f : D \to [R]$ 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 10] Pick x, y uniformly and independently in S ($x = y$ possible). For given h, let

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

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

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

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 11. 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. ■

4.2 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 12. 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 $h : S \to [n]$ 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 9 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 10, 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 $i, j, k \in [t]$, 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 ith (jth, kth) 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 ith (jth, kth) 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 $n^{2-\epsilon''}$. ■

4.3 Reducing C3XOR to listing triangles

Lemma 13. 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 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 12 we hashed to $R = n^{1-\epsilon}$ elements. Now we pick $R := \sqrt{n}$. By the paragraph after Definition 9, 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

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

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. ■

Acknowledgments, cf. [Vio11]. We are very grateful to Rasmus Pagh and Virginia Vassilevska Williams for answering many questions on finding triangles in graphs. Rasmus also pointed us to [PP06, Amo11]. We also thank Siyao Guo for pointing out that a step in a previous proof of Lemma 13 was useless, and Ryan Williams for stimulating discussions.

References

[AC05] Nir Ailon and Bernard Chazelle. Lower bounds for linear degeneracy testing. *J. of the ACM*, 52(2):157–171, 2005.

[AGHP92] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures & Algorithms*, 3(3):289–304, 1992.

[Amo11] Rasmus Resen Amossen. *Scalable Query Evaluation in Realational Databases.* PhD thesis, IT University of Copenhagen, 2011.

[AYZ97] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[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.

[Die96] Martin Dietzfelbinger. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 569–580, 1996.

[Eri99] Jeff Erickson. Lower bounds for linear satisfiability problems. *Chicago J. Theoret. Comput. Sci.*, 1999, 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.

[GV04] Dan Gutfreund and Emanuele Viola. Fooling parity tests with parity gates. In *8thWorkshop on Randomization and Computation (RANDOM)*, pages 381–392. Springer, 2004.

[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 *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 183–237. Amer. Math. Soc., Providence, RI, 1997.

[NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: efficient constructions and applications. *SIAM J. on Computing*, 22(4):838–856, 1993.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. of Computer and System Sciences*, 49(2):149–167, 1994.

[Păt10] Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *ACM Symp. on the Theory of Computing (STOC)*, pages 603–610, 2010.

[PP06] Anna Pagh and Rasmus Pagh. Scalable computation of acyclic joins. In *ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS)*, pages 225–232, 2006.

[PW10] Mihai Pătraşcu and Ryan Williams. On the possibility of faster SAT algorithms. In *ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1065–1075, 2010.

[Var97] Alexander Vardy. The intractability of computing the minimum distance of a code. *IEEE Transactions on Information Theory*, 43(6):1757–1766, 1997.

[Vio11] Emanuele Viola. Reducing 3XOR to listing triangles, an exposition. Available at http://www.ccs.neu.edu/home/viola/, 2011.

[VW09] Virginia Vassilevska and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. In *ACM Symp. on the Theory of Computing (STOC)*, pages 455–464, 2009.

[WW10] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 645–654, 2010.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de