# Listing Triangles

Andreas Björklund[1⋆], Rasmus Pagh[2⋆⋆], Virginia Vassilevska Williams[3⋆⋆⋆], and Uri Zwick[4†]

[1] Department of Computer Science, Lund University, Sweden.
[2] IT University of Copenhagen, Denmark.
[3] Computer Science Department, Stanford University, USA.
[4] Blavatnik School of Computer Science, Tel Aviv University, Israel.

**Abstract.** We present new algorithms for listing triangles in dense and sparse graphs. The running time of our algorithm for dense graphs is $\tilde{\mathcal{O}}(n^\omega + n^{3(\omega-1)/(5-\omega)}t^{2(3-\omega)/(5-\omega)})$, and the running time of the algorithm for sparse graphs is $\tilde{\mathcal{O}}(m^{2\omega/(\omega+1)} + m^{3(\omega-1)/(\omega+1)}t^{(3-\omega)/(\omega+1)})$, where $n$ is the number of vertices, $m$ is the number of edges, $t$ is the number of triangles to be listed, and $\omega < 2.373$ is the exponent of fast matrix multiplication. With the current bound on $\omega$, the running times of our algorithms are $\tilde{\mathcal{O}}(n^{2.373} + n^{1.568}\,t^{0.478})$ and $\tilde{\mathcal{O}}(m^{1.408} + m^{1.222}\,t^{0.186})$, respectively. We first obtain randomized algorithms with the desired running times and then derandomize them using *sparse recovery* techniques. If $\omega = 2$, the running times of the algorithms become $\tilde{\mathcal{O}}(n^2 + nt^{2/3})$ and $\tilde{\mathcal{O}}(m^{4/3} + mt^{1/3})$, respectively. In particular, if $\omega = 2$, our algorithm lists $m$ triangles in $\tilde{\mathcal{O}}(m^{4/3})$ time. Pătraşcu (STOC 2010) showed that $\Omega(m^{4/3-o(1)})$ time is required for listing $m$ triangles, unless there exist subquadratic algorithms for 3SUM. We show that unless one can solve quadratic equation systems over a finite field significantly faster than the brute force algorithm, our triangle listing runtime bounds are tight assuming $\omega = 2$, also for graphs with more triangles.

## 1 Introduction

Algorithmic problems concerning the set of triangles in a graph have recently received much attention, due to applications in various kinds of graph analysis such as the study of social processes [8], community detection [5], and dense subgraph mining [26]. Many of these problems require the listing of all triangles in a graph — see [24, 6, 4] for a number of examples.

We consider simple, directed or undirected graphs with $n$ vertices and $m$ edges. A dense graph may contain $\Theta(n^3)$ triangles, so in terms of $n$ the worst-case

⋆ E-mail: andreas.bjorklund@cs.lth.se.
⋆⋆ E-mail: pagh@itu.dk.
⋆⋆⋆ Research supported by a Stanford School of Engineering Hoover Fellowship, NSF Grant CCF-1417238 and BSF Grant BSF:2012338. E-mail: virgi@cs.stanford.edu
† Research supported by BSF grant no. 2012338 and by the The Israeli Centers of Research Excellence (I-CORE) program, (Center No. 4/11). E-mail: zwick@tau.ac.il.

complexity of the trivial cubic time algorithm is optimal. However, most graphs of interest are not dense. In 1978 Itai and Rodeh [13] obtained an algorithm for listing all triangles in $\mathcal{O}\left(m^{3/2}\right)$, which is always an improvement over the naïve $\mathcal{O}\left(n^3\right)$ algorithm. Their algorithm is optimal as a graph with $m$ edges may contain $\Omega\left(m^{3/2}\right)$ triangles.

In this paper we consider *output sensitive* algorithms for triangle listing, which run asymptotically faster when the number $t$ of triangles is small, with *no additional assumptions* on the input graph. (For example, we do not consider running time bounds in terms of graph parameters such as arboricity.) Our approach is to combine known techniques for *counting* the number of triangles, using fast matrix multiplication, with algebraic and combinatorial techniques that allow us to compute the actual triangles. We initially obtain randomized algorithms which we then derandomize using *sparse recovery* techniques.

Since our focus is on constants in the exponents of running time, we use $\tilde{\mathcal{O}}(\cdot)$ notation to suppress multiplicative factors of size $n^{o(1)}$. For dense graphs, our algorithm runs in $\tilde{\mathcal{O}}\left(n^\omega + n^{3(\omega-1)/(5-\omega)}t^{2(3-\omega)/(5-\omega)}\right)$ time, where $\omega < 2.373$ is the exponent of square matrix multiplication [27, 18]. For sparse graphs, our algorithm runs in $\tilde{\mathcal{O}}\left(m^{2\omega/(\omega+1)} + m^{3(\omega-1)/(\omega+1)}t^{(3-\omega)/(\omega+1)}\right)$ time. Under the assumption $\omega = 2$ algorithms run in $\tilde{\mathcal{O}}\left(n^3\right)$ and $\tilde{\mathcal{O}}\left(m^{3/2}\right)$ algorithms for *every* possible value of $t$. Our dense and sparse algorithms are inter-dependent. The dense algorithm performs a sparsifying steps and calls the dense algorithm, while the sparse algorithm performs a densifying step and calls the dense algorithm.

Pătraşcu [23] has shown that listing $m$ triangles in a graph with $m$ edges requires time $\Omega\left(m^{4/3-\varepsilon}\right)$, for every $\varepsilon > 0$, unless there exists an algorithm for 3SUM running in $\mathcal{O}\left(n^{2-\delta}\right)$ time, for some $\delta > 0$. Our algorithm lists $m$ triangles in $\tilde{\mathcal{O}}\left(m^{2\omega/(\omega+1)}\right)$ time. With the current bound $\omega < 2.373$, our algorithm lists $m$ triangles in $\mathcal{O}\left(m^{1.408}\right)$. Interestingly, if $\omega = 2$, the running time becomes $\tilde{\mathcal{O}}\left(m^{4/3}\right)$, essentially matching the conditional lower bound of Pătraşcu [23]. Significant improvements of the exponents in our results are therefore unlikely.

The best previously available algorithms for triangle listing that we are aware of are the $\mathcal{O}\left(m^{3/2}\right)$ algorithm of Itai and Rodeh [13], from which it is also easy to obtain an $\tilde{\mathcal{O}}\left(n^\omega + \min(n^3, nt, t^{3/2})\right)$ algorithm, and an $\mathcal{O}\left(t^{1-\omega/3}n^\omega\right)$-time algorithm that follows from a reduction by Williams and Williams [28, Corollary G.1] from triangle listing to triangle detection. The running times obtained by our algorithms improve upon both of the aforementioned prior results for all values of $t$.

### 1.1 Related work

Figure 1 compares the results described above, focusing on worst-case time complexity. For completeness we now describe some other related work that is not directly comparable to our results.

Quite a bit of work has been done on triangle listing algorithms that perform well on real-life graphs. The paper of Schank and Wagner [24] contains a good

| Reference | Time bounds | If $\omega = 2$ |
|---|---|---|
| Itai and Rodeh [13] | $\tilde{\mathcal{O}}\left(n^\omega + \min(n^3, nt, t^{3/2})\right)$<br>$\mathcal{O}\left(m^{3/2}\right)$ | $\tilde{\mathcal{O}}\left(n^2 + \min(nt, t^{3/2})\right)$ |
| Williams and Williams [28] | $\tilde{\mathcal{O}}\left(n^\omega t^{1-\omega/3}\right)$ | $\tilde{\mathcal{O}}\left(n^2 t^{1/3}\right)$ |
| Pătraşcu [23] | $\tilde{\Omega}(\min(m^{4/3}, n^2, t^{4/3}))$ * | |
| **This paper** | $\tilde{\mathcal{O}}\left(n^\omega + n^{\frac{3(\omega-1)}{5-\omega}} t^{\frac{2(3-\omega)}{5-\omega}}\right)$<br>$\tilde{\mathcal{O}}\left(m^{\frac{2\omega}{\omega+1}} + m^{\frac{3(\omega-1)}{\omega+1}} t^{\frac{3-\omega}{\omega+1}}\right)$ | $\tilde{\mathcal{O}}\left(n^2 + nt^{2/3}\right)$<br>$\tilde{\mathcal{O}}\left(m^{4/3} + mt^{1/3}\right)$ |

**Fig. 1.** Upper and (conditional) lower bounds for listing $t$ triangles in a graph of $n$ vertices and $m$ edges. The results are stated in terms of the exponent $\omega$ of square matrix multiplication, which is known to be below 2.373 [27]. All bounds hold are for worst-case graphs and hold for every choice of $n, m, t \geq 1$. The rightmost column highlights the upper bounds that would result if $\omega = 2$. The lower bound by Pătraşcu marked by * relies on the assumption that 3SUM requires $\tilde{\Omega}(n^2)$ time.

overview of various algorithms with $\mathcal{O}\left(m^{3/2}\right)$ worst-case running time, and investigates how well these algorithms perform on graphs from various application areas, often running much faster than the worst-case analysis would suggest. One algorithm that is often able to beat the worst-case bound is based on enumerating a set of 2-paths where the degree of the middle vertex is no larger than the degrees of the start and end vertices (this is a simplified version of node-iterator-core from [24]). Recently, Berry et al. [4] gave a theoretical explanation why triangle listing is fast for most graphs, even for graphs with a skewed degree distribution, by studying a class of random graphs.

Recently many authors have studied triangle counting and listing algorithms for massive graphs, using either external memory [10, 20] or the MapReduce framework for distributed computation [1]. However, for worst-case graphs these algorithms all use $\Omega(m^{3/2})$ time, even when the number of triangles is zero.

As mentioned above, Pătraşcu [23] showed a link between triangle listing and the time complexity of 3SUM. Jafargholi and Viola [14] further investigated this connection, showing that surprising algorithms for 3SUM would lead to surprising algorithms for triangle listing.

Alon, Yuster, and Zwick [2] show how to efficiently *detect* the presence of small subgraphs in sparse graphs. For triangles they achieve a time bound of $\mathcal{O}\left(m^{2\omega/(\omega+1)}\right)$, and the algorithm even allows *counting* the number of triangles. The algorithm consists of a densification step that enumerates all 2-paths (i.e., paths with two edges) through vertices with degree at most $\Delta$, for a parameter $\Delta$. In this way all triangles that contain a vertex of degree at most $\Delta$ are found. The number of triangles within the set of vertices of degree larger than $\Delta$ is found by squaring the adjacency matrix, which for every pair of vertices gives

the number of 2-paths that connect them. Summing over all edges we get the number of triangles multiplied by 3.

Many authors have given efficient algorithms for *approximately* counting the number of triangles in a graph, see e.g. [16] and its references. Most of these derive an estimator by some kind of sampling followed by an exact triangle counting algorithm.

### 1.2 Our contributions

Our central contribution is a randomized algorithm that lists (with high probability) all triangles in a graph by alternating two procedures:

- **Densifying:** Eliminate vertices of low degrees by enumerating all 2-paths going through them, and
- **Sparsifiying:** Eliminate edges that are part of few triangles by reporting all such triangles using sparse signal recovery techniques.

We can derandomize the algorithm at a cost of a factor $n^{o(1)}$ in the running time by using known *explicit* constructions from the sparse signal recovery literature. Let $\omega$ denote the exponent of square matrix multiplication. In section 3 we show:

**Theorem 1** *There exists a deterministic algorithm that lists all $t$ triangles in a graph of $n$ vertices in time $\tilde{\mathcal{O}}\left(n^{\omega} + n^{3(\omega-1)/(5-\omega)}t^{2(3-\omega)/(5-\omega)}\right)$.*

With the bound $\omega < 2.373$ [27] we get a time bound of $\mathcal{O}\left(n^{2.373} + n^{1.568}\, t^{0.478}\right)$. In section 3 we also derive the following theorem:

**Theorem 2** *There exists a deterministic algorithm that lists all $t$ triangles in a graph of $m$ edges in time $\tilde{\mathcal{O}}\left(m^{2\omega/(\omega+1)} + m^{3(\omega-1)/(\omega+1)}t^{(3-\omega)/(\omega+1)}\right)$.*

Using the bound on $\omega$ as above we get: $\mathcal{O}\left(m^{1.408} + m^{1.222}\, t^{0.186}\right)$. In particular, listing $m$ triangles in a graph of $m$ edges can be done in time $\mathcal{O}\left(m^{1.408}\right)$.

We note that if $\omega = 2$ the time complexity for listing $m$ triangles reduces to $\tilde{\mathcal{O}}\left(m^{4/3}\right)$, meeting the conditional lower bound of [23] based on hardness of 3SUM. In section 6 we show that unless another seemingly difficult problem has faster algorithms, namely quadratic systems of equations (QES), our two runtime bounds are tight also for graphs with more triangles.

QES is defined as follows. Let $F$ be a finite field and $|F|$ its number of elements. A quadratic equation system over $F^l$ is a set of $k$ quadratic equations in $l$ variables over $F$. It is easy to see that QES is NP-complete, as for instance NAESAT easily reduces to it with one equation per clause already over $F = GF(2)$, and it is a polynomial time task to verify a purported solution.

QES is a well-studied problem. The assumption that QES is intractable even on average has been used to design several important cryptosystems (e.g. [17, 21]). A faster algorithm for QES would help attack these. Some algorithms that work well in practice have been designed (see e.g. [15, 7]), though in the worst

case, these do not improve over the exhaustive search $|F|^l \operatorname{poly}(l,k)$ time algorithm. It is a big open problem whether one can obtain a substantial improvement (of the form $|F|^{(1-\varepsilon)l}$ for some $\varepsilon > 0$) over exhaustive search for QES. We show that if one could improve upon our triangle listing algorithms (and $\omega = 2$), then QES does indeed have faster algorithms over any $F$.

**Theorem 3** *Suppose that for some $\epsilon_1 \geq 0$, $\epsilon_2 \geq 0$ with $\epsilon_1 + \epsilon_2 > 0$, there exists an algorithm that lists all $t$ triangles in an $m$-edge graph in $O(m^{1-\epsilon_1} t^{(1-\epsilon_2)/3})$ time or in an $n$-vertex graph in $O(n^{1-\epsilon_1} t^{(1-\epsilon_2)2/3})$ time. Then, for any finite field $F$, there exists an $|F|^{(1-\delta)l} \operatorname{poly}(l,k)$ time algorithm for $\delta > 0$ that solves $l$-variate quadratic equation systems with $k$ equations over $F^l$.*

The hardness of QES was first used as an assumption to base lower bounds on by Vassilevska and Williams [25] who showed that a fast enough algorithm for determining whether an undirected graph with edge weights from some field $F$ has a $k$-clique of total weight 0 (over $F$) would imply a faster than exhaustive search algorithm for QES. In spirit, the proof of Theorem **??** is similar to the proof in [25].

## 2 Listing light triangles

Let $\Lambda$ be a parameter. We say that an edge is *$\Lambda$-light*, or just *light*, if it participates in at most $\Lambda$ triangles, otherwise, it is said to be *$\Lambda$-heavy*. A triangle is *$\Lambda$-light* if at least one of the edges participating in it is light, otherwise it is *$\Lambda$-heavy*. In this section we describe a simple randomized algorithm for listing all $\Lambda$-light triangles with high probability. This algorithm is used as a building block by our algorithms for listing all triangles in dense and sparse graphs.

We include this simple randomized algorithm for completeness. The ideas behind it have been used before, for instance by Gasieniec et al. [9] who, building upon work of Aumann et al. [3] showed how to find $k$ witnesses for Boolean matrix multiplication in $\tilde{\mathcal{O}}\left(n^2 k + n^\omega k^{(3-\omega-\alpha)/(1-\alpha)}\right)$ time. In Section 4 we describe a novel deterministic version of the algorithm described in this section using *sparse recovery* techniques.

**Theorem 4** *Let $G = (V, E)$ be a graph on $n$ vertices and let $1 \leq \Lambda \leq n$. Then, all $\Lambda$-light triangles in $G$ can be found in $\tilde{\mathcal{O}}\left(n^\omega \Lambda^{3-\omega}\right)$ time, with high probability.*

*Proof.* We assume, without loss of generality, that $V = [n] = \{1, 2, \ldots, n\}$. Let $A$ be the adjacency matrix of the graph. Let $\bar{A}$ be the matrix $A$ in which all the 1s in the $k$-th column of $A$ are replaced by $k$, for $k \in [n]$. Let $S \subseteq V$, let $A[*, S]$ denote the matrix obtained from $A$ by selecting the columns whose indices belong to $S$. Similarly, let $A[S, *]$ denote the matrix obtained by selecting the rows of $A$ whose indices belong to $S$. The rectangular Boolean product $A[*, S]A[S, *]$ tells us, for every $i, j \in [n]$, whether there is a path of length 2 from $i$ to $j$ that passes through a vertex of $S$. If there is only one such 2-path, then the $(i, j)$-th entry of the product $\bar{A}[*, S]A[S, *]$ identifies the $k$ for which $(i, k), (k, j) \in E$.

Suppose now that $(i,j) \in E$ is a $\Lambda$-light edge, and let $T_{i,j} = \{k \in V \mid (i,k),(k,j) \in E\}$ be the set of 'mid-points' of the triangles passing through the edge $(i,j)$. Note that $|T_{i,j}| \leq \Lambda$. Let $S$ be a random subset of $V$ of size $n/\Lambda$. Let $k \in T_{i,j}$. The probability that $|S \cap T_{i,j}| = 1$ is at least $\frac{1}{\Lambda}(1-\frac{1}{\Lambda})^{\Lambda-1} \geq \frac{1}{e\Lambda}$. Thus, if we choose $\mathcal{O}(\Lambda \log n)$ random subsets of size $n/\Lambda$, we can, with high probability, identify all light triangles.

As each product $A[*,S]A[S,*]$ and $\bar{A}[*,S]A[S,*]$, where $|S| = n/\Lambda$, can be computed in $\tilde{O}(n^2(n/\Lambda)^{\omega-2})$ (by decomposing each rectangular matrix product into square matrix products), the $\mathcal{O}(\Lambda \log n)$ products could all be computed in $\tilde{O}(n^\omega \Lambda^{3-\omega})$ time. □

It is not difficult to convert the algorithm into a Las Vegas algorithm whose expected running time is $\tilde{O}(n^\omega \Lambda^{3-\omega})$. The idea is to check that each reported triangle exists, and check for each edge that the number of triangles reported is correct (by comparing to the number of 2-paths connecting its end points). As pointed out by [9], using fast rectangular matrix multiplication [11, 19], one can improve the running time of Theorem 4 to $\tilde{\mathcal{O}}\left(n^\omega \Lambda^{(3-\alpha-\omega)/(1-\alpha)} + \Lambda n^2\right) \leq \tilde{\mathcal{O}}\left(\Lambda n^2 + n^{2.373}\Lambda^{0.464}\right)$. Here $\alpha > 0.303$ is the largest constant such that $n \times n^\alpha$ by $n^\alpha \times n$ matrices can be multiplied in $\tilde{\mathcal{O}}\left(n^2\right)$ time. This implies slight improvements of the time bounds in Theorems 1 and 2.

## 3 Listing all triangles

We next describe two algorithms for listing all triangles in dense and sparse graphs that use each other as subroutines. We let `Dense`$(n,t)$ be the algorithm for listing all triangles in a graph on $n$ vertices containing at most $t$ triangles, and use $D(n,t)$ to denote the running time of `Dense`$(n,t)$. Similarly, we let `Sparse`$(m,t)$ be the algorithm for listing all triangles in a graph with $m$ edges (we assume that the graph has no isolated vertices to make $m$ a proper bound on the size of the graph) containing at most $t$ triangles, and let $S(m,t)$ denote the running time of `Sparse`$(m,t)$. We assume that these algorithms receive an upper bound $t$ on the number of triangles in the input graph. This upper bound can be computed before calling our algorithms, either in $\tilde{\mathcal{O}}(n^\omega)$ time, or in $\mathcal{O}\left(m^{2\omega/(\omega+1)}\right)$ time [2].

`Sparse`$(m,t)$ works as follows. It chooses a parameter $\Delta$ depending on $m$ and $t$. Vertices of degree at most $\Delta$ are said to be *low* degree vertices. Vertices of degree greater than $\Delta$ are said to be *high* degree. The algorithm starts by finding all triangles that contain a low degree vertex. This can be easily done in $\mathcal{O}(m\Delta)$ time by examining for every edge incident on a low degree vertex $x$, the length 2-paths formed by taking another edge out of $x$. Once this is done we can remove all edges incident to low degree vertices. If no edges remain we stop — otherwise, all remaining triangles, i.e., triangles that only include high degree vertices can now be found by a call to `Dense`$(2m/\Delta,t)$, as there are at most $2m/\Delta$ high degree vertices. Thus, ignoring constant factors,

$$S(m,t) \;\leq\; m\Delta + D(2m/\Delta,t)\;. \tag{1}$$

`Dense` $(n, t)$ works as follows. If $n < 3$, it returns no triangles. Otherwise, it chooses a parameter $\Lambda$ depending on $n$ and $t$. It then finds all $\Lambda$-light triangles in $\tilde{\mathcal{O}}\left(n^{\omega}\Lambda^{3-\omega}\right)$ time by Theorem 4 (or its deterministic version from Section 4). Once this is done we can remove all $\Lambda$-light edges. If no edges remain we stop — otherwise, as there can be at most $3t/\Lambda$ $\Lambda$-heavy edges, all $\Lambda$-heavy triangles can be found by a call to `Sparse` $(3t/\Lambda, t)$. Thus, ignoring $n^{o(1)}$ factors,

$$D(n,t) \;\leq\; n^{\omega}\Lambda^{3-\omega} + S(3t/\Lambda, t)\;. \tag{2}$$

To analyze the running times of the two algorithms we set

$$\Lambda = \lceil \max(3,\; 6n^{-(\omega+1)/(5-\omega)}t^{2/(5-\omega)}) \rceil, \text{ and}$$
$$\Delta = \lceil 2\max(m^{(\omega-1)/(\omega+1)},\; m^{2(\omega-2)/(\omega+1)}t^{(3-\omega)/(\omega+1)}) \rceil.$$

Suppose first that $t \geq m$. Notice that since we never change $t$ and the number of edges never increases over the recursive calls, if we ever have $t \geq m$, we have $t \geq m$ in all subsequent calls. We get

$$\Delta = 2m^{2(\omega-2)/(\omega+1)}t^{(3-\omega)/(\omega+1)},\; m\Delta = 2m^{3(\omega-1)/(\omega+1)}t^{(3-\omega)/(\omega+1)}.$$

Consider the first recursive call to the dense algorithm, and suppose that it was called on $n$ nodes, where we know that $n \leq 2m/\Delta$. We get the following:

$$n \leq 2m/\Delta = m^{(5-\omega)/(\omega+1)}t^{-(3-\omega)/(\omega+1)}.$$

$$n^{(\omega+1)/2} \leq (2m/\Delta)^{(\omega+1)/2} = \left(\frac{m^{(5-\omega)/(\omega+1)}}{t^{(3-\omega)/(\omega+1)}}\right)^{(\omega+1)/2} = t\cdot(m/t)^{(5-\omega)/2}.$$

Thus, $t/n^{(\omega+1)/2} \geq (t/m)^{(5-\omega)/2}$ which is $\geq 1$ when $t \geq m$, and since $\Lambda = \max\{3, 6(t/n^{(\omega+1)/2})^{2/(5-\omega)}\}$, we get that

$$\Lambda = 6(t/n^{(\omega+1)/2})^{2/(5-\omega)} \geq 6t/m.$$

We now get:

$$\begin{aligned} n^{\omega}\Lambda^{3-\omega} &= 6^{3-\omega}n^{3(\omega-1)/(5-\omega)}t^{2(3-\omega)/(5-\omega)} \\ &\leq 6^{3-\omega}(2m/\Delta)^{3(\omega-1)/(5-\omega)}t^{2(3-\omega)/(5-\omega)} \\ &= 6^{3-\omega}m^{3(\omega-1)/(\omega+1)}t^{(3-\omega)/(\omega+1)}. \end{aligned} \tag{3}$$

Since $\Lambda \geq 6t/m$, we get that $3t/\Lambda \leq m/2$, and hence $S(3t/\Lambda, t) \leq S(m/2, t)$. By Eq. 1 and Eq. 2 we have

$$\begin{aligned} S(m,t) \;&\leq\; m\Delta + n^{\omega}\Lambda^{3-\omega} + S(3t/\Lambda, t) \\ &\leq\; (2+6^{3-\omega})m^{3(\omega-1)/(\omega+1)}t^{(3-\omega)/(\omega+1)} + S(m/2,t) \\ &\leq\; \sum_{i=1}^{\lceil \log m \rceil} (2+6^{3-\omega})(m/2^i)^{3(\omega-1)/(\omega+1)}t^{(3-\omega)/(\omega+1)} \\ &\in \mathcal{O}\left(m^{3(\omega-1)/(\omega+1)}t^{(3-\omega)/(\omega+1)}\right). \end{aligned}$$

Next assume $t < m$. We get $\Delta = 2m^{(\omega-1)/(\omega+1)}$. By the above analysis we get that $\Lambda \leq 6(t/m)^{(5-\omega)/2}$, and so when $t < m$, we have $3 \leq \Lambda \leq 6$. We also have $n \leq 2m/\Delta = m^{2/(\omega+1)}$. By Eq. 1 and Eq. 2 we have

$$\begin{aligned} S(m,t) &\leq m\Delta + n^{\omega}\Lambda^{3-\omega} + S(3t/\Lambda, t) \\ &\leq (2+6^{3-\omega})m^{2\omega/(\omega+1)} + S(t,t) \\ &\leq (2+6^{3-\omega})m^{2\omega/(\omega+1)} + \mathcal{O}\left(t^{2\omega/(\omega+1)}\right) \\ &\in \mathcal{O}\left(m^{2\omega/(\omega+1)}\right). \end{aligned}$$

Once we have established the complexity of $\texttt{Sparse}\,(m,t)$, it is also easy to establish the complexity of $\texttt{Dense}\,(n,t)$. There are again two cases. If $t \leq n^{(\omega+1)/2}$, then $3 \leq \Lambda \leq 6$ and the running time is

$$D(n,t) \;=\; \mathcal{O}\left(n^{\omega} + S(t,t)\right) = \mathcal{O}\left(n^{\omega}\right) \;.$$

If $t > n^{(\omega+1)/2}$, then $\Lambda = 6n^{-(\omega+1)/(5-\omega)}t^{2/(5-\omega)}$ and then

$$\begin{aligned} D(n,t) &\leq n^{3(\omega-1)/(5-\omega)}t^{2(3-\omega)/(5-\omega)} + S(3t/\Lambda, t) \\ &\leq (n^{3(\omega-1)}t^{2(3-\omega)})^{1/(5-\omega)} + S((t^{(3-\omega)}n^{(\omega+1)})^{1/(5-\omega)}, t) \\ &\leq (n^{3(\omega-1)}t^{2(3-\omega)})^{1/(5-\omega)} + (t^{3-\omega}n^{\omega+1})^{3(\omega-1)/((\omega+1)(5-\omega))}t^{(3-\omega)/(\omega+1)} \\ &= \mathcal{O}\left(n^{3(\omega-1)/(5-\omega)}t^{2(3-\omega)/(5-\omega)}\right) \;, \end{aligned}$$

as required.

## 4 Deterministic algorithm

Randomization was only used by the algorithm for listing light triangles.

We now proceed to show how to list all $\Lambda$-light triangles. This is achieved by computing, for every light edge, the list of at most $\Lambda$ 2-paths connecting its vertices. Each such list can be thought of as a vector $x \in \{0,1\}^n$ with at most $\Lambda$ 1s, corresponding to the connecting nodes. Let $P_\Lambda$ denote the set of such vectors that we would like to compute.

To this end we make use of a *sparse recovery* matrix $T$ with the following properties, for some function $f(n) = n^{o(1)}$:

- $T$ has $d = \Theta(\Lambda f(n))$ rows.
- The number of non-zero entries in $T$ is at most $nf(n)$.
- For every $x \in P_\Lambda$, we can compute $x$ from $Tx$ in time $\mathcal{O}\left(\Lambda f(n)\right)$.

Random sparse 0-1 matrices are known to have these properties with high probability for $f(n) = (\log n)^{\mathcal{O}(1)}$ (see e.g. [22] for an overview of such constructions), and there also exist explicit, deterministic constructions with $f(n) = n^{o(1)}$ [12]. Let $D_i$ denote the *diagonal* matrix where the $j$th entry along the diagonal is equal to $T_{i,j}$.

Let $A$ denote the adjacency matrix of the graph. To find all light triangles we compute, for $i = 1, \ldots, d$, the matrix product $AD_iA$. If $D_i$ has $n_i$ non-zero entries, this reduces to an $n$-by-$n_i$ times $n_i$-by-$n$ matrix product. For every $x \in P_\Lambda$ this gives us the vector $Tx$. Specifically, if $x$ is the set of vertices connecting vertices $a$ and $b$, $(Tx)_i = (AD_iA)_{a,b}$. This means that we can recover each $x \in P_\Lambda$ in time $\mathcal{O}(\Lambda f(n))$.

The matrix product $AD_iA$ can be decomposed into $\mathcal{O}\left((n/n_i)^2\right)$ square matrix products, each taking time $\mathcal{O}(n_i^\omega)$. By choice of $T$ we have $\sum_i n_i \leq df(n)$. So the time for computing all $d$ matrix products is bounded by a constant times

$$\sum_{i=1}^{d} n^2 n_i^{\omega-2} \leq dn^2 \left(\sum_{i=1}^{d} n_i/d\right)^{\omega-2} \leq dn^2(nf(n)/d)^{\omega-2} = n^\omega \Lambda^{3-\omega} f(n)^{\omega-1} \ .$$

The first inequality uses Jensen's inequality and the fact that $n_i^{\omega-2}$ is concave (since $\omega - 2 \in [0; 1]$). The second inequality uses our bounds on $d$ and $\sum_i n_i$. Similarly to our randomized algorithm, the deterministic algorithm can be improved to have runtime $\tilde{\mathcal{O}}\left(n^\omega \Lambda^{(3-\alpha-\omega)/(1-\alpha)} + \Lambda n^2\right)$ using rectangular matrix multiplication.

## 5 Listing some triangles

If a graph contains $T$ triangles and we are only required to list $t$ of them, then an improved running time can be obtained as follows. First assume that the given graph is tripartite by creating three copies of each vertex $v$, $v_I$ in partition $I$, $v_J$ in partition $J$ and $v_K$ in partition $K$. Then each edge $(u, v)$ appears 6 times, once for each pair of copies of $u$ and $v$ in different partitions. Each triangle appears 6 times as well, so it suffices to list $6t$ triangles in this new graph. Suppose now that we want to list $t$ triangles in a tripartite graph with $T > t$ triangles. We design a recursive algorithm as follows. Split $I, J, K$ into 2 parts of $n/2$ nodes each, $I_1, I_2, J_1, J_2, K_1, K_2$. Count the triangles in each of the 8 subgraphs induced by $I_i \cup J_j \cup K_k$, and recurse on the part that has the most triangles. At some point, the number of triangles in the part $I_i \cup J_j \cup K_k$ with most triangles will be $< t$, and at this point we no longer recurse, but use our triangle listing algorithm on the current subgraph $G'$. We know that when we recursed on $G'$, it had at least $t$ triangles, but since each of the 8 triples of subgraphs of $G'$ have $< t$, then $G'$ has $< 8t$ triangles. Consider now the number of nodes of $G'$. Suppose that it is $3n/2^j$ for some $j$, and we have done $j$ recursive steps to find $G'$. In each step the number of triangles goes down by at most a factor of 8, so $G'$ has at least $T/8^j$ triangles. Yet, $G'$ has $< 8t$ triangles, and hence $8^{j+1} > T/t$, and hence the number of nodes in $G'$ is $O(n/(T/t)^{1/3})$. We thus get a running time of

$$\tilde{\mathcal{O}}\left(n^\omega + \left(\left(\frac{t}{T}\right)^{1/3} n\right)^{3(\omega-1)/(5-\omega)} t^{2(3-\omega)/(5-\omega)}\right).$$

Using a similar idea, combined with an approach from [14], we can also get an improvement for sparse graphs (in terms of $m$).

## 6 Consequences of faster triangle listing

In this section we prove Theorem 3. We show that if one could improve upon our triangle listing algorithms (and $\omega = 2$), then QES does indeed have faster algorithms over any $F$.

Let $F$ be a finite field and $q = |F|$ its number of elements. Assume that there is no $q^{(1-\epsilon_3)l}\operatorname{poly}(l,k)$ time algorithm for any $\epsilon_3 > 0$ that solves $l$-variate QES on $k$ equations. Given an instance to QES on $l$ variables with $k$ equations $x'Q_ix + E_ix + S_i = 0$ over $F$, where $Q_i$ are $l \times l$ matrices, $E_i$ are $1 \times l$ vectors, and $S_i$ are scalars, we will show how one can use triangle listing to solve it. Much as Pătraşcu [23] did for 3SUM, we use hashing as a filter to find the solutions to QES. We construct $h$ hashed projections of the equations $x'A_ix + B_ix + C_i = 0$ for $i = 1, 2, \ldots, h$ where

$$A_i = \sum_{j=1}^{k} R_i(j)Q_j, \qquad B_i = \sum_{j=1}^{k} R_i(j)E_j, \qquad C_i = \sum_{j=1}^{k} R_i(j)S_j$$

for a random $R_i \in F^k$ (for a vector $R$ we write $R(j)$ to address its $j$th element). The hashed QES $(A, B, C)$ has the following relations to the original QES:

- Every solution to $(Q, E, S)$ is a solution also to $(A, B, C)$.
- Every non-solution to $(Q, E, S)$ is a solution to $(A, B, C)$ with prob. $q^{-h}$.

This means that if $(Q, E, S)$ has $s$ solutions, $(A, B, C)$ has at most $2 \cdot q^{l-h} + s$ solutions with probability at least $1/2$ by the linearity of expectation and Markov's inequality. We can assume that $s < q^{\epsilon_3 l}$ since if not we can use another algorithm in parallel that simply guesses an assignment and verifies it, which runs in expected time $O(q^l/s)$.

We next construct a graph $G$ that has a triangle for each solution to $(A, B, C)$. Let $a$ be a parameter to be fixed later. The vertex set is the union of three sets:

- $V_1$ has one vertex labeled $(\phi_1)$ for each assignment $\phi_1$ to the first $l - 2a$ variables, in total $q^{l-2a}$ vertices.
- $V_2$ has one vertex labeled $(\phi_2, H_2)$ for each combination of an assignment $\phi_2$ to the next $a$ variables $x_{l-2a+1}, \ldots, x_{l-a}$ and a vector $H_2$ in $F^h$, in total $q^{a+h}$ vertices.
- $V_3$ has one vertex labeled $(\phi_3, H_3)$ for each combination of an assignment $\phi_3$ to the last $a$ variables $x_{l-a+1}, \ldots, x_l$ and a vector $H_3$ in $F^h$, in total $q^{a+h}$ vertices.

We let $0_k$ denote the assignment of $k$ variables to the value 0. The edges are:

- $(\phi_1)$ and $(\phi_2, H_2)$ has an edge iff the assignments $x = \phi_1\phi_2 0_{l-a}$ and $y = \phi_1 0_{2a}$ to the variables give $x'A_ix + B_iy + C_i = -H_2(i)$, i.e. we consider the contribution where we use all quadratic terms associated with the vertices and the linear term associated with the first one. There are $q^{l-a}$ edges.

– $(\phi_2, H_2)$ and $(\phi_3, H_3)$ has an edge iff the assignments $x = 0_{l-2a}\phi_2\phi_3$ and $y = 0_{n-2a}\phi_2 0_a$ to the variables give $x'A_ix + B_iy + C_i = H_2(i) - H_3(i)$. There are $q^{2a+h}$ edges.
– $(\phi_3, H_3)$ and $(\phi_1)$ has an edge iff the assignments $x = \phi_1 0_a \phi_3$ and $y = 0_{n-a}\phi_3$ to the variables give $x'A_ix + B_iy + C_i = H_3(i)$. There are $q^{2a+h}$ edges.

A triangle in the graph corresponds to a solution to $(A, B, C)$ since on the left side we count each term exactly once, and on the right hand side $H_2$ and $H_3$ are counted twice with opposite signs and cancel.

We can use our triangle listing algorithm on $G$ to solve $(Q, E, S)$: for each found triangle $(\phi_1), (\phi_2, H_2), (\phi_3, H_3)$ we verify if $x = \phi_1\phi_2\phi_3$ is also a solution to $(Q, E, S)$. To arrive at the lower bound, we note that the graph $G$ has

– $q^{l-2a} + 2 \cdot q^{a+h}$ vertices.
– $q^{l-a} + 2 \cdot q^{2a+h}$ edges.
– $2 \cdot q^{l-h} + s < 2 \cdot q^{l-h} + q^{\epsilon_3 l} < 2 \cdot q^{l-h+\epsilon_3 l}$ triangles with probability 1/2.

We set $a = (l-h)/3$ to get $m = 3 \cdot q^{2l/3+h/3}$and $n = 3q^{l/3+2h/3}$. By varying $h$ we can control the number of triangles w.r.t. $m$ and $n$.

Now assume there is a $O(m^{1-\epsilon_1}t^{(1-\epsilon_2)/3})$ time algorithm for triangle listing for some $t \geq m$. With our bounds on $m$ and $t$ we get

$$\mathcal{O}\left(q^{l-\epsilon_1(2l/3+h/3)-\epsilon_2(l/3-h/3+\epsilon_3 l)+\epsilon_3 l}\right)$$

time. For small enough constant $\epsilon_3$ we get a contradiction of the assumption of non-existence of any $O(q^{(1-\epsilon_3)l})$ time algorithm for QES. If we instead assume a $n^{1-\epsilon_1}t^{(1-\epsilon_2)2/3}$ time algorithm for triangle listing for some $t$, we get $\mathcal{O}\left(q^{l-\epsilon_1(l/3+2h/3)-\epsilon_2(2l/3-2h/3+\epsilon_3 l)+\epsilon_3 l}\right)$ time, also a contradiction.

**Acknowledgement.** We send our thanks to Mihai Pătraşcu with whom we first discussed the ideas leading to this paper. We thank Eric Price and Jelani Nelson for information on sparse recovery methods.

## References

1. Foto N. Afrati, Dimitris Fotakis, and Jeffrey D. Ullman. Enumerating subgraph instances using Map-Reduce. *Proc. IEEE International Conference on Data Engineering (ICDE)*, 0:62–73, 2013.
2. Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.
3. Yonatan Aumann, Moshe Lewenstein, Noa Lewenstein, and Dekel Tsur. Finding witnesses by peeling. *ACM Transactions on Algorithms*, 7(2):24, 2011.
4. Jonathan Berry, Luke Fostvedt, Daniel Nordman, Cynthia Phillips, C. Seshadhri, and Alyson Wilson. Why do simple algorithms for triangle enumeration work in the real world? In *Proc. Innovations in Theoretical Computer Science*, 2014.
5. Jonathan W. Berry, Bruce Hendrickson, Randall A. LaViolette, and Cynthia A. Phillips. Tolerating the community detection resolution limit with edge weighting. *Phys. Rev. E*, 83:056119, May 2011.

6. Shumo Chu and James Cheng. Triangle listing in massive networks. *ACM Trans. Knowl. Discov. Data*, 6(4):17:1–17:32, 2012.
7. N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Proc. EUROCRYPT*, pages 392–407, 2000.
8. Brooke Foucault Welles, Anne Van Devender, and Noshir Contractor. Is a friend a friend? Investigating the structure of friendship networks in virtual worlds. In *CHI'10 Extended Abstracts on Human Factors in Computing Systems*, pages 4027–4032. ACM, 2010.
9. Leszek Gasieniec, Miroslaw Kowaluk, and Andrzej Lingas. Faster multi-witnesses for boolean matrix multiplication. *Inf. Process. Lett.*, 109(4):242–247, 2009.
10. Xiaocheng Hu, Yufei Tao, and Chin-Wan Chung. Massive graph triangulation. In *Proc. of SIGMOD*, pages 325–336. ACM, 2013.
11. X. Huang and V. Y. Pan. Fast rectangular matrix multiplication and applications. *J. of Complexity*, 14(2):257–299, 1998.
12. Piotr Indyk. Explicit constructions for compressed sensing of sparse signals. In *Proc. of 19th SODA*, pages 30–33, 2008.
13. Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978.
14. Zahra Jafargholi and Emanuele Viola. 3SUM, 3XOR, triangles. *CoRR*, abs/1305.3827, 2013.
15. A. Kipnis and A. Shamir. Cryptanalysis of the HFE public key cryptosystem by relinearization. In *Proc. CRYPTO*, volume 1666, pages 19–30, 1999.
16. Mihail N. Kolountzakis, Gary L. Miller, Richard Peng, and Charalampos E. Tsourakakis. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Mathematics*, 8(1-2):161–185, 2012.
17. S. Landau. Polynomials in the nation's service: using algebra to design the advanced encryption standard. *American Mathematical Monthly*, 111:89–117, 2004.
18. F. Le Gall. Powers of tensors and fast matrix multiplication. *CoRR*, abs:1401.7714, 2014.
19. Francois Le Gall. Faster algorithms for rectangular matrix multiplication. In *Proc. Foundations of Computer Science*, pages 514–523, 2012.
20. Rasmus Pagh and Francesco Silvestri. The input/output complexity of triangle enumeration. *arXiv preprint arXiv:1312.0723*, 2013.
21. J. Patarin. Cryptoanalysis of the matsumoto and imai public key scheme of eurocrypt'88. In *Proc. CRYPTO*, pages 248–261, 1995.
22. Ely Porat and Martin J. Strauss. Sublinear time, measurement-optimal, sparse recovery for all. In *Proc. of 23rd SODA*, pages 1215–1227. SIAM, 2012.
23. Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Proc. of 42nd STOC*, pages 603–610, 2010.
24. Thomas Schank and Dorothea Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In *Experimental and Efficient Algorithms*, pages 606–609. Springer, 2005.
25. V. Vassilevska and R. Williams. Finding, minimizing, and counting weighted subgraphs. In *Proc. STOC*, pages 455–464, 2009.
26. Nan Wang, Jingbo Zhang, Kian-Lee Tan, and Anthony KH Tung. On triangulation-based dense neighborhood graph discovery. *Proc. VLDB Endowment*, 4(2):58–68, 2010.
27. Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proc. of 44nd STOC*, pages 887–898, 2012.

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