


# Listing 4-Cycles

Amir Abboud  

Weizmann Institute of Science, Rehovot, Israel

Seri Khoury 

UC Berkeley, CA, USA

Oree Leibowitz 

Weizmann Institute of Science, Rehovot, Israel

Ron Safier 

Weizmann Institute of Science, Rehovot, Israel

---

## Abstract

We study the fine-grained complexity of listing all 4-cycles in a graph on  $n$  nodes,  $m$  edges, and  $t$  such 4-cycles. The main result is an  $\tilde{O}(\min(n^2, m^{4/3}) + t)$  upper bound, which is best-possible up to log factors unless the long-standing  $O(\min(n^2, m^{4/3}))$  upper bound for *detecting* a 4-cycle can be broken. Moreover, it almost-matches recent 3-SUM-based lower bounds for the problem by Abboud, Bringmann, and Fischer (STOC 2023) and independently by Jin and Xu (STOC 2023). Notably, our result separates 4-cycle listing from the closely related *triangle* listing for which higher conditional lower bounds exist, and rule out such a “detection plus  $t$ ” bound. We also show by simple arguments that our bound cannot be extended to mild generalizations of the problem such as reporting all pairs of nodes that participate in a 4-cycle.

**Independent work:** Jin and Xu [26] also present an algorithm with the same time bound.

**2012 ACM Subject Classification** Theory of computation → Problems, reductions and completeness

**Keywords and phrases** Graph algorithms, cycles listing, subgraph detection, fine-grained complexity

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2023.25

**Funding** *Amir Abboud:* This project has received funding from the European Research Council (ERC) under the European Union’s Horizon Europe research and innovation programme (grant agreement No 101078482). Additionally, Amir Abboud is supported by an Alon scholarship and a research grant from the Center for New Scientists at the Weizmann Institute of Science.

## 1 Introduction

Finding small patterns in large graphs is a classical task. In the *detection* formulation we only want to decide if a small pattern  $H$  exists in a large graph  $G$  (as a not-necessarily induced subgraph); this is known as the *Subgraph Isomorphism* problem and is one of the most extensively studied problems in graph algorithms. Its time complexity has been investigated from various angles, including classical algorithms (e.g. [34, 22, 4, 28, 39]), parameterized complexity (e.g. [19, 32]), streaming algorithms (e.g. [24, 23]), distributed computing (e.g. [14, 13]), quantum algorithms (e.g. [31, 30]) and so on.<sup>1</sup>

In the *listing* formulation we are asked to return all occurrences of the pattern in the graph as efficiently as possible. This problem has received much attention both from the classic graph algorithms literature (e.g. [27, 8, 15, 9]) and also from the large literature on *enumerating answers to queries* in databases (where they focus on the closely related problem of listing all patterns with *small delay* between consecutive outputs), see [18, 37, 20, 10, 11, 12].

---

<sup>1</sup> An extensive list of citations for each of these topics is infeasible.



Our interest is in the *fine-grained complexity* of such problems and we therefore focus on the simplest patterns without near-linear time algorithms: triangles and 4-cycles. Besides the theoretical interest, the enumeration of short cycles has received much attention due to applications, see e.g. [6, 7, 21, 38].

Throughout the paper, we assume that graphs are unweighted, undirected, and have  $n$  nodes and  $m$  edges unless stated otherwise. The classical *Johnson's Algorithm* [27] lists all cycles in a graph in time that is polynomial in  $n$  and in their number  $t$ . Four decades later, its time complexity was improved significantly so that it runs in the optimal  $O(m + t)$  time [8]. One may hope that such an efficient algorithm can list all  $k$ -cycles for a *fixed*  $k$  such as  $k = 3$  or  $k = 4$  in time  $O(m + t)$  (here  $t$  is the number of  $k$ -cycles). Unfortunately, this is not known and is impossible under popular conjectures as we discuss below.

A simple observation (discussed in Section 3.1) shows that we cannot hope for an  $O(T(n, m) + t)$  upper bound for listing unless we can solve the detection problem in time  $O(T(n, m))$ . Moreover, note that the  $+t$  term is necessary because typically the number of occurrences of a pattern could be much larger than  $T(n, m)$ . Thus, our starting point is the state-of-the-art for *detecting* triangles and 4-cycles. The best known algorithms for *detecting* if a graph has at least one pattern take  $O(\min(n^\omega, m^{2\omega/(\omega+1)}))$  for triangle [4], where  $\omega < 2.37188$  is the fast matrix multiplication exponent [3, 17], and  $O(\min(n^2, m^{4/3}))$  for 4-cycle [4]. Note that if  $\omega = 2$  then the two bounds are the same. Thus, the main question is whether we can solve the listing problem with a time complexity that matches these bounds, plus  $t$ . In particular, can we list all 4-cycles in  $O(\min(n^2, m^{4/3}) + t)$  time?

Such a result is unlikely for triangle listing. A reduction by Kopelowitz, Pettie, and Porat [29] that optimizes a construction by Pătraşcu [35] shows that an  $O(n^\omega + t)$  bound or any  $O(\min(n^{3-\varepsilon}, m^{3/2-\varepsilon}) + t)$  algorithm for any  $\varepsilon > 0$  would refute the 3-SUM Conjecture<sup>2</sup>.

The actual state-of-the-art for triangle listing is more complicated. An exhaustive search lists all triangles in  $O(n^3)$  time. An algorithm of time  $O(m^{3/2})$  can be achieved by a simple application of low-degree/high-degree separation technique (see for example [4]). A work by Chiba and Nishizeki [15] shows how to list all triangles (and all 4-cycles) in  $O(ma(G))$  time where  $a(G)$  is the arboricity<sup>3</sup> of the graph  $G$ . This running time is essentially the same as  $O(m^{3/2})$  because  $a(G) \leq m^{1/2}$ , but it shows an improvement for graphs with low arboricity (such as planar graphs). These algorithms are essentially optimal because a graph may contain  $\Theta(\min(n^3, m^{3/2}))$  triangles, but better results can be achieved if we also take the number of triangles into account. Then, denoting the number of triangles in the graph by  $t$ , a more clever algorithm by Björklund, Pagh, Vassilevska Williams, and Zwick [9] solves triangle listing in time  $\tilde{O}(n^\omega + n^{\frac{3(\omega-1)}{5-\omega}} t^{\frac{2(3-\omega)}{5-\omega}})$  and  $\tilde{O}(m^{\frac{2\omega}{\omega+1}} + m^{\frac{3(\omega-1)}{\omega+1}} t^{\frac{3-\omega}{\omega+1}})$ . Assuming  $\omega = 2$  the running times simplify to  $\tilde{O}(n^2 + nt^{2/3})$  and  $\tilde{O}(m^{4/3} + mt^{1/3})$ . A matching tight (assuming  $\omega = 2$ ) lower-bound under the 3-SUM Conjecture exists [35, 29] and also, more recently, under the APSP Conjecture<sup>4</sup> [41].

The hardness of achieving “detection plus  $t$ ” time for triangle listing may suggest that we cannot achieve the desired  $O(\min(n^2, m^{4/3}) + t)$  time for 4-cycle listing. Indeed, in Section 3 we present some simple observations showing that such a bound is conditionally impossible

<sup>2</sup> The 3-SUM problem is as follows: Given a set  $A \subseteq \{-n^3, \dots, n^3\}$  of  $n$  elements, decide if there are three distinct elements  $a, b, c \in A$  such that  $a + b + c = 0$ . The 3-SUM Conjecture is the conjecture that solving the 3-SUM problem requires  $n^{2-o(1)}$  time.

<sup>3</sup> The arboricity of a graph is the minimum number of edge-disjoint spanning forests into which the graph can be decomposed.

<sup>4</sup> All-Pairs Shortest Paths (APSP): Given an  $n$ -node edge-weighted directed graph with positive weights in  $\{1, \dots, n^{o(1)}\}$ , compute for all pairs of nodes the shortest distance between them. The APSP Conjecture is the conjecture that solving All-Pairs Shortest Paths requires  $n^{3-o(1)}$  time.

for mild generalizations of the problem such as in directed graphs, or if the task is to count the number of 4-cycles rather than list them (without a  $+t$  term), or if our task is to report which pairs of nodes participate in 4-cycles. Fortunately, none of these reductions give a lower bound in the basic setting of the problem.

Our main result is a positive resolution to the above question, achieving the best-possible “detection plus  $t$ ” bound for 4-cycle listing. Before our work, only the trivial  $O(n^3 + t)$  bound was known.

► **Theorem 1.** *4-cycle listing can be solved in  $O(\min(n^2 + t, (m^{4/3} + t) \cdot \log^2 n))$  time.*

Notably, this running time is faster than the aforementioned lower bound for triangle listing stating that there is no  $O(\min(n^{3-\varepsilon}, m^{3/2-\varepsilon}) + t)$  algorithm for triangle listing for any  $\varepsilon > 0$  (assuming 3-SUM) and therefore separates the two problems.

Any improvement on our upper bound would break the longstanding upper bound for 4-cycle *detection*, as discussed in Section 3.1. Moreover, a recent “short cycle removal” technique in fine-grained complexity introduced by Abboud, Bringmann, Khoury, and Zamir [2], can prove conditional lower bounds for 4-cycle listing that holds even if 4-cycle detection turns out to be easier. In particular, two independent papers [1, 26] use this technique to prove a  $(\min(n^2, m^{4/3}) + t)^{1-o(1)}$  lower bound under the 3-SUM Conjecture, showing that our algorithm is tight.<sup>5</sup> One interesting aspect of our result (mentioned in [1]) is that it proves, in some sense, the optimality of the lower-bound machinery in these recent works.

The main technical ingredient underlying our results, which distinguish 4-cycle from triangle and from the variants discussed in Section 3 is a (well-known) *supersaturation* result stating that if the number of edges in a graph is larger than a certain threshold, then the graph must have a large number of 4-cycles.

**Independent work:** Jin and Xu [26] independently obtained an algorithm for 4-cycle listing of the same running time (they have fewer log factors) by a similar approach, but some lemmas are proved differently.<sup>6</sup>

## 1.1 Outline of the Paper

We organize this paper as follows. In Section 2 we present and analyze an efficient algorithm for 4-cycle listing. In Section 3 we discuss some hardness results for mild generalizations of the problem. Lastly, in Section 4 we discuss two related open problems which we find interesting. The first open problem is regarding the generalization of our results for longer even cycles. The second open problem is determining which detection problem is harder: triangle or 4-cycle detection.

## 2 Upper Bounds for 4-Cycle Listing

In Section 2.1, we start with a simple extension of the folklore  $O(n^2)$ -time algorithm for 4-cycle detection [36] to an  $O(n^2 + t)$ -time algorithm for 4-cycles listing, where  $t$  is their number.

<sup>5</sup> These lower bounds are for *enumeration* with constant *delay*; our upper bounds also apply in that formulation of the listing problem by standard techniques. We choose to present our results for listing rather than enumeration (which would have made the theorems slightly stronger) because it makes our analysis more intuitive.

<sup>6</sup> The two papers appeared simultaneously on ArXiv, but their results were included in a paper with more results that were published in STOC 2023; their main results were lower bounds.

Then, in Section 2.2, we present our main result, which is an  $\tilde{O}(m^{4/3} + t)$ -time algorithm for 4-cycles listing.

## 2.1 Warm-up: An $O(n^2 + t)$ -time Algorithm

The simple  $O(n^2)$ -time detection algorithm can be described as follows. We enumerate all 2-paths  $(u, x, v)$  in the graph until we notice that some pair  $(u, v)$  has appeared as the endpoints of more than a single 2-path; when that happens, we stop and report the existence of a 4-cycle (since two 2-paths sharing the same endpoints form a 4-cycle). While enumerating the 2-paths, the algorithm marks in an  $n \times n$  table all the pairs  $(u, v)$  that have already been seen as endpoints of 2-paths, and therefore it knows when to stop. Importantly, the algorithm never enumerates more than  $n^2 + 1$  2-paths before stopping, giving the desired upper bound on the running time.

Another important point implicit in the above description is that all 2-paths can be enumerated efficiently, in linear time in  $m$  and the number of 2-paths listed so far: For each node  $a$ , for each neighbor  $b \in N(a)$ , for each neighbor  $c \in N(b)$  enumerate the 2-path  $(a, b, c)$ .

The  $O(n^2 + t)$  listing algorithm does exactly the same thing, except it does not stop after finding the first 4-cycle but continues enumerating all the 2-paths of the graph. (Additionally, it records all 2-paths in the entry of the table corresponding to a pair of nodes  $(u, v)$  so that all 4-cycles involving  $(u, v)$  can be listed.) The following observations analyze the running time.

We start with a worst-case bound on the number of 2-paths in a graph, as a function of the number of 4-cycles.

► **Observation 2.** *Given a graph  $G$ , the number of 2-paths in  $G$  is at most  $n^2 + 4t$  where  $t$  is the number of 4-cycles in  $G$ .*

**Proof.** Let  $p$  be the total number of 2-paths in the graph and, for a pair of nodes  $u, v$ , let  $p_{u,v}$  be the number of 2-paths between  $u$  and  $v$  and notice that

$$4t = 2 \sum_{u,v \in V} \binom{p_{u,v}}{2} \geq \sum_{\substack{u,v \in V \\ p_{u,v} > 1}} p_{u,v} = p - \sum_{\substack{u,v \in V \\ p_{u,v} = 1}} p_{u,v} \geq p - \binom{n}{2} \geq p - n^2. \quad \blacktriangleleft$$

This leads to the  $O(n^2 + t)$  bound on the simple algorithm above that lists all 2-paths in the graph, as formalized in the following observation.

► **Observation 3.** *Given a graph  $G$ , there is an  $O(n^2 + t)$ -time algorithm that lists all the 4-cycles, where  $t$  is their number.*

**Proof.** Recall the algorithm above that lists all 2-paths  $(u, x, v)$  and indexes them by their endpoints  $(u, v)$ , and then lists all 4-cycles formed by two 2-paths sharing the same pair  $(u, v)$  as endpoints. First, note that the algorithm lists each 4-cycle  $(u, x, v, y)$  exactly twice (once for the pair  $(u, v)$  and once for the pair  $(x, y)$ ). The time complexity is thus upper bounded by a constant factor times the number of 2-paths, plus the number of 4-cycles  $t$ , plus the number of pairs  $n^2$ . By the upper bound on the number of 2-paths from Observation 2, we get the  $O(n^2 + t)$  bound.  $\blacktriangleleft$

## 2.2 An $\tilde{O}(m^{4/3} + t)$ -time Algorithm

In order to improve the  $O(n^2 + t)$ -time algorithm, we can't afford listing all 2-paths. For instance, in a star graph, there are  $O(n^2)$  2-paths, but no 4-cycles. Hence, intuitively speaking, we need to narrow our attention to a certain type of 2-paths that are useful for

listing 4-cycles efficiently. Indeed, to overcome the star example, it suffices to note that there is no point in listing 2-paths with endpoints of degree one (leaves), as these 2-paths can't be extended to 4-cycles.

To extend this intuition, perhaps one could try to split the nodes into low-degree and high-degree groups, denoted by  $L$  and  $H$ , respectively, and consider different types of 2-paths with respect to this partitioning. The advantage of such partitioning is that we can narrow our attention to specific types of 2-paths that are more challenging for listing (i.e., types of 2-paths that could be too expensive to list). For instance, one can immediately spot two types of 2-paths that can be listed efficiently: 2-paths with a low-degree node at the center, and 2-paths that use only high-degree nodes. Indeed, to list all the 2-paths with a low-degree node at the center, we can go over all the edges  $e$  incident to a low-degree node, and list the corresponding 2-paths with a low-degree node at the center that  $e$  is part of. This takes  $O(m\Delta)$  time, where  $\Delta$  is the degree threshold. To list all the 2-paths that use only high-degree nodes  $H$ , we can use the  $O(|H|^2 + t) = O((m/\Delta)^2 + t)$ -time algorithm from Observation 3. By picking  $\Delta = m^{1/3}$ , listing these two types of 2-paths takes  $O(m^{4/3} + t)$  time (as shown in Lemma 8). Furthermore, listing these two types of 2-paths suffices for listing all types of 4-cycles, except the 4-cycles that use two overlapping 2-paths of the form  $LHH$  (2-paths with a high degree node at the center, one low degree endpoint, and one high degree endpoint). That is, these 4-cycles are of the form  $LHHL$ . To list these 4-cycles, we need to find a way to list 2-paths of the form  $LHH$ .

Unfortunately, we can't afford listing all 2-paths of the form  $LHH$ . For instance, take a graph where there is a node  $u$  that is connected to  $\Theta(n)$  leaves (low degree nodes) and to  $\Theta(n^{2/3-\varepsilon})$  high-degree nodes, where each of these high-degree nodes is connected to  $\Theta(n^{1/3+\varepsilon})$  leaves. In this example, we have  $m = O(n)$  edges,  $m^{5/3-\varepsilon} \gg m^{4/3}$  2-paths of the form  $LHH$  (the ones that go through  $u$ ), but no 4-cycles.

To overcome such examples, recall that the only remaining type of 4-cycles that we need to list are the ones of the form  $LHHL$ . Since we know how to list 2-paths of the form  $HLL$  (2-paths with a low-degree node at the center) efficiently, it suffices to list *only one of the two*  $LHH$  2-paths that such a 4-cycle consists of. Hence, for the 4-cycles of the form  $LHHL$ , one could wonder: is there a property that one of the two overlapping  $LHH$  paths (that the 4-cycle consists of) must have, that would make it easier to list such 4-cycles?

Indeed, given a 4-cycle of the form  $LHHL$ , consider the two middle high-degree nodes. One of them must have a degree greater or equal to the other. Therefore, it suffices to list 2-paths of the form  $LHH$ , where the degree of the middle node is at most the degree of the third node (the high-degree endpoint). We refer to such 2-paths as  $L \rightarrow H \rightarrow H$  (the orientation from a node  $u$  to a node  $v$  here means that  $u$ 's degree is at most  $v$ 's). The question that remains is: can we afford to list all 2-paths of the form  $L \rightarrow H \rightarrow H$ ? Interestingly, we answer this question affirmatively. Roughly speaking, we show that there can't be too many  $L \rightarrow H \rightarrow H$  paths compared to 4-cycles, unless the number of such 2-paths was small enough for our purposes. Hence, we use a charging argument that allows us to list all such 2-paths.

For instance, going back to the graph construction we discussed with  $m^{5/3-\varepsilon}$   $LHH$  2-paths and no 4-cycles, it contains only  $O(m)$   $L \rightarrow H \rightarrow H$  2-paths.

**A road-map for the technical parts.** First, in Section 2.3, we prove a helpful theorem that shows that there can't be too many 2-paths of the form  $L \rightarrow H \rightarrow H$  compared to 4-cycles. We refer to this theorem as the  $L \rightarrow H \rightarrow H$  theorem. Then, in Section 2.4, we put everything together and prove our main result - an  $\tilde{O}(m^{4/3} + t)$ -time algorithm for 4-cycle listing.

### 2.3 The $L \rightarrow H \rightarrow H$ Theorem

In this section, we prove the following theorem that connects the number of 4-cycles in a graph to the number of 2-paths of a certain type. The degree of a node  $v$  is denoted by  $\deg(v)$ .

► **Theorem 4.** *Given an undirected graph  $G = (V, E)$  with  $m$  edges, let  $H$  be the set of nodes with degree larger than  $m^{1/3}$ , and  $L = V \setminus H$ . Orient all the edges  $\{u, v\} \in E$  from  $u$  to  $v$  if  $\deg(u) \leq \deg(v)$  (break ties arbitrarily). Let  $P$  be the number of directed 2-paths of the form  $L \rightarrow H \rightarrow H$  in  $G$ . It holds that if  $P > 100m^{4/3} \log^2 m$ , then there are at least  $P/(100 \log^2 m)$  4-cycles in  $G$ .*

In order to prove Theorem 4, we use two helper lemmas. In Lemma 5, we show that the number of 4-cycles is  $\Omega(d^4 - n^2)$ , where  $d$  is the average degree. In Lemma 6, we provide a view of the graph that has some nice properties. In particular, this view is a partitioning of the graph that has a useful regularity property, while the number of  $L \rightarrow H \rightarrow H$  2-paths is preserved. The proof of Theorem 4 is provided after the proof of Lemma 6.

► **Lemma 5.** *Any graph with  $n$  nodes and average degree  $d$  has  $\Omega(d^4 - n^2)$  4-cycles.*

**Proof.** Let  $G = (V, E)$  be a graph with  $n$  nodes and average degree  $d$ . Let  $A$  be the adjacency matrix of  $G$ . Denote by  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$  the  $n$  eigenvalues of  $A$ . The top eigenvalue  $\lambda_1$  of  $A$  is at least  $d$ . This is because:

$$\lambda_1 = \max_{v^T v = 1} v^T A v$$

Now, consider  $u = (1/\sqrt{n}, \dots, 1/\sqrt{n})$ . Clearly, it holds that  $u^T u = 1$ . On the other hand, we have that:

$$u^T A u = \sum_{w \in V} \frac{1}{n} \deg(w) = 2|E(G)|/n = d$$

The number of closed 4-walks in  $G$  is at least  $d^4$ . This is because this number is exactly the trace of  $A^4$  and

$$\text{tr}(A^4) = \sum_{i=1}^n \lambda_i^4 \geq \lambda_1^4 \geq d^4$$

Let  $T$  be the number of 4-cycles. To finish the proof, we show that the number of closed 4-walks is at most

$$3T + 2n^2 \tag{1}$$

This would imply that  $T \geq (d^4 - 2n^2)/3$  which would imply that  $T = \Omega(d^4 - n^2)$ . To prove Equation 1, notice that the distinct types of closed 4-walks are exactly 4-cycles, 2-paths and edges. There are  $T$  4-cycles and at most  $n^2$  edges. By Observation 2, the number of 2-paths is bounded by  $n^2 + 2T$ . ◀

► **Lemma 6.** *Given a graph  $G = (V, E)$  with  $m$  edges, let  $H$  be the set of nodes with degree larger than  $m^{1/3}$ , and  $L = V \setminus H$ . Orient all the edges  $\{u, v\}$  from  $u$  to  $v$  if  $\deg(u) \leq \deg(v)$  (break ties arbitrarily). Let  $P$  be the number of directed 2-paths of the form  $L \rightarrow H \rightarrow H$ . There is a partition of a subset of  $H$  into two parts  $H_1$  and  $H_2$  (i.e.  $H_1 \cap H_2 = \emptyset$  and  $H_1 \cup H_2 \subseteq H$ ), such that:*

1. The number of directed 2-paths of the form  $L \rightarrow H_1 \rightarrow H_2$  is at least  $P/(4\log^2 n)$ .
2. Each node in  $H_1$  has the same number of incoming edges from  $L$  up to a multiplicative 2-factor.
3. Each node in  $H_1$  has the same number of outgoing edges to  $H_2$  up to a multiplicative 2-factor.
4. Each node in  $H_2$  has at least one incoming edge from  $H_1$ .

**Proof.** We start by describing a simpler partitioning. This simpler partitioning splits the set of high-degree nodes  $H$  into two sets  $H'_1$  and  $H'_2$  such that the number of 2-paths of the form  $L \rightarrow H'_1 \rightarrow H'_2$  is at least  $P/4$ . Such a partitioning exists by the probabilistic method: Each node in  $H$  joins  $H'_1$  with probability  $1/2$  independently and otherwise it joins  $H'_2$ . Thus, the probability that a 2-path  $u \rightarrow v \rightarrow w$  of the form  $L \rightarrow H \rightarrow H$  survives in  $L \rightarrow H'_1 \rightarrow H'_2$  is  $\Pr(u \in H'_1 \wedge v \in H'_2) = 1/4$ . Hence, in expectation, we get  $P/4$  such paths in  $L \rightarrow H'_1 \rightarrow H'_2$ .

Next, for any node  $u \in H'_1$ , let  $\deg_{left}(u)$  be the number of incoming edges from  $L$ . Similarly, let  $\deg_{right}(u)$  be the number of outgoing edges to  $H'_2$ . Let  $B_{ij} = \{u \in H'_1 : \deg_{left}(u) \in [2^i, 2^{i+1}] \wedge \deg_{right}(u) \in [2^j, 2^{j+1}]\}$ . It holds that the number of 2-paths of the form  $L \rightarrow H'_1 \rightarrow H'_2$  is

$$P/4 = \sum_{u \in H'_1} \deg_{left}(u) \cdot \deg_{right}(u) = \sum_{i,j \in [\log n]} \sum_{u \in B_{ij}} \deg_{left}(u) \cdot \deg_{right}(u)$$

Hence, there are  $i', j' \in [\log n]$  such that

$$\sum_{u \in B_{i'j'}} \deg_{left}(u) \cdot \deg_{right}(u) \geq P/(4\log^2 n),$$

which implies that there are at least  $P/(4\log^2 n)$   $L \rightarrow H'_1 \rightarrow H'_2$  2-paths that go through  $B_{i'j'}$ . Let  $H_1 := B_{i'j'}$  and define  $H_2$  to be all the vertices in  $H'_2$  with at least one incoming edge from  $H_1$ . The sets  $L, H_1$ , and  $H_2$  satisfy the desired properties.  $\blacktriangleleft$

Now we are ready to prove Theorem 4.

**Proof of Theorem 4.** First, take the partitioning from Lemma 6. We know that the number of 2-paths of the form  $L \rightarrow H_1 \rightarrow H_2$  is  $P' \geq P/(4\log^2 m) \geq 25m^{4/3}$ . Recall that each node in  $H_1$  has the same in-degree from  $L$  up to a multiplicative 2-factor. Denote by  $d_{left}$  the minimum over these degrees. Similarly, each node in  $H_1$  has the same out-degree to  $H_2$  (up to a multiplicative 2-factor). Denote the minimum over these degrees by  $d_{right}$ . Furthermore, each node in  $H_2$  has at least one incoming neighbor from  $H_1$ .

We show that the number of 4-cycles is  $\Omega(P')$ . For this, let  $d_0$  be the average degree of the subgraph of  $G$  induced by the nodes in  $H_1 \cup H_2$ . Since we have at most  $2m^{2/3}$  nodes in  $H_1 \cup H_2$ , it suffices to show, by Lemma 5, that  $d_0^4 = \Omega(P')$ .

For this, we split the proof into two cases:

**Case 1:**  $|H_1| > |H_2|$ . Observe that in this case,  $d_0 > d_{right}/2$ . Hence, it suffices to show that  $d_{right}^4 = \Omega(P')$ . For this, recall that  $P' \leq 4|H_1| \cdot d_{left} \cdot d_{right}$  (where the 4 factor is coming from the two 2 factors for  $d_{left}$  and  $d_{right}$ ), and assume towards a contradiction that  $(d_{right})^4 < P'$ , which implies (by substituting  $d_{right}$  with  $P'/(4|H_1| \cdot d_{left})$ ) that  $(P')^3 < (4|H_1| \cdot d_{left})^4$ . Therefore, we get that  $P' < 16m^{4/3}$  (as  $|H_1| \cdot d_{left} \leq m$ ), which is a contradiction.

**Case 2:**  $|H_1| \leq |H_2|$ . In this case, we have that  $d_0 > d_{right} \cdot |H_1| / (2|H_2|)$ . Hence, it suffices to show that  $(d_{right}|H_1|/2|H_2|)^4 > P'$ . Assume towards a contradiction that  $(d_{right}|H_1|/2|H_2|)^4 \leq P'$ . By substituting  $d_{right}$  with  $P'/(4d_{left} \cdot |H_1|)$ , this implies that  $(P')^3 < (8d_{left} \cdot |H_2|)^4$ . Now we want to argue that  $d_{left} \cdot |H_2|$  is at most  $m$  to get a contradiction to  $P' \geq 25m^{4/3}$ . For this, recall that each node in  $H_2$  has at least one incoming edge from  $H_1$ , which implies that the degree (in  $G$ ) of each node in  $H_2$  is at least  $d_{left}$ . Hence, we have that  $m \geq \sum_{u \in H_2} \deg(u) \geq \sum_{u \in H_2} d_{left} = |H_2| \cdot d_{left}$ , as desired.  $\blacktriangleleft$

## 2.4 Listing 4-Cycles

In this section, we prove the following theorem.

► **Theorem 7.** *Listing all the 4-cycles in an undirected graph  $G = (V, E)$  can be done in  $O(m^{4/3} \log^2 m + t \log^2 m)$  time, where  $m$  is the number of edges and  $t$  is the number of 4-cycles.*

The proof of Theorem 7 is based on listing several types of 2-paths efficiently. Lemma 8 shows that we can list each of these types of 2-paths efficiently.

► **Lemma 8.** *Given a graph  $G = (V, E)$  with  $m$  edges and  $t$  4-cycles. Let  $H$  be the set of nodes with degrees larger than  $m^{1/3}$  and  $L = V \setminus H$ .*

1. *Listing all the 2-paths with nodes only from  $H$  can be done in  $O(m^{4/3} + t)$  time.*
2. *Listing all the 2-paths with a node from  $L$  at the center can be done in  $O(m^{4/3})$  time.*
3. *Orient all the edges  $\{u, v\}$  from  $u$  to  $v$  if  $\deg(u) \leq \deg(v)$  (break ties arbitrarily). Listing all the directed 2-paths of the form  $L \rightarrow H \rightarrow H$  can be done in  $O(m^{4/3} \log^2 m + t \log^2 m)$  time.*

**Proof.**

1. Let  $G'$  be the subgraph of  $G$  induced by  $H$ . Denote by  $n'$  the number of nodes in  $G'$  and by  $t'$  the number of 4-cycles in  $G'$ . Observe that  $n' \leq 2m^{2/3}$  and  $t' \leq t$ . By using an argument similar to the one used in Observation 3, we can list all the 2-paths in  $G'$  in time  $O(n'^2 + t') = O(m^{4/3} + t)$ .
2. Scan all the edges in  $G$  and for those with at least one endpoint in  $L$  scan all the neighbors of the endpoints in  $L$ . This procedure finds all the 2-paths with a node from  $L$  in the middle. Each node in  $L$  has at most  $m^{1/3}$  neighbors. Therefore, the running time of this procedure is  $O(m^{4/3})$ .
3. We can list all the  $L \rightarrow H \rightarrow H$  2-paths in time that is linear in their number and the number of edges. This can be done by going over all the nodes  $u \in L$ , and then going over the neighbors  $v \in H$  of  $u$ , and then going over all neighbors of  $v$  with higher degree than  $v$ . This can be done in time that is linear in the number of edges and the number of  $L \rightarrow H \rightarrow H$  2-paths. This is because we can prepare a set of higher degree nodes in  $H$  for all the nodes  $u \in V$ , via a simple  $O(m)$ -time preprocessing step, where we go over all the edges (with at least one endpoint in  $H$ ), detect for each edge the higher degree endpoint, and store it. Therefore, the running time of the algorithm is  $O(m + P)$  where  $P$  is the number of directed  $L \rightarrow H \rightarrow H$  2-paths. Since by Theorem 4 we have that  $P = O(m^{4/3} \log^2 n + t \log^2 n)$ , the claim follows.  $\blacktriangleleft$

**Proof of Theorem 7.** We consider all the different types of 4-cycles (in terms of low-high degree nodes) and show that we can list all of them in the desired running time.

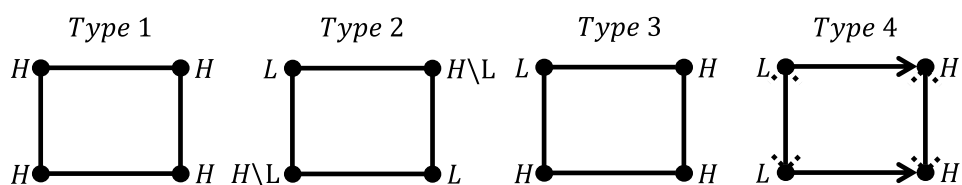


**Type 1: 4-cycles that use only high-degree nodes.** This class of 4-cycles can be decomposed into two 2-paths of all high-degree nodes. These 2-paths can be listed in  $O(m^{4/3} + t)$  time as shown in Lemma 8.

**Type 2: 4-cycles with two non-adjacent low-degree nodes.** This class of 4-cycles can be decomposed into two 2-paths with a low-degree node at the center. These 2-paths can be listed in  $O(m^{4/3})$  time as shown in Lemma 8.

**Type 3: 4-cycles with three high-degree nodes and one low-degree node.** This class can be decomposed into a 2-path of all high-degree nodes and a 2-path with a low-degree node at the center. Using Lemma 8 these 2-paths can be listed in  $O(m^{4/3} + t)$  time.

**Type 4: 4-cycles with two adjacent low-degree nodes and two adjacent high-degree nodes.** These 4-cycles can be decomposed into a directed  $L \rightarrow H \rightarrow H$  2-path and an  $LLH$  2-path. Using Lemma 8 we can list all these 2-paths in  $O(m^{4/3} \log^2 n + t \cdot \log^2 n)$  time. To sum up, we showed how to list all the different types of 4-cycles in  $O(m^{4/3} \log^2 n + t \cdot \log^2 n)$  time, as desired. ◀



■ **Figure 1** The types of 4-cycles. A 4-cycle of the fourth type always consists of an  $L \rightarrow H \rightarrow H$  directed 2-path and a 2-path with a low-degree node at the center.

### 3 Hardness Results

In this section, we give a set of hardness results that more or less follow from folklore reductions (e.g. similar to the hardness reductions for general cyclic queries in databases [5]).

#### 3.1 Lower Bound for 4-Cycle Listing from 4-Cycle Detection

In this section we give a general argument showing that listing cannot be solved faster than the “detection plus  $t$ ” bound. We will prove it specifically for 4-cycle but the proof is more general.

► **Observation 9.** *If 4-cycle listing can be solved in time  $O(T(n, m) + t)$  when  $t$  is the number of 4-cycles in the graph, then 4-cycle detection can be solved in time  $O(T(n, m))$ .*

**Proof.** We will show a fine-grained reduction from 4-cycle detection to 4-cycle listing. Given a graph  $G$ , run the 4-cycle listing algorithm on  $G$ . If the algorithm terminates in  $O(T)$  time then if the algorithm outputs at least one 4-cycle then we also detected a 4-cycle and otherwise, there isn’t a 4-cycle in  $G$ . If the algorithm takes more than  $cT$  time for large enough constant  $c$ , then we know that there are at least  $T$  4-cycles in  $G$ . Therefore we detected a 4-cycle. In conclusion, we have an  $O(T)$  time algorithm for 4-cycle detection as we wanted. ◀

Note that even an  $o(\min(n^2, m^{4/3}))$  time algorithm for 4-cycle is not known [42, 16], so our listing upper bound is tight up to the polylog factor.

### 3.2 Hardness of All-Pairs 4-Cycle

In this section, we define a new natural problem which we call *all-pairs 4-cycle*. We show an algorithm for this problem and a conditional lower-bound from triangle-finding.

► **Definition 10** (all-pairs 4-cycle). *Given an undirected graph  $G$ , all-pairs 4-cycle is the problem of deciding for each pair of distinct vertices  $u, v \in V(G)$  whether there is a 4-cycle  $(u, x, v, y)$  in  $G$  (for  $x, y \in V(G)$ ).*

Note that if 4-cycle listing is in time  $O(T)$  then all-pairs 4-cycle is also in time  $O(T)$ , as there is a trivial reduction from all-pairs 4-cycle to 4-cycle listing. But, we will see that this problem cannot be solved in  $n^{\omega-\varepsilon}$  time as long as triangle detection cannot be solved in  $n^{\omega-\varepsilon}$  time.

▷ **Claim 11.** There is an algorithm for all-pairs 4-cycle in  $O(n^\omega)$ -time. In addition, if there is an  $O(n^{2+\varepsilon})$  time algorithm for all-pairs 4-cycle then there is an  $O(n^{2+\varepsilon})$  time algorithm for triangle detection.

*Proof.* The all-pairs 4-cycle problem can be solved in  $O(n^\omega)$  time by simply computing  $A^2$  where  $A$  is the adjacency matrix of  $G$  and checking for any pair  $u, v$  whether  $A^2[u, v] \geq 2$ .

Assume there is an  $O(n^{2+\delta})$  time algorithm (for  $\delta \geq 0$ ) for all-pairs 4-cycle. We will show that we can solve triangle detection in  $O(n^{2+\delta})$  time. Let  $G$  be an  $n$ -node graph. We define  $G^*$  as an extension of  $G$  by adding a new vertex  $x$  and connecting it to all the vertices in  $G$ . If there is a 4-cycle  $(u, t, v, w)$  in  $G^*$ , and  $uv \in E(G)$  then  $u, v$  participate in a triangle in  $G$  (because at most one of the vertices  $t, w$  can be  $x$ ). For the other direction, if  $u, v$  participate in a triangle  $(u, t, v)$  then after adding  $x$  they must participate in a 4-cycle  $(u, t, v, x)$  and  $uv \in E(G)$ . Therefore, we can solve triangle detection by running all-pairs 4-cycle on  $G^*$ , and check if there is a pair of distinct vertices  $u, v \in V(G)$  such that all-pairs 4-cycle returned “yes” on them and  $u, v \in E(G)$ . The time required is  $O(n^{2+\delta} + n^2) = O(n^{2+\delta})$ . ◁

### 3.3 4-Cycle Counting is Triangle Counting Hard

We now present a folklore reduction from triangle counting to 4-cycle counting.

► **Observation 12.** *There is a fine-grained reduction from triangle counting to 4-cycle counting. The reduction time is quadratic in  $n$  and linear in  $n + m$ .*

*Proof.* We will show a reduction from triangle counting to 4-cycle counting. Let  $G = (V, E)$  be a graph and our goal is to count the triangles in  $G$ . We construct a new graph  $G'$  by duplicating the vertices of  $G$  three times into three parts, and connecting by an edge a pair of vertices from distinct parts if they are a duplication of adjacent vertices in  $G$ . More formally,  $G' = (V_A \cup V_B \cup V_C, E')$  where  $V_A = \{v_A : v \in V\}$ ,  $V_B = \{v_B : v \in V\}$ ,  $V_C = \{v_C : v \in V\}$  and  $E' = \{u_A v_B : uv \in E\} \cup \{u_B v_C : uv \in E\} \cup \{u_C v_A : uv \in E\}$ . Clearly, every triangle  $(u, v, w)$  in  $G$  has exactly six matching triangles in  $G'$ .

Let  $G''$  be the graph formed from  $G'$  by duplicating  $V_A$  and put edges between a vertex in  $V_A$  to its duplication. That is,  $G'' = (V_A \cup V_{A'} \cup V_B \cup V_C, E'')$  such that  $V_{A'} = \{v_{A'} : v_A \in V_A\}$  is a duplication of  $v_A$ , and the set of edges  $E''$  is defined as follows:  $E'' = \{v_A v_{A'} : v_A \in A\} \cup (E' \cap V_A \times V_C) \cup (E' \cap V_B \times V_C) \cup \{u_{A'} v_B : u_A v_B \in E'\}$ . Notice that any triangle in  $G'$  becomes a 4-cycle in  $G''$ . But, there may be 4-cycles in  $G''$  that are not corresponding to triangles in  $G'$ . However, these 4-cycles can't include edges in  $\{v_A v_{A'} : v_A \in A\}$ . Let  $G^*$  be  $G''$  after removing the edges in  $\{v_A v_{A'} : v_A \in A\}$ . The 4-cycles in  $G^*$  are exactly the

4-cycles in  $G''$  that are not corresponding to triangles in  $G'$ . So, by counting the number of 4-cycles in  $G''$  and subtracting the number of 4-cycles in  $G^*$  we get the exact number of triangles in  $G'$ . Since there are six triangles in  $G'$  for each triangle in  $G$ , we can get the number of triangles in  $G$ .

The running time of the reduction is  $O(n + m) = O(n^2)$  for constructing the graphs  $G'$ ,  $G''$  and  $G^*$ . ◀

► **Corollary 13.** *An  $O(\min(n^{\omega-\varepsilon}, m^{\frac{2\omega}{\omega+1}-\varepsilon}))$ -time algorithm for 4-cycle counting (for some  $\varepsilon > 0$ ) would imply an  $O(\min(n^{\omega-\varepsilon}, m^{\frac{2\omega}{\omega+1}-\varepsilon}))$ -time algorithm for triangle counting.*

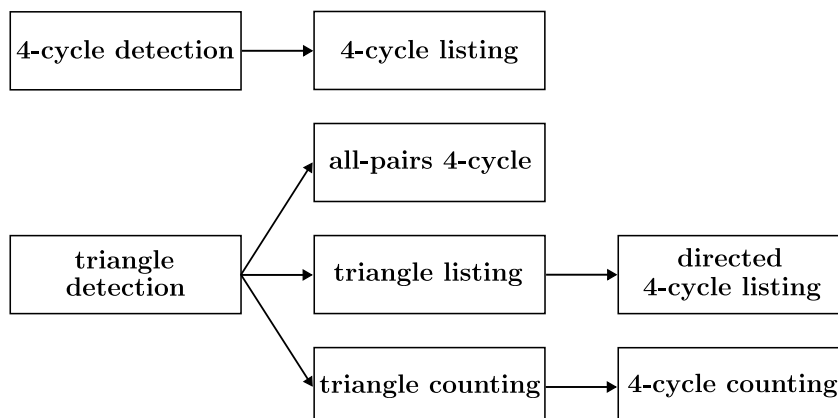
### 3.4 Hardness of Directed 4-Cycle Listing from Triangle Listing

With a similar reduction to the previous one, we can show the hardness of 4-cycle listing in *directed* graphs via a reduction from triangle listing.

► **Definition 14** (directed 4-cycle listing). *Given a directed graph  $G$ , directed 4-cycle listing is the problem of listing all the directed 4-cycles in  $G$ .*

▷ **Claim 15.** If directed 4-cycle listing is in time  $O(T(n, m) + t)$  then triangle listing is also in time  $O(T(n, m) + t)$ .

Proof. Given a triangle listing instance  $G$  we create a graph  $G''$  as in the previous proof. Now, we create a *directed* graph  $G'''$  from  $G''$  by directing all the edges as follows:  $V_A \rightarrow V_{A'} \rightarrow V_B \rightarrow V_C \rightarrow V_A$ . Thus, any directed 4-cycle in this new graph corresponds to a triangle in  $G'$ , and recall that any triangle in  $G$  becomes six triangles in  $G'$ . Therefore, the number of 4-cycles in  $G'''$  and the number of triangles in  $G$  have a one-to-six correspondence, and so any directed 4-cycle listing algorithm can be used to list triangles with an additional factor six overhead in  $t$ . ◀



■ **Figure 2** Overview of the reductions in Section 3.

## 4 Open Problems

We conclude with two open problems that we find interesting.

### 4.1 Generalization for Larger Even Cycles

A natural follow-up question for our work is, how hard is it to list all the cycles of length  $2k$  for constant  $k > 2$ ?

Yuster and Zwick developed an algorithm for  $2k$ -cycle detection in time  $O(n^2)$  [42]. Later on, Dahlgaard et al. showed an algorithm in time  $O(m^{\frac{2k}{k+1}})$  [16]. It is natural to ask whether can we generalize our 4-cycle listing algorithm to an algorithm for  $2k$ -cycle listing for any  $k \geq 2$  in the same time as  $2k$ -cycle detection plus the number of cycles.

► **Open Question 16.** *Can  $2k$ -cycle listing be solved in time  $O(n^2 + t)$  where  $n$  is the number of vertices and  $t$  is the number of  $2k$ -cycles in  $G$ ?*

► **Open Question 17.** *Can  $2k$ -cycle listing be solved in time  $O(m^{\frac{2k}{k+1}} + t)$  where  $m$  is the number of edges and  $t$  is the number of  $2k$ -cycles in  $G$ ?*

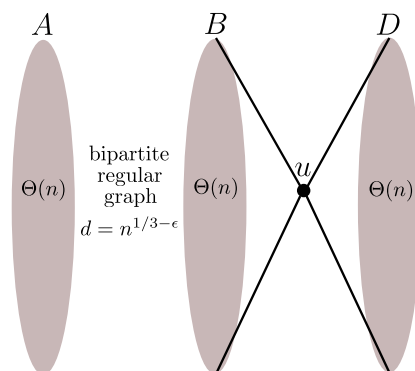
The  $O(n^2 + t)$ -algorithm for 4-cycle listing is pretty simple: just list all the 2-paths and construct 4-cycles from the 2-paths. The key observation of the analysis is that any new 2-path we found after  $n^2$  2-paths were already found must form at least one more 4-cycle.

A naive candidate for an algorithm for  $2k$ -cycle listing in  $O(n^2 + t)$  is to list all the  $k$ -paths and construct all the  $2k$ -cycles from them by looking for pairs of  $k$ -paths between a certain pair of vertices. But notice that there is a problem. In the 4-cycle case, any pair of 2-paths between a certain pair of vertices form a 4-cycle. However, this is not the case in the general  $2k$ -cycle case. For example, a pair of 3-paths between a certain pair of vertices may collide in the middle and thus form a 4-cycle with an additional edge instead of a 6-cycle. Therefore, there can be too many 3-paths but not enough 6-cycles.

For regular graphs, this simple algorithm achieves  $O(n^2 + t)$  time. The analysis is simple: The number of  $k$ -paths is at most  $nd^k$ . If  $d = O(n^{1/k})$  this number is at most  $O(n^2)$ . Otherwise, the graph is “supersaturated” and the number of  $2k$ -cycles is  $t = \Omega(d^{2k})$  [25, 33] and  $t > nd^k$  (otherwise  $d = O(n^{1/k})$ ). Therefore, in this case, the number of  $k$ -paths is at most  $O(t)$ . Hence, we can always bound the number of  $k$ -paths by  $O(n^2 + t)$  and bound the running time of the algorithm by  $O(n^2 + t)$ .

To show that this simple algorithm doesn’t work for general graphs we show the following example. Consider the following 4-layered graph  $G = (A \cup B \cup \{u\} \cup D, E)$ , such that  $A = B = D = \Theta(n)$ , which is defined as follows.  $A \cup B$  is a bipartite regular graph with degree  $n^{1/3-\varepsilon}$  and  $n^{2-6\varepsilon}$  6-cycles. This graph exists by a probabilistic argument. Consider a random bipartite graph such that each edge exists with probability  $n^{-2/3-\varepsilon}$ . In expectation, each vertex has a degree of  $n^{1/3-\varepsilon}$  and the number of 6-cycles is  $n^{2-6\varepsilon}$ . Every vertex in  $B$  is connected to  $u$  and every vertex in  $D$  is connected to  $u$  (i.e.  $B \cup \{u\}$  and  $\{u\} \cup D$  are full bipartite graphs). The example is illustrated in Figure 3.

The number of 3-paths in this graph is at least  $n^{2+1/3-\varepsilon}$  while the number of 6-cycles is at most  $n^{2-6\varepsilon} + n \cdot n^{4 \cdot (\frac{1}{3}-\varepsilon)} = O(n^{2+1/3-4\varepsilon})$ . This is because there are  $n^{2-6\varepsilon}$  6-cycles in the bipartite graph  $A \cup B$ , and every 4-path in  $A \cup B$  that starts in  $B$  generates a 6-cycle with  $u$  and there are at most  $n \cdot n^{4 \cdot (\frac{1}{3}-\varepsilon)}$  such 4-paths. Notice that there are no other 6-cycles. To conclude, the number of 3-paths is at least  $n^{2+1/3-\varepsilon}$  and it is asymptotically bigger than  $n^2 + t$  where the number of 6-cycles is  $t = O(n^{2+1/3-4\varepsilon})$ . Therefore the running time of the algorithm we suggested is lower bounded by  $n^{2+1/3-o(1)} + \Omega(t)$ .



■ **Figure 3** An example of a graph with  $O(n^{2+1/3-\epsilon})$  3-paths but  $O(n^{2+1/3-4\epsilon})$  6-cycles. The example shows that 6-cycle listing by listing 3-paths must take  $n^{2+1/3-o(1)} + \Omega(t)$  time.

## 4.2 4-Cycles vs. Triangles: Which is Harder?

As we already mentioned, triangles and 4-cycles are the smallest patterns that are not trivial to find. As such, it is natural to ask which of these patterns are harder to detect. 4-cycle detection can be solved in  $O(\min(n^2, m^{\frac{4}{3}}))$  time while triangle detection can be solved in  $O(\min(n^\omega, m^{\frac{2\omega}{\omega+1}}))$  time, which is higher as long as  $\omega > 2$ . This supports the intuition that triangle detection is harder, but this is not formally known. In order to show hardness formally, we have to show a reduction.

► **Open Question 18.** *Is there a (fine-grained) reduction from 4-cycle to triangle? Concretely, would an  $O(m^{4/3-\epsilon})$  algorithm for triangle detection (for some  $\epsilon > 0$ ) imply an  $O(m^{4/3-\delta})$  algorithm for 4-cycle detection (for some  $\delta = \delta(\epsilon) > 0$ )?*

Even a weak reduction is not known: Suppose that triangle detection is in  $O(m)$  time. Does it imply an  $o(m^{4/3})$  algorithm for 4-cycle detection?

In an attempt to develop such a reduction we encountered the following barrier: If such a reduction would preserve the number of occurrences (i.e. the number of triangles in the input graph is the same as the number of 4-cycles in the graph generated by the reduction), then this reduction works also for other variations of problems such as *listing* and *counting*.

In this paper, we showed that 4-cycle listing is indeed easier than triangle listing (assuming 3-SUM conjecture). But there is still a gap in the counting problems. The state of the art running time for triangle counting is  $O(\min(n^\omega, m^{\frac{2\omega}{\omega+1}}))$  [4] while the best known running time for 4-cycle counting is  $O(\min(n^\omega, m^{\frac{4\omega-1}{2\omega+1}}))$  [43, 40]. Under the assumption  $\omega = 2$  the running times in the sparse cases are  $O(m^{4/3})$  and  $O(m^{7/5})$  correspondingly. This means that at least in sparse graphs, 4-cycle counting actually seems harder than triangle counting.

► **Open Question 19.** *Is there an algorithm for 4-cycle counting in time  $O(m^{\frac{2\omega}{\omega+1}})$ ? Or alternatively, can we show a hardness result for 4-cycle counting?*

---

## References

- 1 Amir Abboud, Karl Bringmann, and Nick Fischer. Stronger 3-sum lower bounds for approximate distance oracles via additive combinatorics. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 391–404. ACM, 2023. doi:10.1145/3564246.3585240.

## 25:14 Listing 4-Cycles

- 2 Amir Abboud, Karl Bringmann, Seri Khoury, and Or Zamir. Hardness of approximation in  $p$  via short cycle removal: cycle detection, distance oracles, and beyond. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1487–1500. ACM, 2022. doi:10.1145/3519935.3520066.
- 3 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.
- 4 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.
- 5 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *International Workshop on Computer Science Logic*, pages 208–222. Springer, 2007.
- 6 R. Bar-Yehuda and S. Even. On approximating a vertex cover for planar graphs. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 303–309, New York, NY, USA, 1982. Association for Computing Machinery. doi:10.1145/800070.802205.
- 7 Jonathan Berry, Bruce Hendrickson, Randall LaViolette, and Cynthia Phillips. Tolerating the community detection resolution limit with edge weighting. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 83:056119, May 2011. doi:10.1103/PhysRevE.83.056119.
- 8 Etienne Birmelé, Rui Ferreira, Roberto Grossi, Andrea Marino, Nadia Pisanti, Romeo Rizzi, and Gustavo Sacomoto. Optimal listing of cycles and st-paths in undirected graphs. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1884–1896. SIAM, 2013.
- 9 Andreas Björklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. Listing triangles. In *International Colloquium on Automata, Languages, and Programming*, pages 223–234. Springer, 2014.
- 10 Nofar Carmeli and Markus Kröll. On the enumeration complexity of unions of conjunctive queries. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 134–148, 2019.
- 11 Nofar Carmeli and Markus Kröll. Enumeration complexity of conjunctive queries with functional dependencies. *Theory of Computing Systems*, 64(5):828–860, 2020.
- 12 Nofar Carmeli and Luc Segoufin. Conjunctive queries with self-joins, towards a fine-grained enumeration complexity analysis. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 277–289, 2023.
- 13 Keren Censor-Hillel. Distributed subgraph finding: progress and challenges. *arXiv preprint*, 2022. arXiv:2203.06597.
- 14 Yi-Jun Chang, Seth Pettie, and Hengjie Zhang. Distributed triangle detection via expander decomposition. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 821–840. SIAM, 2019.
- 15 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985. doi:10.1137/0214017.
- 16 Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Morten Stöckel. Finding even cycles faster via capped  $k$ -walks. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 112–120, 2017.
- 17 Ran Duan, Hongxun Wu, and Renfei Zhou. Faster matrix multiplication via asymmetric hashing, 2022. doi:10.48550/arXiv.2210.10173.
- 18 Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Transactions on Computational Logic (TOCL)*, 8(4):21–es, 2007.
- 19 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1-3):57–67, 2004.

- 20 Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Constant delay algorithms for regular document spanners. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 165–177, 2018.
- 21 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*, CHI EA '10, pages 4027–4032, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1753846.1754097.
- 22 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 1–10, 1977.
- 23 Rajesh Jayaram and John Kallaugher. An optimal algorithm for triangle counting in the stream. *arXiv preprint*, 2021. arXiv:2105.01785.
- 24 Madhav Jha, C. Seshadhri, and Ali Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In Inderjit S. Dhillon, Yehuda Koren, Rayid Ghani, Ted E. Senator, Paul Bradley, Rajesh Parekh, Jingrui He, Robert L. Grossman, and Ramasamy Uthrusamy, editors, *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, pages 589–597. ACM, 2013. doi:10.1145/2487575.2487678.
- 25 Tao Jiang and Liana Yepremyan. Supersaturation of even linear cycles in linear hypergraphs. *Combinatorics, Probability and Computing*, 29(5):698–721, 2020. doi:10.1017/S0963548320000206.
- 26 Ce Jin and Yinzhan Xu. Removing additive structure in 3sum-based reductions. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 405–418. ACM, 2023. doi:10.1145/3564246.3585157.
- 27 Donald B Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84, 1975.
- 28 Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Information Processing Letters*, 74(3-4):115–121, 2000.
- 29 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *SODA*, 2016.
- 30 François Le Gall. Improved quantum algorithm for triangle finding via combinatorial arguments. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 216–225. IEEE, 2014.
- 31 Frédéric Magniez, Miklos Santha, and Mario Szegedy. Quantum algorithms for the triangle problem. *SIAM Journal on Computing*, 37(2):413–424, 2007.
- 32 Dániel Marx and Michał Pilipczuk. Everything you always wanted to know about the parameterized complexity of subgraph isomorphism (but were afraid to ask). *arXiv preprint*, 2013. arXiv:1307.2187.
- 33 Robert Morris and David Saxton. The number of  $c2\ell$ -free graphs. *Advances in Mathematics*, 298:534–580, 2016. doi:10.1016/j.aim.2016.05.001.
- 34 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.
- 35 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 603–610, 2010.
- 36 Dana Richards and Arthur L. Liestman. Finding cycles of a given length. In B.R. Alspach and C.D. Godsil, editors, *Annals of Discrete Mathematics (27): Cycles in Graphs*, volume 115 of *North-Holland Mathematics Studies*, pages 249–255. North-Holland, 1985. doi:10.1016/S0304-0208(08)73019-6.
- 37 Luc Segoufin. Constant delay enumeration for conjunctive queries. *ACM SIGMOD Record*, 44(1):10–17, 2015.

## 25:16 Listing 4-Cycles

- 38 Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517, 1977. doi:10.1137/0206036.
- 39 Virginia Vassilevska Williams, Joshua R Wang, Ryan Williams, and Huacheng Yu. Finding four-node subgraphs in triangle time. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on discrete algorithms*, pages 1671–1680. SIAM, 2014.
- 40 Virginia Vassilevska Williams, Joshua R. Wang, Ryan Williams, and Huacheng Yu. Finding four-node subgraphs in triangle time. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '15*, pages 1671–1680, USA, 2015. Society for Industrial and Applied Mathematics.
- 41 Virginia Vassilevska Williams and Yinzhan Xu. Monochromatic triangles, triangle listing and apsp. *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 786–797, 2020.
- 42 Raphael Yuster and Uri Zwick. Finding even cycles even faster. *SIAM Journal on Discrete Mathematics*, 10(2):209–222, 1997.
- 43 Raphael Yuster and Uri Zwick. Detecting short directed cycles using rectangular matrix multiplication and dynamic programming. In *SODA*, volume 4, pages 254–260, 2004.