Listing 6-Cycles

Ce Jin[*] Virginia Vassilevska Williams[†] Renfei Zhou[‡]

Abstract

Listing copies of small subgraphs (such as triangles, 4-cycles, small cliques) in the input graph is an important and well-studied problem in algorithmic graph theory. In this paper, we give a simple algorithm that lists t (non-induced) 6-cycles in an n-node undirected graph in $\widetilde{O}(n^2 + t)$ time. This nearly matches the fastest known algorithm for detecting a 6-cycle in $O(n^2)$ time by Yuster and Zwick (1997). Previously, a folklore $O(n^2 + t)$-time algorithm was known for the task of listing 4-cycles.

1 Introduction

Listing (also called enumerating) cycles in a graph is an important algorithmic task with a variety of applications (e.g. in computational biology [29] and social networks [25]). A line of works dating back to the 1970s developed efficient algorithms for listing all simple cycles in a (directed or undirected) input graph [37, 28, 35, 33, 6, 7, 26], and some of these algorithms have been engineered in practice [9]. These algorithms consider cycles of arbitrarily large length; however, in many natural scenarios, one only cares about listing cycles whose length is a small given constant, and this will be the focus of this paper.

Formally, we are given an *undirected* simple graph G of n nodes and m edges, and a small pattern graph H (in our case, H is the k-cycle C_k for some small fixed k). The H-*detection* problem asks whether G contains a copy of H as a (*not necessarily induced*) subgraph. In the H-*listing* problem, we are additionally given a parameter t, and we need to output t distinct copies of H in G (if G contains fewer than t copies of H, then we need to output all copies).

Known results on C_k-detection and C_k-listing. The simplest cycle is a triangle (C_3). A long line of works in graph algorithms and fine-grained complexity has led to a complete picture of triangle listing algorithms, up to natural hardness hypotheses: [8] gave an algorithm for listing t triangles in $\widetilde{O}(\min\{n^2+nt^{2/3}, m^{4/3}+mt^{1/3}\})$ time[1], assuming that the matrix multiplication exponent is $\omega = 2$. This running time is likely to be optimal: for $t = 1$ it matches the best known triangle detection algorithm in $O(m^{2\omega/(\omega+1)})$ time [5], and when t is sufficiently large in terms of n (or m) this running time is conditionally optimal under the 3SUM hypothesis [34, 30] and even more believable hypotheses [39]. See also [22] for more fine-grained reductions between triangle listing and other problems.

Now we review known results for cycle length $k > 3$. A common phenomenon is that finding *even* length cycles appears to be easier than finding odd cycles. Finding a cycle of any fixed odd length in undirected graphs is as difficult as the problem in directed graphs (see e.g. [38]), whereas finding even cycles is easier in undirected graphs than in directed graphs. Bondy and Simonovits [10] showed that for any integer $k \geq 2$, any n vertex graph with at least $100kn^{1+1/k}$ edges must contain a $2k$-cycle. Meanwhile, the complete bipartite graph on n nodes in each partition does not contain any cycles of odd length. This difference has been exploited to obtain faster algorithms for finding a cycle of any given even length [41, 17] that are faster than the known algorithms for detecting even the smallest length odd cycle, a triangle.

Let us review the previous works on the easier cycle *detection* problem in more detail. Yuster and Zwick [41] gave an $O(n^2)$-time algorithm that detects C_k in an undirected n-node graph for any fixed *even* integer k. Detecting C_k when $k = O(1)$ is odd can be done in $\widetilde{O}(n^\omega)$ time (e.g. via color-coding [4]).

When graph sparsity is taken into account, and the running time is in the number of edges m, the fastest running times are as follows. The fastest algorithms for C_k-detection for odd k in terms of m run

[*]MIT. cejin@mit.edu. Partially supported by NSF Grant CCF-2129139.

[†]MIT. virgi@mit.edu. Supported by NSF Grants CCF-2129139 and CCF-2330048 and BSF Grant 2020356.

[‡]Tsinghua University. zhourf20@mails.tsinghua.edu.cn.

[1]We use $\widetilde{O}(f)$ to denote $O(f \cdot \text{polylog}(f))$.

in $O(m^{\omega(k+1)/(2\omega+k-1)})$ time [19] and $O(m^{2-\frac{2}{k+1}})$ time [5], where the former is faster than the latter if $\omega \leq 2 + 2/(k-1)$; for the current best value of ω [21, 40] this is the case for $k \leq 5$. For even $k \geq 4$, the best known running time for detecting C_k is $O(m^{2-\frac{4}{k+2}})$ [17, 5]; this running time was shown to be conditionally optimal for $k \equiv 2 \pmod 4$ for algorithms that do not use fast matrix multiplication [32, 17].

While the listing variant of the problem is well understood for triangles (C_3), there isn't very much known for larger k. The problem of C_4-listing was very recently independently studied by [3] and [27], who gave an algorithm in $\widetilde{O}(\min\{n^2 + t, m^{4/3} + t\})$ time. For $t = 1$ this matches the known C_4-detection algorithms in $O(n^2)$ time [41] and $O(m^{4/3})$ time [5]. It was also shown to be conditionally optimal under 3SUM hypothesis independently by [1] and [27] (building on [2]), and later shown to be conditionally optimal under the Exact Triangle hypothesis (which is weaker than 3SUM hypothesis and APSP hypothesis) [15].

Our results. Given the previous works on the fine-grained complexity of listing triangles and 4-cycles, a natural question is to investigate the fine-grained complexity of C_k-listing for larger k. In this paper, we make the first step by considering $k = 6$, the smallest even number after 4. Our result is summarized in the following theorem.

Theorem 1.1. (List t 6-Cycles) *There is a deterministic algorithm that lists t copies of C_6 in an n-node undirected simple graph in $\widetilde{O}(n^2 + t)$ time.*

The running time of our algorithm nearly matches the algorithm of Yuster and Zwick [41] for detecting a C_6 in $O(n^2)$ time, for any $t \leq O(n^2)$.

At a high level, our algorithm for listing 6-cycles follows a similar idea as in the folklore 4-cycle detection/listing algorithm (e.g., [41, 27, 3]), where a 4-cycle $a \begin{smallmatrix} \diagup b_1 \diagdown \\ \diagdown b_2 \diagup \end{smallmatrix} c$ is formed by pasting together two paths $a - b_1 - c$ and $a - b_2 - c$ (where $b_1 \neq b_2$) which are stored in a table indexed by (a, c). Analogously, our algorithm forms a 6-cycle $a \begin{smallmatrix} \diagup b_1 - c_1 \diagdown \\ \diagdown b_2 - c_2 \diagup \end{smallmatrix} d$ by pasting together two paths $a - b_1 - c_1 - d$ and $a - b_2 - c_2 - d$, but the new challenge here is to avoid wasting time on degenerate cases $b_1 = b_2$ or $c_1 = c_2$ which do not produce valid 6-cycles.

We leave it as an open question to extend Theorem 1.1 from C_6 to C_{2k} for larger k.

Further related works. The problem of listing short cycles was also studied in restricted graph classes such as low-arboricity graphs [16] and planar graphs [31].

Another way to generalize the classical triangle listing results is to consider k-cliques of larger size k. A very recent work [18] considered the problem of listing k-cliques for small k.

More generally, enumeration algorithms are widely studied in database theory, and recently there is growing interest in the fine-grained complexity of enumeration algorithms; see e.g., [24, 36, 23, 13, 12, 11, 20, 14].

In the literature of enumeration algorithms, it is common to study the *delay* between outputting two consecutive answers. For example, the 4-cycle listing algorithm of [27] actually achieves $O(1)$ delay after $O(\min\{n^2, m^{4/3}\})$-time preprocessing; this is stronger than listing t 4-cycles in $O(\min\{n^2, m^{4/3}\} + t)$ time. Whether our 6-cycle listing algorithm (Theorem 1.1) can be upgraded to an enumeration algorithm with $\operatorname{poly}\log(n)$-delay after $\widetilde{O}(n^2)$-time preprocessing is left for future investigation.

2 Algorithm for Listing All 6-Cycles

Let $G = (V, E)$ be an undirected graph with n nodes. We first focus on a slightly easier variant of the 6-cycle listing problem which asks to list *all* 6-cycles in G. Our algorithm for this variant is summarized in the following theorem.

Theorem 2.1. (List All 6-Cycles) *There is a deterministic algorithm that lists all 6-cycles of the n-node input graph G in $O\big((n^2 + t)\log n\big)$ time, where t denotes the total number of 6-cycles in G.*

We will show how Theorem 2.1 implies our main Theorem 1.1 (which has a given parameter t possibly much smaller than the total cycle count) in Section 3.

To prove Theorem 2.1, we consider a color-coded version of the problem: each node in G receives one out of four possible colors, and thus the node set is partitioned into four color classes $V = A \sqcup B \sqcup C \sqcup D$.

Then, our task is to list all 6-cycles $a \begin{smallmatrix} \diagup b_1 - c_1 \diagdown \\ \diagdown b_2 - c_2 \diagup \end{smallmatrix} d$, where $a \in A$, $b_1 \neq b_2 \in B$, $c_1 \neq c_2 \in C$, and $d \in D$. Solving this task is the main part of our algorithm, as summarized by the following lemma:

Lemma 2.1. *Given a 4-partite undirected graph $G = (V, E)$ where $V = A \sqcup B \sqcup C \sqcup D$, we can list all 6-cycles whose 6 nodes are in A, B, C, D, C, B respectively in the order they appear on the cycle, in $O(n^2 + t)$ total time, where t denotes the total number of 6-cycles in G.*

We stress that in the statement of Lemma 2.1, t is the total number of *all* 6-cycles in G, not just those following the specified color pattern A, B, C, D, C, B.

Note that Lemma 2.1 implies Theorem 2.1 by the standard color-coding technique [4]: if we color each node in G with one of the four colors chosen independently at random, then any fixed 6-cycle in G satisfies the specified color pattern (and hence will be reported by Lemma 2.1) with probability $\geq 4^{-6} = \Omega(1)$, so repeating $O(\log n)$ rounds suffices to report all 6-cycles of G with $1 - 1/\operatorname{poly}(n)$ success probability and total running time $O\big((n^2 + t)\log n\big)$. This color-coding can also be derandomized using the perfect hashing technique described in [4, Section 4], with the same asymptotic time complexity.

The rest of this section is devoted to the proof of Lemma 2.1. Our algorithm runs in two stages: in the first stage (Section 2.1) we preprocess several tables, and in the second stage (Section 2.2) we report 6-cycles based on the information in these tables. Then Section 2.3 contains the key argument for bounding the time complexity of the algorithm.

2.1 Stage I: Compute Tables.

We are given an input graph $G = (V, E)$ where $V = A \sqcup B \sqcup C \sqcup D$. We use t to denote the total number of 6-cycles in G. By convention, we will use lowercase letters a, b, c, d to denote nodes in A, B, C, D respectively, if not otherwise stated. We use $N_x := \{u : (u, v) \in E\}$ to denote the set of neighbors of node $x \in V$.

We define the following tables:

- $N_{a,c}$ is the set of common neighbors $b \in B$ of a and c. Formally, $N_{a,c} := N_a \cap N_c \cap B$. Similarly, we define another table $N_{b,d} := N_b \cap N_d \cap C$.
- Let $P_{a,d}$ be the set of $b \in N_a \cap B$ such that $|N_{b,d}| \geq 2$, that is, the set of b that can form the shape $a{-}b{<}\begin{smallmatrix}c_1\\ c_2\end{smallmatrix}{>}d$. Similarly, let $Q_{a,d}$ be the set of $c \in N_d \cap C$ such that $|N_{a,c}| \geq 2$, that is, the set of c that can form $a{<}\begin{smallmatrix}b_1\\ b_2\end{smallmatrix}{>}c{-}d$.
- Let $R_{a,d}$ be the set of edges $(b, c) \in E \cap ((B \cap N_a) \times (C \cap N_d))$ such that $|N_{a,c}| = |N_{b,d}| = 1$. In other words, there is a path $a - b - c - d$ without *replacements* of b and c (there does not exist any other path $a - b' - c - d$ or $a - b - c' - d$ with $b' \neq b$ or $c' \neq c$).

Each of these tables N, P, Q, R consists of $O(n^2)$ entries, where each entry is a set of nodes (or node pairs). We say the size of the table is the total number of nodes (or node pairs) in all its entries. The first stage of our algorithm is to compute all these tables in $O(n^2 + \text{table size})$ time, shown in Algorithm 1. (Later in Section 2.3 we will show that the total size of the tables is also $O(n^2 + t)$.)

It is straightforward to verify that Algorithm 1 correctly computes all the tables N, P, Q, R according to their definitions. To analyze its time complexity, we only need to notice that every time Lines 5, 7, 9, 11 and 16 are executed, there will be a new element inserted to some table entry, so the running time of these lines add up to $O(\text{table size})$. The time consumed by Lines 13 and 14 is bounded by the number of paths $a - b - c$ and $b - c - d$ in the graph, which equals the total size of the tables $N_{a,c}$ and $N_{b,d}$, which is again $O(\text{table size})$. The other lines of Algorithm 1 take $O(|E|) = O(n^2)$ time.

2.2 Stage II: Report 6-Cycles.

In the second stage, we use the precomputed tables N, P, Q, R to help us report all 6-cycles of the form $a{<}\begin{smallmatrix}b_1 - c_1\\ b_2 - c_2\end{smallmatrix}{>}d$, where $a \in A$, $b_1 \neq b_2 \in B$, $c_1 \neq c_2 \in C$, and $d \in D$.

We think of each element $b \in P_{a,d}$ as representing a collection of paths $a - b - (?) - d$ that share the same b (where the ? mark can be any node in $N_{b,d}$), and similarly each $c \in Q_{a,d}$ represents a collection of paths sharing c, and each $(b, c) \in R_{a,d}$ represents a single path from a to d. Every possible path $a - b - c - d$ is contained in at least one of these three tables: if there is a replacement for b or c, then this path appears in $Q_{a,d}$ and/or $P_{a,d}$;

Algorithm 1: Compute the tables

```
1  Let $N_{a,c} = N_{b,d} = \varnothing$ for all $a \in A$, $b \in B$, $c \in C$, $d \in D$
2  Let $P_{a,d} = Q_{a,d} = R_{a,d} = \varnothing$ for all $a \in A$, $d \in D$
3  for $(b,c) \in E \cap (B \times C)$ do                      ▷ Compute tables $N, P, Q$
4      for $a \in N_b \cap A$ do
5          Insert $b$ to $N_{a,c}$                              ▷ $b$ is a common neighbor of $a$ and $c$
6          if $|N_{a,c}| = 2$ then                              ▷ Found two paths $a - b - c$, $a - b' - c$ ($b' \neq b$)
7              Insert $c$ to $Q_{a,d}$ for all $d \in N_c \cap D$
8      for $d \in N_c \cap D$ do                                ▷ Symmetric to the above
9          Insert $c$ to $N_{b,d}$
10         if $|N_{b,d}| = 2$ then
11             Insert $b$ to $P_{a,d}$ for all $a \in N_b \cap A$
12 for $(b,c) \in E \cap (B \times C)$ do                      ▷ Compute table $R$
13     Compute $S_a \coloneqq \{a \in N_b \cap A : |N_{a,c}| = 1\}$
14     Compute $S_d \coloneqq \{d \in N_c \cap D : |N_{b,d}| = 1\}$
15     for $(a,d) \in S_a \times S_d$ do
16         Insert $(b,c)$ to $R_{a,d}$
```

otherwise, this path is included in $R_{a,d}$. For a desired 6-cycle $a \begin{smallmatrix} b_1 - c_1 \\ b_2 - c_2 \end{smallmatrix} d$, we make a case distinction according to which tables the two paths $a - b_1 - c_1 - d$ and $a - b_2 - c_2 - d$ belong to:

1. One path belongs to P, and the other path belongs to Q.
2. Both paths belong to P (or both paths belong to Q).
3. Both paths belong to R.
4. One path belongs to P (or Q), while the other belongs to R.

In the following, we separately consider each case and describe our algorithm for listing all the 6-cycles in that case. (These cases are not disjoint, but this will only cause each cycle to be reported $O(1)$ times, which effectively blows up the total time complexity by a constant factor.)

Case 1. For every a and d, we want to report all 6-cycles formed by pasting together two paths $a - b_1 - c_1 - d$ and $a - b_2 - c_2 - d$ that belong to P and Q respectively. To do this, we iterate over all $b_1 \in P_{a,d}$ and $c_2 \in Q_{a,d}$, and report all possible choices of $c_1 \in N_{b_1,d}$ and $b_2 \in N_{a,c_2}$ such that $c_1 \neq c_2$ and $b_1 \neq b_2$. The pseudocode is given in Algorithm 2. It is clear that Algorithm 2 correctly lists all desired 6-cycles of Case 1.

Algorithm 2: Report 6-Cycles of Case 1

```
1  for each $a \in A$, $d \in D$ such that $|P_{a,d}| \geq 1$ and $|Q_{a,d}| \geq 1$ do
2      for each $b_1 \in P_{a,d}$ and $c_2 \in Q_{a,d}$ do
3          for each $c_1 \in N_{b_1,d}$ and $b_2 \in N_{a,c_2}$ do      ▷ List cycles given $a, d, b_1, c_2$
4              if $b_1 \neq b_2$ and $c_1 \neq c_2$ then
5                  Report cycle $a \begin{smallmatrix} b_1 - c_1 \\ b_2 - c_2 \end{smallmatrix} d$
```

Next, we analyze the time complexity of Algorithm 2. First, the number of iterations of the for-loop on Line 1 is bounded by $O(n^2)$. We then show that, for every $O(1)$ time we spend on the for-loop on Line 3 to 5, we can report one 6-cycle, thus the time complexity for this part is $O(t)$. At every time we enter this inner for-loop on

Line 3, the time we spend equals $O(|N_{b_1,d}| \cdot |N_{a,c_2}|)$, and the number of 6-cycles we report equals the number of $(c_1, b_2) \in N_{b_1,d} \times N_{a,c_2}$ such that $b_1 \neq b_2$, $c_1 \neq c_2$, which is

$$\left|\left(N_{b_1,d} \setminus \{c_2\}\right) \times \left(N_{a,c_2} \setminus \{b_1\}\right)\right| \geq (|N_{b_1,d}| - 1)(|N_{a,c_2}| - 1) \geq \frac{1}{4}|N_{b_1,d}| \cdot |N_{a,c_2}|,$$

where the last inequality uses the fact that $|N_{b_1,d}|, |N_{a,c_2}| \geq 2$ due to the definition of P, Q. This implies the desired time complexity $O(n^2 + t)$.

Case 2. The algorithm for Case 2 is similar, see Algorithm 3. (We only give the pseudocode for cycles whose both paths belong to P; the case for Q is symmetric.)

Algorithm 3: Report 6-Cycles of Case 2

```
1 for each $a \in A$, $d \in D$ such that $|P_{a,d}| \geq 2$ do
2     for each $b_1, b_2 \in P_{a,d}$ where $b_1 \neq b_2$ do
3         for each $c_1 \in N_{b_1,d}$ and $c_2 \in N_{b_2,d}$ do          ▷ List cycles given $a, d, b_1, b_2$
4             if $c_1 \neq c_2$ then
5                 Report cycle $a \begin{smallmatrix} \diagup b_1 - c_1 \diagdown \\ \diagdown b_2 - c_2 \diagup \end{smallmatrix} d$
```

Similar to Case 1, the correctness is clear, and we only need to show that every $O(1)$ time spent on Line 3 to 5 will report a 6-cycle. At every time we enter the inner for-loop, we spend $O(|N_{b_1,d}| \cdot |N_{b_2,d}|)$ time, and the number of reported cycles equals the number of $(c_1, c_2) \in N_{b_1,d} \times N_{b_2,d}$ such that $c_1 \neq c_2$, which is

$$|N_{b_1,d}| \cdot |N_{b_2,d}| - |N_{b_1,d} \cap N_{b_2,d}| \geq |N_{b_1,d}| \cdot (|N_{b_2,d}| - 1) \geq \frac{1}{2}|N_{b_1,d}| \cdot |N_{b_2,d}|,$$

where we again used the fact that $|N_{b_2,d}| \geq 2$ due to the definition of P. Therefore, the time consumption on the inner loop is $O(t)$, and the total time complexity is the same as Case 1, $O(n^2 + t)$.

Case 3. For Case 3, where both paths belong to R, we use a straightforward enumeration as shown in Algorithm 4.

Algorithm 4: Report 6-Cycles of Case 3

```
1 for each $a \in A$, $d \in D$ such that $|R_{a,d}| \geq 2$ do
2     for each $(b_1, c_1), (b_2, c_2) \in R_{a,d}$ where $(b_1, c_1) \neq (b_2, c_2)$ do
3         Report cycle $a \begin{smallmatrix} \diagup b_1 - c_1 \diagdown \\ \diagdown b_2 - c_2 \diagup \end{smallmatrix} d$
```

We need to show that the cycles reported by Algorithm 4 are valid. Suppose to the contrary that $b_1 = b_2$, then $c_1 \neq c_2$, and hence $|N_{b_1,d}| \geq 2$. Since $(b_1, c_1) \in R_{a,d}$, this contradicts the definition of $R_{a,d}$. Hence, we must have $b_1 \neq b_2$, and similarly $c_1 \neq c_2$, so the reported 6-cycle is valid. Clearly, the running time of Algorithm 4 is also $O(n^2 + t)$.

Case 4. Case 4 is similar to Case 3 where we use the fact that paths in R have no replacements for nodes b and c, thus $b_1 = b_2$ or $c_1 = c_2$ will never happen in the straightforward enumeration. We omit the pseudocode. The time complexity of this case is $O(n^2 + t)$ as before.

To summarize, we have shown that our stage 2 algorithm lists all desired 6-cycles in $O(n^2 + t)$ total time.

2.3 Bounding the Total Table Size.

We have shown that stage 1 runs in $O(n^2 + \text{table size})$ time and stage 2 runs in $O(n^2 + t)$ time. To bound the total running time, it remains to show that the total size of the tables P, Q, R, N does not exceed $O(n^2 + t)$.

First, we argue that the sizes of tables P, Q, R are $O(n^2+t)$. For any fixed a, d, when the set $P_{a,d}$ contains $x \geq 2$ elements, it produces at least $\binom{x}{2} \geq x-1$ valid 6-cycles as we have seen in Case 2 of the proof in Section 2.2. So $t \geq \sum_{a,d}(|P_{a,d}|-1) \geq (\sum_{a,d}|P_{a,d}|) - n^2$. A similar argument applies to tables Q (Case 2 in Section 2.2) and R (Case 3 in Section 2.2) as well. Hence, the total size of P, Q, R is $O(n^2+t)$.

It remains to analyze the size of table N. Our analysis on the sizes of $N_{a,c}$ (and symmetrically, $N_{b,d}$) relies on the following lemma:

Lemma 2.2. *If node $a \in A$ satisfies $\sum_{c\in C}|N_{a,c}| \geq 100n + k$, then G contains at least k 6-cycles in which a is the only node in A.*

Proof. [2] For any fixed $a \in A$, we consider the subgraph G_a of G defined by taking the union of the edges in all paths $a - b - c$ where $c \in C$ and $b \in N_{a,c}$. Since these paths have distinct (b, c) edges, we know $|E(G_a)| \geq \sum_{c\in C}|N_{a,c}| \geq 100n + k$. In the following, we first prove that there exists at least one 6-cycle provided $|E(G_a)| \geq 100n$.

We iteratively remove nodes in G_a whose current degree is less than 3, and denote the resulting graph by G'_a, with number of edges $|E(G'_a)| \geq |E(G_a)| - 2n \geq 98n$. As shown in Fig. 1, G'_a is composed of a bipartite graph between $V(G'_a)\cap B$ and $V(G'_a)\cap C$, and edges connecting a with *every* node in $V(G'_a)\cap B$. We know a itself is not removed, since otherwise $|V(G'_a)\cap B| \leq 2$ and the graph G'_a can only have at most $\sum_{b\in V(G'_a)\cap B}\deg(b) \leq 2n < 98n$ edges.

Next, we find a 6-cycle in G'_a of the form $a - b_1 - c_1 - b_2 - c_2 - b_3 - a$ in a greedy fashion: starting from an arbitrary $b_1 \in V(G'_a)\cap B$, in each step we go to an arbitrary unvisited neighbor of the current node (which must exist since every node has degree ≥ 3 in G'_a) as the next node on the cycle. Hence, we have found a 6-cycle containing a in G_a.

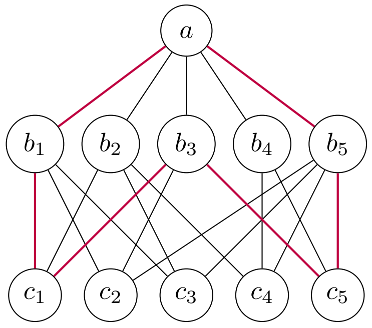

Figure 1: An example of G'_a. We are guaranteed that every node including a has degree at least 3, so we can find a 6-cycle step-by-step (drawn in thick purple lines).

Take the edge (b_1, c_1) from the 6-cycle that we have just found, and remove (b_1, c_1) from G_a, so that this 6-cycle no longer appears in the new G_a. Then, the number of edges $|E(G_a)|$ decreases by 1; if it is still $\geq 100n$, we can repeat the above argument to find a second 6-cycle. This process is repeated for k times as the initial G_a has $100n + k$ edges, and k cycles are reported in total, which proves the lemma. ∎

According to Lemma 2.2, for any fixed $a \in A$, $\sum_{c\in C}|N_{a,c}| - 100n$ is a lower bound on the number of 6-cycles in G in which a is the only node in A. Thus, $\sum_{a\in A}\sum_{c\in C}|N_{a,c}| - 100n^2$ is a lower bound on the total number of 6-cycles in G. In other words, the total size of $N_{a,c}$ over all a, c is $O(n^2+t)$. A symmetric argument shows that the total size of $N_{b,d}$ over all b, d is also $O(n^2+t)$. Hence, the total size of all tables N, P, Q, R is bounded by $O(n^2+t)$. So our algorithm in stage 1 also takes $O(n^2 + \text{table size}) \leq O(n^2+t)$ time. This finishes the proof of Lemma 2.1.

[2] We thank an anonymous reviewer for suggesting this simpler proof; our original proof had to invoke a lemma from [41] and [10].

3 Algorithm for Listing t 6-Cycles

It remains to show how our algorithm for listing all 6-cycles (Theorem 2.1) can be modified into an algorithm for listing t 6-cycles in $\widetilde{O}(n^2 + t)$ time, for any given t.[3]

Let the nodes of $G = (V, E)$ be $v_1, v_2, \ldots, v_n$ in arbitrary order. Let G_i be the subgraph of G induced by the first i nodes $v_1, \ldots, v_i$. Note that we can decide whether the number of 6-cycles in G_i is at least t, in $\widetilde{O}(|V(G_i)|^2 + t) \leq \widetilde{O}(n^2 + t)$ time: we simply attempt to list all 6-cycles in G_i using the algorithm from Theorem 2.1, and we terminate the algorithm if the running time is too long. This allows us to use binary search to find the largest index $1 \leq i \leq n$ such that the number of 6-cycles in G_i is at most t, in $\widetilde{O}(n^2 + t)$ time. Now there are three cases:

- Case 1: $i = n$.

  This means the total number of 6-cycles in G is at most t, and can be listed by Theorem 2.1 in $\widetilde{O}(n^2 + t)$ time.

- Case 2: $i < n$.

  This means G_{i+1} contains more than t 6-cycles. We then decide whether G_{i+1} has at most $2t$ 6-cycles, in $\widetilde{O}(n^2 + t)$ time. Based on the outcome there are two cases:

  – Case 2(1): G_{i+1} has at most $2t$ 6-cycles.

    In this case we can list all 6-cycles in G_{i+1} in $\widetilde{O}(n^2 + t)$ time, and output t of them as the answer.

  – Case 2(2): G_{i+1} has more than $2t$ 6-cycles.

    Since G_i has at most t 6-cycles, we know v_{i+1} is contained in more than $2t - t = t$ 6-cycles.

    In this case, we simply apply the following standard lemma to v_{i+1}, and output t 6-cycles containing v_{i+1} in $\widetilde{O}(m + t) = \widetilde{O}(n^2 + t)$ time.

LEMMA 3.1. (SIMPLE ADAPTATION OF [4, LEMMA 3.1]) *For fixed k, given an n-node m-edge graph G with a special node v, and a parameter t, we can list t k-cycles in G containing v in $O((m + t) \log n)$ time.*

Proof Sketch. Using the color coding technique [4], we can assume the nodes in G are partitioned as $V(G) = \{v\} \sqcup V_1 \sqcup V_2 \sqcup \cdots \sqcup V_{k-1}$, and we only need to list k-cycles with nodes $v, v_1 \in V_1, \ldots, v_{k-1} \in V_{k-1}$ in order. By a dynamic-programming-like procedure in $O(m)$ time, we can compute all the vertices $v_i \in V_i$ that are reachable from v, as well as all their parents $v_{i-1} \in V_{i-1}$ reachable from v and adjacent to v_i. To report k-cycles, we start with every neighbor of v in V_{k-1}, and follow the parent pointers to go back to v using DFS. In this way we output t k-cycles in $O(kt) = O(t)$ time. □

References

[1] A. ABBOUD, K. BRINGMANN, AND N. FISCHER, *Stronger 3-sum lower bounds for approximate distance oracles via additive combinatorics*, in Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023, ACM, 2023, pp. 391–404.

[2] A. ABBOUD, K. BRINGMANN, S. KHOURY, AND O. ZAMIR, *Hardness of approximation in P via short cycle removal: cycle detection, distance oracles, and beyond*, in STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022, ACM, 2022, pp. 1487–1500.

[3] A. ABBOUD, S. KHOURY, O. LEIBOWITZ, AND R. SAFIER, *Listing 4-cycles*, CoRR, abs/2211.10022 (2022).

[4] N. ALON, R. YUSTER, AND U. ZWICK, *Color-coding*, J. ACM, 42 (1995), p. 844–856.

[5] N. ALON, R. YUSTER, AND U. ZWICK, *Finding and counting given length cycles*, Algorithmica, 17 (1997), pp. 209–223.

[3]A natural attempt is to subsample the edges of G at some rate p and obtain a subgraph G' whose expected C_6 count (which equals p^6 times the original count) is close to t, and then apply Theorem 2.1 on G'. This does not directly work since the C_6 count of G' can have high variance (for example, when some edge participates in many C_6s in G) and can be far from t with very high probability. Our approach here instead uses a binary search and is deterministic.

[6] G. J. Bezem and J. van Leeuwen, *Enumeration in graphs*, Tech. Rep. RUU-CS-87-07, Utrecht University, 1987.

[7] E. Birmelé, R. A. Ferreira, R. Grossi, A. Marino, N. Pisanti, R. Rizzi, and G. Sacomoto, *Optimal listing of cycles and st-paths in undirected graphs*, in Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013, SIAM, 2013, pp. 1884–1896.

[8] A. Björklund, R. Pagh, V. Vassilevska Williams, and U. Zwick, *Listing triangles*, in Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I, vol. 8572 of Lecture Notes in Computer Science, Springer, 2014, pp. 223–234.

[9] J. Blanusa, P. Ienne, and K. Atasu, *Scalable fine-grained parallel cycle enumeration algorithms*, in SPAA '22: 34th ACM Symposium on Parallelism in Algorithms and Architectures, Philadelphia, PA, USA, July 11 - 14, 2022, ACM, 2022, pp. 247–258.

[10] A. Bondy and M. Simonovits, *Cycles of even length in graphs*, J. Combin. Theory, Ser. B, 16 (1974), pp. 97–105.

[11] K. Bringmann and N. Carmeli, *Unbalanced triangle detection and enumeration hardness for unions of conjunctive queries*, CoRR, abs/2210.11996 (2022).

[12] K. Bringmann, N. Carmeli, and S. Mengel, *Tight fine-grained bounds for direct access on join queries*, in PODS '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022, ACM, 2022, pp. 427–436.

[13] N. Carmeli and M. Kröll, *On the enumeration complexity of unions of conjunctive queries*, ACM Trans. Database Syst., 46 (2021), pp. 5:1–5:41.

[14] N. Carmeli and L. 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, PODS 2023, Seattle, WA, USA, June 18-23, 2023, ACM, 2023, pp. 277–289.

[15] T. M. Chan and Y. Xu, *Simpler reductions from exact triangle*, CoRR, abs/2310.11575 (2023). To appear in SOSA 2024.

[16] N. Chiba and T. Nishizeki, *Arboricity and subgraph listing algorithms*, SIAM J. Comput., 14 (1985), pp. 210–223.

[17] S. Dahlgaard, M. B. T. Knudsen, and M. Stöckel, *Finding even cycles faster via capped k-walks*, in Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, New York, NY, USA, 2017, Association for Computing Machinery, p. 112–120.

[18] M. Dalirrooyfard, S. Mathialagan, V. Vassilevska Williams, and Y. Xu, *Listing cliques from smaller cliques*, CoRR, abs/2307.15871 (2023).

[19] M. Dalirrooyfard, T.-D. Vuong, and V. Vassilevska Williams, *Graph pattern detection: Hardness for all induced patterns and faster noninduced cycles*, SIAM J. Comput., 50 (2021), pp. 1627–1662.

[20] S. Deng, S. Lu, and Y. Tao, *On join sampling and the hardness of combinatorial output-sensitive join algorithms*, in Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2023, Seattle, WA, USA, June 18-23, 2023, ACM, 2023, pp. 99–111.

[21] R. Duan, H. Wu, and R. Zhou, *Faster matrix multiplication via asymmetric hashing*, in Proc. 64th IEEE Symposium on Foundations of Computer Science (FOCS), 2023.

[22] L. Duraj, K. Kleiner, A. Polak, and V. Vassilevska Williams, *Equivalences between triangle and range query problems*, in Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020, SIAM, 2020, pp. 30–47.

[23] A. Durand, *Fine-grained complexity analysis of queries: From decision to counting and enumeration*, in Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020, ACM, 2020, pp. 331–346.

[24] A. Durand and E. Grandjean, *First-order queries on structures of bounded degree are computable with constant delay*, ACM Trans. Comput. Log., 8 (2007), p. 21.

[25] P. Giscard, P. Rochet, and R. C. Wilson, *Evaluating balance on social networks from their simple cycles*, J. Complex Networks, 5 (2017), pp. 750–775.

[26] R. Grossi, *Enumeration of paths, cycles, and spanning trees*, in Encyclopedia of Algorithms, Springer, 2016, pp. 640–645.

[27] C. Jin and Y. Xu, *Removing additive structure in 3SUM-based reductions*, in Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023, ACM, 2023, pp. 405–418.

[28] D. B. Johnson, *Finding all the elementary circuits of a directed graph*, SIAM J. Comput., 4 (1975), pp. 77–84.

[29] S. Klamt and A. von Kamp, *Computing paths and cycles in biological interaction graphs*, BMC Bioinform., 10 (2009).

[30] T. Kopelowitz, S. Pettie, and E. Porat, *Higher lower bounds from the 3SUM conjecture*, in Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016, SIAM, 2016, pp. 1272–1287.

[31] L. Kowalik, *Short cycles in planar graphs*, in Graph-Theoretic Concepts in Computer Science, 29th International Workshop, WG 2003, Elspeet, The Netherlands, June 19-21, 2003, Revised Papers, vol. 2880 of Lecture Notes in Computer Science, Springer, 2003, pp. 284–296.

[32] A. Lincoln and N. Vyas, *Algorithms and lower bounds for cycles and walks: Small space and sparse graphs*, in 11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA, vol. 151 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 11:1–11:17.

[33] P. Mateti and N. Deo, *On algorithms for enumerating all circuits of a graph*, SIAM J. Comput., 5 (1976), pp. 90–99.

[34] M. Pătraşcu, *Towards polynomial lower bounds for dynamic problems*, in Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010, ACM, 2010, pp. 603–610.

[35] R. C. Read and R. E. Tarjan, *Bounds on backtrack algorithms for listing cycles, paths, and spanning trees*, Networks, 5 (1975), pp. 237–252.

[36] Y. Strozecki, *Enumeration complexity*, Bull. EATCS, 129 (2019).

[37] R. E. Tarjan, *Enumeration of the elementary circuits of a directed graph*, SIAM J. Comput., 2 (1973), pp. 211–216.

[38] V. Vassilevska Williams, *Efficient Algorithms for Path Problems in Weighted Graphs*, PhD thesis, Carnegie Mellon University, 2008.

[39] V. Vassilevska Williams and Y. Xu, *Monochromatic triangles, triangle listing and APSP*, in Proc. 61st IEEE Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 2020, pp. 786–797.

[40] V. Vassilevska Williams, Y. Xu, Z. Xu, and R. Zhou, *New bounds for matrix multiplication: from Alpha to Omega*, in Proc. 35th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2024. To appear.

[41] R. Yuster and U. Zwick, *Finding even cycles even faster*, SIAM Journal on Discrete Mathematics, 10 (1997), pp. 209–222.