Fully Dynamic Four-Vertex Subgraph Counting

Kathrin Hanauer
University of Vienna, Faculty of Computer Science, Vienna, Austria

Monika Henzinger
University of Vienna, Faculty of Computer Science, Vienna, Austria

Qi Cheng Hua
University of Vienna, Faculty of Computer Science, Vienna, Austria

Abstract

This paper presents a comprehensive study of algorithms for maintaining the number of all connected four-vertex subgraphs in a dynamic graph. Specifically, our algorithms maintain the number of paths of length three in deterministic amortized $\mathcal{O}(m^{\frac{1}{2}})$ update time, and any other connected four-vertex subgraph which is not a clique in deterministic amortized update time $\mathcal{O}(m^{\frac{2}{3}})$. Queries can be answered in constant time. We also study the query times for subgraphs containing an arbitrary edge that is supplied only with the query as well as the case where only subgraphs containing a vertex s that is fixed beforehand are considered. For length-3 paths, paws, 4-cycles, and diamonds our bounds match or are not far from (conditional) lower bounds: Based on the OMv conjecture we show that any dynamic algorithm that detects the existence of paws, diamonds, or 4-cycles or that counts length-3 paths takes update time $\Omega(m^{1/2-\delta})$.

Additionally, for 4-cliques and all connected induced subgraphs, we show a lower bound of $\Omega(m^{1-\delta})$ for any small constant $\delta > 0$ for the amortized update time, assuming the static combinatorial 4-clique conjecture holds. This shows that the $\mathcal{O}(m)$ algorithm by Eppstein et al. [9] for these subgraphs cannot be improved by a polynomial factor.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Dynamic graph algorithms

Keywords and phrases Dynamic Graph Algorithms, Subgraph Counting, Motif Search

Related Version A short version is to appear at SAND'22.

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101019564, "The Design of Modern Fully Dynamic Data Structures (MoDynStruct)"), as well as from the Austrian Science Fund (FWF) and netIDEE SCIENCE project P 33775-N.

Acknowledgements The authors want to thank Leonhard Paul Sidl for careful proofreading.

1 Introduction

Detecting or counting subgraphs is an important question in social network analysis, where dense subgraphs usually represent communities, as well as in telecommunication network surveillance, and computational biology. This can also be seen in a recent study by Sahu et al. [21]: finding and counting fixed subgraphs was the fourth most popular graph computation in practice, only superseded by finding connected components, computing shortest paths, and answering queries about the degree of neighbors. Furthermore the same study showed that the dynamic setting is important in practice as 65% of the graphs were dynamic. Thus, the goal of this paper is to advance the study of subgraph counting problems in dynamic graphs.

Algorithmic problems in dynamic graphs are usually modeled by the following data structure question. Given a potentially non-empty initial graph and a fixed subgraph pattern

Table 1 Upper and conditional lower bounds on the time per update and query for counting different subgraphs, where $\delta > 0$ is an arbitrarily small constant and $h \in \mathcal{O}(\sqrt{m})$. Update times are amortized, query times are worst-case. Results in blue are new or improved.
* Read: For polynomial preprocessing time and $\mathcal{O}(\cdot)$ query time, the update time is $\Omega(\cdot)$.
† The previous space complexity for 3-cycles and length-3 paths was $\mathcal{O}(mh)$ with amortized update time $\mathcal{O}(h)$ [10] and $\mathcal{O}(m)$ with amortized update time $\mathcal{O}(m^{\frac{1}{2}})$ for 3-cycles [14]; for (other) 4-vertex subgraphs, it was $\mathcal{O}(mh^2)$ with amortized update time $\mathcal{O}(h^2)$ [9].
^{a} Thm 4, ^{b} Thm 5, ^{c} Cor. 19, ^{d} Thm 20, ^{e} Thm 21, ^{f} Cor. 25, ^{g} Thm 26, $^\alpha$ [10], $^\beta$ [9], $^\zeta$ [7].

	Lower Bounds*		Update Time		Query Time		Space†
Subgraph	**Update**	**Query**	**ours**	**previous**	**all**	$e \in E$	
Non-induced subgraphs and s-subgraphs							
connected, $n = 4$			$\mathcal{O}(m)^a$		$\mathcal{O}(1)$	$\mathcal{O}(1)^a$	$\mathcal{O}(1)/\mathcal{O}(m)^a$
claw ⅄	$\Omega(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)^{\alpha\beta}$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
length-3 path ⟋	$\Omega(m^{\frac{1}{2}-\delta})^g$	$\mathcal{O}(m^{1-\delta})^g$	$\mathcal{O}(m^{\frac{1}{2}})^b$	$\mathcal{O}(h)^{\alpha\beta}$	$\mathcal{O}(1)$	$\mathcal{O}(m^{\frac{1}{2}})^b$	$\mathcal{O}(\min(n^2, m^{1.5}))^b$
paw ⊲	$\Omega(m^{\frac{1}{2}-\delta})^g$	$\mathcal{O}(m^{1-\delta})^g$	$\mathcal{O}(m^{\frac{2}{3}})^b$	$\mathcal{O}(h^2)^\beta$	$\mathcal{O}(1)$	$\mathcal{O}(m^{\frac{2}{3}})^b$	$\mathcal{O}(n^2)^b$
3-cycle △	$\Omega(m^{\frac{1}{2}-\delta})^g$	$\mathcal{O}(m^{1-\delta})^g$	$\mathcal{O}(m^{\frac{1}{2}})^c$	$\mathcal{O}(h)^\alpha$	$\mathcal{O}(1)$	$\mathcal{O}(m^{\frac{1}{2}})^c$	$\mathcal{O}(\min(n^2, m^{1.5}))^c$
4-cycle ◇	$\Omega(m^{\frac{1}{2}-\delta})^g$	$\mathcal{O}(m^{1-\delta})^g$	$\mathcal{O}(m^{\frac{2}{3}})^b$	$\mathcal{O}(h^2)^\beta$	$\mathcal{O}(1)$	$\mathcal{O}(m^{\frac{2}{3}})^b$	$\mathcal{O}(n^2)^b$
k-cycle, $k \geq 5$	$\Omega(m^{\frac{1}{2}-\delta})^g$	$\mathcal{O}(m^{1-\delta})^g$					
diamond ⬖	$\Omega(m^{\frac{1}{2}-\delta})^g$	$\mathcal{O}(m^{1-\delta})^g$	$\mathcal{O}(m^{\frac{2}{3}})^b$	$\mathcal{O}(h^2)^\beta$	$\mathcal{O}(1)$	$\mathcal{O}(m^{\frac{2}{3}})^b$	$\mathcal{O}(\min(nm, m^{\frac{5}{3}}))^b$
4-clique ⊠	$\Omega(m^{1-\delta})^e$	$\mathcal{O}(m^{2-\delta})^e$	$\mathcal{O}(m)^\zeta$	$\mathcal{O}(h^2)^\beta$	$\mathcal{O}(1)$	$\mathcal{O}(m)^\zeta$	$\mathcal{O}(1)$
s-claw ⅄	$\Omega(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)^d$		$\mathcal{O}(1)^d$		$\mathcal{O}(1)^d$
s-length-3-path ⟋	$\Omega(m^{\frac{1}{2}-\delta})^g$	$\mathcal{O}(m^{1-\delta})^g$	$\mathcal{O}(m^{\frac{1}{2}})^d$		$\mathcal{O}(1)^d$		$\mathcal{O}(n)^d$
s-paw ⊲	$\Omega(m^{\frac{1}{2}-\delta})^g$	$\mathcal{O}(m^{1-\delta})^g$	$\mathcal{O}(m^{\frac{2}{3}})^d$		$\mathcal{O}(1)^d$		$\mathcal{O}(n^2)^d$
s-3-cycle △	$\Omega(m^{\frac{1}{2}-\delta})^g$	$\mathcal{O}(m^{1-\delta})^g$	$\mathcal{O}(m^{\frac{1}{2}})^d$		$\mathcal{O}(1)^d$		$\mathcal{O}(n)^d$
s-4-cycle ◇			$\mathcal{O}(m^{\frac{2}{3}})^d$		$\mathcal{O}(1)^d$		$\mathcal{O}(n^2)^d$
s-k-cycle, $k \geq 5$, odd	$\Omega(m^{\frac{1}{2}-\delta})^g$	$\mathcal{O}(m^{1-\delta})^g$					
s-diamond ⬖	$\Omega(m^{\frac{1}{2}-\delta})^g$	$\mathcal{O}(m^{1-\delta})^g$	$\mathcal{O}(m^{\frac{2}{3}})^d$		$\mathcal{O}(1)^d$		$\mathcal{O}(n^2)^d$
s-4-clique ⊠	$\Omega(m^{\frac{1}{2}-\delta})^g$	$\mathcal{O}(m^{1-\delta})^g$	$\mathcal{O}(m)^d$		$\mathcal{O}(1)^d$		$\mathcal{O}(1)^d$
Induced subgraphs							
connected, $n = 4$	$\Omega(m^{1-\delta})^e$	$\mathcal{O}(m^{2-\delta})^e$	$\mathcal{O}(m)^f$	$\mathcal{O}(h^2)^\beta$	$\mathcal{O}(1)$		$\mathcal{O}(1)^f$

$\mathcal{P}$ (such as a k-clique) maintain a data structure that allows the following updates to the current graph G:

- Insert(u, v): Insert the edge $\{u, v\}$ into G.
- Delete(u, v): Delete the edge $\{u, v\}$ from G.
- Query(): Return the number of subgraphs of pattern $\mathcal{P}$ in G.

Given a subgraph pattern $\mathcal{P}$ that is not a clique, there are two variants of this problem: One variant, called *induced subgraph counting*, counts a subgraph if it is exactly equivalent to $\mathcal{P}$ and does *not* contain any additional edges. In the *non-induced* version, a subgraph is counted if it contains $\mathcal{P}$ and potentially additional edges (but *not* additional vertices). Eppstein and Spiro [10] studied subgraph counting for all possible connected three-vertex patterns in both the induced and the non-induced variant and gave a dynamic algorithm with amortized update time $\mathcal{O}(h)$, where h is the *h-index* of G, i.e., the maximum number such that the graph contains h vertices of degree at least h. Note that h is $\mathcal{O}(\sqrt{m})$, where m always is the current number of edges in the graph.

There are six connected graphs on four vertices, which we refer to as *length-3 path*, *claw*, *paw*, *4-cycle*, *diamond*, and *4-clique*. Eppstein et al. [9] extended the method of [10] to maintain counts of any (induced and non-induced) connected four-vertex subgraph in amortized time $\mathcal{O}(h^2) = \mathcal{O}(m)$ and $\mathcal{O}(mh^2)$ space. This paper contains a comprehensive study of the complexity of dynamically counting all possible connected four-vertex subgraphs. We present new improved dynamic algorithms and give the first conditional lower bounds.

Upper bounds. We show how to maintain the number of any connected four-vertex non-induced subgraph that is not a clique (such as a paw, a 4-cycle, or a diamond) in update time $\mathcal{O}(m^{2/3})$ and at most $\mathcal{O}(nm)$ space. For graphs with an h-index larger than $\mathcal{O}(m^{1/3})$, our algorithms are hence faster than the $\mathcal{O}(h^2)$ algorithm by Eppstein et al. [9]. Besides, our data structure can also be used to count all s-triangles, i.e., triangles that contain a fixed vertex s, in $\mathcal{O}(m^{1/2})$ update time, constant query time, and $\mathcal{O}(n)$ space, and likewise for s-length-3 paths. The update time is in $\mathcal{O}(m^{2/3})$ for s-4-cycles, s-paws, and s-diamonds, with $\mathcal{O}(n^2)$ space, and $\mathcal{O}(m)$ for s-4-cliques with constant space. For $\varepsilon \in [0, 1]$, our data structure supports queries on the number of triangles containing an arbitrary vertex or edge in $\mathcal{O}(\min(m^{2\varepsilon}, n^2))$ or $\mathcal{O}(\min(m^{1-\varepsilon}, n))$ worst-case time, respectively, with an amortized update time of $\mathcal{O}(m^{\max(\varepsilon,1-\varepsilon)})$ or $\mathcal{O}(m^{\varepsilon})$, respectively, and $\mathcal{O}(n^2)$ space. We also show how to maintain length-3 paths in time $\mathcal{O}(m^{1/2})$, but this result was already stated in [9]. See Table 1 for an overview. All our algorithms are deterministic and the running time bounds are amortized unless stated otherwise.

Lower bounds. We also give the first conditional lower bounds for counting various four-vertex subgraphs based on two popular hypotheses: the Online Boolean Matrix-Vector Multiplication (OMv) conjecture [12] and the Combinatorial k-Clique hypothesis. In the OMv conjecture we are given a Boolean $n \times n$ matrix M that can be preprocessed. Then, an online sequence of vectors $v_1, \ldots, v_n$ is presented and the goal is to compute each Boolean product Mv_i (using conjunctions and disjunctions) before seeing the next vector v_{i+1}.

▶ **Conjecture 1** (OMv). *For any constant $\delta > 0$, there is no $\mathcal{O}(n^{3-\delta})$-time algorithm that solves OMv with error probability at most $1/3$ in the word-RAM model with $O(\log n)$ bit words.*

Based on the OMv conjecture we show that *detecting* (with probability at least $2/3$ in the word-RAM model with $O(\log n)$ bit words) the existence of (non-induced) paws, diamonds, 4-cliques, or k-cycles for any $k \geq 3$ in a graph with edge insertions and deletions takes amortized update time $\Omega(m^{1/2-\delta})$ or query time $\Omega(m^{1-\delta})$ if only polynomial preprocessing time is allowed. This lower bound applies also to the worst-case update time of any insertions-only or deletions-only algorithm. Note that this lower bound does not only apply to *counting* the number of such subgraphs but already to *detecting* whether such a subgraph exists. Let s be a fixed vertex in the graph. The same lower bounds apply to algorithms that detect whether a diamond, 4-clique, or k-cycle with odd k containing s exists. Finally, we also show a lower bound for *counting* the number of length-3 paths and length-3 s-paths. We remark that the conditional lower bounds for (s-)3-cycles were already known before [12].

We also use the Combinatorial k-Clique hypothesis which is defined as follows and has become popular in recent years (e.g. [20, 1, 5, 4]).

▶ **Conjecture 2** (Combinatorial k-Clique). *For any constant $\delta > 0$, for an n-vertex graph there is no $\mathcal{O}(n^{k-\delta})$ time combinatorial algorithm for k-clique detection with error probability at most $1/3$ in the word-RAM model with $\mathcal{O}(\log n)$ bit words.*

Let $\delta > 0$ be a small constant. Based on the 4-clique conjecture we show that (with probability at least 2/3 in the word-RAM model with $O(\log n)$ bit words) there does not exist a combinatorial algorithm that counts *any connected induced four-vertex subgraph* in a dynamic graph with amortized update time $O(n^{4-2\delta}/m)$, which is $O(m^{1-\delta})$, and query preprocessing time $O(n^{4-2\delta})$. This bound applies also to any insertions-only algorithm. The bound can be extended to any k-clique with $k > 4$ showing that the amortized update time is $\Omega(n^{k-2\delta}/m)$ with $\Omega(n^{k-2\delta})$ preprocessing and query time.

Technical contribution. For the upper bounds we extend and improve upon Eppstein et al. [9] both with respect to running time and space. The high-level idea is as follows: We partition the vertices into (few) *high-degree* and (many) *low-degree* vertices and then maintain for each vertex, vertex pair, or vertex triple certain information in a data structure such as the number of certain paths up to length 3 that contain low-degree vertices. When an edge $\{u, v\}$ is updated, four-vertex subgraphs that contain u, v, and two other low-degree vertices can be quickly counted using the information in the data structure. On the other side, subgraphs that contain a high-degree vertex in addition to u and v can often be counted "from scratch" after each update as there are few high-degree vertices. The more challenging case is the situation where relationships involving two or more high-degree vertices in the subgraph need to be checked or maintained. How to deal with this depends on the subgraph to count. For diamonds, e.g., this requires to keep certain information about triples of vertices.

For the conditional lower bounds based on the combinatorial k-clique conjecture we first directly deduce the lower bound for incremental 4-clique counting. Then we use the fact that (a) we have a lower bound for 4-cliques, (b) we developed algorithms with $\mathcal{O}(m^{2/3})$ update time for all non-induced subgraphs, and (c) there exist "counting formulas" that allow to compute the number of any induced subgraph based on the number of 4-cliques and the number of non-induced subgraphs. Thus, if the number of an induced subgraph pattern could be computed in $\mathcal{O}(m^{1-\delta})$ time per update for some small $\delta > 0$ then we could use the corresponding counting formula and our algorithms for non-induced subgraphs to dynamically maintain the number of 4-cliques, contradicting our dynamic lower bound for 4-cliques.

For the conditional lower bounds based on the OMv conjecture we construct for each subgraph pattern $\mathcal{P}$ based on an 1-uMv instance (which is a variant of OMv) a suitable graph based on $\mathcal{P}$ with $\mathcal{O}(n)$ vertices and $\mathcal{O}(n^2)$ edges such that detecting the existence of the (non-induced) version of $\mathcal{P}$ in the graph equals finding the answer for the 1-uMv instance. Then the lower bound follows as in [12]. The challenge is to construct such a graph. We show how to do this for detecting non-induced (s)-paws, (s)-diamonds, (s)-4-cliques, and (s)-k-cycles for $k \geq 3$ and for counting non-induced length-3 (s)-paths.

Our paper gives in Section 2 the preliminaries, in Section 3 our new algorithms, and in Section 4 our lower bounds. Proofs that have been omitted in the main part are given in the appendix.

2 Preliminaries

Basic Definitions

We consider an undirected dynamic graph $G = (V, E)$ and use n to denote the number of vertices and m for the current number of edges. Two vertices $u \neq v$ are *adjacent* if there is an edge $e = \{u, v\} \in E$. In this case, u and v are *incident* to e. The *neighborhood* $N(v)$ of a vertex v is defined as $\{u \mid \{u, v\} \in E\}$ and v's *degree* is $\deg(v) = |N(v)|$. As a shorthand notation to exclude just one vertex, we use $N_{\bar{w}}(v) = N(v) \setminus \{w\}$ and $\deg_{\bar{w}}(v) = |N_{\bar{w}}(v)|$, i.e., $\deg_{\bar{w}}(v) = \deg(v) - 1$ if $w \in N(v)$ and $\deg_{\bar{w}}(v) = \deg(v)$ otherwise. A *k-path* (also length-k path) is a sequence of distinct edges $\langle\{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{k-1}, v_k\}\rangle$ of length k, where $v_i \neq v_j$ for all $0 \leq i, j \leq k$. A *k-cycle* is a k-path where as an only exception the first vertex equals the last, i.e., $v_0 = v_k$. A 3-cycle is also called *triangle*. A *claw* is a graph consisting of a vertex x, called the *central* vertex, and three edges incident to it. A *paw* is a graph consisting of a triangle together with an additional edge attached to one of the vertices of the triangle. This vertex is called the *central* vertex of the paw and the additional edge the *arm*. A *diamond* is a 4-cycle with a *chord*, i.e., an additional edge connecting one of the two pairs of non-adjacent vertices, which creates two triangles sharing the chordal edge. A *k-clique* is the complete graph K_k on k vertices.

A graph $G' = (V', E')$ is a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E$. A subgraph $G' = (V', E')$ is said to be *induced* if $E' = \{\{u, v\} \in E \mid u, v \in V'\}$. The term *non-induced subgraph* or just *subgraph* without an adjective refers to all subgraphs, induced and not. For a static graph $\mathcal{P}$, also called *pattern*, we denote by $\mathbf{c}(G, \mathcal{P})$ the number of (non-induced) subgraphs of G that are isomorphic to $\mathcal{P}$, and by $\mathbf{c_I}(G, \mathcal{P})$ the number of induced subgraphs that are isomorphic to $\mathcal{P}$, in each case divided by the number of automorphisms of $\mathcal{P}$. For a vertex $s \in V$, we further denote by $\mathbf{c}(G, \mathcal{P}, s)$ the number of non-induced subgraphs of G that are isomorphic to $\mathcal{P}$ and contain s. We denote by $\mathbb{P}_k$ the set of connected subgraphs on k vertices. There are six connected graphs in $\mathbb{P}_4$, which we refer to as *length*-3 *path*, *claw*, *paw*, 4-*cycle* ◇, *diamond*, and 4-*clique* ⊠.

Given a pattern $\mathcal{P}$, we study the problem of *maintaining* the number of occurrences of $\mathcal{P}$ as a subgraph or induced subgraph of G, called the *non-induced subgraph count* $\mathbf{c}(G, \mathcal{P})$ or the *induced subgraph count* $\mathbf{c_I}(G, \mathcal{P})$, respectively, of $\mathcal{P}$ in G, and the analogous problem of *maintaining* the count of non-induced subgraphs containing a specific predefined vertex $v \in V$ or edge $e \in E$, $\mathbf{c}(G, \mathcal{P}, v)$ or $\mathbf{c}(G, \mathcal{P}, e)$, respectively. Unless stated otherwise, *maintaining* a count implies that we can retrieve it by a query in constant time. We also study the closely related problem of *querying* the number of subgraphs containing a specific vertex or edge that is given only with the query.

Further Related Work

Detecting (or counting) subgraphs, also known as the *subgraph isomorphism* problem, generalizes the *clique* or *Hamiltonian cycle* problem and is hence $\mathcal{NP}$-hard. Nevertheless, it can be solved efficiently if the subgraphs to detect or count are restricted.

Static algorithms. Algorithms counting numbers of subgraphs and solving related problems have been studied extensively for static graphs. Alon, Yuster and Zwick [2] developed an algorithm to count the number of triangles and other circles (up to seven vertices) in a graph in time $\mathcal{O}(n^\omega)$, where n denotes the number of vertices and $\omega < 2.373$ [19] is the fast matrix multiplication exponent, i.e., the smallest value such that two $n \times n$ matrices can be multiplied in $\mathcal{O}(n^\omega)$ time. Kloks, Kratsch and Müller [16] showed how to compute the number of 4-cliques in time $\mathcal{O}(m^{(\omega+1)/2})$ (m denotes the number of edges) and the number of any other subgraph of size 4 in time $\mathcal{O}(n^\omega + m^{(\omega+1)/2})$. There is also a large body of work on parallel algorithms for counting subgraphs (see e.g. [18]) and to develop approximate algorithms in the streaming setting (see e.g. [6, 3]).

Dynamic algorithms. A more recent development is counting subgraph numbers for dynamic graphs. Kara et al. [14] provided an algorithm for counting triangles in amortized time

$\mathcal{O}(\sqrt{m})$ per update and $\mathcal{O}(m)$ space, which can also enumerate them with constant time delay. Dhulipala et al. [7] extended it to a batch-dynamic parallel algorithm with $\mathcal{O}(\Delta\sqrt{\Delta + m})$ amortized work and $\mathcal{O}(\text{polylog}(\Delta + m))$ depth *w.h.p.* for a batch of Δ updates. Based on a static algorithm to enumerate cliques, they show how to obtain a dynamic algorithm for maintaining the number of k-cliques for a fixed $k > 3$ with expected $\mathcal{O}(\Delta(m + \Delta)\alpha^{k-4})$ work and $\mathcal{O}(\log^{-2} n)$ depth *w.h.p.* per update and $\mathcal{O}(m + \Delta)$ space, where $\alpha \in \mathcal{O}(\sqrt{m})$ is the arboricity of the graph. They also give a parallel fast matrix multiplication algorithm with $\mathcal{O}(\min(\Delta m^{(2k-1)\omega_p/(3\omega_p+3)}, (\Delta + m)^{2(k+1)\omega_p/(3\omega_p+3)}))$ amortized work and $\mathcal{O}(\log(\Delta + m))$ depth, with parallel matrix multiplication constant ω_p. Eppstein et al. [9] also count all three-vertex subgraphs in directed graphs in amortized time $\mathcal{O}(h)$. For specific graph classes, namely *bounded expansion* graphs, Dvorak and Tuma [8] gave a different algorithm for maintaining counts for arbitrary graph patterns $\mathcal{P}$ of k vertices the number of induced subgraphs of pattern $\mathcal{P}$ in amortized time $\mathcal{O}(\log^{(k^2-k)/2-1} n)$ and in amortized time $\mathcal{O}(n^{\epsilon})$ for any constant $\epsilon > 0$ in *no-where dense* graphs.

In recent subsequent work [13], it was shown that counting 4-cycles is hard also in random graphs, i.e., with $\Omega(m^{1/2-\delta})$ update time or $\Omega(m^{1-\delta})$ query time.

3 New Counting Algorithms for Subgraphs on Four Vertices

Counting the number of claws [10] and 4-cliques is fairly straightforward [7]. The latter extends to 4-vertex subgraphs in general, where all work is either done during the updates or queries.

▶ **Observation 3** ([10]). *Let $G = (V, E)$ be a dynamic graph and $\mathcal{P}$ be the claw. Then, $\mathbf{c}(G, \mathcal{P}) = \sum_{v \in V, \deg(v) \geq 3} \binom{\deg(v)}{3}$. The count can be initialized in $\mathcal{O}(n)$ time and maintained in constant time and space. We can query $\mathbf{c}(G, \mathcal{P}, \{u, v\})$ for an arbitrary edge $\{u, v\} \in E$ in constant time.*

▶ **Theorem 4.** *Let $G = (V, E)$ be a dynamic graph and $\mathcal{P} \in \mathbb{P}_4$. We can*
(i) *maintain $\mathbf{c}(G, \mathcal{P})$ in worst-case $\mathcal{O}(m)$ update time and constant space.*
(ii) *query $\mathbf{c}(G, \mathcal{P}, \{u, v\})$ for an arbitrary edge $\{u, v\} \in E$ in worst-case $\mathcal{O}(m)$ time, with constant update time and space.*
(iii) *query $\mathbf{c}(G, \mathcal{P}, \{u, v\})$ for an arbitrary edge $\{u, v\} \in E$ in worst-case constant time, with $\mathcal{O}(m)$ worst-case update time and $\mathcal{O}(m)$ space.*

For connected 4-vertex subgraphs other than the claw and the clique, we can invest space to achieve speedups in running time. In the following, we present our data structure and show how to update it efficiently. We then use different parts of the data structure to count different subgraphs. Specifically, we prove the following result:

▶ **Theorem 5.** *Let G be a dynamic graph and $\mathcal{P} \in \mathbb{P}_4$. We can maintain $\mathbf{c}(G, \mathcal{P})$ in*
(i) *amortized $\mathcal{O}(\sqrt{m})$ update time with $\mathcal{O}(\min(m^{1.5}, n^2))$ space and query $\mathbf{c}(G, \mathcal{P}, e)$ for an arbitrary edge $e \in E$ in worst-case $\mathcal{O}(\sqrt{m})$ time if $\mathcal{P}$ is the length-3 path,*
(ii) *amortized $\mathcal{O}(m^{2/3})$ update time with $\mathcal{O}(n^2)$ space and query $\mathbf{c}(G, \mathcal{P}, e)$ for an arbitrary edge $e \in E$ in worst-case $\mathcal{O}(m^{2/3})$ time if $\mathcal{P}$ is the paw or the 4-cycle,*
(iii) *amortized $\mathcal{O}(m^{2/3})$ update time with $\mathcal{O}(\min(nm, m^{5/3}))$ space and query $\mathbf{c}(G, \mathcal{P}, e)$ for an arbitrary edge $e \in E$ in worst-case $\mathcal{O}(m^{2/3})$ time if $\mathcal{P}$ is the diamond.*

Our algorithm makes use of a standard technique in dynamic graph algorithms that partitions vertices into *high-degree* and *low-degree* vertices. We adapt it to our needs as follows: Let m_0 be the number of edges of G at construction or when recomputing from

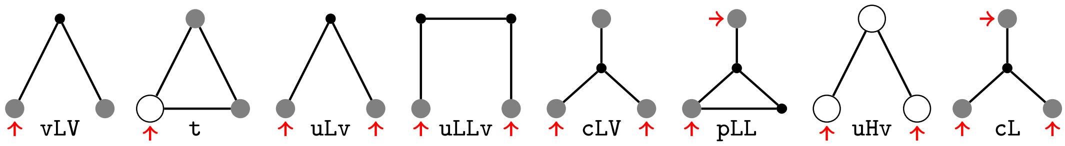

Figure 1 Subgraph structures of $\mathcal{D}_\varepsilon$. Small and filled vertices have low degree, large and empty vertices high degree, medium-sized and shaded vertices can have either high or low degree. Anchors are marked by arrows.

scratch and let $M = 2m_0$. A recomputation from scratch and re-initialization of the partition is triggered whenever the current number of edges $m < \lfloor \frac{M}{4} \rfloor$ or $m \geq M$. Initially and at each recomputation from scratch, a vertex $v \in V$ is classified as *high-degree* and added to partition $\mathcal{H}$ if $\deg(v) \geq \theta$ and otherwise as *low-degree* and added to partition $\mathcal{L} := V \setminus \mathcal{H}$, for some threshold θ. As G evolves, a high-degree vertex v is reclassified as low and moved to $\mathcal{L}$ only if $\deg(v) < \frac{1}{2}\theta$. Vice-versa, a low-degree vertex v is reclassified as high and moved to $\mathcal{H}$ only if $\deg(v) \geq \frac{3}{2}\theta$. We call such a partition $(\mathcal{H}, \mathcal{L})$ a *dynamic vertex partition with threshold* θ. If $\theta = M^\varepsilon$ for some $\varepsilon \in [0, 1]$, we call the partition an *ε-partition.*

▶ **Theorem 6** (ε-Partition [15]). *Let $\varepsilon \in [0, 1]$ and consider an ε-partition $(\mathcal{H}, \mathcal{L})$ for a dynamic graph $G = (V, E)$. Then, $|\mathcal{H}| \in \mathcal{O}(m^{1-\varepsilon})$. The partition can be constructed in $\mathcal{O}(n)$ time and maintained in amortized constant time per update with amortized $\mathcal{O}(m^{-\varepsilon})$ changes to the partition per update and $\Omega(m)$ updates between two recomputations from scratch. The required space is $\mathcal{O}(n)$.*

Data Structure $\mathcal{D}_\varepsilon$

We assume that the algorithm can access the degree of a vertex v in constant time and, for each pair of vertices u, v determine in constant time whether $\{u, v\} \in E$. In addition, we maintain the following data structure $\mathcal{D}_\varepsilon$ or a subset of it, if we are not only interested in counting some specific subgraphs on four vertices. All subgraph structures that are part of $\mathcal{D}_\varepsilon$ are non-induced. See Figure 1 for visualizations.

- an ε-partition $(\mathcal{H}, \mathcal{L})$
- For each vertex $v \in V$: `vLV[`v`]`: the number of 2-paths $\langle\{v, x\}, \{x, y\}\rangle$ with $x \in \mathcal{L}$
- For each vertex $v \in \mathcal{H}$: `t[`v`]`: the number of 3-cycles $\langle\{v, x\}, \{x, y\}, \{y, v\}\rangle$
- For each distinct, unordered pair of vertices $u, v \in V$:
  - `uLv[`u, v`]`: the number of 2-paths $\langle\{u, x\}, \{x, v\}\rangle$ with $x \in \mathcal{L}$
  - `uLLv[`u, v`]`: the number of length-3 paths $\langle\{u, x\}, \{x, y\}, \{y, v\}\rangle$ with $x, y \in \mathcal{L}$
  - `cLV[`u, v`]`: the number of claws with a central vertex $x \in \mathcal{L}$, $u \neq x \neq v$
  - `pLL[`u, v`]`: the number of paws with a central vertex $x \in \mathcal{L}$, $u \neq x \neq v$, u or v at the other end of the arm, and fourth vertex $y \in \mathcal{L}$
- For each distinct, unordered pair of vertices $u, v \in \mathcal{H}$: `uHv[`u, v`]`: the number of length-2 paths $\langle\{v, x\}, \{x, v\}\rangle$ with $x \in \mathcal{H}$
- For each distinct, unordered triple of vertices $u, v, w \in V$: `cL[`u, v, w`]`: the number of claws with a fourth vertex $x \in \mathcal{L}$ at the center

For each auxiliary subgraph whose count is maintained by the data structure, we call vertices of the set that acts as key *anchors*, e.g., u and v are anchors for the 2-paths $\langle\{u, x\}, \{x, v\}\rangle$ with $x \in \mathcal{L}$, which are counted by `uLv[`u, v`]`. We use hash tables with $\mathcal{O}(1)$ amortized access time and only store non-zero counts. Note that `uLv`, `uLLv`, `pLL`, and `cL` correspond to s_2, s_3, s_5, and s_7, respectively, in the algorithm by Eppstein et al. [9], whereas `vLV`, `t`, and `cLV` are modifications of s_0, s_1, and s_4, and `uHv` has no equivalent at all. However, Eppstein et al. [9] use a different partitioning scheme, where there are at most $\mathcal{O}(h)$ vertices of degree $\Omega(h)$, whereas in our case, there are at most $\mathcal{O}(m^{1-\varepsilon})$ vertices of degree $\Omega(m^{\varepsilon})$, which requires a different running time analysis also for the common auxiliary counts.

We generally assume that in case of an edge insertion, the auxiliary counts are updated immediately *before* the counts of interest, and in reverse order for an edge deletion. The update of the ε-partition can either happen first or last (but not in between). We also assume that we start with an empty graph and all counts are initialized to zero.

Maintaining the Data Structure $\mathcal{D}_\varepsilon$

Given a dynamic graph $G = (V, E)$ and $\varepsilon \in [0, 1]$, we show how the components of the data structure $\mathcal{D}_\varepsilon$ can be updated after an edge insertion or deletion and if a vertex changes partition. We start with a helper lemma:

▶ **Lemma 7.** *Let* `aux` *be an auxiliary subgraph count in $\mathcal{D}_\varepsilon$ with worst-case update time $\mathcal{E}_{aux}$ after an edge insertion or deletion, worst-case update time $\mathcal{V}_{aux}$ after a vertex changes partition, and $\mathcal{S}_{aux}$ space. Then, $\mathcal{D}_\varepsilon$ with* `aux` *can be maintained in amortized update time $\mathcal{O}(\mathcal{E}_{aux} + \mathcal{V}_{aux} \cdot m^{-\varepsilon})$ with $\mathcal{O}(n + \mathcal{S}_{aux})$ space.*

Proof. By Theorem 6, the ε-partition can be maintained in $\mathcal{O}(n)$ space and such that there are $\Omega(m)$ updates between two recomputations of the partition from scratch. After each such complete repartitioning, we set `aux` to zero and re-insert all edges one-by-one. The total recomputation time hence is $\mathcal{O}(m \cdot \mathcal{E}_{\mathtt{aux}})$ and amortization over $\Omega(m)$ edge updates results in an amortized edge update time of $\mathcal{O}(\mathcal{E}_{\mathtt{aux}})$. By Theorem 6, there are amortized $\mathcal{O}(m^{-\varepsilon})$ vertices changing partition per edge update, hence the claim follows. ◀

As the insertion and deletion operations are entirely symmetric and only differ in whether a certain amount is added or subtracted from the stored counts, we only give the details for edge insertions in the following. Similarly, we only consider the case that a vertex v changes from $\mathcal{L}$ to $\mathcal{H}$; the other case is symmetric. Note that if v is about to change partitions, $\deg(v) \in \Theta(m^{\varepsilon})$.

▶ **Lemma 8.** *$\mathcal{D}_\varepsilon$ with* `vLV` *can be maintained in amortized $\mathcal{O}(m^{\varepsilon})$ update time and $\mathcal{O}(n)$ space.*

Proof. Let $\{u, v\}$ be the newly inserted edge. If $u \in \mathcal{L}$ ($v \in \mathcal{L}$), increase `vLV[`w`]` by one for each $w \in N_{\bar{v}}(u)$ ($w \in N_{\bar{u}}(v)$) and increase `vLV[`v`]` by $\deg(u) - 1$ (`vLV[`u`]` by $\deg(v) - 1$). This takes $\mathcal{O}(m^{\varepsilon})$ time.

If a vertex $v \in \mathcal{L}$ changes to $\mathcal{H}$, this affects all length-2 paths where v is the central, low-degree vertex. For each neighbor $w \in N(v)$, decrease `vLV[`w`]` by $\deg(v) - 1$. The running time is $\mathcal{O}(\deg(v)) = \mathcal{O}(m^{\varepsilon})$.

As each vertex may be adjacent to at least one low-degree vertex, the space requirement is $\mathcal{O}(n)$. By Lemma 7, $\mathcal{D}_\varepsilon$ with `vLV` can hence be maintained in $\mathcal{O}(m^{\varepsilon})$ amortized update time and $\mathcal{O}(n)$ space. ◀

▶ **Lemma 9.** *$\mathcal{D}_\varepsilon$ with* `uLv` *can be maintained in amortized $\mathcal{O}(m^{\varepsilon})$ time per update and $\mathcal{O}(\min(m^{1+\varepsilon}, n^2))$ space.*

Proof. Let $\{u, v\}$ be the newly inserted edge. If $u \in \mathcal{L}$ ($v \in \mathcal{L}$), increment `uLv[`v, w`]` (`uLv[`u, w`]`) by one for each $w \in N_{\bar{v}}(u)$ ($w \in N_{\bar{u}}(v)$). This takes $\mathcal{O}(m^{\varepsilon})$ time.

If a vertex $v \in \mathcal{L}$ changes to $\mathcal{H}$, this affects all length-2 paths where v is the central, low-degree vertex. For each pair of distinct neighbors $x, y \in N(v)$, decrease `uLv`$[x,y]$ by 1. The running time is $\mathcal{O}(\deg(v)^2) = \mathcal{O}(m^{2\varepsilon})$.

Each edge may be incident to at least one low-degree vertex v and form $\mathcal{O}(m^\varepsilon)$ length-2 paths with the other edges incident to v. The space requirement hence is $\mathcal{O}(\min(m^{1+\varepsilon}, n^2))$. By Lemma 7, $\mathcal{D}_\varepsilon$ with `uLv` can hence be maintained in $\mathcal{O}(m^\varepsilon + m^{2\varepsilon}m^{-\varepsilon}) = \mathcal{O}(m^\varepsilon)$ amortized update time and $\mathcal{O}(\min(m^{1+\varepsilon}, n^2))$ space. ◀

▶ **Lemma 10.** *$\mathcal{D}_\varepsilon$ with `t` can be maintained in amortized $\mathcal{O}(m^{\max(1-\varepsilon,\varepsilon)})$ time per update and $\mathcal{O}(\min(m^{1+\varepsilon}, n^2))$ space.*

Proof. Let $\{u, v\}$ be the newly inserted edge. For each $h \in \mathcal{H}$, increment `t[h]` by one if h is adjacent to both u and v. If $u \in \mathcal{H}$ ($v \in \mathcal{H}$): Increment `t[u]` (`t[v]`) by one for each $h \in \mathcal{H}$ that is adjacent to both u and v, and increment `t[u]` (`t[v]`) by `uLv`$[u,v]$. This takes $\mathcal{O}(|\mathcal{H}|) = \mathcal{O}(m^{1-\varepsilon})$ time.

If a vertex $v \in \mathcal{L}$ changes to $\mathcal{H}$, then for each pair of distinct neighbors $x, y \in N(v)$ such that $\{x, y\} \in E$, we increase `t`$[v]$ by one. Otherwise, if v changes from $\mathcal{H}$ to $\mathcal{L}$, set `t`$[v] := 0$. The running time is $\mathcal{O}(\deg(v)^2) = \mathcal{O}(m^{2\varepsilon})$.

By Lemma 9, $\mathcal{D}_\varepsilon$ with `uLv` can be maintained in amortized $\mathcal{O}(m^\varepsilon)$ update time and $\mathcal{O}(\min(m^{1+\varepsilon}, n^2))$ space. As $|\mathcal{H}| \in \mathcal{O}(m^{1-\varepsilon})$, the space requirement for `t` is $\mathcal{O}(\min(m^{1+\varepsilon}, n^2))$. By Lemma 7, $\mathcal{D}_\varepsilon$ with `t` can be maintained in $\mathcal{O}(m^{1-\varepsilon} + m^{2\varepsilon}m^{-\varepsilon}) + \mathcal{O}(m^\varepsilon) = \mathcal{O}(m^{\max(1-\varepsilon,\varepsilon)})$ amortized update time and $\mathcal{O}(\min(m^{1+\varepsilon}, n^2))$ space. ◀

▶ **Lemma 11.** *$\mathcal{D}_\varepsilon$ with `uLLv` can be maintained in amortized $\mathcal{O}(m^{2\varepsilon})$ time per update and $\mathcal{O}(\min(m^{1+2\varepsilon}, n^2))$ space.*

Proof. Let $\{u, v\}$ be the newly inserted edge. If $u \in \mathcal{L}$ ($v \in \mathcal{L}$), we count the length-3 paths starting/ending with $\{u, v\}$ as follows: For each low-degree neighbor $w \in N_{\bar{v}}(u) \cap \mathcal{L}$ ($w \in N_{\bar{u}}(v) \cap \mathcal{L}$), increment `uLLv`$[v,x]$ (`uLLv`$[u,x]$) by one for each $x \in N(w) \setminus \{u, v\}$. This takes $\mathcal{O}(m^{2\varepsilon})$ time. If both $u \in \mathcal{L}$ and $v \in \mathcal{L}$, we additionally count the length-3 paths having $\{u, v\}$ as centerpiece in $\mathcal{O}(\deg(u)^2) = \mathcal{O}(m^{2\varepsilon})$ time: For each pair of distinct vertices x, y with $x \in N_{\bar{v}}(u)$, $y \in N_{\bar{u}}(v)$, increment `uLLv`$[x,y]$ by one.

If a vertex $v \in \mathcal{L}$ changes to $\mathcal{H}$, we iterate over all pairs of distinct vertices y, w, where $y \in N_{\bar{v}}(x)$ for some low-degree neighbor $x \in N(v) \cap \mathcal{L}$ and $w \in N_{\bar{x}}(v)$, and decrease `uLLv`$[w,y]$ by one. As v has $\mathcal{O}(m^{2\varepsilon})$ pairs of neighbors and each low-degree neighbor has in turn $\mathcal{O}(m^\varepsilon)$ neighbors, the running time is in $\mathcal{O}(m^{3\varepsilon})$.

Each edge may be incident to two low-degree vertices and hence form $\mathcal{O}(m^{2\varepsilon})$ length-3 paths with the other edges incident to the end vertices. The space requirement hence is $\mathcal{O}(\min(m^{1+2\varepsilon}, n^2))$. By Lemma 7, $\mathcal{D}_\varepsilon$ with `uLLv` can be maintained in $\mathcal{O}(m^{2\varepsilon} + m^{3\varepsilon}m^{-\varepsilon}) = \mathcal{O}(m^{2\varepsilon})$ amortized update time and $\mathcal{O}(\min(m^{1+2\varepsilon}, n^2))$ space. ◀

▶ **Lemma 12.** *$\mathcal{D}_\varepsilon$ with `cLV` can be maintained in amortized $\mathcal{O}(m^{2\varepsilon})$ time per update and $\mathcal{O}(n \min(n, m^{2\varepsilon}))$ space.*

Proof. Let $\{u, v\}$ be the newly inserted edge. If $u \in \mathcal{L}$ ($v \in \mathcal{L}$), we update the number of claws where u (v) is the central vertex as follows: For each pair of distinct neighbors $x, y \in N_{\bar{v}}(u)$ ($x, y \in N_{\bar{u}}(v)$), increment `cLV`$[x,y]$ by one. This accommodates for the claws where v (u) is not an anchor vertex and takes $\mathcal{O}(m^{2\varepsilon})$ time. For the other case, increment `cLV`$[v,w]$ (`cLV`$[u,w]$) by $\deg(u) - 2$ ($\deg(v) - 2$) for each $w \in N_{\bar{v}}(u)$ ($w \in N_{\bar{u}}(v)$) in $\mathcal{O}(m^\varepsilon)$ time.

If a vertex $v \in \mathcal{L}$ changes to $\mathcal{H}$, we decrease `cLV[`x, y`]` by $\deg(v) - 2$ for each pair of distinct neighbors $x, y \in N(v)$ in total $\mathcal{O}(\deg(v)^2) = \mathcal{O}(m^{2\varepsilon})$ time.

As each vertex may be adjacent to at least one low-degree vertex and we store the count for all pairs, the space requirement is in $\mathcal{O}(n^2)$. On the other hand, each low-degree vertex has at most $\mathcal{O}(m^{2\varepsilon})$ neighbors that can serve as anchors, which yields a space requirement of $\mathcal{O}(nm^{2\varepsilon})$. By Lemma 7, $\mathcal{D}_\varepsilon$ with `cLV` can be maintained in $\mathcal{O}(m^{2\varepsilon} + m^{2\varepsilon}m^{-\varepsilon}) = \mathcal{O}(m^{2\varepsilon})$ amortized update time and $\mathcal{O}(n \min(n, m^{2\varepsilon}))$ space. ◀

▶ **Lemma 13.** *$\mathcal{D}_\varepsilon$ with `pLL` can be maintained in amortized $\mathcal{O}(m^{2\varepsilon})$ time per update and $\mathcal{O}(\min(n^2, nm^{2\varepsilon}))$ space.*

Proof. Let $\{u, v\}$ be the newly inserted edge.

If $u \in \mathcal{L}$ ($v \in \mathcal{L}$): First, we update all paws where u (v) is the central vertex and v (u) is the anchor vertex at the arm: For each ordered pair of distinct neighbors $x, y \in N_{\bar{v}}(u)$ ($x, y \in N_{\bar{u}}(v)$) such that $x \in \mathcal{L}$ and $\{x, y\} \in E$, increment `pLL[`v, y`]` (`pLL[`u, y`]`) by one. This can be done in $\mathcal{O}(m^{2\varepsilon})$ time. Second, we update all paws where u (v) is the central vertex and v (u) is the anchor vertex in the triangle: For each ordered pair of distinct neighbors $x, y \in N_{\bar{v}}(u)$ ($x, y \in N_{\bar{u}}(v)$) such that $x \in \mathcal{L}$ and $\{x, v\} \in E$ ($\{x, u\} \in E$), increment `pLL[`v, y`]` (`pLL[`u, y`]`) by one. This again can be done in $\mathcal{O}(m^{2\varepsilon})$ time. Third, we update all paws where u (v) is the non-anchor, non-central vertex in the triangle and v (u) is the anchor vertex in the triangle: For each neighbor $x \in N_{\bar{v}}(u)$ ($x \in N_{\bar{u}}(v)$) with $x \in \mathcal{L}$ and $\{v, x\} \in E$ ($\{u, x\} \in E$), increment `pLL[`v, y`]` (`pLL[`u, y`]`) by one for each $y \in N(x) \setminus \{u, v\}$. The running time is in $\mathcal{O}(m^{2\varepsilon})$, as $u, x \in \mathcal{L}$.

If both $u \in \mathcal{L}$ and $v \in \mathcal{L}$, we update all paws where $\{u, v\}$ connects the central vertex to the non-anchor vertex in the triangle: For each ordered pair of distinct neighbors $x, y \in N_{\bar{v}}(u)$ with $\{x, v\} \in E$ and each ordered pair of distinct neighbors $x, y \in N_{\bar{u}}(v)$ with $\{x, u\} \in E$, increment `pLL[`x, y`]` by one. The running time is in $\mathcal{O}(\deg(u)^2 + \deg(v)^2) = \mathcal{O}(m^{2\varepsilon})$.

If a vertex $v \in \mathcal{L}$ changes to $\mathcal{H}$: For all paws where v was the central vertex, we iterate over all unordered pairs of neighbors $y, z \in N(v)$ and every neighbor $x \in N(v) \cap \mathcal{L}$ such that $y \neq x \neq z$. If $\{x, y\} \in E$, we decrease `pLL[`y, z`]` by one, and if $\{x, z\} \in E$, we also decrease `pLL[`y, z`]` by one. For all paws where v was the low-degree, non-central vertex in the triangle, we iterate over all pairs of distinct neighbors $x, y \in N(v)$ such that $\{x, y\} \in E$ and $x \in \mathcal{L}$, and, for each $z \in N(x) \setminus \{v, y\}$, decrease `pLL[`$y, z$`]` by one. In this case, $\{x, z\}$ forms the arm. As $\deg(x) \in \mathcal{O}(m^\varepsilon)$ in the second case, the total running time is $\mathcal{O}(\deg(v)^3 + \deg(v)^2 \cdot m^\varepsilon) = \mathcal{O}(m^{3\varepsilon})$.

The argument for the space requirement is the same as for to `cLV` and $\mathcal{O}(n \min(n, m^{2\varepsilon}))$ by Lemma 12. By Lemma 7, $\mathcal{D}_\varepsilon$ with `pLL` can be maintained in $\mathcal{O}(m^{2\varepsilon} + m^{3\varepsilon}m^{-\varepsilon}) = \mathcal{O}(m^{2\varepsilon})$ amortized update time and $\mathcal{O}(n \min(n, m^{2\varepsilon}))$ space. ◀

▶ **Lemma 14.** *$\mathcal{D}_\varepsilon$ with `uHv` can be maintained in amortized $\mathcal{O}(m^{\max(1-\varepsilon, \varepsilon)})$ time per update and $\mathcal{O}(n + \min(n^2, m^{2-2\varepsilon}))$ space.*

Proof. Let $\{u, v\}$ be the newly inserted edge. If $u, v \in \mathcal{H}$, we iterate over all $h \in \mathcal{H} \setminus \{u, v\}$. If h is adjacent to u (v), increment `uHv[`h, v`]` (`uHv[`u, h`]`), respectively, by one. The running time is $\mathcal{O}(|\mathcal{H}|) = \mathcal{O}(m^{1-\varepsilon})$.

If a vertex $v \in \mathcal{L}$ changes to $\mathcal{H}$ (analogously vice-versa): For each pair of distinct high-degree neighbors $x, y \in N(v) \cap \mathcal{H}$, increment `uHv[`$x, y$`]` by one in total $\mathcal{O}(\deg(v)^2) = \mathcal{O}(m^{2\varepsilon})$ time. *Only if* v changes from $\mathcal{L}$ to $\mathcal{H}$: For every high-degree neighbor $w \in N(v) \cap \mathcal{H}$, we iterate over all $h \in \mathcal{H}$ and increase `uHv[`v, h`]` by one if $\{w, h\} \in E$ in total $\mathcal{O}(\deg(v) \cdot |\mathcal{H}|) =$

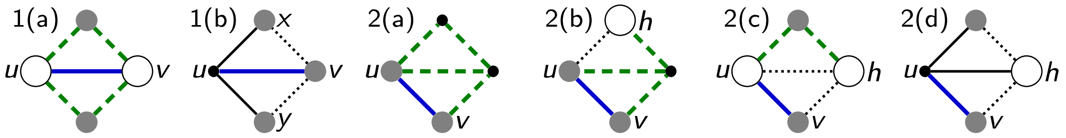

Figure 2 Counting the number of diamonds that contain an edge $\{u, v\}$. Green, dashed edges belong to paths that are considered via auxiliary counts, whereas dotted edges are edges whose presence is looked up by the algorithm. As before, small and filled vertices have low degree, large and empty vertices high degree, medium-sized and shaded vertices can have either high or low degree.

$\mathcal{O}(m^\varepsilon \cdot m^{1-\varepsilon}) = \mathcal{O}(m)$ time. *Only if* v changes from $\mathcal{H}$ to $\mathcal{L}$, we set $\texttt{uHv}[v, h] := 0$ for each $h \in \mathcal{H}$ in total $\mathcal{O}(|\mathcal{H}|) = \mathcal{O}(m^{1-\varepsilon})$ time. The overall time is hence $\mathcal{O}(m^{\max(2\varepsilon,1)})$.

There are $\mathcal{O}(\min(n, m^{1-\varepsilon}))$ high-degree vertices, which results in $\mathcal{O}(\min(n^2, m^{2-2\varepsilon}))$ pairs of anchor vertices. $\mathcal{D}_\varepsilon$ with uHv can hence be maintained in $\mathcal{O}(m^{1-\varepsilon} + m^{\max(2\varepsilon,1)}m^{-\varepsilon}) = \mathcal{O}(m^{\max(1-\varepsilon,\varepsilon)})$ amortized update time and $\mathcal{O}(n + \min(n^2, m^{2-2\varepsilon}))$ space by Lemma 7. ◀

▶ **Lemma 15.** *$\mathcal{D}_\varepsilon$ with* cL *can be maintained in amortized $\mathcal{O}(m^{2\varepsilon})$ time per update and $\mathcal{O}(\min(n^3, nm^{3\varepsilon}, m^{1+2\varepsilon}))$ space.*

Proof. Let $\{u, v\}$ be the newly inserted edge. If $u \in \mathcal{L}$ ($v \in \mathcal{L}$), increment $\texttt{cL}[v, x, y]$ ($\texttt{cL}[u, x, y]$) by one for each pair of distinct neighbors $x, y \in N_{\bar{v}}(u)$ ($x, y \in N_{\bar{u}}(v)$). This takes $\mathcal{O}(m^{2\varepsilon})$ time.

If a vertex $v \in \mathcal{L}$ changes to $\mathcal{H}$: For each triple of distinct neighbors $x, y, z \in N(v)$, we decrease $\texttt{cL}[x, y, z]$ by one in total $\mathcal{O}(\deg(v)^3) = \mathcal{O}(m^{3\varepsilon})$ time.

As each edge may be incident to a low-degree vertex v, the number of triples with non-zero count for cL is in $\mathcal{O}(\min(n^3, nm^{3\varepsilon}, m^{1+2\varepsilon}))$. By Lemma 7, $\mathcal{D}_\varepsilon$ with cL can be maintained in $\mathcal{O}(m^{2\varepsilon} + m^{3\varepsilon-\varepsilon}) = \mathcal{O}(m^{2\varepsilon})$ amortized update time and $\mathcal{O}(\min(n^3, m^{1+2\varepsilon}))$ space. ◀

Non-Induced Subgraph Counts

We are now ready to prove Theorem 5 and show for each connected subgraph on four vertices how to count it using the data structure $\mathcal{D}_\varepsilon$.

▶ **Lemma 16.** *Let $G = (V, E)$ be a dynamic graph, $\varepsilon \in [0, 1]$, and $\mathcal{P}$ be the diamond ◇. We can query $\mathbf{c}(G, \mathcal{P}, \{u, v\})$ for an arbitrary edge $\{u, v\} \in E$ in $\mathcal{O}(\min(m^{\max(1-\varepsilon,2\varepsilon)}, n^2))$ worst-case time if we maintain the data structure $\mathcal{D}_\varepsilon$ with auxiliary counts* uLv, pLL, uHv, *and* cL.

Proof. Edge $\{u, v\}$ can either be the chord of the diamond or be part of the 4-cycle. See Figure 2 for an illustration.

For the first case, where $\{u, v\}$ is the chord: (a) If $u, v \in \mathcal{H}$, we can obtain the number of length-2 paths p between u and v as $p = \texttt{uLv}[u, v] + \texttt{uHv}[u, v]$. As each pair of length-2 paths forms a diamond with $\{u, v\}$, the total number of diamonds is $\binom{p}{2}$. (b) Otherwise, $\{u, v\} \cap \mathcal{L} \neq \emptyset$. W.l.o.g., $u \in \mathcal{L}$. We then iterate over all distinct, unordered pairs of neighbors $x, y \in N_{\bar{v}}(u)$ in $\mathcal{O}(\deg(u)^2) = \mathcal{O}(\min(m^{2\varepsilon}, n^2))$ time. For each such pair with $\{x, v\}, \{y, v\} \in E$, we count one diamond.

For the second case, where $\{u, v\}$ is part of the cycle, we distinguish between the degrees of the other two vertices. (a) The number of diamonds where the other two vertices have low degree is given by $\texttt{pLL}[u, v]$. Note that either u or v is incident to the chord. (b) The number of diamonds where the other vertex incident to the chord has low degree and the fourth vertex has high degree can be obtained by iterating over all $h \in \mathcal{H} \setminus \{u, v\}$ in

$\mathcal{O}(|\mathcal{H}|) = \mathcal{O}(\min(m^{1-\varepsilon}, n))$ time. If either $\{h, u\} \in E$ or $\{h, v\} \in E$, we have `cL`$[u, v, h]$ more diamonds. If both $\{h, u\}, \{h, v\} \in E$, we add 2`cL`$[u, v, h]$ to the number of diamonds. (c, d) The number of diamonds where the other vertex incident to the chord has high degree can be obtained as follows: (c) If $u \in \mathcal{H}$ ($v \in \mathcal{H}$), the number of diamonds where the chord is incident to u (v) can be obtained by iterating over all $h \in \mathcal{H} \setminus \{u, v\}$ in $\mathcal{O}(|\mathcal{H}|) = \mathcal{O}(\min(m^{1-\varepsilon}, n))$ time. For each such vertex h, we check whether $\{u, h\}, \{v, h\} \in E$ and add `uLv`$[u, h]$ + `uHv`$[u, h] - 1$ (`uLv`$[v, h]$ + `uHv`$[v, h] - 1$) to the count. The correction by 1 is necessary because the auxiliary counts also contain the path $\langle \{u, v\}, \{v, h\} \rangle$ ($\langle \{h, u\}, \{u, v\} \rangle$). (d) If $u \in \mathcal{L}$ ($v \in \mathcal{L}$), we iterate over all high-degree neighbors $h \in N_{\bar{v}}(u) \cap \mathcal{H}$ ($h \in N_{\bar{u}}(v) \cap \mathcal{H}$) and in each case over all $x \in N(u) \setminus \{v, h\}$ ($x \in N(v) \setminus \{u, h\}$) in total $\mathcal{O}(\min(m^{2\varepsilon}, n^2))$ time and count one diamond each if $\{h, x\}, \{h, v\} \in E$ ($\{h, x\}, \{h, u\} \in E$). ◀

▶ **Lemma 17.** *Let $G = (V, E)$ be a dynamic graph and $\mathcal{P}$ be the diamond. We can maintain $\mathbf{c}(G, \mathcal{P})$ in amortized $\mathcal{O}(m^{2/3})$ update time and $\mathcal{O}(\min(nm, m^{\frac{5}{3}}))$ space. We can query $\mathbf{c}(G, \mathcal{P}, e)$ for an arbitrary edge $e \in E$ in worst-case $\mathcal{O}(m^{2/3})$ time.*

Proof. After an edge $\{u, v\}$ was inserted or before an edge $\{u, v\}$ is removed, the number of diamonds containing it can be obtained in $\mathcal{O}(\min(m^{\max(1-\varepsilon, 2\varepsilon)}, n^2))$ time worst-case time by Lemma 16 if $\mathcal{D}_\varepsilon$ with auxiliary counts `uLv`, `pLL`, `uHv`, and `cL` is maintained. By Lemma 9, Lemma 13, Lemma 14, and Lemma 15, this can be done in amortized $\mathcal{O}(m^\varepsilon + m^{\max(1-\varepsilon,\varepsilon)} + m^{2\varepsilon}) = \mathcal{O}(m^{\max(1-\varepsilon, 2\varepsilon)})$ time and $\mathcal{O}(\min(n^3, \max(n^2, nm^{3\varepsilon}, m^{2-2\varepsilon}, m^{1+2\varepsilon})))$ space. Together with the cost for the query, this yields a total amortized update time of $\mathcal{O}(m^{\max(1-\varepsilon, 2\varepsilon)}) = \mathcal{O}(m^{\frac{2}{3}})$ for $\varepsilon = \frac{1}{3}$ and $\mathcal{O}(\min(nm, m^{\frac{5}{3}}))$ space. By Lemma 16, the worst-case time to query $\mathbf{c}(G, \mathcal{P}, e)$ for an arbitrary edge $e \in E$ then is $\mathcal{O}(m^{2/3})$. ◀

Queries with Vertices and Edges and Non-Induced s-Subgraph Counts

With similar techniques, we can count non-induced triangles containing a specified vertex or edge as well as maintain s-subgraph counts for patterns with up to four vertices.

▶ **Theorem 18.** *Let $G = (V, E)$ be a dynamic graph, $\mathcal{P}$ be the 3-cycle, and $\varepsilon \in [0, 1]$. We can query $\mathbf{c}(G, \mathcal{P}, a)$ for an arbitrary vertex or edge a in*

- **(i)** *worst-case $\mathcal{O}(\min(m^{2\varepsilon}, n^2))$ time with $\mathcal{O}(m^{\max(\varepsilon, 1-\varepsilon)})$ amortized update time and $\mathcal{O}(\min(n^2, m^{1+\varepsilon}))$ space if $a \in V$,*
- **(ii)** *worst-case time $\mathcal{O}(\min(m^{1-\varepsilon}, n))$ with $\mathcal{O}(m^\varepsilon)$ amortized update time and $\mathcal{O}(\min(n^2, m^{1+\varepsilon}))$ space if $a \in E$.*

▶ **Corollary 19.** *Let $G = (V, E)$ be a dynamic graph and $\mathcal{P}$ the 3-cycle. We can maintain $\mathbf{c}(G, \mathcal{P})$ with an amortized update time of $\mathcal{O}(\sqrt{m})$ and $\mathcal{O}(\min(n^2, m^{1.5}))$ space and query $\mathbf{c}(G, \mathcal{P}, e)$ for $e \in E$ arbitrary in worst-case $\mathcal{O}(\sqrt{m})$ time. We can query $\mathbf{c}(G, \mathcal{P}, v)$ for arbitrary $v \in V$ in worst-case $\mathcal{O}(m^{2/3})$ time with an amortized update time of $\mathcal{O}(m^{2/3})$ and $\mathcal{O}(\min(n^2, m^{4/3}))$ space.*

▶ **Theorem 20.** *Let $G = (V, E)$ be a dynamic graph, $s \in V$, and $\mathcal{P}$ be a connected subgraph. We can maintain the non-induced s-subgraph count $\mathbf{c}(G, \mathcal{P}, s)$ in*

- **(i)** *worst-case constant update time and constant space if $\mathcal{P}$ is the claw,*
- **(ii)** *amortized update time $\mathcal{O}(\sqrt{m})$ and $\mathcal{O}(n)$ space if $\mathcal{P}$ is the 3-cycle or the length-3 path,*
- **(iii)** *amortized update time $\mathcal{O}(m^{2/3})$ and $\mathcal{O}(n^2)$ space if $\mathcal{P}$ is the paw, the 4-cycle, or the diamond,*
- **(iv)** *worst-case update time $\mathcal{O}(m)$ and constant space if $\mathcal{P}$ is the 4-clique.*

4 Lower Bounds

We give new lower bounds for detecting and counting induced and non-induced subgraphs.

Induced Subgraph Counts

Our results for counting induced subgraphs on four vertices are conditioned on the combinatorial k-clique conjecture:

▶ **Theorem 21.** *Let G be a dynamic graph, $\mathcal{P} \in \mathbb{P}_4$, and let $\gamma > 0$ be a small constant. There is no incremental or fully dynamic combinatorial algorithm with preprocessing time $\mathcal{O}(m^{2-\gamma})$ for maintaining $\mathbf{c_I}(G, \mathcal{P})$ in amortized update time $\mathcal{O}(m^{1-\gamma})$ and query time $\mathcal{O}(m^{2-\gamma})$, unless the k-clique conjecture fails.*

Before we turn to the proof, we recall the following relations between subgraph counts.

▶ **Lemma 22** ([9]). *For each pair $P, P' \in \mathbb{P}_4$, the non-induced subgraph count $\mathbf{c}(P, P') = 1$ if $P = P'$ and otherwise nonzero only in the following cases:*

$\mathbf{c}(\lhd\!\!-, \sqcap) = 2$, $\mathbf{c}(\diamond, \sqcap) = 4$, $\mathbf{c}(\ominus, \sqcap) = 6$, $\mathbf{c}(\boxtimes, \sqcap) = 12$,
$\mathbf{c}(\lhd\!\!-, \curlywedge) = 1$, $\mathbf{c}(\ominus, \curlywedge) = 2$, $\mathbf{c}(\boxtimes, \curlywedge) = 4$, $\mathbf{c}(\ominus, \lhd\!\!-) = 4$,
$\mathbf{c}(\boxtimes, \lhd\!\!-) = 12$, $\mathbf{c}(\ominus, \diamond) = 1$, $\mathbf{c}(\boxtimes, \diamond) = 3$, $\mathbf{c}(\boxtimes, \ominus) = 6$.

▶ **Proposition 23** ([17]). *Let $G, \mathcal{P}$ be graphs and let k be the number of vertices of $\mathcal{P}$. Then, $\mathbf{c}(G, \mathcal{P}) = \sum_{P \in \mathbb{P}_k} \mathbf{c_I}(G, P) \cdot \mathbf{c}(P, \mathcal{P})$.*

▶ **Lemma 24.** *Let $G = (V, E)$ be a graph. The following relationships between induced and non-induced subgraph counts hold:*

$$\mathbf{c_I}(G, \boxtimes) = \mathbf{c}(G, \boxtimes)$$
$$\mathbf{c_I}(G, \ominus) = \mathbf{c}(G, \ominus) - 6\mathbf{c_I}(G, \boxtimes)$$
$$\mathbf{c_I}(G, \diamond) = \mathbf{c}(G, \diamond) - \mathbf{c}(G, \ominus) + 3\mathbf{c_I}(G, \boxtimes)$$
$$\mathbf{c_I}(G, \lhd\!\!-) = \mathbf{c}(G, \lhd\!\!-) - 4\mathbf{c}(G, \ominus) + 12\mathbf{c_I}(G, \boxtimes)$$
$$\mathbf{c_I}(G, \curlywedge) = \mathbf{c}(G, \curlywedge) - \mathbf{c}(G, \lhd\!\!-) + 2\mathbf{c}(G, \ominus) - 4\mathbf{c_I}(G, \boxtimes)$$
$$\mathbf{c_I}(G, \sqcap) = \mathbf{c}(G, \sqcap) - 2\mathbf{c}(G, \lhd\!\!-) - 4\mathbf{c}(G, \diamond) + 6\mathbf{c}(G, \ominus) - 12\mathbf{c_I}(G, \boxtimes)$$

Proof. Let $\mathcal{P} \in \mathbb{P}_4$. The statement follows from Proposition 23 and Lemma 22 by substituting equations and solving them for $\mathbf{c_I}(G, \mathcal{P})$ in the order as listed. ◀

▶ **Corollary 25.** *Let $G = (V, E)$ be a dynamic graph and $\mathcal{P} \in \mathbb{P}_4$. We can maintain $\mathbf{c_I}(G, \mathcal{P})$ with $\mathcal{O}(m)$ worst-case update time and constant space.*

Proof of Theorem 21. First consider the case that $\mathcal{P} = \boxtimes$. Suppose there is an incremental or fully dynamic algorithm $\mathcal{A}$ that maintains $\mathbf{c_I}(G, \mathcal{P})$ in time $\mathcal{O}(m^{1-\gamma})$ with query time $\mathcal{O}(m^{2-\gamma})$ for some $\gamma > 0$. Construct an algorithm $\mathcal{A}'$ for static 4-clique detection as follows: Run $\mathcal{A}$ on an initially empty graph, insert all edges one-by-one in total $\mathcal{O}(m^{2-\gamma})$ time, and query the result in $\mathcal{O}(m^{2-\gamma})$ time. As $\mathcal{O}(m^{2-\gamma}) = \mathcal{O}(n^{4-2\gamma})$ this contradicts Conjecture 2.

For the remaining five induced four-vertex subgraphs, let $\mathcal{P}$ be such a subgraph. We construct a deterministic algorithm $\mathcal{A}'$ for static 4-clique detection as follows: $\mathcal{A}'$ executes the above operations for our non-induced subgraph counting algorithm from Sect. 3, which can maintain the number of all connected subgraphs on four vertices with $\mathcal{O}(m^{2/3})$ amortized update time and $\mathcal{O}(1)$ query time by Theorem 5. It thus takes $\mathcal{O}(m^{5/3})$ time in total to compute $\mathbf{c}(G, \mathcal{P}')$ for all $\mathcal{P}' \in \mathbb{P}_4 \setminus \{\boxtimes\}$. Assume by contradiction that there exists an algorithm

$$M = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$
$$u^\top = (\,1\ 1\ 0\,)$$
$$v^\top = (\,0\ 1\ 1\ 0\,)$$

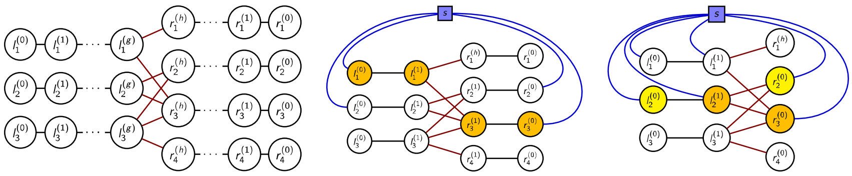

Figure 3 Construction of $G_{M,g,h}$ and G for $u^\top Mv$ for (s)-5-cycle and (s)-diamond detection.

$\mathcal{A}^*$ that maintains $\mathbf{c}_\mathbf{I}(G, \mathcal{P})$ in update time $\mathcal{O}(m^{1-\gamma})$ and query time $\mathcal{O}(m^{2-\gamma})$. Then $\mathcal{A}'$ also executes the same operations with $\mathcal{A}^*$ to compute $\mathbf{c}_\mathbf{I}(G, \mathcal{P})$. Using the formula for $\mathcal{P}$ in Lemma 24, $\mathcal{A}'$ can solve the static 4-clique detection problem in time $\mathcal{O}(m^{\max(5/3,2-\gamma)}) \subseteq \mathcal{O}(n^{4-\delta})$ time for $\delta = \min(2\gamma, 2/3) > 0$, a contradiction to Conjecture 2. ◀

Non-Induced Subgraph Counts

In this section, we give new lower bounds for detecting (and thus counting) cycles of arbitrary length, paws, diamonds, and 4-cliques[1], as well as counting length-3 paths. Our results are based on the OMv conjecture. In [12] it is proven that instead of reducing from OMv directly it suffices to reduce from the following 1-uMv version: For any positive integer parameters n_1, n_2, given an $n_1 \times n_2$ matrix M, there is no algorithm with preprocessing time polynomial in n_1 and n_2 that computes for an n_1-dimensional vector u and an n_2-dimensional vector v the product $u^\top Mv$ in time $\mathcal{O}(n_1 n_2^{1-\delta} + n_1^{1-\delta} n_2)$ for any small constant $\delta > 0$ with error probability of at most $\frac{1}{3}$ in the word-RAM model with $\mathcal{O}(\log n)$ bit words.

▶ **Theorem 26.** *Let G be a partially dynamic graph and let $\mathcal{P}$ be the paw ⧏, the diamond ◇, the 4-clique ⊠, or a k-cycle with $k \geq 3$. On condition of Conjecture 1, there is no partially dynamic algorithm to maintain whether $\mathbf{c}(G, \mathcal{P}) > 0$ with polynomial preprocessing time and worst-case update time $\mathcal{O}(m^{1/2-\delta})$ and query time $\mathcal{O}(m^{1-\delta})$ with an error probability of at most $1/3$ for any $\delta > 0$. This also holds for fully dynamic algorithms with amortized update time and for paws, diamonds, 4-cliques, or odd k-cycles containing a specific vertex s, as well as for maintaining the number of length-3 paths ⟋ and the number of length-3 paths containing a specific vertex s.*

Our constructions build on the following graph (see Figure 3 for an example).

▶ **Definition 27** ($G_{M,g,h}$). *Given a matrix $M \in \{0,1\}^{n_1 \times n_2}$ and two integers $g, h \geq 0$, we denote by $G_{M,g,h} = (\bigcup_{0 \leq p \leq g} L^{(p)} \cup \bigcup_{0 \leq q \leq h} R^{(q)}, E_L \cup E_R \cup E_M)$ the $(g + h + 2)$-partite graph with*

$$L^{(p)} = \{l_1^{(p)}, \ldots, l_{n_1}^{(p)}\}, \quad 0 \leq p \leq g; \qquad E_L = \{(l_i^{(p)}, l_i^{(p+1)}) \mid 1 \leq i \leq n_1 \wedge 0 \leq p < g\}$$
$$R^{(q)} = \{r_1^{(q)}, \ldots, r_{n_2}^{(q)}\}, \quad 0 \leq q \leq h; \qquad E_R = \{(r_j^{(q)}, r_j^{(q+1)}) \mid 1 \leq j \leq n_2 \wedge 0 \leq q < h\}$$

and $E_M = \{(l_i^{(g)}, r_j^{(h)}) \mid M_{ij} = 1\}$. $G_{M,g,h}$ has $(g+1) \cdot n_1 + (h+1) \cdot n_2$ vertices and at most $n_1 n_2 + g \cdot n_1 + h \cdot n_2$ edges. All vertices x in $L^{(p)}$ for $1 \leq p < g$ and in $R^{(q)}$ for $1 \leq q < h$ have $\deg(x) = 2$, whereas all vertices y in $L^{(0)}$ and in $R^{(0)}$ have $\deg(y) = 1$.

For convenience, we set $L := L^{(0)}$, $R := R^{(0)}$, $l_i := l_i^{(0)}$, and $r_j := r_j^{(0)}$.

▶ **Observation 28.** *Every cycle in $G_{M,g,h}$ is even and has length at least 4.*

---

[1] Theorem 21 applies also to 4-cliques, but it is based on a different assumption.

Let s be a fixed vertex in the graph. The s-k-cycle detection problem requires the algorithm to detect whether a k-cycle containing s exists. We use the same notation for the other subgraphs as well.

▶ **Lemma 29.** *Given a partially dynamic algorithm $\mathcal{A}$ for one of the problems listed below, one can solve* 1-uMv *with parameters n_1 and n_2 by running the preprocessing step of $\mathcal{A}$ on a graph with $\mathcal{O}(m + \sqrt{m} \cdot k)$ edges and $\Theta(\sqrt{m} \cdot k)$ vertices, and then making $\mathcal{O}(\sqrt{m})$ insertions (or $\mathcal{O}(\sqrt{m})$ deletions) and 1 query, where m is such that $n_1 = n_2 = \sqrt{m}$. The problems are*

(a) *(s-)k-Cycle Detection for odd k*
(b) *(s-)Paw Detection*
(c) *(s-)Diamond Detection*
(d) *(s-)k-Clique Detection for $k = 4$*
(e) *(s-)length-k Path Counting for $k = 3$*
(f) *k-Cycle Detection*

Proof of Case (c). We only prove the decremental case. Consider a 1-uMv problem with $n_1 = n_2 = \sqrt{m}$. Given M, we construct the tripartite graph G from $G_{M,1,0}$ by adding to it a vertex s and connecting it by an edge to every vertex in $G_{M,1,0}$. Thus, the total number of edges is at most $n_1 n_2 + 3n_1 + n_2 = \mathcal{O}(m)$. Once u and v arrive, we delete $\{s, l_i\}$ and $\{s, l_i^{(1)}\}$ iff $u_i = 0$ and delete $\{r_j, s\}$ iff $v_j = 0$. See Figure 3 for an example.

Consider the case that G contains a diamond with chord e. As every triangle must be incident to s by Observation 28, $e = \{s, x\}$ for some $x \in L \cup L^{(1)} \cup R$. Furthermore, all vertices in L have degree at most two and x has degree at least three, so $x \notin L$. If $x = r_j \in R$, then by construction, x must have two neighbors $l_i^{(1)}, l_{i'}^{(1)} \in L^{(1)}$ such that there are edges $\{s, l_i^{(1)}\}$ and $\{s, l_{i'}^{(1)}\}$ in G. Again by construction, there are thus also edges $\{s, l_i\}$ and $\{s, l_{i'}\}$ and a diamond $\{s, l_i, l_i^{(1)}, x = r_j\}$ with chord $\{s, l_i^{(1)}\}$. As each vertex in $L^{(1)}$ is adjacent to exactly one vertex in L, every diamond must contain a vertex $r \in R$ and the edge $\{r, s\}$. Hence, we have $u^\top M v = 1$ iff there is a diamond in G iff there is a diamond incident to s. In total, we need to do $2n_1 + n_2 = \mathcal{O}(\sqrt{m})$ updates and 1 query. ◀

5 Conclusion

Our focus in this work was especially on non-induced and induced four-vertex subgraphs. We gave improved both upper and lower bounds for detecting or counting four-vertex subgraphs in the dynamic setting, thereby closing the gap (w.r.t. improvements by a polynomial factor) for counting non-induced length-3 paths and narrowing it considerably for non-induced paws, 4-cycles, and diamonds. For counting induced subgraphs, we showed that the update time of the algorithm by Eppstein et al. [9] cannot be improved by a polynomial factor, but that a better space complexity can be achieved in the worst case.

Many of our lower bounds also apply to subgraphs with more than four vertices, but to the best of our knowledge, only algorithms for cliques have been considered here so far. Hence, besides closing the gap for four-vertex subgraphs, the complexity of detecting and counting subgraphs with five or more vertices would be an interesting field for future work.

We also investigated the complexity of querying the number of subgraphs containing a specific edge, as such queries are relevant, e.g., to measure the similarity of graphs via histograms. This can similarly be done for vertices. As a by-product of our results for four-vertex subgraphs, we showed for 3-cycles that such vertex queries can be answered in $\mathcal{O}(m^{\frac{2}{3}})$. The complexity of vertex queries for larger subgraphs remains an open question.

Further interesting lines to follow regard the complexity of approximate counting, counting all approximately densest subgraphs, as well as the complexity of *enumerating* subgraphs.

Our work was strongly motivated also by the practical relevance of counting subgraphs in the dynamic setting. For this reason, we consider an experimental evaluation of dynamic subgraph counting algorithms a relevant and very interesting task for future work.

References

1. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is valiant's parser. *SIAM Journal on Computing*, 47(6):2527–2555, 2018.
2. N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, Mar 1997. doi:10.1007/BF02523189.
3. Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, page 623–632, USA, 2002. Society for Industrial and Applied Mathematics.
4. Thiago Bergamaschi, Monika Henzinger, Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New techniques and fine-grained hardness for dynamic near-additive spanners. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1836–1855. SIAM, 2021. doi:10.1137/1.9781611976465.110.
5. Karl Bringmann, Nick Fischer, and Marvin Künnemann. A fine-grained analogue of schaefer's theorem in p: Dichotomy of $\exists k \forall$-quantified first-order graph properties. In *34th Computational Complexity Conference*, pages 1–27. Schloss Dagstuhl, 2019.
6. Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *Proceedings of the Twenty-Fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '06, page 253–262, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1142351.1142388.
7. Laxman Dhulipala, Quanquan C. Liu, Julian Shun, and Shangdi Yu. Parallel batch-dynamic k-clique counting. *CoRR*, abs/2003.13585, 2020. URL: https://arxiv.org/abs/2003.13585, arXiv:2003.13585.
8. Zdeněk Dvořák and Vojtěch Tůma. A dynamic data structure for counting subgraphs in sparse graphs. In Frank Dehne, Roberto Solis-Oba, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures*, pages 304–315, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
9. David Eppstein, Michael T. Goodrich, Darren Strash, and Lowell Trott. Extended dynamic subgraph statistics using h-index parameterized data structures. *Theor. Comput. Sci.*, 447:44–52, 2012. doi:10.1016/j.tcs.2011.11.034.
10. David Eppstein and Emma S. Spiro. The h-index of a graph and its application to dynamic subgraph statistics. *J. Graph Algorithms Appl.*, 16(2):543–567, 2012. doi:10.7155/jgaa.00273.
11. Kathrin Hanauer, Monika Henzinger, and Qi Cheng Hua. Fully dynamic four-vertex subgraph counting. *CoRR*, abs/2106.15524, 2021. URL: https://arxiv.org/abs/2106.15524, arXiv:2106.15524.
12. Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, pages 21–30, 2015.
13. Monika Henzinger, Andrea Lincoln, and Barna Saha. The complexity of average-case dynamic subgraph counting. In *Proceedings of the Thirty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Alexandria, Virginia, USA, January 9-12, 2022*. SIAM, 2022. to appear.
14. Ahmet Kara, Hung Q. Ngo, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Counting Triangles under Updates in Worst-Case Optimal Time. In Pablo Barcelo and Marco Calautti, editors, *22nd International Conference on Database Theory (ICDT 2019)*, volume 127 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:18, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10306, doi:10.4230/LIPIcs.ICDT.2019.4.

15 Ahmet Kara, Hung Q. Ngo, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Maintaining triangle queries under updates. *ACM Trans. Database Syst.*, 45(3):11:1–11:46, 2020. doi:10.1145/3396375.

16 T. Kloks, D. Kratsch, and H. Müller. Finding and counting small induced subgraphs efficiently. In Manfred Nagl, editor, *Graph-Theoretic Concepts in Computer Science*, pages 14–23, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

17 William L Kocay. Some new methods in reconstruction theory. In *Combinatorial Mathematics IX*, pages 89–114. Springer, 1982.

18 Tamara G. Kolda, Ali Pinar, Todd Plantenga, C. Seshadhri, and Christine Task. Counting triangles in massive graphs with mapreduce. *SIAM Journal on Scientific Computing*, 36(5):S48–S77, 2014. arXiv:https://doi.org/10.1137/13090729X, doi:10.1137/13090729X.

19 F. Le Gall. Powers of tensors and fast matrix multiplication. In K. Nabeshima, K. Nagasaka, F. Winkler, and Á. Szántó, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303. ACM, 2014. doi:10.1145/2608628.2608664.

20 Andrea Lincoln, Virginia Vassilevska Williams, and Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1236–1252. SIAM, 2018.

21 Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M. Tamer Özsu. The ubiquity of large graphs and surprising challenges of graph processing: extended survey. *VLDB J.*, 29(2-3):595–618, 2020. doi:10.1007/s00778-019-00548-x.

A Omitted Proofs from Sect. 3

Proof of Theorem 4. If $\mathcal{P}$ is the claw ⅄, the number of claws affected by the insertion or deletion by an edge $\{u, v\}$ can be obtained directly from the degrees of u and v, and the edge counts can be updated in total $\mathcal{O}(n)$ time by iterating over all $x \in N_{\bar{v}}(u)$ ($y \in N_{\bar{u}}(v)$).

For the other subgraphs, consider the following algorithm $\mathcal{A}$: Given an edge $\{u, v\}$, it iterates over all edges $\{x, y\} \in E$, such that $\{u, v\} \cap \{x, y\} = \emptyset$, tests in constant time whether $\{u, v, x, y\}$ form $\mathcal{P}$, and updates any global or edge counts related to one of the up to six possible edges in $\mathcal{O}(m)$ total time.

If $\mathcal{P}$ is the 4-cycle ◇ or the 4-clique ⊠, $\mathcal{A}$ can be used to enumerate all subgraphs that contain an edge $\{u, v\}$.

If $\mathcal{P}$ is the length-3 path ⟋, $\mathcal{A}$ only enumerates those paths where $\{u, v\}$ is *not* the centerpiece. For the remaining length-3 paths, we iterate over all $x \in N_{\bar{v}}(u)$. If $\{v, x\} \in E$, the number of length-3 paths containing $\{u, v\}$ and $\{v, x\}$ equals $\deg(v) - 2$, and $\deg(v) - 1$ if $\{v, x\} \notin E$. The same can be done symmetrically for all $y \in N_{\bar{u}}(v)$ and takes $\mathcal{O}(n)$ time in total. This procedure can be used to update the edge counts for all edges that form a length-3 path with $\{u, v\}$ as centerpiece.

If $\mathcal{P}$ is the paw ⊲, $\mathcal{A}$ only enumerates those paws where $\{u, v\}$ is either the arm or the edge in the triangle not incident to the central vertex. For the remaining paws where u is the central vertex, we iterate over all $x \in N_{\bar{u}}(v)$ and test whether $\{x, u\} \in E$. The number of paws containing the triangle $\{u, v\}$, $\{v, x\}$, and $\{x, u\}$ then equals $\deg(u) - 2$. We store the number of such triangles t_u and iterate over all $y \in N_{\bar{v}}(u)$. If $\{v, y\} \notin E$, the number of paws containing $\{u, v\}$ and $\{u, y\}$ as arm is t_u, and $t_u - 1$ otherwise. The same can be done symmetrically for the case where v is the central vertex in total $\mathcal{O}(n)$ time, and, together with $\mathcal{A}$, it can be used to update the edge counts for all edges that form a paw with $\{u, v\}$.

If $\mathcal{P}$ is the diamond ⟠, $\mathcal{A}$ only enumerates those diamond where $\{u, v\}$ is *not* the chord. For the remaining diamonds, we iterate over all $x \in N_{\bar{u}}(v)$ and test whether $\{x, u\} \in E$. We store the number of such triangles t and repeat the same iteration. The number of diamonds containing $\{u, v\}$, $\{u, x\}$, and $\{v, x\}$ then equals $t - 1$.

We can use the subgraph specific additional procedures plus algorithm $\mathcal{A}$ in all cases except the claw ⅄ to (i) update $\mathbf{c}(G, \mathcal{P})$ on each edge update, (ii) query $\mathbf{c}(G, \mathcal{P}, \{u, v\})$ without using additional precomputed information, (iii) store and update $\mathbf{c}(G, \mathcal{P}, e)$ for each edge $e \in E$. ◀

A.1 Counting Non-Induced Subgraphs

Counting 3-cycles

▶ **Lemma 30.** *Let $G = (V, E)$ be a dynamic graph, $\varepsilon \in [0, 1]$, and let $\mathcal{P}$ be the 3-cycle △. We can query $\mathbf{c}(G, \mathcal{P}, \{u, v\})$ for an arbitrary edge $\{u, v\} \in E$ in worst-case $\mathcal{O}(\min(m^{1-\varepsilon}, n))$ time if we maintain the data structure $\mathcal{D}_\varepsilon$ with auxiliary count* `uLv`.

Proof. The number of 3-cycles where the third vertex has low degree is given directly by `uLv`$[u, v]$. To count all 3-cycles where the third vertex has high degree, we iterate over all $h \in \mathcal{H}$ and test whether $\{u, h\}, \{v, h\} \in E$. The running time is hence $\mathcal{O}(|\mathcal{H}|) = \mathcal{O}(\min(m^{1-\varepsilon}, n))$. ◀

▶ **Corollary 31.** *Let $G = (V, E)$ be a dynamic graph, $\varepsilon \in [0, 1]$, and let $\mathcal{P}$ be the 3-cycle △. We can query $\mathbf{c}(G, \mathcal{P}, \{u, v\})$ for an arbitrary edge $\{u, v\} \in E$ in worst-case $\mathcal{O}(\min(m^{1-\varepsilon}, n))$ time, with $\mathcal{O}(m^\varepsilon)$ amortized update time and $\mathcal{O}(\min(m^{1+\varepsilon}, n^2))$ space.*

Proof. Follows directly from Lemma 30 and Lemma 9. ◀

▶ **Lemma 32.** *Let $G = (V, E)$ be a dynamic graph, $\varepsilon \in [0, 1]$, and let $\mathcal{P}$ be the 3-cycle $\triangle$. We can query $\mathbf{c}(G, \mathcal{P}, v)$ for an arbitrary vertex $v \in V$ in worst-case $\mathcal{O}(\min(m^{2\varepsilon}, n^2))$ time if we maintain the data structure $\mathcal{D}_\varepsilon$ with auxiliary count* `t`.

Proof. If $v \in \mathcal{H}$, the number of 3-cycles containing v is given directly by `t[`v`]`.

If $v \in \mathcal{L}$, we iterate over all unordered pairs of distinct neighbors $x, y \in N(v)$ and test whether $\{x, y\} \in E$. This takes $\mathcal{O}(\deg(v)^2) = \mathcal{O}(\min(m^{2\varepsilon}, n^2))$ time. ◀

▶ **Corollary 33.** *Let $G = (V, E)$ be a dynamic graph, $\varepsilon \in [0, 1]$, and let $\mathcal{P}$ be the 3-cycle $\triangle$. We can query $\mathbf{c}(G, \mathcal{P}, v)$ for an arbitrary vertex $v \in V$ in worst-case $\mathcal{O}(\min(m^{2\varepsilon}, n^2))$ time, with $\mathcal{O}(m^{\max(\varepsilon, 1-\varepsilon)})$ amortized update time and $\mathcal{O}(\min(m^{1+\varepsilon}, n^2))$ space.*

Proof. Follows directly from Lemma 32 and Lemma 10. ◀

Counting Length-3 paths

▶ **Lemma 34.** *Let $G = (V, E)$ be a dynamic graph, $\varepsilon \in [0, 1]$, and let $\mathcal{P}$ be the length-3 path. We can query $\mathbf{c}(G, \mathcal{P}, \{u, v\})$ for an arbitrary edge $\{u, v\} \in E$ in worst-case $\mathcal{O}(\min(m^{1-\varepsilon}, n))$ time if we maintain the data structure $\mathcal{D}_\varepsilon$ with auxiliary counts* `vLV` *and* `uLv`.

Proof. The number of length-3 paths containing $\{u, v\}$ as centerpiece is $\deg_{\bar{v}}(u) \cdot \deg_{\bar{u}}(v)$ minus the number of 3-cycles containing $\{u, v\}$, which can be obtained in worst-case $\mathcal{O}(\min(m^{1-\varepsilon}, n))$ time by Lemma 30 if $\mathcal{D}_\varepsilon$ is maintained with auxiliary count `uLv`.

The number of length-3 paths that start or end with $\{u, v\}$ and contain two further vertices x, y, i.e., $u - v - x - y$ or $v - u - x - y$, equals `vLV[`v`]` $- \deg_{\bar{v}}(u) +$ `vLV[`u`]` $- \deg_{\bar{u}}(v)$ if x has low degree. For the case where x has high degree, we iterate over all $h \in \mathcal{H} \setminus \{u, v\}$. If h is adjacent to u, we add $\deg_{\bar{v}}(h) - 1$ and if h is adjacent to v, we add $\deg_{\bar{u}}(h) - 1$. This takes $\mathcal{O}(|\mathcal{H}|) = \mathcal{O}(\min(m^{1-\varepsilon}, n))$ time. ◀

▶ **Corollary 35.** *Let $G = (V, E)$ be a dynamic graph, $\varepsilon \in [0, 1]$, and let $\mathcal{P}$ be the length-3 path. We can query $\mathbf{c}(G, \mathcal{P}, \{u, v\})$ for an arbitrary edge $\{u, v\} \in E$ in worst-case $\mathcal{O}(\min(m^{1-\varepsilon}, n))$ time, with $\mathcal{O}(m^\varepsilon)$ amortized update time and $\mathcal{O}(\min(m^{1+\varepsilon}, n^2))$ space.*

Proof. Follows directly from Lemma 34, Lemma 8, and Lemma 9. ◀

▶ **Lemma 36.** *Let $G = (V, E)$ be a dynamic graph and $\mathcal{P}$ the length-3 path. We can maintain $\mathbf{c}(G, \mathcal{P})$ in amortized $\mathcal{O}(\sqrt{m})$ update time and $\mathcal{O}(\min(m^{1.5}, n^2))$ space. We can query $\mathbf{c}(G, \mathcal{P}, e)$ for an arbitrary edge $e \in E$ in worst-case $\mathcal{O}(\sqrt{m})$ time.*

Proof. After an edge $\{u, v\}$ was inserted or before an edge $\{u, v\}$ is removed, the number of length-3 paths containing $\{u, v\}$ can be obtained in $\mathcal{O}(\min(m^{1-\varepsilon}, n))$ worst-case time by Lemma 34 if $\mathcal{D}_\varepsilon$ is maintained with auxiliary counts `vLV` and `uLv`. By Lemma 8 and Lemma 9, this can be done with $\varepsilon = \frac{1}{2}$ in amortized $\mathcal{O}(\sqrt{m})$ time and $\mathcal{O}(\min(m^{1.5}, n^2))$ space. The additional time for the query is $\mathcal{O}(m^{1-\varepsilon}) = \mathcal{O}(\sqrt{m})$. By Lemma 34, the worst-case time to query $\mathbf{c}(G, \mathcal{P}, e)$ for an arbitrary edge $e \in E$ then is $\mathcal{O}(\sqrt{m})$. ◀

Counting Claws

▶ **Lemma 37.** *Let $G = (V, E)$ be a dynamic graph, $\varepsilon \in [0, 1]$. We can query the number of claws containing two arbitrary vertices u and v, $u \neq v \in V$, as non-central vertices in worst-case $\mathcal{O}(m^{1-\varepsilon})$ time if we maintain the data structure $\mathcal{D}_\varepsilon$ with auxiliary count* `cLV`*.*

Proof. The number of claws where the central vertex has low degree is given directly by `cLV[`u, v`]`. To count all claws where the central vertex has high degree, we iterate over all $h \in \mathcal{H}$, test whether $\{u, h\}, \{v, h\} \in E$, and add up $\deg(h) - 2$. The running time is hence $\mathcal{O}(|\mathcal{H}|) = \mathcal{O}(m^{1-\varepsilon})$. ◀

Counting Paws

▶ **Lemma 38.** *Let $G = (V, E)$ be a dynamic graph, $\varepsilon \in [0, 1]$, and $\mathcal{P}$ be the paw −◁. We can query $\mathbf{c}(G, \mathcal{P}, \{u, v\})$ for an arbitrary edge $\{u, v\} \in E$ in worst-case $\mathcal{O}(\min(m^{\max(1-\varepsilon,\varepsilon)}, n))$ time if we maintain the data structure $\mathcal{D}_\varepsilon$ with auxiliary counts* `uLv`*,* `t`*, and* `cLV`*.*

Proof. Edge $\{u, v\}$ may be either (1) the arm, (2) one of the two triangle edges connected to the central vertex, or (3) the third triangle edge. Let $t_{\{u,v\}}$ be the number of 3-cycles containing $\{u, v\}$, which can be obtained in $\mathcal{O}(\min(m^{1-\varepsilon}, n))$ worst-case time by Lemma 30 if we maintain $\mathcal{D}_\varepsilon$ with auxiliary count `uLv`. Let t_u (t_v) be the number of triangles containing u (v), which can be obtained by Lemma 32 in $\mathcal{O}(\min(m^{2\varepsilon}, n^2))$ worst-case time by Lemma 32 if we maintain the data structure $\mathcal{D}_\varepsilon$ with auxiliary count `t`.

In case (1), the number of paws equals the number of 3-cycles containing either v or u, but not both, i.e., $t_u + t_v - 2t_{\{u,v\}}$. In case (2), the number of paws equals $t_{\{u,v\}}$ multiplied by $\deg_{\bar{v}}(u) - 1 + \deg_{\bar{u}}(v) - 1$, and in the case (3), it is the number of claws with low-degree central vertex, `cLV[`u, v`]`, plus those with a high-degree central vertex. This number can be obtained in $\mathcal{O}(|\mathcal{H}|) = \mathcal{O}(\min(m^{1-\varepsilon}, n))$ time by computing the sum of $\deg(h) - 2$ for all $h \in \mathcal{H}$ with $\{u, h\}, \{v, h\} \in E$. ◀

▶ **Lemma 39.** *Let $G = (V, E)$ be a dynamic graph and $\mathcal{P}$ the paw −◁. We can maintain $\mathbf{c}(G, \mathcal{P})$ in amortized $\mathcal{O}(m^{2/3})$ update time and $\mathcal{O}(n^2)$ space and query $\mathbf{c}(G, \mathcal{P}, e)$ for an arbitrary edge $e \in E$ in worst-case $\mathcal{O}(m^{2/3})$ time. Alternatively, we can maintain $\mathbf{c}(G, \mathcal{P})$ in amortized $\mathcal{O}(m)$ update time and $\mathcal{O}(n^2)$ space and query $\mathbf{c}(G, \mathcal{P}, e)$ for an arbitrary edge $e \in E$ in worst-case $\mathcal{O}(\sqrt{m})$ time.*

Proof. After an edge $\{u, v\}$ was inserted or before an edge $\{u, v\}$ is removed, the number of paws containing it can be obtained in $\mathcal{O}(\min(m^{\max(1-\varepsilon,2\varepsilon)}, n^2))$ worst-case time by Lemma 38 if $\mathcal{D}_\varepsilon$ with auxiliary counts `uLv`, `t`, and `cLV` is maintained. By Lemma 9, Lemma 10, and Lemma 12, this can be done in amortized $\mathcal{O}(m^\varepsilon + m^{\max(1-\varepsilon,\varepsilon)} + m^{2\varepsilon}) = \mathcal{O}(m^{\max(1-\varepsilon,2\varepsilon)})$ time and $\mathcal{O}(n^2)$ space. Together with the cost for the query, this yields an amortized update time of $\mathcal{O}(m^{\max(1-\varepsilon,2\varepsilon)}) = \mathcal{O}(m^{2/3})$ with $\varepsilon = \frac{1}{3}$. By Lemma 38, the worst-case time to query $\mathbf{c}(G, \mathcal{P}, e)$ for an arbitrary edge $e \in E$ then is $\mathcal{O}(m^{2/3})$. ◀

Counting 4-Cycles

▶ **Lemma 40.** *Let $G = (V, E)$ be a dynamic graph, $\varepsilon \in [0, 1]$, and $\mathcal{P}$ be the 4-cycle ◇. We can query $\mathbf{c}(G, \mathcal{P}, \{u, v\})$ for an arbitrary edge $\{u, v\} \in E$ in worst-case $\mathcal{O}(\min(m^{\max(1-\varepsilon,2\varepsilon)}, n^2))$ time if we maintain the data structure $\mathcal{D}_\varepsilon$ with auxiliary counts* `uLLv`*,* `uLv`*, and* `uHv`*.*

Proof. The 4-cycles containing an edge $\{u, v\}$ can be distinguished by the number of low-degree vertices among the other two vertices a, b forming the cycle $\langle\{u, v\}, \{v, a\}, \{a, b\}, \{b, u\}\rangle$.

The number of cycles where $a, b \in \mathcal{L}$ is given by $\texttt{uLLv[}u, v\texttt{]}$.

For the number of cycles with $a \in \mathcal{L}, b \in \mathcal{H}$ (analogously vice-versa with swapped rules for u and v), we use length-2 paths as follows: We iterate over all $h \in \mathcal{H} \setminus \{v\}$ such that $\{u, h\} \in E$ and obtain $\texttt{uLv[}v, h\texttt{]}$ in total $\mathcal{O}(|\mathcal{H}|) = \mathcal{O}(\min(m^{1-\varepsilon}, n))$ time. Each such vertex h is a possible choice for b. If $u \in \mathcal{H}$, the number of cycles equals the sum of $\texttt{uLv[}v, h\texttt{]}$ over all such h. Otherwise, $\texttt{uLv[}v, h\texttt{]}$ also counts the length-2 path $\langle\{h, u\}, \{u, v\}\rangle$, so we instead sum up $\texttt{uLv[}v, h\texttt{]} - 1$ for all such h.

For all cycles with $a, b \in \mathcal{H}$, we proceed as follows: If $u, v \in \mathcal{L}$, we iterate over all pairs of neighbors $x \in N_{\bar{v}}(u) \cap \mathcal{H}$ and $y \in N_{\bar{u}}(v) \cap \mathcal{H}$ in $\mathcal{O}(\deg(u) \cdot \deg(v)) = \mathcal{O}(\min(m^{2\varepsilon}, n^2))$ time. The number of 4-cycles equals the number of pairs x, y with $\{x, y\} \in E$. Otherwise, assume w.l.o.g. that $u \in \mathcal{H}$. If $v \in \mathcal{L}$, the number of 4-cycles is the sum of $\texttt{uHv[}u, h\texttt{]}$ over all $h \in \mathcal{H}$ with $\{h, v\} \in E$, and the sum of $(\texttt{uHv[}u, h\texttt{]} - 1)$ otherwise. This can be done in $\mathcal{O}(|H|) = \mathcal{O}(\min(m^{1-\varepsilon}, n))$ time. ◀

▶ **Lemma 41.** *Let $G = (V, E)$ be a dynamic graph and $\mathcal{P}$ the 4-cycle $\diamond$. We can maintain $\mathbf{c}(G, \mathcal{P})$ in amortized $\mathcal{O}(m^{2/3})$ update time and $\mathcal{O}(n^2)$ space. We can query $\mathbf{c}(G, \mathcal{P}, e)$ for an arbitrary edge $e \in E$ in worst-case $\mathcal{O}(m^{2/3})$ time.*

Proof. After an edge $\{u, v\}$ was inserted or before an edge $\{u, v\}$ is removed, the number of 4-cycles containing it can be obtained in $\mathcal{O}(\min(m^{\max(1-\varepsilon, 2\varepsilon)}, n^2))$ worst-case time by Lemma 40 if $\mathcal{D}_\varepsilon$ with auxiliary counts $\texttt{uLv}$, $\texttt{uLLv}$, and $\texttt{uHv}$ is maintained. By Lemma 9, Lemma 11, and Lemma 14, this can be done in amortized $\mathcal{O}(m^\varepsilon + m^{2\varepsilon} + m^{\max(1-\varepsilon, \varepsilon)}) = \mathcal{O}(m^{\max(1-\varepsilon, 2\varepsilon)})$ time and $\mathcal{O}(n^2)$ space. Together with the cost for the query, this yields a total amortized update time of $\mathcal{O}(m^{\max(1-\varepsilon, 2\varepsilon)}) = \mathcal{O}(m^{2/3})$ with $\varepsilon = \frac{1}{3}$. By Lemma 40, the worst-case time to query $\mathbf{c}(G, \mathcal{P}, e)$ for an arbitrary edge $e \in E$ then is $\mathcal{O}(m^{2/3})$. ◀

A.2 Counting Non-Induced s-Subgraphs

▶ **Lemma 42.** *Let $G = (V, E)$ be a dynamic graph, $\mathcal{P}$ be a 3-cycle $\triangle$ and $s \in V$. We can maintain $\mathbf{c}(G, \mathcal{P}, s)$ in amortized $\mathcal{O}(\sqrt{m})$ update time and $\mathcal{O}(n)$ space.*

Proof. If $\{u, v\}$ is inserted or deleted and $u \neq s \neq v$, the count changes by one if $\{u, s\}, \{v, s\} \in E$. Otherwise, if $\{s, x\}$ is inserted or deleted, the count changes by the number of triangles containing $\{s, x\}$, which can be obtained in $\mathcal{O}(m^{1-\varepsilon})$ time by Lemma 30 if we maintain $\mathcal{D}_\varepsilon$ with $\texttt{uLv}$.

By Lemma 9, $\mathcal{D}_\varepsilon$ with $\texttt{uLv}$ can be maintained in amortized $\mathcal{O}(m^\varepsilon)$ update time and $\mathcal{O}(n^2)$ space. In total, we can maintain 3-cycles containing s in amortized update time $\mathcal{O}(m^{\max(1-\varepsilon, \varepsilon)}) = \mathcal{O}(\sqrt{m})$ with $\varepsilon = \frac{1}{2}$ and $\mathcal{O}(n^2)$ space. The space can be reduced to $\mathcal{O}(n)$ by restricting $\texttt{uLv}$ such that it only maps single vertices and uses s as the second. ◀

▶ **Lemma 43.** *Let $G = (V, E)$ be a dynamic graph, $\mathcal{P}$ be a length-3 path, and $s \in V$. We can maintain $\mathbf{c}(G, \mathcal{P}, s)$ in amortized $\mathcal{O}(\sqrt{m})$ update time and $\mathcal{O}(n)$ space.*

Proof. If an edge $\{s, x\}$ is inserted or deleted, we count the number of affected paths in worst-case $\mathcal{O}(\min(m^{1-\varepsilon}, n))$ time by Lemma 34 if we maintain the data structure $\mathcal{D}_\varepsilon$ with auxiliary counts $\texttt{vLV}$ and $\texttt{uLv}$.

If an edge $\{u, v\}$ is inserted or deleted and $u \neq s \neq v$, the count changes by $\deg(s) - 1 + \deg_{\bar{u}}(v)$ if $\{s, u\} \in E$ and $\{s, v\} \notin E$. Vice versa, if $\{s, v\} \in E$ and $\{s, u\} \notin E$, it changes by $\deg(s) - 1 + \deg_{\bar{v}}(u)$. If both $\{s, u\}, \{s, v\} \in E$, the count changes by $2(\deg(s) - 2) + \deg_{\bar{u}}(v) - 1 + \deg_{\bar{v}}(u) - 1$. This accounts for paths where s is adjacent to either u or v. For paths via an intermediate low-degree vertex between s and u, the count changes by `uLv`$[s,u]$, subtracting 1 if $\{s,v\} \in E$, $\{u,v\}$ was added, and $v \in \mathcal{L}$. Symmetrically, the count changes by `uLv`$[s,v]$, subtracting 1 if $\{s,u\} \in E$, $\{u,v\}$ was added, and $u \in \mathcal{L}$. For paths with an intermediate high-degree vertex between s and u or v, we count all $h \in \mathcal{H} \setminus \{v,s\}$ with $\{h,s\},\{h,u\} \in E$ as well as all $h' \in \mathcal{H} \setminus \{u,s\}$ with $\{h',s\},\{h',g\} \in E$.

By Lemma 8 and Lemma 9, we can maintain $\mathcal{D}_\varepsilon$ with `vLV` and `uLv` in $\mathcal{O}(m^\varepsilon)$ amortized time and $\mathcal{O}(n^2)$ space. Iterating over all high-degree vertices can be done in $\mathcal{O}(|\mathcal{H}|) = \mathcal{O}(m^{1-\varepsilon})$ time, thus yielding an amortized update time of $\mathcal{O}(m^{\max(\varepsilon,1-\varepsilon)}) = \mathcal{O}(\sqrt{m})$ time for $\varepsilon = \frac{1}{2}$. The space complexity can be reduced to $\mathcal{O}(n)$ by only maintaining `uLv` for those vertex pairs where one vertex is s. ◀

▶ **Lemma 44.** *Let $G = (V,E)$ be a dynamic graph, $\mathcal{P}$ be the claw ⅄, and $s \in V$. We can maintain $\mathbf{c}(G,\mathcal{P},s)$ in worst-case $\mathcal{O}(1)$ update time and constant space.*

Proof. Only the insertion or deletion of an edge $\{s,x\}$ can change the number of claws containing s as the central vertex. In this case, we update the count by $\frac{1}{2}\deg_{\bar{x}}(s)\cdot(\deg_{\bar{x}}(s)-1)$ if $\deg_{\bar{x}}(s) \geq 2$. Symmetrically, for the number of claws where s is one of the non-central vertices and x is the center, we update the count by $\frac{1}{2}\deg_{\bar{s}}(x)\cdot(\deg_{\bar{s}}(x)-1)$ if $\deg_{\bar{s}}(x) \geq 2$.

In case that an edge $\{u,v\}$ is inserted or deleted and $u \neq s \neq v$, the count changes by $\deg_{\bar{v}}(u) - 1$ if $\deg_{\bar{v}}(u) \geq 1$ and $\{s,u\} \in E$, and by $\deg_{\bar{u}}(v) - 1$ if $\deg_{\bar{u}}(v) \geq 1$ and $\{s,v\} \in E$.

All steps can be done in constant time and space. ◀

▶ **Lemma 45.** *Let $G = (V,E)$ be a dynamic graph, $\mathcal{P}$ be the paw ⊲, and $s \in V$. We can maintain $\mathbf{c}(G,\mathcal{P},s)$ in amortized $\mathcal{O}(m^{2/3})$ time and $\mathcal{O}(n^2)$ space.*

Proof. There are three possibilities for a paw to contain s: (1) at the center, (2) at the other end of the arm, or (3) as one of the two non-central triangle vertices. Let t_v be the number of triangles containing a vertex v after the insertion or before the deletion of an edge, respectively.

If an edge $\{s,x\}$ is inserted or deleted, then by Lemma 38, we obtain the number of paws containing $\{s,x\}$ in worst-case $\mathcal{O}(\min(m^{\max(1-\varepsilon,2\varepsilon)}, n^2))$ time if we maintain the data structure $\mathcal{D}_\varepsilon$ with auxiliary counts `uLv`, `t`, and `cLV`.

If an edge $\{u,v\}$ is inserted or deleted and $u \neq s \neq v$, we distinguish two cases. If $\{s,u\} \in E$ ($\{s,v\} \in E$), the count changes by the difference in the number of paws containing $\{s,u\}$ ($\{s,v\}$) before and after the update. The difference can be obtained in at most four queries using again Lemma 38. If both $\{s,u\},\{s,v\} \in E$, we subtract $\deg(s)-2+\deg(u)-2+\deg(v)-2$ from the sum of both differences to avoid double-counting paws that contain the triangle s,u,v. By this, we cover possibility (1), (3), and partially also (2). The remaining situation is that s is the non-central vertex incident to the arm and $\{u,v\}$ is not incident to the central vertex. The number of paws in this case equals the number of claws containing u, v, and s with a low-degree central vertex, which is given by `cL`$[u,v,s]$, and the number of claws with a high-degree central vertex, which we obtain by counting the number of vertices $h \in \mathcal{H}$ with $\{h,s\},\{h,u\},\{h,v\} \in E$ in $\mathcal{O}(|\mathcal{H}|) = \mathcal{O}(m^{1-\varepsilon})$ time.

By Lemma 9, Lemma 10, and Lemma 12, we can maintain `uLv`, `t`, and `cLV` in amortized $\mathcal{O}(m^\varepsilon + m^{\max(1-\varepsilon,\varepsilon)} + m^{2\varepsilon}) = \mathcal{O}(m^{\max(1-\varepsilon,2\varepsilon)})$ time and $\mathcal{O}(n^2)$ space. `cL` can be maintained in $\mathcal{O}(m^{2\varepsilon})$ time and $\mathcal{O}(n\min(n^2,m^{3\varepsilon}))$ space by Lemma 15. With additional $\mathcal{O}(\min(m^{\max(1-\varepsilon,2\varepsilon)}, n^2) + m^{1-\varepsilon})$ time to update the count, we obtain a total amortized update time of $\mathcal{O}(m^{\frac{2}{3}})$ with $\varepsilon = \frac{1}{3}$. The total space complexity can be reduced to $\mathcal{O}(n^2)$ by not maintaining `cL` for arbitrary triples of vertices, but only for pairs and fixing s as third vertex. ◀

▶ **Lemma 46.** *Let $G = (V, E)$ be a dynamic graph, $\mathcal{P}$ be the 4-cycle ◇ and $s \in V$. We can maintain $\mathbf{c}(G, \mathcal{P}, s)$ in amortized $\mathcal{O}(m^{2/3})$ time and $\mathcal{O}(n^2)$ space.*

Proof. If an edge $\{s, x\}$ is inserted or deleted, the count changes by the number of 4-cycles containing $\{s, x\}$, which can be obtained in worst-case $\mathcal{O}(\min(m^{\max(1-\varepsilon,2\varepsilon)}, n^2))$ time by Lemma 40 if $\mathcal{D}_\varepsilon$ with auxiliary counts `uLLv`, `uLv`, and `uHv` is maintained.

If an edge $\{u, v\}$ is inserted or deleted and $u \neq s \neq v$, and $\{s, u\} \in E$, the count changes by the number of length-2 paths between s and v besides $\langle\{s, u\}, \{u, v\}\rangle$, which the sum of `uLv`$[s, v]$ and the number of high-degree vertices $h \in \mathcal{H}$ with $\{s, h\}, \{h, v\}$ minus 1. Analogously, if $\{s, v\} \in E$, the count changes by the number of length-2 paths between s and u besides $\langle\{s, v\}, \{v, u\}\rangle$, which is `uLv`$[s, u]$ plus the number of high-degree vertices $h \in \mathcal{H}$ with $\{s, h\}, \{h, u\}$ minus 1. This can be done in $\mathcal{O}(|\mathcal{H}|) = \mathcal{O}(m^{1-\varepsilon})$ time.

By Lemma 9, Lemma 11, and Lemma 14, we can maintain $\mathcal{D}_\varepsilon$ with an amortized update time of $\mathcal{O}(m^{\max(2\varepsilon,1-\varepsilon)})$ and $\mathcal{O}(n^2)$ space. Together with the additional time for updating the count, we arrive at an amortized update time of $\mathcal{O}(m^{\max(2\varepsilon,1-\varepsilon)}) = \mathcal{O}(m^{2/3})$ with $\varepsilon = \frac{1}{3}$ and a space complexity of $\mathcal{O}(n^2)$. ◀

▶ **Lemma 47.** *Let $G = (V, E)$ be a dynamic graph, $\mathcal{P}$ be the diamond ⬦ and $s \in V$. We can maintain $\mathbf{c}(G, \mathcal{P}, s)$ in amortized $\mathcal{O}(m^{2/3})$ update time and $\mathcal{O}(n^2)$ space.*

Proof. If an edge $\{s, x\}$ is inserted or deleted, the change in the number of diamonds can be obtained in $\mathcal{O}(\min(m^{\max(1-\varepsilon,2\varepsilon)}, n^2))$ worst-case time by Lemma 16 if we maintain the data structure $\mathcal{D}_\varepsilon$ with auxiliary counts `uLv`, `pLL`, `uHv`, and `cL`.

If an edge $\{u, v\}$ is inserted or deleted and $u \neq s \neq v$, we distinguish three cases: (1) $\{u, v\}$ is the chord, (2) s is incident to the chord, (3) u or v is incident to the chord, but not s and $\{u, s\}$ or $\{v, s\} \in E$. For case (1), if $\{u, s\}, \{v, s\} \in E$, the count changes by the number of triangles containing $\{u, v\}$ minus 1 to correct for the triangle containing s. For case (2), if $\{u, s\}, \{v, s\} \in E$, the count changes by the number of triangles containing $\{s, u\}$ minus 1 plus the number of triangles containing $\{s, v\}$ minus 1, correcting with minus 1 again for triangles containing v and u, respectively. For case (3), the count changes by the number of claws `cL`$[s, u, v]$ with a low-degree fourth vertex plus the number of high-degree vertices $h \in \mathcal{H}$ such that $\{h, s\}, \{h, u\}, \{h, v\} \in E$ if either $\{s, u\}$ or $\{s, v\} \in E$, and by twice this amount if both $\{s, u\}, \{s, v\} \in E$.

The number of triangles containing an edge $\{u, v\}$, $\{s, u\}$, or $\{s, v\}$ can be queried in $\mathcal{O}(m^{1-\varepsilon})$ worst-case time if $\mathcal{D}_\varepsilon$ with `uLv` is maintained by Lemma 30. We hence need to maintain $\mathcal{D}_\varepsilon$ with auxiliary counts `uLv`, `pLL`, `uHv`, and `cL`. By Lemma 9, Lemma 13, Lemma 14, and Lemma 15 and together with the query times to update the count, we arrive at an amortized total update time of $\mathcal{O}(m^{\max(2\varepsilon,1-\varepsilon)}) = \mathcal{O}(m^{2/3})$ with $\varepsilon = \frac{1}{3}$. The space complexity can be reduced to $\mathcal{O}(n^2)$ by maintaining `cL` only for those triplets that contain s. ◀

▶ **Lemma 48.** *Let $G = (V, E)$ be a dynamic graph, $\mathcal{P}$ be the 4-clique ⊠, and $s \in V$. We can maintain $\mathbf{c}(G, \mathcal{P}, s)$ in amortized $\mathcal{O}(m)$ update time and constant space.*

Proof. If an edge $\{s, x\}$ is inserted or deleted, we obtain the change in the number of 4-cliques by querying $\mathbf{c}(G, \mathcal{P}, \{s, x\})$, which can be done in $\mathcal{O}(m)$ worst-case time and constant space by Theorem 4. If an edge $\{u, v\}$ is inserted or deleted, $u \neq s \neq v$, and $\{u, s\}, \{v, s\} \in E$, we iterate over all $y \in V$ and test whether $\{s, u, v, y\}$ form a 4-clique. This takes $\mathcal{O}(n)$ time. In total, we arrive at a worst-case update time of $\mathcal{O}(m)$ and constant space. ◀

B Omitted Proofs from Sect. 4

▶ **Observation 49.** *Every path in $G_{M,g,h}$ that contains two distinct vertices $l_i \neq l_{i'} \in L$ (resp. $r_j \neq r_{j'} \in R$) has length at least $2g+2$ (resp. $2h+2$).*

Each case in Lemma 29 corresponds to one of the following lemmas:

(s-)k-Odd Cycle Detection

▶ **Lemma 50** ((s-)k-Odd Cycle Detection). *Given a partially dynamic algorithm $\mathcal{A}$ for (s-)k-cycle detection with $k \geq 3$ and k odd, one can solve 1-uMv with parameters n_1 and n_2 by running the preprocessing step of $\mathcal{A}$ on a graph with $\mathcal{O}(m + \sqrt{m} \cdot k)$ edges and $\Theta(\sqrt{m} \cdot k)$ vertices, and then making $\mathcal{O}(\sqrt{m})$ insertions (or $\mathcal{O}(\sqrt{m})$ deletions) and 1 query, where m is such that $n_1 = n_2 = \sqrt{m}$.*

Proof. We only prove the decremental case, the incremental case has a symmetric proof. Consider a 1-uMv problem with $n_1 = n_2 = \sqrt{m}$. Given M, we construct the $(k-1)$-partite graph G from $G_{M,g,h}$ with $g = h = \frac{1}{2}(k-3)$ by adding to it a vertex s and edges $\{s, l_i\}$, $\{r_j, s\}$ for all $r_j \in R$, $l_i \in L$. Thus, the total number of edges is at most $n_1 n_2 + \frac{1}{2}(k-1)(n_1+n_2) = \mathcal{O}(m + k \cdot \sqrt{m})$. Once u and v arrive, we delete $\{s, l_i\}$ iff $u_i = 0$ and delete $\{r_j, s\}$ iff $v_j = 0$. See Figure 3 for an example with $k = 5$.

We have $u^\top M v = 1$ iff there is a cycle of length k in G iff there is a cycle of length k incident to s: By Observation 28, every cycle that does not contain s is even. By Observation 49, every cycle that contains s but does not contain both a vertex in L and R has length at least $k+1$. In total, we need to do $n_1 + n_2 = \mathcal{O}(\sqrt{m})$ updates and 1 query. ◀

(s-)Paw Detection

▶ **Lemma 51** ((s-)Paw Detection). *Given a partially dynamic algorithm $\mathcal{A}$ for (s-)paw detection, one can solve 1-uMv with parameters n_1 and n_2 by running the preprocessing step of $\mathcal{A}$ on a graph with $\mathcal{O}(m + \sqrt{m} \cdot k)$ edges and $\Theta(\sqrt{m} \cdot k)$ vertices, and then making $\mathcal{O}(\sqrt{m})$ insertions (or $\mathcal{O}(\sqrt{m})$ deletions) and 1 query, where m is such that $n_1 = n_2 = \sqrt{m}$.*

Proof. Construct G as in the proof of Lemma 50 for $k = 3$ and add an additional vertex t and edge $\{s, t\}$. Update G as in the proof of Lemma 50. Then, $u^\top M v = 1$ iff there is a paw in G iff there is a paw incident to s and we need to do $n_1 + n_2 = \mathcal{O}(\sqrt{m})$ updates and 1 query. ◀

▶ **Lemma 52** ((s-)4-Clique Detection). *Given a partially dynamic algorithm $\mathcal{A}$ for (s-)4-clique detection, one can solve 1-uMv with parameters n_1 and n_2 by running the preprocessing step of $\mathcal{A}$ on a graph with $\mathcal{O}(m)$ edges and $\Theta(\sqrt{m})$ vertices, and then making $\mathcal{O}(\sqrt{m})$ insertions (or $\mathcal{O}(\sqrt{m})$ deletions) and 1 query, where m is such that $n_1 = n_2 = \sqrt{m}$.*

Proof of Lemma 52. We only prove the decremental case. Consider a 1-uMv problem with $n_1 = n_2 = \sqrt{m}$. Given M, we construct the tripartite graph G from $G_{M,1,0}$ by adding to it a vertex s and edges $\{s, l_i\}$, $\{s, l_i^{(1)}\}$, $\{r_j, s\}$, $\{l_i, r_j\}$ for all $r_j \in R$, $l_i \in L$. Thus, the total number of edges is at most $2n_1 n_2 + 3n_1 + n_2 = \mathcal{O}(m)$. Once u and v arrive, we delete $\{s, l_i\}$ and $\{s, l_i^{(1)}\}$ iff $u_i = 0$ and delete $\{r_j, s\}$ iff $v_j = 0$. See Figure 3 for an example.

Consider the case that G contains a 4-clique (i.e., K_4). Note that each edge of the clique is shared by exactly two triangles, whose vertices together induce K_4. In consequence of Observation 28, every triangle must be incident to s or contain an edge $\{l_i, r_j\}$, $l_i \in L, r_j \in R$

for some $1 \leq i \leq n_1$, $1 \leq j \leq n_2$. As s can be only one vertex of K_4 and the three remaining vertices also form a triangle in G, K_4 must contain an edge $\{l_i, r_j\}, l_i \in L, r_j \in R$ for some $1 \leq i \leq n_1$, $1 \leq j \leq n_2$. By construction, each vertex $l_i \in L$ has an edge to $l_i^{(1)}$ and possibly s, and apart from that has only neighbors in R. Thus, $\{l_i, r_j\}$ is always part of a triangle $\{l_i, l_i^{(1)}, r_j\}$, and possibly a second triangle $\{s, l_i, r_j\}$. However, by construction, no triangle $\{l_i, r_j, r_{j'}\}$ with $j \neq j'$ can exist, as $G_{M,1,0}$ is tripartite and no edge between two vertices in R has been added. Thus, if $\{l_i, r_j\}$ is part of K_4, there are triangles $\{l_i, l_i^{(1)}, r_j\}$ and $\{s, l_i, r_j\}$. By construction, G then also contains $\{s, l_i^{(1)}\}$, which implies that K_4 is induced by the vertices $\{s, l_i, l_i^{(1)}, r_j\}$.

Hence, we have $u^\top M v = 1$ iff there is a 4-clique in G iff s is a vertex of a 4-clique in G. In total, we need to do $2n_1 + n_2 = \mathcal{O}(\sqrt{m})$ updates and 1 query. ◀

▶ **Lemma 53** ((s-)Length-3 Path Counting). *Given a partially dynamic algorithm $\mathcal{A}$ for counting the number of (s-)length-3 paths, one can solve 1-uMv with parameters n_1 and n_2 by running the preprocessing step of $\mathcal{A}$ on a graph with $\mathcal{O}(m + \sqrt{m} \cdot k)$ edges and $\Theta(\sqrt{m} \cdot k)$ vertices, and then making $\mathcal{O}(\sqrt{m})$ insertions (or $\mathcal{O}(\sqrt{m})$ deletions) and 1 query, where m is such that $n_1 = n_2 = \sqrt{m}$.*

Proof. We only prove the incremental case, the decremental case is symmetric. Consider a 1-uMv problem with $n_1 = n_2 = \sqrt{m}$. Given M, we construct the tripartite graph G from $G_{M,g,h}$ with $g = h = 0$ by adding two isolated vertices s, t. We also count for each vertex $v \in R \cup L$ the number of paths of length 2 starting at v and the total number of length-3 paths in G. Once u and v arrive, we add $\{s, l_i\}$ iff $u_i = 1$ and $\{r_j, t\}$ iff $v_j = 1$. Furthermore, we count the number k_u of 1-entries in u and the number k_v of 1-entries in v. If $u_i = 1$, we increment the length-3 path counter by the number of paths of length 2 starting at l_i plus $(k_u - 1) \cdot \deg(l_i)$. Likewise, if $v_j = 1$, we increment the length-3 path counter by the number of paths of length 2 starting at r_j plus $(k_v - 1) \cdot \deg(r_j)$. We have $u^\top M v = 1$ iff the number of 3-paths in G is greater than the length-3 path counter. In total, we need to do $n_1 + n_2 = \mathcal{O}(\sqrt{m})$ updates and 1 query. ◀

For lower bounds on detecting and counting cycles of even length, we modify $G_{M,g,h}$ by duplicating the left-hand side vertices as well as the edges depending on M (see Figure 4 for an example):

▶ **Definition 54** ($H_{M,g,h}$). *Given a matrix $M \in \{0,1\}^{n_1 \times n_2}$ and two integers $g, h \geq 1$, we denote by $H_{M,g,h} = (T^{(0)} \cup \cdots \cup T^{(g)} \cup B^{(0)} \cup \cdots \cup B^{(g)} \cup R^{(0)} \cup \cdots \cup R^{(h)}, E_L \cup E_R \cup E_{M_T} \cup E_{M_B})$ the $(g + h + 2)$-partite graph where*

$$
\begin{aligned}
T^{(p)} &= \left\{t_1^{(p)}, \ldots, t_{n_1}^{(p)}\right\}, \quad B^{(p)} = \left\{b_1^{(p)}, \ldots, b_{n_1}^{(p)}\right\}, \quad 0 \leq p \leq g, \\
E_L &= \left\{\{t_i^{(p)}, t_i^{(p+1)}\}, \{b_i^{(p)}, b_i^{(p+1)}\} \mid 1 \leq i \leq n_1 \wedge 1 \leq p < g\right\} \\
R^{(q)} &= \left\{r_1^{(q)}, \ldots, r_{n_2}^{(q)}\right\}, \quad 0 \leq q \leq h; \qquad E_R = \{(r_j^{(q)}, r_j^{(q+1)}) \mid 1 \leq j \leq n_2 \wedge 1 \leq q < h\} \\
E_{M_T} &= \left\{\{t_i^{(0)}, r_j^{(0)}\}, \{t_i^{(g)}, r_j^{(h)}\} \mid M_{ij} = 1 \wedge i \leq j\right\} \\
E_{M_B} &= \left\{(b_i^{(0)}, r_j^{(0)}), (b_i^{(h)}, r_j^{(h)}) \mid M_{ij} = 1 \wedge i > j\right\}.
\end{aligned}
$$

$H_{M,g,h}$ has $2 \cdot (g + 1) \cdot n_1 + (h + 1) \cdot n_2$ vertices and at most $2n_1n_2 + 2 \cdot g \cdot n_1 + h \cdot n_2$ edges. All vertices x in $T^{(p)}$, $B^{(p)}$ for $1 \leq p < g$ and in $R^{(q)}$ for $1 \leq q < h$ have $\deg(x) = 2$.

▶ **Observation 55.** *$H_{M,g,h}$ is acyclic (a forest).*

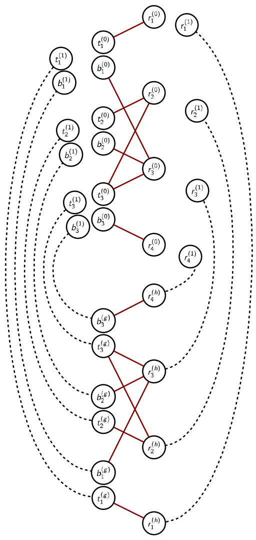

Figure 4 Construction of $H_{M,g,h}$ for $M = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix}$.

▶ **Observation 56.** *Every path in $H_{M,g,h}$ that contains two distinct vertices $x_i \neq x_j \in T^{(1)} \cup B^{(1)}$ ($r_i \neq r_j \in R^{(1)}$) has length at least $2g$ ($2h$).*

▶ **Lemma 57** (k-Cycle Detection)**.** *Given a partially dynamic algorithm $\mathcal{A}$ for k-cycle detection with $k \geq 4$, one can solve* 1-uMv *with parameters n_1 and n_2 by running the preprocessing step of $\mathcal{A}$ on a graph with $\mathcal{O}(m + \sqrt{m} \cdot k)$ edges and $\Theta(\sqrt{m} \cdot k)$ vertices, and then making $\mathcal{O}(\sqrt{m})$ insertions (or $\mathcal{O}(\sqrt{m})$ deletions) and 1 query, where m is such that $n_1 = n_2 = \sqrt{m}$.*

Proof. Consider a 1-uMv problem with $n_1 = n_2 = \sqrt{m}$. Given M, we construct the k-partite graph H from $H_{M,g,h}$ with $g = \lceil \frac{k}{2} \rceil - 1$, $h = \lfloor \frac{k}{2} \rfloor - 1$ by adding edges $\{t_i^{(0)}, t_i^{(1)}\}$, $\{b_{i'}^{(0)}, b_{i'}^{(1)}\}$, $\{r_j^{(0)}, r_j^{(1)}\}$, for all $t_i^{(0)} \in T, b_{i'}^{(0)} \in B, r_j \in R^{(0)}$. Thus, the total number of edges is at most $2n_1n_2 + 2(\lceil \frac{k}{2} \rceil - 1) \cdot n_1 + (\lfloor \frac{k}{2} \rfloor - 1) \cdot n_2 = \mathcal{O}(m + k \cdot \sqrt{m})$. Once u and v arrive, we delete $(t_i^{(0)}, t_i^{(1)})$ and $\{b_i^{(0)}, b_i^{(1)}\}$ iff $u_i = 0$ and delete $\{r_j^{(0)}, r_j^{(1)}\}$ iff $v_j = 0$. See Figure 5 for an example with $k = 4$.

We have $u^\top M v = 1$ iff there is a cycle of length k in H: By Observation 55, every cycle must contain at least one vertex in $T^{(1)}$, $B^{(1)}$, or $R^{(1)}$. By Observation 56, every cycle that contains a vertex in $T^{(1)} \cup B^{(1)}$, but not in $R^{(1)}$ or vice versa has length at least $2(\lfloor \frac{k}{2} \rfloor - 1) + 4 \geq k + 1$. In total, we need to do $2n_1 + n_2 = \mathcal{O}(\sqrt{m})$ updates and 1 query. ◀

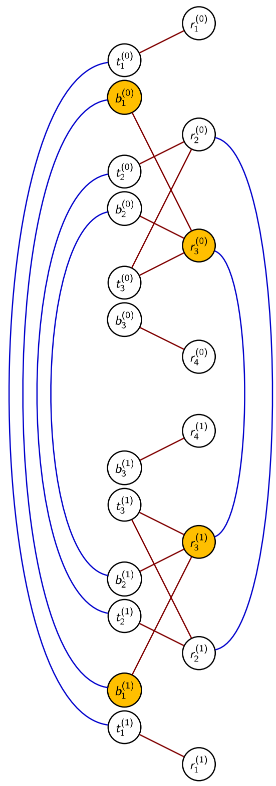

Figure 5 Construction of H for 4-cycle detection with $M = \left(\begin{smallmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{smallmatrix}\right)$, $u^\top = \begin{pmatrix} 1 & 1 & 0 \end{pmatrix}$, and $v^\top = \begin{pmatrix} 0 & 1 & 1 & 0 \end{pmatrix}$. The orange vertices $b_1^{(0)}, r_3^{(0)}, r_3^{(1)}, b_1^{(1)}$ form a 4-cycle.