Smoothed Analysis of Dynamic Graph Algorithms

Uri Meir * Ami Paz †

Abstract

Recent years have seen significant progress in the study of dynamic graph algorithms, and most notably, the introduction of strong lower bound techniques for them (e.g., Henzinger, Krinninger, Nanongkai and Saranurak, STOC 2015; Larsen and Yu, FOCS 2023). As worst-case analysis (adversarial inputs) may lead to the necessity of high running times, a natural question arises: in which cases are high running times really necessary, and in which cases these inputs merely manifest unique pathological cases?

Early attempts to tackle this question were made by Nikoletseas, Reif, Spirakis and Yung (ICALP 1995) and by Alberts and Henzinger (Algorithmica 1998), who considered models with very little adversarial control over the inputs, and showed fast algorithms exist for them. The question was then overlooked for decades, until Henzinger, Lincoln and Saha (SODA 2022) recently addressed uniformly random inputs, and presented algorithms and impossibility results for several subgraph counting problems.

To tackle the above question more thoroughly, we employ *smoothed analysis*, a celebrated framework introduced by Spielman and Teng (J. ACM, 2004). An input is proposed by an adversary but then a noisy version of it is processed by the algorithm instead. This model of inputs is parameterized by the amount of adversarial control, and fully interpolates between worst-case inputs and a uniformly random input. Doing so, we extend impossibility results for some problems to the smoothed model with only a minor quantitative loss. That is, we show that partially-adversarial inputs suffice to impose high running times for certain problems. In contrast, we show that other problems become easy even with the slightest amount of noise. In addition, we study the interplay between the adversary and the noise, leading to three natural models of smoothed inputs, for which we show a hierarchy of increasing difficulty stretching between the average-case and the worst-case complexities.

^{*}Tel-Aviv University, Tel-Aviv, Israel. Email: urimeir.cs@gmail.com

[†]LISN — CNRS & Université Paris-Saclay. Email: ami.paz@lisn.fr

Contents

1	Intr	roduction	1							
	1.1	Smoothed dynamic graphs	2							
	1.2	Models of smoothing	3							
		1.2.1 Oblivious adversaries	3							
		1.2.2 Adaptive adversary	4							
	1.3	Main results	4							
		1.3.1 Decision problems	5							
		1.3.2 Counting small subgraphs	6							
		1.3.3 Hierarchy between smoothing models	7							
	1.4	A technical overview of the subgraph counting lower bounds	8							
	1.5	Related work	12							
		1.5.1 Dynamic graph algorithms	12							
		1.5.2 Beyond worst-case analysis of dynamic graph algorithms	12							
		1.5.3 Smoothed analysis in different models	13							
0	D	1* *	10							
2	Pre	liminaries	13							
3	The	e relative power of the adversaries	15							
4	Upp	Upper bounds								
	4.1	Easy problems with an oblivious flip adversary	18							
	4.2	Counting small subgraphs with any adversary	24							
		4.2.1 Counting short s - t paths	24							
		4.2.2 Counting short cycles through a node	26							
5	Low	ver bounds via embedding	28							
	5.1	Polynomial lower bounds	28							
	5.2	Connectivity lower bound								
	5.3	Lower bounds with oblivious add/remove adversary	33							
6	Low	ver bounds for counting small subgraphs	35							
Ū	6.1	Solving OuMv by counting s-t 3-paths on P_3 -partite graphs								
	0.1	6.1.1 The setting								
		6.1.2 The reduction	37							
		6.1.3 Analysis	37							
	6.2	Reducing P_3 -partite graphs to general graphs	41							
	٠	6.2.1 Simulating a single step	42							
		6.2.2 The actual reduction	43							
		6.2.3 Analysis	43							
	6.3	Putting it all together	47							
	6.4	Lower bounds for other counting problems	47							
Α	Too	ols from probability theory	49							
В	was	ssaging the OMv problem	50							

1 Introduction

We study dynamic graph algorithms—data structures for handling graphs that undergo changes. The input is composed of an initial n-node graph, followed by a sequence for edge changes (addition or removal) and queries. The goal is to maintain enough information on the graph in order to promptly answer a predetermined query (e.g., is the graph connected, or how many paths of length 3 connect two predefined nodes).

Recent years have seen a large body of work regarding lower bounds for dynamic graph algorithms. One line of work concerns unconditional lower bounds [5,30,34,36]; for example, deciding the connectivity of a dynamic graph (i.e. whether the graph is connected) requires $\Omega(\log n)$ time per update provided the query time is constant [34]. Another celebrated line of work proves polynomial conditional lower bounds [1,11,20,27,35,37]; for example, even the seemingly simple problem of counting paths of length 3 between two predefined nodes s and t (henceforth, s-t 3-paths) requires $\Omega(n^{1-\epsilon})$ time per update¹ unless queries take roughly quadratic time [20], assuming the OMv conjecture (see Section 1.4). Similarly to other algorithmic models, it is natural to ask whether known results in worst-case analysis² encapsulate inherent impossibilities, or just superficially allow for pathological yet unrealistic examples.

Preceding the advances in lower bound techniques, a couple of early works proposed algorithms specialized for distributional inputs of restricted form, with the number of edges (the graph density) playing a crucial role. In [33], the input sequence is assumed to be entirely stochastic (no adversarial choices) and follow simple rules to ensure a predefined density. In [2], an adversary determines ahead of time which steps will add an edge and which will remove one, but the choice of edges to add or remove is made uniformly at random. Consequently, the graph in each round is distributed uniformly over all graphs of a given density, a fact that is crucial in the analysis.

Following advances in lower bound techniques for dynamic graph algorithms (and for fine-grained complexity in general), a recent work [21] revived the average-case analysis of dynamic graph algorithms, offering upper and lower bounds for subgraph counting problems under uniformly random inputs. For example, it shows that s-t 3-paths can be counted with O(1) update and query times on such inputs, while maintaining the number of such paths of length 5 requires either $\Omega(n^{1-\epsilon})$ time per update or roughly quadratic time per query. Note that some classical problems in the field, such as connectivity, are trivial for a uniformly random input: one can blindly answer each query positively, since a random graph is connected w.h.p.

These results point to the fragility of existing lower bounds, but they only apply with very little adversarial control over the input. In our work, we strive to capture input models in which the adversary has more, but not perfect control. In particular, we are led by the following question.

How robust are lower bounds for dynamic graph algorithms?

To address this question, we apply *smoothed analysis*, a celebrated theoretical framework introduced by Spielman and Teng [40, 41]. In this approach, a smoothed input is created by considering an adversarial input and then perturbing it by adding random noise; the complexity is computed with respect to the smoothed input (e.g., on expectation). Smoothed analysis was applied to different types of inputs; when used on graphs, a random perturbation is applied to an adversarially-chosen graph by independently flipping the (in)existence of each edge with a certain

¹That is, cannot be solved in $O(n^{1-\epsilon})$ time per update, for any constant $\epsilon > 0$

²In the literature of dynamic graph algorithms, worst-case running time is sometimes contrasted with amortized running time. Here and all throughout, worst-case simply refers to an adversarial input, as opposed to a uniformly random (average-case) input or a smoothed input.

probability [14, 28, 29]. A more recent line of work applied smoothed analysis to a graph-based computational model closer to ours called dynamic networks, where the computation also occurs in rounds (note that this is a model of distributed computation, while ours is centralized). There, each round is executed with an entirely new communication graph, and a smoothed input is simply defined by perturbing each such graph independently as above [9, 10, 31].

We continue the conceptual line of applying random perturbations. In dynamic graph algorithms each round consists of a *single* edge change, which is too subtle to invoke noise through a global perturbation as in the aforementioned cases. Instead, we apply smoothing per round: with some probability and independently of other rounds, we replace the adversarial edge with a uniformly random edge. Over many changes, this accumulates to a global perturbation, naturally extending previous graph smoothing models. This approach yields diverse input models and results.

In addition to its theoretical merits, smoothed analysis is often argued to be a realistic model of inputs. This applies for dynamic graphs as well as to any computation model, since inputs are prone to minor perturbations before being processed (e.g., due to communication failures or human errors). That is, even if (due to chance or malicious intentions) the intended input should impose high running times, it is well-incentivized to analyze the de facto performance of an algorithm assuming a perturbed version of the input is processed instead (i.e., a smoothed input).

1.1 Smoothed dynamic graphs

We consider fully dynamic graphs with a fixed set of nodes V = [n] and changing edges. The input consists of the edges of an initial graph, followed by a sequence of edges to be flipped. The complexity of a dynamic graph algorithm is measured by the time $t_{\rm pre}$ it needs for processing the initial graph, the time $t_{\rm u}$ for processing an edge change, and the time $t_{\rm q}$ for answering a query.

We next define a p-smoothed input, for a parameter $0 \le p \le 1$, which we name the *oblivious flip adversary* model of smoothing. First, the adversary fixes an initial graph and a sequence of edge flips. The initial graph is smoothed as follows: each node pair remains consistent with the adversarial choice with probability p, and otherwise re-sampled (uniformly from $\{0,1\}$). Then, each edge flip remains unchanged with probability p, and otherwise re-sampled uniformly from $\binom{[n]}{2}$. Clearly, when p=0 the process coincides with an average-case (uniformly random) input, while for p=1 it results in the standard adversarial (worst case) input.

Study cases. Smoothed analysis can be relevant to any problem presenting a gap between the worst-case and average-case complexities. We consider two classes of such problems: counting small subgraphs, and deciding certain graph properties. Interestingly, the different classes exhibit fundamentally different smoothed complexities as a function of p, the fraction of adversarial changes.

Two examples are depicted in Table 1 for constant values of p. The complexity of counting s-t 3-paths increases quickly with p, e.g., for all values $p \in [0.01, 1]$ the complexity is asymptotically identical to the worst-case (p = 1). In contrast, the complexity of connectivity remains as in the average case (p = 0) for any constant p; e.g., it is constant for any $p \in [0, 0.99]$. This already exemplifies different behaviors in terms of noise robustness: the hardness of counting subgraphs hinges on a small set of adversarial edges and therefore persists even with a lot of randomness, while the decision problems we consider concern with global properties, and even a small amount of noise guarantees that a blind positive answer to all queries is correct w.h.p.

Problem	Average case $(p=0)$	Smoothed case $(0$	Worst case $(p=1)$
s-t 3-paths	O(1) [21]	$\Omega(n^{1-\epsilon}), O(n)$ T.6.11, L.4.8	$\Omega(n^{1-\epsilon}), O(n)$ [20], [18]
Connectivity	O(1) L.4.6	O(1) L.4.6	$\Omega(\log n), \tilde{O}(\log n)$ [34], [22]

Table 1: The update time complexities of s-t 3-paths and connectivity, with a non-adaptive flip adversary and constant $0 . The update times are for <math>t_q = O(1)$ and $t_{pre} = O(n^{3-\epsilon})$. The s-t 3-paths lower bounds are conditioned on the OMv conjecture.

A smooth transition. A recent work on average-case complexity [21] focuses on counting small subgraphs. Perhaps the strongest message in it is that some problems, such as counting s-t 5-paths, are hard not only in the worst-case but also on average. In contrast, counting s-t 2-paths is easy even in the worst-case (that is, solvable with constant update and query times). Our focus is the third option presented: for a few problems, including counting s-t 3-paths and s-t 4-paths, there is a gap: these problems can be solved quickly in the average case, although they admit strong (conditional) lower bounds in the worst-case.

For constant p, we show that counting s-t 3-paths and s-t 4-paths are as hard as the worst case and require linear update time. More interestingly, for sub-constant p, we show that both can be solved with O(pn) update time and constant query time, while $\Omega(pn^{1-\epsilon})$ time per update is necessary even if we allow almost quadratic query time. That is, the complexity linearly depends on p, the fraction of adversarial changes, as stated in Table 2. In particular, the problem is as easy as the average case for very small values of p (e.g., p = 1/n), but becomes asymptotically as hard as the worst case already with p = 0.01, with a gradual transition between the two.

1.2 Models of smoothing

Smoothed analysis was originally introduced for studying the simplex algorithm, where the inputs are continuous and the added noise is Gaussian. The field has greatly evolved since: first, it is now typical to analyze the complexity of a computational problem, rather than that of a specific solution (algorithm), leading to more robust results. Second, the model of random noise was adjusted to discrete inputs, which are common throughout computer science. Lastly, the framework has also been applied to several different computational models, each posing its own unique challenges.

In our case, the inputs are discrete and arrive in an online manner. This gives rise to several natural models of smoothing, differing in the adaptivity of the adversary to the noise and in the way changes are encoded. In the following we present three variants of p-smoothed inputs: oblivious flip adversary, oblivious add/remove adversary, and an adaptive adversary. Each model constitutes a different intermediate model between worst-case and average-case, and the relations between them is depicted in Fig. 3. In Fig. 5 we exemplify how the complexities of concrete problems increase differently as we transition from easier to harder models of input.

1.2.1 Oblivious adversaries

Operation types in worst and average cases. A change in a dynamic graph can be seen as choosing a potential edge and flipping it, or as choosing a potential edge and whether to add or to remove it. In the typical worst-case analysis, both views coincide: an algorithm can always check the edge's status in O(1) time and do nothing if it remained unchanged, so an adversary will never choose to leave an edge unchanged. Hence, worst-case complexity does not depend on the operation type. Prior work concerning average-case complexity does not address the relation between flip and add/remove operations. In Section 3 we show that up to constant factors, the average-case complexity of any problem is also independent of the operations' encoding.

Problem	Average case	Smoothed case		Worst case	
s-t 2-paths	O(1)				[21]
s-t 3-paths	O(1) [21]	$\Omega(pn^{1-\epsilon}), O(pn)$	T.6.11, L.4.8	$\Omega(n^{1-\epsilon}), O(n)$	[20], [18]
s-t 4-paths	O(1) [21]	$\Omega(pn^{1-\epsilon}), O(pn)$	T.6.12, L.4.9	$\Omega(n^{1-\epsilon}), O(n)$	[20], C.4.10
s-t 5-paths	$\Omega(n^{1-\epsilon})$				[21]

Table 2: The update complexities $t_{\rm u}$ of short paths counting, for $t_{\rm q}=O(1)$ and $t_{\rm pre}=O(n^{3-\epsilon})$, as a function of p. For these problems the smoothed complexities with all adversaries coincide. The lower bounds are conditioned on the OMv conjecture.

Operation types in smoothed analysis. We have already described the oblivious flip adversary, which chooses a sequence of *edges to flip*, and then some of these edges are re-sampled uniformly. The *oblivious add/remove adversary* is similarly defined, but instead of edges to flip, it chooses *edges to add or remove* before some of its operations are replaced by random ones.

A priori, one might suspect that an oblivious add/remove adversary is stronger than an oblivious flip one, as the former can, e.g., fixate on certain edges and persistently remove them. But one might also suspect the opposite, as an oblivious flip adversary is guaranteed to make a change at each round, while an add/remove adversary might "waste" rounds by trying to add an existing edge or remove a missing one. As it turns out, the first intuition is true: if p is high enough, an add/remove adversary can simulate a hard case instance on a small subgraph, which we use in order to prove lower bounds for it. Furthermore, using a proxy model with lazy flips we show that the oblivious add/remove adversary is always at least as strong as the flip one: the p-smoothed complexity in the oblivious flip model is not higher than the p'-smoothed complexity in the oblivious add/remove model, for some $p' \in [p/2, p]$.

1.2.2 Adaptive adversary

Another issue that emerges when defining smoothing models is *adaptivity to the noise*. Indeed, in a process that involves both adversarial and random choices, a present ("online") adversary aware of the random noise applied might disrupt the algorithm more than an oblivious one. Similar distinctions were recently made in smoothed analysis of dynamic networks [31] and online learning [17].

Unlike the oblivious adversaries, an *adaptive adversary* can choose its edge changes in an online manner — it chooses an edge to flip knowing all previous changes (adversarial or randomly-chosen). We show that this is sometimes enough in order to cause much slower running times, separating the oblivious and the adaptive smoothed models. As in the worst case, an adaptive adversary never asks to add an existing edge or to remove a non-existing one, thus defining it with flip operations or with add/remove ones makes no difference.

Queries. Queries are part of the input, and are sometimes even crucial for devising worst-case lower bounds, especially for problems where queries are parametrized. This is the case, e.g., in the lower bound for (u, v)-connectivity [34], where queries are made with parameters u, v and ask whether the nodes u and v are currently in the same connected component. Hence, parameterized queries could also be studied in a smoothed manner (and in the average-case), yet the problems studied here have no parameterized queries and the issue is left for future work.

1.3 Main results

Table 4 compares our results to known results in average-case and worst-case models, focusing on update times of algorithms with constant query times. We discuss these results below.

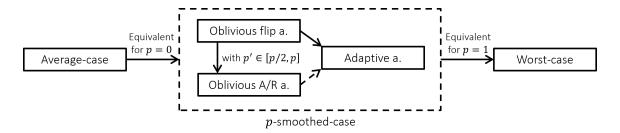


Figure 3: Relations between models (Section 1.2 and Section 1.3.3). An arrow from model A to model B represents that model B is at least as hard as model A (that is, the complexity of a problem can only increase). A solid arrow represents that this relation is strict, i.e., for some problem and range of p, the asymptotic complexity increases from A to B.

1.3.1 Decision problems

We consider several decision problems, such as connectivity and bipartite perfect matching (deciding whether a bipartite graph contains a perfect matching). These are trivial to solve on a random static graph since they hold w.h.p. In Section 4.1 we point out that these hold w.h.p. also at each round of an average-case dynamic graph, and are thus also easy to decide in this setting. We extend this argument to p-smoothed dynamic graphs with an oblivious flip adversary, show it holds even with very little randomness (p being almost 1), and thus reach a trivial constant-time algorithm in this setting as well (Lemma 4.6). Specifically, we show that the random perturbation of the initial graph guarantees the first (polynomially many) graphs in the sequence hold such properties with a polynomially small error, allowing for a trivial "yes" answer. To deal with longer sequences of changes, we argue even more randomness affects the later rounds, and each graph now holds the desired properties with only an exponentially small error. Despite the simple intuition, the formal argument here is not completely straightforward, as the graph at each round is in fact statistically far from a uniformly random graph (e.g., the parity of the number of edges distinguishes the two).

Theorem A (informal). For any $p \in [0, 1 - \frac{26 \log n}{n})$, connectivity and bipartite perfect matching can be decided in constant update and query times under the oblivious flip adversary.

This theorem might raise the question whether the oblivious flip adversary has any power at all, or does it coincide with the average-case complexity. We later separate the models by showing they induce different complexities for subgraph counting problems.

While the trivial "yes" algorithm works well for bipartite perfect matching with an oblivious flip adversary, it fails for the stronger adversaries — oblivious add/remove adversary, and adaptive adversary. To see this, consider an adversary that repeatedly removes all the edges touching a fixed node; if p is relatively large, this adversary will manage to isolate the node at least in some of the rounds, rendering the trivial algorithm wrong.

We formalize this intuition by proving lower bounds using a new embedding technique, which might also be applicable in other settings of smoothing of graphs. The adversary focuses on a small set of $\hat{n} = \hat{n}(n,p)$ nodes and controls the induced subgraph on these nodes, as well as its connections with the rest of the graph. It embeds a worst-case lower bound graph on these \hat{n} nodes, such that the decision on the entire graph hinges on the \hat{n} -node subgraph.

An adaptive adversary can embed a worst-case construction for bipartite perfect matching on $\hat{n} = pn/20$ nodes: between every two designated queries, it changes the internal structure of the subgraph to be as in an \hat{n} -node worst case, and in parallel "fixes" any random edge change that touches these nodes. This results in Theorem 5.3, giving a polynomial lower bound on the update time of bipartite perfect matching.

Problem	Average	case	Oblivious flip a.	Oblivious AR a. Adaptive adv.		Worst case model			
Small subgraph of	Small subgraph counting								
s-t 3-paths	O(1)	[21]	$\Omega(pn^{1-\epsilon}), O(pn)$			T.6.11	, L.4.8	$\Omega(n^{1-\epsilon}), O(n)$	[20], [18]
s-t 4-paths	O(1)	[21]	$\Omega(pn^{1-\epsilon}), O(pn)$			T.6.12	, L.4.9	$\Omega(n^{1-\epsilon}), O(n)$	[20], C.4.10
s triangles	O(1)	[21]	$\Omega(pn^{1-\epsilon}), O(pn)$	$P(pn^{1-\epsilon}), O(pn)$ T.6.12, L.4.11		$\Omega(n^{1-\epsilon}), O(n)$	[20]		
s 4-cycles	O(1)	[21]	$\Omega(pn^{1-\epsilon}), O(pn)$			T.6.12,	L.4.12	$\Omega(n^{1-\epsilon}), O(n)$	[21], C.4.13
Other key problems									
Connectivity	O(1)		L.4.6	_		$\Omega(\log pn)$	T.5.6	$\Omega(\log n), \tilde{O}(\log n)$	n) [34], [22]
Perfect matching	O(1)		L.4.6	$\Omega(n^{\delta/3-\epsilon})$ T	`.5.8	$\Omega(p^{2-\epsilon}n^{1-\epsilon})$	T.5.3	$\Omega(n^{1-\epsilon}), O(n^{1.49})$	⁹⁵) [20], [38]

Table 4: Update times $t_{\rm u}$ of different problems assuming $t_{\rm q}=O(1)$ and $t_{\rm pre}=o(n^{3-\epsilon})$. The complexity of deciding connectivity with an oblivious add/remove adversary (marked —) remains open. The lower bound for perfect matching with an oblivious add/remove adversary is for a constant $0 < \delta < 1$ and $1-n^{-\delta} \le p \le 1$. All the polynomial lower bounds are conditioned on the OMv conjecture.

Surprisingly, a similar strategy works for the oblivious add/remove adversary, even though it is not aware of the random changes. This requires larger values of p and assures a control of a smaller subgraph, yet Theorem 5.8 provides a polynomial lower bound on the update time for this model.

Theorem B (informal). For any $p \in [0,1]$, there is no algorithm for **bipartite perfect matching** under an adaptive adversary with $t_{\rm u}(n) = O(p^{2-\epsilon}n^{1-\epsilon})$ and $t_{\rm q}(n) = O((pn)^{2-\epsilon})$ for any $\epsilon > 0$, unless the OMv conjecture fails.

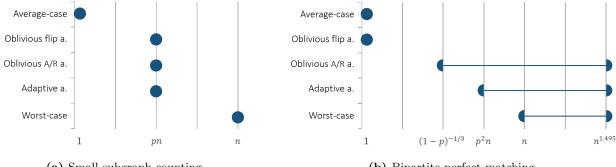
If $p \in [1 - n^{-\delta}, 1]$ for a constant $0 < \delta < 1$, then there is no algorithm for **bipartite perfect** matching under an oblivious add/remove adversary with $t_u(n) = O(n^{\frac{\delta}{3} - \epsilon})$ and $t_q(n) = O(n^{\frac{2\delta}{3} - \epsilon})$ for any $\epsilon > 0$ unless the OMv conjecture fails.

By choosing the best δ for a given value of $p \in [0,1]$, Theorem B implies that there is no algorithm under the oblivious add/remove model with running times $t_{\rm u}(n) = O(\min\left\{\frac{1}{1-p},n\right\}^{1/3-\epsilon})$ and $t_{\rm q}(n) = O(\min\left\{\frac{1}{1-p},n\right\}^{2/3-\epsilon})$.

Finally, we remark that both theorems also apply for **bipartite maximum matching** and **bipartite minimum vertex cover**, and Theorem B also applies for **counting** s k-cycles for $k \geq 3$ **odd** (and hence s-triangle detection), **counting** s-t 3-paths and **counting** s-t 4-paths; the latter two are subsumed by Theorem D below.

1.3.2 Counting small subgraphs

Four subgraph counting problems were shown to have constant average-case update time [21], but $\Omega(n^{1-\epsilon})$ worst-case update time. These problems are counting s-t 3-paths, s-t 4-paths, cycles of length 3 through a fixed node s (s triangles), and cycles of length 4 through a fixed node s (s 4-cycles). We establish that their p-smoothed update time is essentially identical across all three smoothed models (Fig. 5a). Specifically, in lemmas 4.8, 4.9, 4.11 and 4.12 we present simple algorithms (a la [21]) that achieve O(pn) expected update time even with an adaptive adversary (the strongest adversary). Roughly speaking, a change to an edge that touch s (or t) is "expensive"—it affect many short paths starting from s (or t) and thus results in an O(n) update time; other changes take only O(1) time to process. Expensive changes occur at random only 1/n of the time, but an adversary (of any type) can perform only them, resulting in probability p + 1/n for an O(n)-update time change, and an O(pn+1) expected update time overall (which translates to the same amortized update time).



(a) Small subgraph counting

(b) Bipartite perfect matching

Figure 5: The asymptotic update time of two problems under different input models, exemplifying the separations depicted in Fig. 3. Separations between smoothed and non-smoothed models arise from Fig. 5a using, e.g., $p = 1/\sqrt{n}$. Fig. 5b separates the different smoothed adversaries: adaptive adversary can force higher update time for any $p = \omega(1/\sqrt{n})$, but the two oblivious adversaries are only separated for high smoothing parameter $(p = 1 - 1/\operatorname{poly}(n))$.

Theorem C (informal). There is an algorithm for counting s-t 3-paths, s-t 4-paths, s triangles and s 4-cycles, with $t_u = O(pn+1)$ (expected) and $t_q = O(1)$ under any p-smoothing model.

Substituting p=1 in the theorem gives a worst-case algorithm with O(n) update time. This improves upon the state of the art for s-t 4-paths [8] and s 4-cycles [18], (corollaries 4.10 and 4.13).

In theorems 6.11 and 6.12 we complement these upper bounds by almost-matching conditional lower bounds on the update time. Specifically, we show an $\Omega(p \cdot n^{1-\epsilon})$ lower bound on the update time, for any $\epsilon > 0$, that applies even to an oblivious flip adversary (the weakest one). This is achieved by a series of reductions, starting from the OMv conjecture. This part is the most technically involved in this work, and we discuss it further in Section 1.4.

Theorem D (informal). Any algorithm for the problems from Theorem C must satisfy

$$t_{\text{pre}}(n) + \frac{n}{p} \cdot t_{\text{u}}(n) + n \cdot t_{\text{q}}(n) + \frac{n^2}{p} = \tilde{\Omega}(n^{3-\epsilon}),$$

for any $\epsilon > 0$, under any p-smoothing model, unless the OMv conjecture fails. Specifically, no algorithm can simultaneously have $t_{pre} = O(n^{3-\epsilon})$, $t_u = O(pn^{1-\epsilon})$, and $t_q = O(n^{2-\epsilon})$.

This completes the picture for s-t k-paths and the transition between k=2 and k=5 (see Table 2), showing how the complexity escalates with both the path length and the degree of adversarial control (parameter p).

1.3.3 Hierarchy between smoothing models

While all our models coincide with the worst-case model at p=1 and an average-case input at p=0, in Section 3 we show a hierarchy among the smoothing models for different regimes of 0 . The adaptive adversary can be defined with either operation type, and can disregard theadditional information it is privy to, making it at least as strong as both its oblivious counterparts. The oblivious add/remove adversary can simulate the oblivious flip adversary (while using p'= $p/(2-p) \in [p/2, p]$, as stated in Claim 3.2). This induces a (non-strict) hierarchy between the models (Fig. 3). We next discuss separation results for these models.

For some problems, such as counting small subgraphs, the three smoothing models coincide with one another (Fig. 5a): they all admit an O(pn) algorithm (Theorem C) and a conditional $\Omega(pn^{1-\epsilon})$ lower bound (Theorem D). These bounds separate the smoothing models from the worst case for $0 \le p < n^{-\delta}$ and from the average case for $n^{-\delta} , for any constant <math>\delta > 0$.

For other problems the three smoothing models demonstrate separations, i.e., different complexities for the same problem. This is the case with perfect bipartite matching (Fig. 5b): under an oblivious flip adversary with $p \in [0, 1 - \frac{26 \log n}{n})$, the problem admits a trivial "yes" algorithm since the graph contains enough random edges at all times (Theorem A). In contrast, an adaptive adversary can embed a worst-case lower bound construction within a small set of nodes, leading to a polynomial lower bound of $\Omega(p^{2-\epsilon}n^{1-\epsilon})$ for the problem (Theorem B). This separates the oblivious flip adversary from the adaptive one for almost any value of p, e.g., for $p \in \left[\frac{\log n}{\sqrt{n}}, 1 - \frac{26 \log n}{n}\right)$.

The more surprising separation is between the oblivious flip adversary and the oblivious add/remove one, as operation types have no effect on the complexity in all other regimes (worst-case, averagecase, and adaptive smoothed model). Using our embedding technique, Theorem B gives a superconstant lower bound on the update time of bipartite perfect matching when $p \approx 1 - o(1)$. This separates the two oblivious models, e.g., when $1 - p \in \left(\frac{26 \log n}{n}, \frac{1}{\log n}\right]$. Finally, we conjecture that the oblivious add/remove adversary is strictly weaker than the

adaptive adversary, but this is left unresolved for the moment.

A technical overview of the subgraph counting lower bounds

The most technically involved proof is the lower bounds for counting small subgraphs, even for the oblivious flip adversary, which we overview in this section. The general outline is based on that of lower bounds for counting larger subgraphs in the average case [21], consisting of a chain of finegrained reductions. Evidently, these tools alone do not suffice for the small subgraphs we consider here, as these are easy to count in the average case. Our hardness results utilize an adversarial strategy, and in its core lies a reduction that generates a sequence of changes resembling certain psmoothed sequences of changes, pinpointing the dependency on p. To quantify this resemblance we use the standard notion of statistical distance between distributions (or random variables). In our arguments we repeatedly use the *Poissonization* technique, described below, to break dependencies between key random variables. This technique is well-suited for our smoothed models, and we believe it should prove useful for other smoothed models of computation as well.

Poisson distribution. We recall the Poisson distribution $Pois(\lambda)$ is supported on $\mathbb{N}_{>0}$ and parameterized by its mean value λ (see [32, Section 8.4]). In a nutshell, Pois(λ) is the number of successful coin tosses when tossing infinitely many times a coin with infinitesimally small success probability, such that the expected number of successful attempts is λ . Formally, this is the limit for $n \to \infty$ of the binomial distribution $B(n, \lambda/n)$. Useful properties of this distribution are summarized in Fact 2 in Appendix A.

The Poissonization technique. A particularly useful property of the Poisson distribution is known as *Poissonization* (Item 3 of Fact 2), and we next describe it in our setting. Consider a distribution $\mathcal D$ over a set E of edges. A multi-set of edges sampled from $\mathcal D$ can be described by its histogram $\mathcal{H}: E \to \mathbb{N}_{>0}$, where $\mathcal{H}(e_i)$ represents the number of times the edge $e_i \in E$ is sampled. When drawing r such samples, the histogram values $\mathcal{H}(e_1), \mathcal{H}(e_2), \ldots$ exhibit a slight dependency among them, merely due to the fact that their sum is fixed to r. This dependency poses complications in probabilistic and information-theoretic analysis, especially if additional constraints on the histogram values exist. To address this, we independently sample the value $\mathcal{H}(e_i)$ for each edge from $Pois(r \cdot \mathcal{D}(e_i))$, where $\mathcal{D}(e_i)$ represents the probability of e_i in \mathcal{D} . This yields a total of t = Pois(r) samples, which might not be precisely r samples, but remains well-concentrated around r and even allows sampling conditioned on individual histogram values having e.g., a desired parity.

We apply Poissonization in several ways in our proofs, most prominently in Step 2 below. This helps to simplify and clear up some hardness results of [21] and correct inaccuracies. More crucially, Poissonization helps us overcome the additional challenges that arise due to the adversarial presence.

Sections 6.1, 6.2 and 6.3 together prove a lower bound for counting s-t 3-paths, which is extended to other problems in Section 6.4. The bounds are conditioned on the OMv conjecture [20], which formalizes the common belief that a truly sub-cubic combinatorial algorithm for matrix multiplication does not exist.

Step 1: massaging the OMv conjecture

The first step reduces between online algebraic matrix-vector multiplication problems. It consists of an algebraic reduction that was presented in [20] and is standard by now, and two reductions that were presented in [21]. The latter are reviewed in Appendix B.

Definition 1 (OMv problem). The OMv problem is the following online problem. Fix an integer n. The initial inputs are a Boolean n-vector v_0 and a Boolean $n \times n$ matrix M. Then, Boolean vectors v_i of dimension n arrive online, for i = 1, ..., n. An algorithm solves the OMv problem correctly if it outputs the Boolean vector Mv_i before the arrival of v_{i+1} , for i = 0, 1, ..., n-1, and outputs Mv_n .

Conjecture 1 (OMv conjecture [20, Conjecture 1.1]). For any constant $\epsilon > 0$, there is no $O(n^{3-\epsilon})$ -time algorithm that solves OMv with error probability at most 1/3 in the word-RAM model with $O(\log n)$ -bit words.

After a standard reduction from the OMv problem to the OuMv problem, two other reductions are made. The first reduces to OuMv with operations modulo 2 (that is, over \mathbb{F}_2 instead of \mathbb{R}), as defined below. The second reduction is a typical worst-case to average-case reduction over \mathbb{F}_2 , showing the problem is as hard on random inputs. We define the OuMv problem and state the reduction to conclude this step.

Definition 2 (parity OuMv problem). The parity OuMv problem is the following online problem. Fix an integer n. The initial inputs are two Boolean n-vectors u_0, v_0 , and a Boolean $n \times n$ matrix M. Then, pairs of Boolean vectors (u_i, v_i) of dimension n arrive online, for $i = 1, \ldots, n$. An algorithm solves the parity OuMv problem correctly if it outputs the Boolean value $u_i^T M v_i$ (over \mathbb{F}_2) before the arrival of (u_{i+1}, v_{i+1}) , for $i = 0, 1, \ldots, n-1$, and outputs $u_n^T M v_n$.

An algorithm solves the average-case parity OuMv problem if it answers correctly, with error probability at most 1/20, over inputs that are chosen independent and uniformly at random (that is, all entries of the matrix M and the vectors u_i, v_i are i.i.d. random bits).

The sequence of reductions concludes in the following lemma.

Lemma 1.1 ([21, Section 2]). For any constant $\epsilon > 0$, there is no $O(n^{3-\epsilon})$ -time algorithm that solves the average-case parity OuMv problem, unless the OMv conjecture fails.

Remark. At this point, [21, Section 2.1] reduces the former problem to a newly defined algebraic online problem, the biased average-case OuMv problem. Unfortunately, this leads to some inconsistencies in their proof (e.g., the statements regarding this online problem use the notions of update and query times, which are irrelevant to online problems). To circumvent this, our argument diverges from theirs, directly reducing the parity OuMv problem to a dynamic graph problem.

Step 2: from OuMv to counting s-t 3-paths in restricted graphs

In Section 6.1, we show that average-case parity OuMv can be solved using an algorithm that counts s-t 3-paths in P_3 -partite p-smoothed dynamic graphs with an oblivious flip adversary.

A P_3 -partite graph is composed of n' = 2n + 2 nodes partitioned as $V = \{s\} \sqcup A \sqcup B \sqcup \{t\}$, where |A| = |B| = n, and can only have edges of types sA, AB, Bt (that is, $E \subseteq (\{s\} \times A) \cup (A \times B) \cup (B \times \{t\})$). To represent the OuMv problem using this graph, consider u, v and M as the indicators of the sA, Bt and AB edges, respectively (e.g., $(s, A_j) \in E \iff u(j) = 1$). It is not hard to see that the product $u^T M v$ is exactly the number of s-t 3-paths.

Assume we have a P_3 -partite graph that represents M and (u_{i-1}, v_{i-1}) as above, and we get the next pair (u_i, v_i) and want to update the graph accordingly. In worst-case analysis, such reduction can use adversarial choices: flip the sA and Bt edges to represent (u_i, v_i) , leave the AB edges intact, and then query the number of s-t 3-paths to retrieve $u_i^T M v_i$. When random changes are involved in the process, however, undesired changes to AB edges are bound to occur. Moreover, such changes are more likely than others, as there are n^2 potential edges of types AB and only 2n potential edges of the other types (sA and Bt). Next, we describe how to overcome this challenge.

For the mere problem of having any changes in AB edges, a standard solution used in [21] serves here as well: the reduction creates 3 copies of the graph, such that whenever a query is made, each has different AB edges but they have exactly the same sA and Bt edges. To be precise, the AB edges in the 3 graphs will represent 3 matrices M_1, M_2, M_3 such that $M \equiv_2 M_1 + M_2 + M_3$, while the sA and Bt edges in all the graphs represent u_i and v_i . This assures that the sum modulo 2 of the number of s-t 3-paths in the three graphs gives $u_i^T M v_i$ with the original matrix M:

$$u_i^T M v_i \equiv_2 u_i^T M_1 v_i + u_i^T M_2 v_i + u_i^T M_3 v_i.$$
 (1)

The harder challenge is to synthesize 3 such correlated change sequences that resemble authentic p-smoothed change sequences. To this end, consider an *oblivious* adversary that chooses an sA or Bt edge u.a.r. After p-smoothing, each change has the following distribution over the graph edges

$$\mathcal{D}^p_{\mathrm{adv}}(e) := \begin{cases} \frac{p}{2n} + \frac{1-p}{n(n+2)}, & \text{if } e \text{ is of type } sA \text{ or } Bt \\ \frac{1-p}{n(n+2)}, & \text{if } e \text{ is of type } AB \end{cases}.$$

This guarantees the highest chances for a change of type sA and Bt after smoothing, which is crucial for avoiding redundancies in the reduction and obtaining a tight lower bound. Moreover, since all edge changes are drawn from $\mathcal{D}_{\text{adv}}^p$, we can simulate a sequence by generating a multi-set of edges and later randomize the order in which they are changed. We create a sequence of roughly $t = 5n \log(n)/p$ changes using Poissonization. This breaks dependencies and allows us to construct a multi-set of samples by drawing the number of appearances of each edge independently.

Edges of types sA and Bt. Recall that changes to sA and Bt are shared by all 3 copies; hence, we create one multi-set of such changes, S_0 . Let $u_{\text{dif}} = u_i - u_{i-1}$ and $v_{\text{dif}} = v_i - v_{i-1}$ be the difference vectors, and denote by z_e the number of times the sA edge $e = (s, a_j)$ is added to S_0 . Our goal is to ensure that for each such sA edge, $z_e \equiv_2 u_{\text{dif}}(j)$ (and similarly for Bt edges and v_{dif}). Naively using rejection sampling (resampling the multi-set S_0 until all conditions are met) would take $2^{\Omega(n)}$ attempts. Instead, we use Poissonization: for each sA edge e, draw a Poisson number of samples z_e independently, conditioned on the desired parity, i.e., $z_e \sim \text{Pois}(\mathcal{D}_{\text{adv}}^p(e) \cdot t)|_{z_e \equiv_2 u_{\text{dif}}(e)}$. Rejection sampling of each z_e value separately only requires $\Theta(n)$ attempts overall.

Edges of type AB. Drawing z_e for each AB edge e in a similar way would require $\Omega(n^2)$ time, which is too costly for this fine-grained reduction. However, only $O(t) = o(n^2)$ changes of type AB are expected, so we instead draw the total number of AB changes from a Poisson distribution with parameter $t_{AB} = (1 - p)nt/(n + 2)$, and then assign an AB edge u.a.r. for each of these changes. In actuality, we wish to create 3 correlated copies, where AB edges cancel out. To this end, we draw three multi-sets S_1, S_2, S_3 , each composed of $Pois(t_{AB}/2)$ edges of type AB, where each edge is chosen u.a.r. Each copy uses exactly two of the sets S_1, S_2, S_3 , such that each set is used twice. This ensures that (1) holds.

Adding S_0 to the multi-set of each copy and shuffling each of them separately produces 3 sequences of edge changes that are statistically close to a sequence of Pois(t) edge changes drawn from \mathcal{D}_{adv}^p . Bounding the statistical distance limits the additional error from the imperfect simulation, guaranteeing the error probability on the output $u_i^T M v_i$ is only slightly higher than the error probability of 3 executions of the s-t 3-paths counting algorithm on genuine p-smoothed inputs.

Remark. The simulation above is perhaps the crux of the reduction, generating inputs for the dynamic graph algorithm from inputs to the OuMv problem. We note that the simulation used in [21] was much simpler and easily shown to be efficient, but its proof of correctness was incomplete. One place that is lacking is a false proof for Claim C.2. While this is fixable for their specific case, it stems in an inescapable subtlety of handling dependencies. Our use of Poissonization in both the simulation and the analysis tackles these subtleties and formally prove our claims, as well as theirs.

Step 3: from restricted graphs to general graphs

We continue in Section 6.2, by showing how to count s-t 3-paths in the restricted class of P_3 -partite graphs using an algorithm that counts such paths in general graphs. This reduction uses inclusion-edgeclusion: it takes a P_3 -partite graph \mathcal{G} , and constructs 16 (general) graphs by adding edges that were previously not allowed (exterior edges). The same approach was taken in the average-case analysis, but smoothed inputs require more attention due to the adversarial presence.

To account for the adversarial presence, the transition from a p-smoothed P_3 -partite graph to p'-smoothed graphs must use $p' \leq p$ that sufficiently weakens the adversary. Still, this can be done using $p' \in [p/2, p]$, which is crucial for a lossless reduction leading to (nearly) tight lower bounds.

The 16 new dynamic graphs all use the original node set $V = \{s\} \sqcup A \sqcup B \sqcup \{t\}$, and initially all have the same interior edges as the P_3 -partite graph. Their initial exterior edges are correlated in the following sense: they properly partition the four exterior edge sets E_{At} , E_{AA} , E_{BB} , E_{sB} . That is, there exists four partitions, one for each edge set, such that the exterior edges of each of the 16 initial graphs constitute a unique combination of the parts, one from each edge set.

We next describe how to construct a single change, using a parameter $\alpha(n,p) \approx \frac{1}{2-p} \in [1/2,1]$. We flip a coin $C \sim Bin(\alpha)$; if C=1 we make an interior change, the next change in the p-smoothed sequence of \mathcal{G} , while if C=0 we change a random exterior edge. Choosing α carefully and $p'=\alpha \cdot p$ guarantees that a sequence of such changes is p'-smoothed on a general graph, for an appropriate adversarial strategy. We apply this sequence of changes to each of the 16 graphs.

Since the graphs undergo the exact same changes, they properly partition the four exterior edge sets at all times. This invariant ensures that the total number of exterior s-t 3-paths (i.e., not of type sABt) in all 16 graphs is always easy to compute. After querying all graphs for their s-t 3-paths count, we sum the answers, subtract the exterior paths, and divide by 16. This gives the number of sABt paths in \mathcal{G} , since each interior path appears in all graph and was counted 16 times.

To summarize, steps 1-3 dictate a chain of fine-grained reductions as follows. An algorithm for counting s-t 3-paths in general graphs can be used to count them in a P3-partite graph, which in

turn can be used to solve the average-case parity OuMv problem. A solution for the average-case parity OuMv problem implies a solution to the OMv problem, contradicting the OMv conjecture.

1.5 Related work

1.5.1 Dynamic graph algorithms

Dynamic graph algorithms constitute a deep and well-studied research topic, as was recently presented in the thorough survey of Hanauer, Henzinger and Schulz [19]. The most recent dynamic algorithm for connectivity is by Huang, Huang, Kopelowitz, Pettie and Thorup [22], and has $t_{\rm u}=O(\log n\log^2\log n)$ amortized update time and $t_{\rm q}=O(\log n/\log\log\log n)$ time per query. Small subgraph counting was recently studied by Hanauer, Henzinger and Hua [18], who showed, e.g., that counting s-t 3-paths and s-triangles (triangles through a given node s) can be done with $t_{\rm u}=O(n)$ and constant query time (see also Section 5.1 below). Kara, Nikolic, Ngo, Olteanu and Zhang [24] studied databases that support enumeration of triangles in trinary relations, and their work implies triangle counting in dynamic graphs with $t_{\rm pre}=O(m_0^{3/2}), t_{\rm u}=O(m^{1/2})$, and $t_{\rm q}=O(1)$.

Pătrașcu and Demaine [34] proved the first unconditional lower bound for dynamic graph algorithms, focusing on connectivity problems in undirected graphs. One of their results asserts that there is no dynamic algorithm for deciding whether the graph is connected with a constant query time and $t_{\rm u} = o(\log n)$ amortized time (see Lemma 5.5), a results that remains the best known to date. Follow-up works on their work [5, 30, 36], considered, e.g., the case of parameterized queries (querying whether two given nodes are in the same connected component) and directed graphs.

Following a line of work on conditional lower bounds in dynamic graphs [1,11,27,35,37], which used several different conjectures, Henzinger, Krinninger, Nanongkai and Saranurak [20] introduced the OMv conjecture as a unified conjecture. They showed, e.g., that s-triangle detection, bipartite matchings and bipartite vertex cover cannot be done with polynomial preprocessing time, $t_{\rm u} = O(n^{1-\epsilon})$ and $t_{\rm q} = O(n^{2-\epsilon})$, for any $\epsilon > 0$ and even amortized and randomized, unless the OMv conjecture is false. The main focus of the works until this point was on worst-case analysis, and average-case analysis was rarely applied, as described next.

1.5.2 Beyond worst-case analysis of dynamic graph algorithms

The first to study average-case complexity of dynamic graph algorithms were Nikoletseas, Reif, Spirakis and Yung [33]. They consider an initial empty graph, a phase of addition of random edges, and then a phase where random edges are added or remove with equal probabilities. In this setting, they present an algorithm for (u, v)-connectivity, i.e., where a query includes a pair (u, v) of nodes and has to determine if they are in the same connected component.

Following this, Alberts and Henzinger [2] defined the "restricted randomness" model. To this end, they note that each change consists of a type (add or remove) and a parameter (which edge to add or to remove). In their model, an adversary chooses a sequence of types (add or remove) for the changes, and then the parameter (edge) of each change is chosen uniformly at random from the relevant set (existing edges for a remove operation, node-pairs without edges for an add operation). They show that several problems have better amortized time complexities in the restricted randomness model compared to the best worst-case bounds known at the time (almost no lower bounds where known at the time). The problems considered include connectivity and its variants (edge and node connectivity), maximum cardinality matching, minimum spanning forest, and bipartiteness.

Indeed, one could see this model as an average-case model, but we prefer to use this term only for uniformly random inputs (as in [21, 33]). It can also be seen as a smoothed analysis model,

though without the ability to parameterize the amount of randomness.

As this model consists in an adversary and a random process, it should not come as a surprise that both an oblivious adversary and an adaptive one can be considered in this setting. Here, an adversary that chooses the sequence of types in advance is called an oblivious one, and an adversary that can choose each type according to the outcome of the random process so far is an adaptive one. These variants are not equivalent: for example, an adaptive adversary can make sure that all the n-edge graphs in the sequence are connected (by adding an n-th edge to an (n-1)-edge graph only if it is connected), while with an oblivious adversary every n-edge graph has equal probability, and most of them are not connected. The distinction between adversaries is not made in [2], but the proofs use the fact that each graph has equal probability (conditioned on the number of edges), thus the model there is oblivious.

The average-case complexity of dynamic graph algorithms was disregarded for decades, until recently Henzinger, Lincoln and Saha [21] approached it again with new tools. They studied subgraph counting problems, presented some simple algorithms inspired by [2], and focused mainly on proving conditional lower bounds. Their proofs use ideas from fine-grained complexity [20], and specifically lower bounds for average-case complexity in this setting [4,7]. For some problems, such as counting s-t 5-paths, they show that the average-case complexity cannot be better than the worst-case one (up to an n^{ϵ} factor), while for others, such as s-t 3-paths, they prove a big gap exists. As smoothed analysis lie between the worst case and the average one, we focus the current work only on problems of the last type, where a gap exists. Some of our lower bound proofs use techniques from [21]; for example, while they do not give a lower bound for s-t 3-paths (since it is easy in the average case) we give a lower bound for this problem in the smoothed case by extending the technique they used for proving a lower bound for s-t 5-paths.

1.5.3 Smoothed analysis in different models

Smoothed analysis was introduced by Spielman and Teng [40] in relation to using pivoting rules in the simplex algorithm. Since then, it have received much attention in sequential algorithm design. In particular interest to us are smoothed analysis results for static graphs [14, 28, 29] and dynamic networks [9, 10, 15, 31], which we use as inspiration for our model of noise.

In recent years, smoothed analysis was also applied in settings of a repeated game, where inputs are chosen successively. These include dynamic networks, population protocols [39] and online learning [16,17]. The effect of the adversary's adaptivity was first studied with regard to information spreading in dynamic networks [31], which is shown to be polynomially slower when the adversary is adaptive compared to an oblivious adversary. The adaptivity question was also studied in the context of online learning [17], and a handful of problems were shown to have the same smoothed complexity whether the adversary is adaptive or oblivious. From these, the dynamic networks works are the most relevant to ours, as they defined similar smoothing models. Their results, however, focus on round complexity and not computation time, and thus are incomparable with ours.

2 Preliminaries

We use [n] for the set $\{1, 2, ..., n\}$, and \triangle to denote the symmetric difference between two sets. T denotes the number of edge changes in the dynamic graph.

We extend the definition of a dynamic graph to the case where some updates are chosen at random while others are adversarial. While we explore variants on this model, the basic definition is for the *adaptive flip adversary*:

Definition 3 (p-smoothed dynamic graph, adaptive). A p-smoothed dynamic graph with an adaptive adversary is a dynamic graph $\mathcal{G} = (G_0, \dots, G_T)$ created by the following random process.

- Initially, H_0 is proposed by an adversary. For each node-pair $e \in \binom{V}{2}$, with probability p it is included in G_0 according to H_0 , and with probability 1-p it is re-sampled uniformly, i.e., included in G_0 as an edge with probability 1/2.
- At step $i \in [T]$, a node-pair $e \in {V \choose 2}$ is proposed by the adversary. With probability p this e remains unchanged, and with probability 1-p it is discarded and a new e is chosen uniformly at random from ${V \choose 2}$. The new graph G_i has edge set $E_i = E_{i-1} \triangle \{e\}$.

We also consider *oblivious* adversaries, that fix all their choices ahead of time, but then in execution each choice is replaced by a random choice with probability 1-p. In this model we allow for either a flip adversary or an add/remove adversary:

Definition 4 (p-smoothed dynamic graph, oblivious). A p-smoothed dynamic graph with an oblivious adversary is a dynamic graph $\mathcal{G} = (G_0, \dots, G_T)$ created by the following random process.

- An adversary proposes an initial graph H_0 and a sequence of T changes, where each change consists of a node-pair $e_i \in \binom{V}{2}$. An add/remove adversary also chooses an action a_i : add or remove.
- For each node-pair $e \in \binom{V}{2}$, with probability p it is included in G_0 according to H_0 , and with probability 1-p it is re-sampled uniformly, i.e., included in G_0 as an edge with probability 1/2.
- At step $i \in [T]$, randomly choose $c_i \sim \text{Ber}(p)$ and act accordingly:
 - if $c_i = 1$: use the edge e_i . For a flip adversary, flip it, and the new edge set is $E_i = E_{i-1} \triangle \{e_i\}$. For an add/remove adversary use action a_i and accordingly the resulting edge set is either $E_i = E_{i-1} \setminus \{e_i\}$ or $E_i = E_{i-1} \cup \{e_i\}$.
 - if $c_i = 0$: an edge e is chosen uniformly at random from $\binom{V}{2}$, and the new graph G_i has the edge set $E_i = E_{i-1} \triangle \{e\}$.

Note that a graph created by this process can also be thought of as a graph created step by step as in the adaptive adversary case, but by an adversary (flip or add/remove) that does not see any of the random choices in the initial graph and previous steps.

The initial graph. Smoothing the initial graph is crucial for the model to extrapolate between worst-case and average-case inputs, but it seem to make little difference in all our results. Our algorithms sometime assume an adversarial initial graph (that is, their analysis does not use the smoothing of the initial graph). Our lower bounds rely on a uniformly random initial graph G_0 , using the following simple observation:

Observation 1. The adversarial strategy that proposes an initial graph H_0 uniformly at random results in a smoothed initial graph G_0 which is uniformly random as well. This strategy can be employed by any type of adversary and any parameter $0 \le p \le 1$.

Restricted dynamic graphs. In Section 6, we consider a variant where the graph edges are restricted to a set $R_H \subseteq {V \choose 2}$ of potential edges. In this model *all changes*, both random and adversarial, as well as the initial graph G_0 , are on R_H instead of the entire set ${V \choose 2}$.

We particularly focus on (H,Π) -partite graphs, defined as follows.

Definition 5 $((H,\Pi)$ -partite graph). Fix a simple graph H on node set [k], and consider a graph G on node set [n]. We say that G is an (H,Π) -partite graph with a mapping $\Pi:[n] \to [k]$ if any edge in G is mapped to an edge in H, i.e. if $(u,v) \in E_G$ then $(\Pi(u),\Pi(v)) \in E_H$.

If all the graphs G of a dynamic graph \mathcal{G} on [n] are (H,Π) -partite, we say \mathcal{G} is (H,Π) -partite. In this case we use R_H to denote the set of edges that are allowed in \mathcal{G} , i.e.

$$R_H := \{(u, v) \mid (\Pi(u), \Pi(v)) \in E_H\}.$$

For example, if H is composed of two nodes and a single edge and G is a (H,Π) -partite graph for some Π , then G is bipartite. In some cases we describe G on a node set with k parts, calling it H-partite (where H has k nodes). In these cases Π and H are implicit for ease of presentation. For example, in Section 6 we discuss P_3 -partite graphs where G uses node set $V = \{s\} \coprod A \coprod B \coprod \{t\}$. It is then clear from context that $\Pi(A)$ is fixed ($\Pi(B)$ as well), and H is a 3-path (P_3) graph: $\Pi(s) - \Pi(A) - \Pi(B) - \Pi(t)$.

For a partition of the node set of \mathcal{G} , we naturally categorize edges and longer paths using the different parts. For example, in a general graph on 4 sets of nodes there could be many edge types, but if the graph is P_3 -partite (with the mapping above) then only edges of types sA, AB, Bt exist and 3-paths from s to t are all of type sABt.

Our proofs use some probabilistic tools, detailed in Appendix A and shortly described next. We make an extensive use of the *Poisson distribution* and more importantly the *Poissonization* technique (see, e.g., [32, Section 8.4]). We use Pois(t) for the Poisson distribution with parameter t and sometimes for a random variable drawn from this distribution. Some useful properties of the Poisson distribution are laid out in Fact 2. For the distance between two distribution P, Q over the same domain we use the well-known *statistical distance* (sometimes called total-variation distance), denoted by SD(P, Q). Some useful properties of the statistical distance can be found in Fact 3.

3 The relative power of the adversaries

Adaptive vs. oblivious adversaries. While we assume an adversary is aware of the problem in hand, the smoothing parameter p and other parameters of the problem, awareness to the actual noisy changes occurred is a completely different matter (as is the case in other multi-round smoothing models). We show that for some problems, an adaptive adversary can cause higher running times compared to the oblivious adversaries, separating the models. Perhaps surprisingly, we show that for counting small subgraphs this is not the case, and adaptivity has little to no affect on the complexity of the problem.

Flip vs. add/remove operations

A priori, it is unclear whether an algorithm that handles edge-flip operations can deal with add/remove operations, or vice versa. This is typically a non-issue, as the vast majority of research in dynamic graph algorithm sticks to worst-case analysis, where the adversary has full knowledge of the (in)existence of edges and picks additions or removals accordingly. Slightly more subtle is the case of average-case analysis, where the location of each change is chosen at random. The recent work defining average-case analysis of dynamic graphs [21] assumed an edge chosen at random is always flipped, i.e., an average-case flip model. Let us define the average-case add/remove model as one where in each round, both an edge and an operation type are chosen at random (this may sound similar to [2] where the type was chosen by an adversary and the edge at random, but the models are actually very different). Unsurprisingly, we show below that the average-case model with flip and with add/remove operations are equivalent, up to a constant factor.

Claim 3.1 (Equivalence between flips and add/remove models in average-case analysis). An average-case adversary with flip operations has the same power as an average-case adversary with add/remove operations. That is, they have the same success probability up to $1/n^c$ additive error, and the same amortized times up to constant factors, over any sequence of $\Omega(\log n)$ changes.

Proof. The key for the proof is translating add/remove changes to flip changes with laziness.

For the sake of the proof, consider a third average-case model, *lazy flip*. In each step of this model no change occurs with probability 1/2, call this 'null', and otherwise a uniformly random edge is flipped.

This lazy flip model is entirely equivalent to the average-case add/remove model³. Indeed, first choose the edge e at random. In the former model we choose whether to flip it or not, while in the latter we choose whether to try adding or try removing it. In both models the edge is consequently flipped with probability 1/2.

We next show equivalence between the lazy and non-lazy flip models. Any algorithm for the non-lazy flip model with amortized time $t_{\rm u}$ can be run on the lazy model, doing nothing when the step is 'null', and the amortized update time can only decrease (and does, roughly by a factor of 2). Formally, any sequence of s steps consists of $s' \leq s$ flips. The total computation is at most $t_{\rm u} \cdot s' \leq t_{\rm u} \cdot s$, or at most $t_{\rm u}$ amortized. For correctness, note that the distribution of each of the s' changes is uniform over the edges, just as in the non-lazy model, leading to the exact same error probability.

For the other side, consider an algorithm \mathcal{A} with update procedure $u(\cdot)$ for the lazy model. This algorithm might use the 'null' steps to perform computation. We can instead use a new algorithm \mathcal{A}' with the following update procedure $u'(\cdot)$, where a variable *counter* was initialized to 0 in pre-processing:

- If the step is 'null', increase *counter* by 1.
- If the step is an edge e, run u(null) for counter times, then u(e), and set counter = 0.

Evidently, \mathcal{A} and \mathcal{A}' do the exact same computations over any lazy input sequence.

Next, consider an algorithm \mathcal{B} for the non-lazy flip model. The algorithm simulates \mathcal{A}' , with update operation $u_{\mathcal{B}}(e)$:

- Toss random fair coins until the first heads appears, use t for the number of tosses.
- Run u'(e) with couner = t 1.

Amortized update time. Consider an execution of \mathcal{B} on a sequence of $s = \Omega(\log n)$ edge updates, and denote by T the total amount of coin tosses made by \mathcal{B} in $u_{\mathcal{B}}(e)$ operations. The random variable T is simply the amount of fair coin tosses required for s heads. By a standard Chernoff bound, since $s = \Omega(\log n)$, then 4s fair tosses have at least s heads with probability at least 1 - 1/n, which gives a lower bound on the probability that $T \leq 4s$. Note that each execution of $u'(\cdot)$ with counter = t - 1 executes $u(\cdot)$ for t times. Therefore, \mathcal{B} calls $u(\cdot)$ exactly T times over s updates, and its amortized time is of at most $(t_{\mathbf{u}} \cdot T)/s$, and w.h.p. $4t_{\mathbf{u}}$.

Correctness. Note that each of the T executions of $u(\cdot)$ gets as input 'null' with probability 1/2 and a change with probability 1/2, so the guaranteed error is kept, but for the added 1/n factor for the event $\{T > 4s\}$.

³Note the equivalence can only hold for an add/remove model with equal probabilities. Otherwise, the graph tends to a different density, while a lazy flip tends to 1/2 with any laziness factor.

Operation type in smoothed models.

In both worst-case and average-case analysis the model can be equivalently defined with flip operations or add/remove operations. Perhaps surprisingly, it turns out this is not quite the case for the smoothed model. First, we note that an adaptive adversary, which is fully aware of the current graph, can use either operations type (same as the worst-case adversary). This leaves us with a single adaptive smoothed model, where the adversary is at least as strong as the oblivious one (no matter the operation type used).

For oblivious adversaries, which choose their entire input ahead of the execution, we show below that the add/remove adversary is not weaker than the flip adversary (i.e., the complexity of any problem in the presence of an add/remove adversary is at least as high as with a flip adversary). We later show the other direction is false. The intuition here is that an oblivious flip adversary quickly loses track of the graph, whereas an oblivious add/remove adversary can control a small part of it by "insisting" on some edges begin added or removed. Note that, by definition, whenever randomness controls a change, it does not draw an action with equal probability but instead always flips this edge. This is justified by Claim 3.1 above: smoothing the operation type as well as the choice of the edge would have simply resulted in a lazy model that chooses a null operation with probability $p/2 \le 1/2$, affecting the amortized times by a factor of at most 2.

We now focus on p-smoothed models with oblivious adversaries. A strategy similar to the proof of Claim 3.1 is applied here: we define a proxy lazy model, with laziness only over adversarial flip changes, and show it is equivalent (up to constants) to the non-lazy flip model. However, in this setting an add/remove adversary is not forced to choose the operation at random (although it can), making this adversarial model potentially stronger. Our results from Section 4.1 and Section 5 show that this is indeed the case, separating the models.

Our first result shows that any lower bound shown using a flip oblivious adversary can be translated to a similar lower bound (up to constant factors) using an add/remove oblivious adversary.

Claim 3.2 (Add/remove is stronger than flip for oblivious adversary in a p-smoothed model). If there exists an algorithm \mathcal{A} that solves a problem \mathcal{P} with error δ over a p-smoothed sequence of ssteps with an oblivious add/remove adversary, then there exists an algorithm \mathcal{B} that solves \mathcal{P} with error probability at most $\delta + 1/n$ over a p'-smoothed sequence of $\Theta(s)$ steps with an oblivious flip adversary, for $p' = p/(2-p) \in [p/2, p]$, and provided that $s = \Omega(\log n)$.

Proof. As before, we use a proxy model we call the oblivious *lazy*-flip p-smoothed model: every step in the sequence is 'null' with probability p/2, flips an adversarially chosen edge with probability p/2, and flips a uniformly random edge with probability 1-p. This is equivalent to giving the adversary probability p, but then with probability 1/2 nullifying the step it chose.

On the one hand, an oblivious add/remove adversary can simulate any oblivious lazy-flip adversary. This is done by copying the strategy to choose e, but then choosing to add or remove it with probability 1/2. Doing so, each step consists of a random flip with probability 1-p, or the adversarial edge e with probability p. If an adversarial edge is chosen, it is only flipped with probability 1/2. This is the exact same distribution as the proxy lazy model above.

On the other hand, as before, the laziness can be dealt with, as formalized next. Assume an algorithm \mathcal{A} with update procedure $u(\cdot)$ for the p-smoothed model with an oblivious AR adversary. In particular the same algorithms works in the lazy p-smoothed model with an oblivious flip adversary. As before, we define \mathcal{A}' with update procedure $u'(\cdot)$ that uses a counter of null steps, and execute $u(\cdot)$ multiple time upon the next actual change.

Consider an algorithm \mathcal{B} for the non-lazy p'-smoothed model with flip adversary for p' that we set later. The update procedure of \mathcal{B} does the following on edge e:

- Toss random coins with bias 1 p/2 towards heads, until the first heads appears. Call the number of coins used t.
- Run u'(e) with plugged in value counter = t 1.

Amortized update time. Note that the probability for head is $1 - p/2 \ge 1/2$, and so for a sequence of length $s = \Omega(\log n)$ changes, the total amount of coin tosses T is at most 4s with probability at least 1 - 1/n. Hence, the total time for updates is at most $t_{\rm u} \cdot T$, and the amortized update time is bounded by $t_{\rm u} \cdot (T/s)$, and w.h.p. by $4t_{\rm u}$.

Correctness. In the sequence of T calls to $u(\cdot)$, each change is null with probability p/2, an adversarial flip with probability p'(1-p/2), and a random flip otherwise. Choosing p'=p/(2-p) (note that $p' \in [p/2, p]$), the probability for the adversarial flip is p/2, meaning \mathcal{A} only errs with probability δ . Adding the error for the event $\{T > 4s\}$, we get overall error $\delta + 1/n$.

4 Upper bounds

In this section we present dynamic algorithms for the problems studied in this work. We start with proving that some problems become trivial in the presence of input randomness, then present algorithms for counting short paths, and finally use the latter algorithms to also count cycles.

We use s and t to denote two different fixed nodes (i.e., that are hard-coded in the algorithms), and u and v for arbitrary nodes, or nodes that can come as inputs. All the paths and cycles we consider are simple. We use $m_0 = |E(G_0)|$.

4.1 Easy problems with an oblivious flip adversary

This section deals with properties that hold w.h.p. for random graphs, e.g., connectivity. In essence, we show the randomness in the model guarantees that such properties hold w.h.p. at all times, which allows for a trivial algorithm that outputs "yes". While this is intuitive for p = 0 (i.e., the average-case model of [21]), we extend this intuition to almost any value of $p \in [0, 1)$.

Our first claim is applicable to monotone graph properties, and relies solely on the random changes occurred in the initial graph.

Claim 4.1. Fix a monotonely increasing graph property \mathcal{P} and real numbers $\alpha \in (0,1]$ and $q \in [0,1/2]$, such that on an Erdős-Rényi graph $G \sim G_{n,q}$,

$$\Pr_{G \sim G_{n,q}}[G \text{ does not satisfy } \mathcal{P}] \leq \alpha.$$

Then for every smoothing parameter $p \le 1 - 2q$ and every round $r \ge 0$ in a p-smoothed dynamic graph in the oblivious flip model, it holds that

$$\Pr[G_r \text{ does not satisfy } \mathcal{P}] \leq \alpha.$$

Proof. We denote by C_i the random string used for the smoothing process at round i, and by C_0 for the random string used to smooth the initial graph. Initially, the adversary proposes H_0 , and the smoothing process chooses each node-pair w.p. 1-p and then flips it w.p. 1/2 (equivalent to re-sampling w.p. 1/2), independently of other edges. That is, the randomness C_0 samples a graph $F_0 \sim G_{n,(1-p)/2}$ of flips, and changes the initial graph to $G_0 = H_0 \oplus F_0$.

Next, we fix the (oblivious flip) adversarial strategy. The smoothed sequence of changes $e_1, \ldots e_r$ applied on an empty graph results a (random) graph that only depends on C_1, \ldots, C_r . Denoting this graph by G', we have that

$$G_r = G_0 \oplus G' = F_0 \oplus (H_0 \oplus G').$$

The crucial point is that the adversarial strategy is chosen in advance, and therefore it does not depend on F_0 . Furthermore, the random strings C_1, \ldots, C_r are independent of C_0 and therefore of F_0 . This means the distribution of G_r can be described as follows: first draw C_1, \ldots, C_r which combined with the adversarial strategy gives $H_0 \oplus G'$, and then XOR this graph with $F_0 \sim G_{n,(1-p)/2}$.

For any outcome of C_1, \ldots, C_r , the graph $H_0 \oplus G'$ is fixed and G_r then depends only on F_0 : every node-pair e is an edge in G_r independently of the others, w.p. (1-p)/2 if $e \notin H_0 \oplus G'$ and w.p. (1+p)/2 if $e \in H_0 \oplus G'$. Using the inequalities $(1+p)/2 \ge (1-p)/2 \ge q$ and the monotonicity of \mathcal{P} , we have the following:

$$\Pr_{C_0,\dots,C_r}[G_r \text{ does not satisfy } \mathcal{P}] = \underset{C_1,\dots,C_r}{\mathbb{E}} \left[\Pr_{C_0}[G_r \text{ does not satisfy } \mathcal{P}] \right] \\
\leq \underset{C_1,\dots,C_r}{\mathbb{E}} \left[\Pr_{C_0}[F_0 \text{ does not satisfy } \mathcal{P}] \right] \\
= \underset{C_0}{\Pr}[F_0 \in \mathcal{P}] \\
\leq \underset{C_0}{\Pr}[G \text{ does not satisfy } \mathcal{P}] \\
\leq \alpha.$$

We can leverage the claim to get the following result.

Claim 4.2. Fix C > 2. For any parameter $p \in [0, 1 - \frac{2C \log n}{n})$, the problems of connectivity, bipartite perfect matching, bipartite maximum matching and bipartite minimum vertex cover admit constant update and query time algorithms with an oblivious flip adversary, with probability of at least 1 - 1/n to succeed through all T rounds, provided that $T \le n^{\frac{C-3}{2}}$ rounds.

Proof. The premise of Claim 4.1 holds for connectivity with $q = \frac{C \log n}{n}$ and $\alpha = 2n^{-(C-1)/2}$, for any C > 2. A similar result holds for bipartite perfect matching with $q = \frac{C \log n}{n}$ and $\alpha = n^{-(C-1)}$, for any C > 2 (see [23, Remark 4.3]).

for any C>2 (see [23, Remark 4.3]). Thus, for any $p\in[0,1-\frac{2C\log n}{n})$, a p-smoothed dynamic graph is disconnected at round r for any $r\in[n^{(C-3)/2}]$ with probability at most $2n^{-(C-1)/2}$ (and similarly for a bipartite graph having a perfect matching at all times). This allows for a trivial constant-time algorithm that does nothing during updates and outputs "yes" upon any query.

Using a union bound over $T \leq n^{\frac{C-3}{2}}$ rounds, the algorithm succeeds in *all* rounds with probability at least 1 - 1/(2n).

For maximum matching, the existence of a perfect matching immediately implies a matching on all nodes, i.e., of size n (for a bipartite graph with n nodes in each side). Hence, an algorithm for maximum matching can answer n to all queries.

Similarly, picking all the nodes of one side gives a vertex cover of size n, while any cover must touch all the edges of the perfect matching described above and thus must have size at least n. An algorithm for minimum vertex cover can thus answer n to all queries as well.

Before we turn to another argument, we digest the meaning of the claim above in the context of comparison between models.

Oblivious add/remove adversary. The proof of Claim 4.1 would indeed fail if the adversary uses add/remove operations. This is for a good reason: in Theorem 5.8 we show that such an adversary can disrupt any fast algorithm with. e.g., $p = 1 - 1/\sqrt{n}$. This in fact separates the two models.

Average-case model. Looking at Claim 4.1, one might wonder if the p-smoothed oblivious flip adversary can pose any difficulty or is it simply equivalent to not having no adversary at all (the average-case, p=0). In Section 6, we show the two models are separated for several subgraph counting problems. We prove a polynomial lower bound for the update time in the p-smoothed oblivious flip model whenever $p=\Omega(n^c)$ (for fixed c>0), which stands in contrast to the upper bound of O(1) for the average case shown by [21].

The argument in Claim 4.1 relies only on the randomness injected to the initial graph, and for p close to 1 it can only be useful for a polynomial number of rounds (hence the restriction on T). This restriction, however, can be lifted. Indeed, more randomness is being injected to the model with time, and we next show that once poly(n)/(1-p) rounds have passed, each graph exhibits a "typical" behavior of a uniformly random graph. This is true even though each such graph might be statistically far from an actual uniformly random graph (to see this, consider the number of edges mod 2 as the distinguisher).

Claim 4.3. Fix a graph property \mathcal{P} and a constant $\alpha > 0$ such that on a uniformly random graph $G \sim G_{n,\frac{1}{\alpha}}$,

$$\Pr[G \text{ does not satisfy } \mathcal{P}] \leq \alpha.$$

Then for every smoothing parameter p < 1 and $r_p := n \cdot \binom{n}{2}/(1-p)$, every round $r \ge r_p$ in a p-smoothed dynamic graph in the oblivious flip model satisfies

$$\Pr[G_r \text{ does not satisfy } \mathcal{P}] \leq 4r \cdot \left(\alpha + \frac{n^2}{e^{2n}}\right).$$

A key element in the proof is a stochastic graph G_{ℓ} . First, we pick $\ell = \operatorname{Pois}(r)$, and consider the graph after ℓ rounds, G_{ℓ} (which is a random variable depending both on the smoothing process and the choice of ℓ). On the one hand, if $r \geq r_p$ then G_{ℓ} has negligible statistical distance from a uniformly random graph $(G_{n,\frac{1}{2}})$. On the other hand, $\ell = r$ with probability at least $1/\operatorname{poly}(r)$, so events with an exponentially negligible probability in G_{ℓ} must have an exponentially negligible probability in G_r as well. Combining both parts shows that any event with a negligible probability in a uniformly random graph (e.g., being disconnected), also have a negligible probability in G_r .

Proof. Fix any sequence of changes (adversarial strategy) e_1, \ldots, e_t and a round r, and focus on the first $\ell = \text{Pois}(r)$ steps. We shall analyze the distribution of the stochastic graph G_{ℓ} , where randomness comes from the choice of ℓ and the smoothing process.

Our first goal is to show that G_{ℓ} is distributed almost uniformly, that is,

$$SD\left(G_{\ell}, G_{n, \frac{1}{2}}\right) \le \frac{n^2}{e^{2n}}.\tag{2}$$

Using the Poissonization technique, we construct G_{ℓ} by independently drawing the number of random changes chosen by nature, $\operatorname{Pois}((1-p)\cdot r)$, and the number of adversarial changes, $\operatorname{Pois}(p\cdot r)$. Next, we draw a random permutation σ on their order, choosing which rounds use random changes and which follow the adversarial choice. Finally, each random change draws a uniformly random edge to flip.

As the adversarial strategy is fixed, any permutation σ entirely fixes the set of adversarial changes made throughout the entire process. We first focus on the initial graph G_0 and apply only these adversarial flips, obtaining a stochastic graph $G'_{\ell} = (V, E'_{\ell})$ whose randomness only comes from G_0 and σ .

Next we focus on the edges chosen by the randomness. Each edge is drawn with probability $1/\binom{n}{2}$, and so we can use Poissonization for the set of $\operatorname{Pois}((1-p)\cdot r)$ random changes. Indeed, this draw is equivalent to drawing independently the number of randomness-chosen flips for each edge $z_e \sim \operatorname{Pois}((1-p)\cdot r/\binom{n}{2})$. If we start from an empty graph and make these changes, we end up with a stochastic graph $G''_{\ell} = (V, E''_{\ell})$ that represents all random flips made throughout the process.

The order in which edges are flipped does not affect the resulting graph $G_{\ell} = (v, E_{\ell})$. Thus, starting from the initial graph G_0 and applying both adversarial and random flips, we can write

$$E_{\ell} = E_{\ell}^{\prime} \triangle E_{\ell}^{\prime\prime},$$

where both sides are random variables according to the smoothing process and the choice of ℓ .

We are now ready to analyze G_{ℓ} via G''_{ℓ} . Choose $r_p = n\binom{n}{2}/(1-p)$, and consider any round $r \geq r_p$. The process that produces G''_{ℓ} flips each potential edge in the graph $z_e \sim \operatorname{Pois}(\lambda_r)$ times, where $\lambda_r = (1-p) \cdot r/\binom{n}{2}$. For any round $r \geq r_p$ we have $\lambda_r \geq \lambda_{r_p} = n$. By Item 2 in Fact 2, we know the parity of z_e (and the existence or inexistence of the edge e) is almost uniform:

$$\Pr[z_e \equiv_2 0] = \frac{1 + e^{-2\lambda_r}}{2} \in \left[\frac{1}{2}, \frac{1 + e^{-2n}}{2}\right]$$

Denoting the parity of z_e by $\mathbb{1}_e$, we get an indicator for the existence edge e in G''_{ℓ} . We also use U_1 for a uniformly random bit, thus the previous inequality can be written in terms of statistical distance:

$$SD(\mathbb{1}_e, U_1) \le e^{-2n}.$$

Due to Poissonization, all z_e are independent from one another, and so are the indicators $\mathbb{1}_e$, implying the distribution of G''_{ℓ} is a product of all $\mathbb{1}_e$. Similarly, a uniformly random graph $G_{n,\frac{1}{2}}$ is a product of U_1 for each potential edge. By sub-additivity of statistical distance for product measures (Item 1 in Fact 3), we get:

$$\operatorname{SD}\left(G_{\ell}'', G_{n, \frac{1}{2}}\right) \le \sum_{e} \operatorname{SD}(\mathbb{1}_{e}, U_{1}) \le \frac{n^{2}}{e^{2n}}.$$

Lastly, recall that G_{ℓ} is made by flipping the edges of G'_{ℓ} in G''_{ℓ} . That is, any fixed permutation σ sets the adversarial flips and in turn sets a bijection between G_{ℓ} and G''_{ℓ} . This bijection only only re-orders the probability vector, attaching each probability to a new graph instead of the old one (in a bijective manner). For this reason, it does not affect the statistical distance from a uniformly random graph (where all probabilities are the same). Finally, averaging over all permutations σ we obtain

$$SD\left(G_{\ell}, G_{n, \frac{1}{2}}\right) \le \frac{n^2}{e^{2n}},$$

which proves (2).

Since the two graphs are statistically close, Item 2 in Fact 3 implies

$$\Pr[G_{\ell} \text{ does not satisfy } \mathcal{P}] \le \Pr[G_{n,\frac{1}{2}} \text{ does not satisfy } \mathcal{P}] + \operatorname{SD}(G_{\ell}, G_{n,\frac{1}{2}}) \le \alpha + \frac{n^2}{e^{2n}}.$$
 (3)

Finally, to connect G_{ℓ} and G_r , recall that ℓ is distributed according to Pois(r), and so it has a rather high probability of choosing exactly round r:

$$\Pr[\text{Pois}(r) = r] = \frac{r^r e^{-r}}{r!} \ge \frac{1}{2\sqrt{\pi r}} \ge \frac{1}{4r},$$

with the first inequality deriving from Stirling's approximation.

This means that G_{ℓ} is somewhat likely to choose the graph G_r . By the total law of probability:

$$\Pr[G_{\ell} \text{ does not satisfy } \mathcal{P}] = \sum_{i \in \mathbb{N}} \Pr[\ell = i] \cdot \Pr[G_i \text{ does not satisfy } \mathcal{P}]$$

$$\geq \Pr[\ell = r] \cdot \Pr[G_r \text{ does not satisfy } \mathcal{P}]$$

$$\geq \frac{1}{4r} \cdot \Pr[G_r \text{ does not satisfy } \mathcal{P}].$$

Combining this inequality with Eq. (3), the proof is concluded.

Using the above claim we can guarantee a p-smoothed dynamic graph stays connected w.h.p. even over a near-exponential number of rounds. As mentioned above, $G_{n,q}$ satisfies some nice properties w.h.p. even for q = o(1) (see [23]). For a uniformly random graph (q = 1/2) we quantify the negligible probability that these properties don't hold, following similar arguments as in [12,13].

Claim 4.4. There exists N_0 such that the following hold for any $n \geq N_0$:

- 1. The probability of an Erdős-Rényi graph over n nodes w.p. 1/2 for each edge $(G_{n,\frac{1}{2}})$ to be disconnected is at most $\frac{2n}{2^{n/2}}$.
- 2. The probability of a random bipartite graph with n nodes on each side and each edge w.p. 1/2 to have no perfect matching is at most $\frac{4(n+1)^2}{2(n+1)/2}$.

Proof. Any disconnected graph must have a connected component of size $k \leq n/2$ nodes. The probability of a set of k nodes to indeed be disconnected from the rest of the graph $G_{n,\frac{1}{2}}$ is exactly $2^{-k(n-k)} \leq 2^{-\frac{nk}{2}}$.

We can union bound over all sets of size k and over all sizes k. The probability of a graph being disconnected is thus at most

$$\sum_{k=1}^{\lceil n/2 \rceil} \binom{n}{k} \cdot \left(\frac{1}{2}\right)^{nk/2} \le \sum_{k=1}^{\lceil n/2 \rceil} n^k \cdot \left(\frac{1}{2^{n/2}}\right)^k = \sum_{k=1}^{\lceil n/2 \rceil} \left(\frac{n}{2^{n/2}}\right)^k \le \frac{2n}{2^{n/2}},$$

where the last inequality is since the sum of an infinite geometric series is at most twice its first element (assuming $n/2^{n/2} \le 1/2$ which holds for $n \ge 8$).

For the second bullet, consider a graph with n nodes on each side, call the sides A, B. Any graph with no perfect matching must violate Hall's condition, meaning there exists $S \subseteq A$ of size |S| = k with neighbor set $|\Gamma(S)| < k$. Differently put, there exists $S \subseteq A$, |S| = k, and a set $T \subseteq B$, |T| = n - k + 1 such that all edges between S and T are missing. We union bound again over all possible pairs S, T with sizes k, n - k + 1, and over all values of k to find an upper bound for the

probability that Hall's condition is violated:

$$\sum_{k=1}^{n} \binom{n}{k} \binom{n}{n-k+1} \left(\frac{1}{2}\right)^{k(n-k+1)} \le \sum_{k=1}^{n+1} \binom{n+1}{k} \binom{n+1}{n-k+1} \left(\frac{1}{2}\right)^{k(n-k+1)}$$

$$\le 2 \sum_{k=1}^{\lceil (n+1)/2 \rceil} \binom{n+1}{k} \binom{n+1}{n-k+1} \left(\frac{1}{2}\right)^{k(n-k+1)}$$

$$\le 2 \sum_{k=1}^{\lceil (n+1)/2 \rceil} \binom{n+1}{k}^2 \left(\frac{1}{2}\right)^{k(n+1)/2}$$

$$\le 2 \sum_{k=1}^{\lceil (n+1)/2 \rceil} \left(\frac{(n+1)^2}{2^{(n+1)/2}}\right)^k$$

$$\le \frac{4(n+1)^2}{2^{(n+1)/2}}.$$

In the second inequality we used symmetry of the addends (potentially adding twice the addend for k=(n+1)/2 when n is even). The two last inequalities are similar to those in the previous bullet, using $(n+1)^2/2^{(n+1)/2} \le 1/2$ for $n \ge 20$. This proves the claim with $N_0 = 20$.

While Claim 4.2 is limited to $p \in [0, 1 - \frac{2C \log n}{n})$, Claim 4.3 allows us to handle the regime $1 - p = o(\log n/n)$, where very few perturbations are added to the initial graph, leaving it close to the worst-case. The trick is to execute a known algorithm for the worst-case during the first $\approx n^3/(1-p)$ rounds, and only then rely on the randomness to kick in. Over a long enough sequence, the amortized running times will become similar to those of the average-case.

Lemma 4.5. Fix $p \in [0, 1-n^{-c}]$ for some c > 0. For the following problems, there exists an algorithm with all amortized running times O(1) on any sequence of polynomial length $\Omega(n^6/(1-p))$ which succeeds w.h.p. on p-smoothed dynamic graphs with an oblivious flip adversary: **bipartite** perfect matching, bipartite maximum matching and bipartite minimum vertex cover. The same holds for connectivity with a sequence of polynomial length at least $\tilde{\Omega}(n^3/(1-p))$.

Proof. For all problems but connectivity, until round $r_p = n \binom{n}{2}/(1-p)$, we run a known algorithm that works in the worst-case. All problems admit such algorithms with $O(n^3)$ amortized running time (by solving each round separately). For connectivity, we use the algorithm of [22] that has $\tilde{O}(\log n)$ update time and $\tilde{O}(\log n)$ query time.

For connectivity and bipartite perfect matching, starting round r_p the algorithm simply answers "yes" on all queries.

Combining Claim 4.3 with Claim 4.4, we know that for all steps $r_p \leq r \leq T$ we have for connectivity

$$\Pr[G_r \text{ is not connected}] \le 4r \left(\frac{2n}{2^{n/2}} + \frac{n^2}{e^{2n}}\right) \le \frac{2T \cdot n^2}{2^{n/2}}.$$

Similarly, for bipartite perfect matching:

$$\Pr[G_r \text{ does not have perfect a matching}] \leq 4r \left(\frac{4(n+1)^2}{2^{(n+1)/2}} + \frac{n^2}{e^{2n}}\right) \leq \frac{8T \cdot (n+1)^2}{2^{(n+1)/2}}.$$

That is, for both problems, the failure probability of the "yes" algorithm on each round is exponentially small. Taking a union bound over at most T rounds, and as since T = poly(n), we get that the "yes" algorithm succeeds on all rounds between r_p and T with probability 1 - o(1).

For the complexity, recall that $T \ge n^6/(1-p)$ which implies $T \ge n^3 \cdot r_p$. The total running time of the first r_p rounds is $n^3 \cdot r_p$, and the rest trivially answers "yes", taking $O(T-r_p)$ total computation. Over all T rounds, this amortizes to O(1), meaning constant amortized update and query times.

As in the proof of Claim 4.2, an algorithm for maximum matching can answer n on all queries after round r_p , and algorithm for minimum vertex cover can answer n on all queries.

Finally, for a slightly more modest range for p we improve upon Claim 4.2:

Lemma 4.6. For any $p \in [0, 1 - \frac{26 \log n}{n})$, there are algorithms for connectivity, bipartite perfect matching, bipartite maximum matching and bipartite minimum vertex cover with constant update and query times with an oblivious adversary, with success probability at least 1 - 1/n through all T rounds, provided that $T \le 2^{n/3}$ rounds.

Proof. For connectivity we prove that the "yes" algorithm is correct on all T rounds with probability at least 1-1/n. The argument for bipartite perfect matching is almost identical, and the extension to maximum matching and minimum vertex cover is as in the proof of Claim 4.2.

Using Claim 4.1 with $q = \frac{13 \log n}{n}$ and $\alpha = 2n^{-6}$ for all rounds $r = 0, \ldots, n^4$ the graph G_r is disconnected with probability at most $2n^{-6}$, and by a union bound over all rounds the algorithm accumulate an error probability of at most 1/(2n). We next apply Claim 4.3, noting that $1-p \geq 1/n$ and therefore $r_p \leq n^4$. We thus apply the claim for all rounds $r = n^4 + 1, \ldots, T$, combined with Claim 4.4. This shows that each graph G_r in these rounds is disconnected with probability at most $poly(n)/2^{n/2} \leq 1/(2n2^{n/3})$, and by a union bound over all rounds, the algorithm accumulates an error probability of at most 1/(2n). Thus, the algorithm errs on any round between 1 and T with probability at most 1/n.

4.2 Counting small subgraphs with any adversary

4.2.1 Counting short s-t paths

In this section we show that it is easy to count s-t paths of length 1, 2, 3 or 4 in smoothed dynamic graphs, with any adversary type. Our algorithms in this section and the following one are similar to the average-case algorithms of [21], with a faster preprocessing procedure, as well as careful treatment of non-simple paths (such as s-t 4-paths of the form (s, u, v, s, t)) that seem to have been previously overlooked. Roughly speaking, most algorithms have "cheap" and "expensive" changes, where the latter rarely happen at random but might be consistently chosen by an adversary. We use an amortization argument to bound the running times in the presence of an adversary, pinpointing the dependency on p.

The update time of our algorithms is constant for any p = O(1/n), and then gradually increases with p, reaching O(n) when p is an arbitrarily small constant. In particular, this coincides with previous bounds for the worst case (p = 1) and the average case (p = 0). For the average case, the initialization time is reduced to $O(m_0)$, where m_0 is the number of edges in the initial graph, compared to previous algorithms [21] that have initialization times of $O(n^2)$ for s-t 3-paths and s -triangles, and $O(n^{\omega})$ for s-t 4-paths, and s 4-cycles.

Counting s-t paths of length 1 or 2 is trivial even in the worst case [21], and hence also in smoothed and random dynamic graphs. In order to count s-t 3-paths, we use an auxiliary dynamic algorithm that counts s-u 2-paths, for a fixed node s and a node u given as an input in each query.

Lemma 4.7. There is an algorithm for p-smoothed dynamic graphs with any adversary type that has fixed nodes s and t, receives a node u in each query, and has $O(m_0)$ preprocessing time, O(pn + 1)

expected update time, and can answer a query with the number of s-u 2-paths excluding the edge (s,t) in O(1) time. In addition, the algorithm maintains a counter c_{s2u} of such s-u 2-paths for each node u in the graph. The algorithm can also be used without specifying a node t, in which case no edge is excluded.

The expected running can be thought of as the amortized running time, as for a reasonable amount of changes the total update time would concentrate around its expectation.

Proof. The preprocessing is a simple BFS from s up to depth 2, ignoring the edge (s, t), in $O(m_0)$ time.

Upon an insertion or a removal of an edge (v, u) (where $u, v \neq s$), if $v \neq t$ then the algorithm checks the existence of the edge (s, v) and update c_{s2u} accordingly (increments c_{s2u} if (v, u) was added and (s, v) exists, decrements c_{s2u} if (v, u) was removed and (s, v) exists, and ignores the change otherwise). If $u \neq t$, the algorithm similarly checks the existence of (s, u) and updates c_{s2v} .

Upon an update of an (s, v) edge, $v \neq t$, the algorithm checks for all nodes $u \neq s$ if the edge (v, u) exists and update c_{s2u} accordingly, which takes O(n) time. A random update will update an (s, v) edge with probability 1/n; an adversarial change, on the other hand, might perform such an update at each time, and an adversarial change happens w.p. p.

Upon a query with a node u, the algorithm naturally returns c_{s2u} . Note that $c_{s2s} = 0$ at all times.

To conclude, for each update w.p. 1-p it takes O(1) time, and w.p. p it takes O(n), for a total of O(pn+1) expected time.

With this algorithm in hand, counting s-t 3-paths is an easy task.

Lemma 4.8. There is an algorithm for p-smoothed dynamic graphs with any adversary type that counts the number of s-t 3-paths with preprocessing time $O(m_0)$, update time O(pn+1) in expectation, and query time O(1).

Note that we only consider simple paths, so the edge (s,t) can never participate in an s-t 3-path.

Proof. The algorithm uses the algorithm for maintaining the counter c_{s2u} of s-u 2-paths from Lemma 4.7. In addition, it maintains a counter c of s-t 3-paths, initialized to 0.

Upon initialization, the algorithm initializes the algorithm from Lemma 4.7, and while doing so it lists all the nodes u with $c_{s2u} \neq 0$. Then, for each node u in the list, if the edge (u,t) exists, it increases c by c_{s2u} . Since each s-t 3-path must be of the form (s, v, u, t) with $u, v \neq s, t$, the counter c now contains exactly the number of s-t 3-paths.

Upon an update of an edge e not touching t, the algorithm updates the c_{s2u} counters, and every time it changes a counter c_{s2u} , it checks if the edge (u,t) exists. If so, it updates c by the change of c_{s2u} (which can be positive or negative). Upon an update of an edge (u,t), the algorithm updates c by c_{s2u} .

For a query, the algorithm naturally returns c.

Regarding the complexity, note that the preprocessing phase only adds O(1) operations for each node u that was updated by the preprocessing phase of the algorithm from Lemma 4.7, and thus they have the same asymptotic preprocessing time. Similarly, upon an update not involving t, the algorithm only performs O(1) additional operations for each operation of the algorithm from Lemma 4.7, and does only O(1) operations upon an update involving t, thus again the algorithms have the same asymptotic update time.

Finally, we present an algorithm that maintains the number of s-t 4-paths.

Lemma 4.9. There is an algorithm for p-smoothed dynamic graphs with any adversary type that counts the number of s-t 4-paths with preprocessing time $O(m_0)$, update time O(pn+1) in expectation, and query time O(1).

Proof. The algorithm uses the algorithm from Lemma 4.7 twice, in order to keep track of the number c_{s2u} of s-u 2-paths and the number c_{t2u} of t-u 2-paths (in both cases, excluding (s,t)). In addition, it maintains a counter c of s-t 4-paths, initialized to 0.

In the preprocessing, the algorithm first performs preprocessing of the two 2-path counters, and while doing so keeps a list of all nodes u with $c_{s2u} > 0$. It then goes over this list, and for each u with $c_{s2u} > 0$ it adds $c_{s2u}c_{t2u}$ to c, thus counting all s-t 4-paths but including degenerate paths of the form (s, v, u, v, t). Finally, it goes over all neighbors v of s, and for each such v that is also a neighbor of t, it subtracts $\deg(v) - 2$ from c. This is done in order to account for all (s, v, u, v, t) paths v is a part of. In total, the preprocessing takes no more than the $O(m_0)$ preprocessing time of the 2-path databases, and the last part also takes $O(\deg(s)) = O(m_0)$ time.

Upon an update of an edge (u,v) (with $\{u,v\} \cap \{s,t\} = \emptyset$) the algorithm preforms 4 types of updates, corresponding to 4 types of s-t 4-paths through (u,v), as follows. All the updates are done before updating the 2-path counters.

To account for paths of type (s, u, v, x, t), $x \notin \{s, t, u, v\}$, the algorithm first checks that the edge (s, u) exists. If so, if (u, v) is added, the algorithm adds c_{t2v} to c; if (u, v) is removed and the edge (v, t) does not exist, the algorithm subtracts c_{t2v} from c; finally, if (u, v) is removed and the edge (v, t) exists, the algorithm subtracts $c_{t2v} - 1$ from c. The reason for this subtle difference is that in the first two cases we do not have to account for a degenerate path of the form (s, u, v, u, t) since the 2-path (v, u, t) is not counted in c_{t2v} in these cases, while in the last case the degenerate 4-path (s, u, v, u, t) is not counted in c while the 2-path (v, u, t) is counted in c_{t2v} so we should not subtract 1 from c for this 2-path.

The other 3 types of updates account for paths of types (s, v, u, x, t), (s, x, u, v, t), and (s, x, v, u, t), and are completely analogous. All these updates take constant time.

Upon an update of an edge (s, u), the algorithm must go through all neighbors $v \neq s$ of u, and for each such neighbor update c by c_{t2v} if the edge (u, t) does not exist in the graph, and update it by $c_{t2v}-1$ otherwise. This counts all paths of the form (s, u, v, x, t) changed by the update of (s, u), where the last case makes sure degenerate paths are not counted. An update of an edge of the form (t, v) is analogous. In both cases, the 2-path counters are updated after c is update, and the time complexity of these updates is O(n).

Observe that both in the above proof and in the proof of Lemma 4.7, any update takes O(n) time even in the worst case; in fact, this can already be observed by considering the proof of [21] in detail. This results in a deterministic worst-case algorithm with O(n) (non-amortized) update time and constant query time. The state of the art for the problem is $O(n^2 \log^3 n)$ update time and $O(\#_4P + \log n)$ query time, where $\#_4P$ is the output value. This is achieved using a technique of [8] for maintaining a set of shortest paths between any pair of nodes.

Corollary 4.10. There is a deterministic worst-case algorithm that counts the number of s-t 4-paths with preprocessing time $O(m_0)$, update time O(n), and query time O(1).

4.2.2 Counting short cycles through a node

We now move from counting paths to counting cycles, and start by counting triangles (3-cycles) through a fixed node s.

Lemma 4.11. There is an algorithm for p-smoothed dynamic graphs with any adversary type that counts the number of triangles through s with preprocessing time $O(m_0)$, update time O(pn+1) in expectation, and query time O(1).

Proof. We use the algorithm from Lemma 4.7 in order to keep track of the number c_{s2u} of s-u 2-paths. In addition, it maintains a counter c of twice the number of triangles through s, initialized to 0.

The algorithm first initializes all the c_{s2u} counters and keeps a list of the nodes u with $c_{s2u} > 0$. It then goes over this list and for each such u, if u is a neighbor of s, it adds c_{s2u} to c.

Upon an update, the algorithm keeps track of all updates of c_{s2u} variables, and for each such update, if u is a neighbor of s, it updates c by the same amount.

Note that each triangle (s, u, v, s) through s is counted exactly twice, both in the initialization and during the algorithm: such a triangle is counted once as (s, u, v, s) when updating c_{s2v} , and once (s, v, u, s) when updating c_{s2u} . Accordingly, the algorithm answers each query with c/2. The overhead over each operation of the algorithm from Lemma 4.7 is constant, giving the desired preprocessing, update and query times.

Next, we give an algorithm for counting 4-cycles through a fixed node s. While it is tempting to use the s-t 4-path counting algorithm with s=t for this goal, as suggested in [21], the situation is a bit more subtle: the 4-path algorithm first counts also degenerate paths (e.g. of the form (s, u, v, u, t)) and then deduce them from the total count, and this deduction step assumes $s \neq t$. In addition, when s=t it will count each cycle twice, once at each direction, but this is easily fixable by dividing the output by 2. A simpler approach is maintaining two copies of s in the graph, denoted s and s', and counting s-s' 4-paths. We suggest yet another approach, that does not use the 4-path algorithm at all.

Lemma 4.12. There is an algorithm for p-smoothed dynamic graphs with any adversary type that counts the number of 4-cycles through a given node s with preprocessing time $O(m_0)$, update time O(pn+1) in expectation, and query time O(1).

Proof. The algorithm uses the algorithm from Lemma 4.7 in order to keep track of the number c_{s2u} of s-u 2-paths (without a node t and an excluded edge (s,t)). In addition, it maintains a counter c of 4-cycles through s, initialized to 0.

The algorithm first initializes all the c_{s2u} counters and keeps a list of the nodes u with $c_{s2u} > 0$. It then goes over this list and for each such u it adds $\binom{c_{s2u}}{2}$ to c.

Upon an update, the algorithm keeps track of all updates of c_{s2u} variables, and for each such update, is subtracts $\binom{c_{s2u}}{2}$ from c before the update, and adds $\binom{c_{s2u}}{2}$ to it after.

Since each 4-cycles through s has exactly one mid-point u, and the number of 4-cycles through s with a midpoint u is exactly $\binom{c_s 2u}{2}$, the algorithm is correct. The overhead over each operation of the algorithm from Lemma 4.7 is constant.

Finally, we observe that the above proof also implies a better worst-case algorithm than known, as in the case of s-t 4-paths. We get a deterministic worst-case algorithm with O(n) update time and constant query time, improving upon the state of the art $O(m^{2/3})$ update time [18] for any $m = \omega(n^{3/2})$.

Corollary 4.13. There is a deterministic worst-case algorithm that counts the number of 4-cycles through a given node s with preprocessing time $O(m_0)$, update time O(n), and query time O(1).

5 Lower bounds via embedding

In the case of an adaptive or oblivious add/remove adversary, we can achieve lower bounds for many problems by embedding a known worst-case lower bound graph as a subgraph of the entire smoothed graph. To this end, an adaptive adversary can choose $\hat{n} = \Theta(pn)$ nodes, keep them disconnected from the rest of the graph, and fully control the structure of their induced subgraph. An oblivious add/remove adversary can do the same, but on $\hat{n} = n^{\delta/3}$ nodes for a small constant δ .

5.1 Polynomial lower bounds

We start by noting that an adaptive adversary can "take control" of parts of the graph, i.e., choose a set \hat{V} of nodes, make sure it is disconnected from the rest of the graph, and set the edges inside \hat{V} as it desires. We prove a more general lemma: given a graph G, a set R of potential edges and a set $R' \subseteq R$, the adversary can reach a new graph G' satisfying $(G \triangle G') \cap R = R'$, i.e. a graph where the edges of R' are flipped, the edges of $R \setminus R'$ are as in the original graph, and the rest of the edges may change arbitrarily.

Lemma 5.1 (Embedding a graph). Consider an n-node graph G, a set $R \subseteq {V \choose 2}$ of r = |R| potential edges, and a subset $R' \subseteq R$ of size |R'| = r'. An adaptive adversary can reach a new graph G' satisfying $(G \triangle G') \cap R = R'$ within at most ℓ steps with probability at least $1 - 2\exp(-p\ell/40)$, provided that $\ell \ge \frac{10r'}{p}$ and $r \le \frac{pn^2}{18}$.

We later use this lemma both to control a subset of nodes (their internal edges and the edges connecting them to the rest of the graph), and to simulate a sequence of worst-case changes on these nodes.

The adversary puts all its efforts on changing a subset of the edges of R' and maintaining the edges of R by reverting any random changes to them. We analyze this as a 1-dimensional random walk process: the nature starts with a budget of at most r' (edges that the adversary wants to change), and then a random change might add 1 to the sum by flipping an edge of $R \setminus R'$ or an edge of R' that was already changed, while an adversarial change reduces it by 1 by reverting a change or fixing an edge of R'. If the probability of the adversary to take a step is larger the probability of a random change to hit R, then by gambler's ruin (see [32, Section 7.2.1]) the process will eventually converge to 0 (edges that the adversary wants to change), as desired.

Proof. We say that a change is *effective* if it changes an edge of R. The adversary will change an edge of R as long as there is a change to make, which happens w.p. p. The probability that a random change happens and hits R is $(1-p)\cdot r/\binom{n}{2}$. By our assumption, this is at most p/9. A sequence of ℓ steps is expected to have at least $p\ell$ effective steps (just by the adversarial turns), and by Chernoff, the probability of having less than $p\ell/2$ effective steps is at most $\exp(-p\ell/8)$.

Consider the first $\ell' = p\ell/2$ effective steps, and let f(i) be the number of edges that need to be flipped after i effective steps. Conditioned on such a step, the probability of an adversarial change is at least 9 times that of a random one, and therefore at least 9/10. Thus, if f(i-1) > 0 then with probability⁴ at least 9/10 we have f(i) = f(i-1) - 1, and otherwise f(i) = f(i-1) + 1. In particular $f(i-1) - f(i) \in \{-1, 1\}$.

Denote by E_t the event that $f(i) \neq 0$ for any $i \leq t$ (otherwise we are done within t effective steps). Define for each step $y_i := (f(i-1) - f(i) + 1)/2$, then $y_1, \dots t_{\ell'}$ are Bernoulli (assuming

 $^{^{4}}$ We need f(i-1) > 0 to allow the adversary to make a required change. For simplicity, the calculations are as if a random change is never in our favor.

 $E_{\ell'-1}$), each with expectation at least 9/10 and overall $\mathbb{E}[\sum y_i] \ge 9\ell'/10$. By telescopic sum we can write:

$$\Pr[f(\ell') > 0] = \Pr[f(0) - f(\ell') < r']$$

$$= \Pr\left[\sum_{i \in [\ell']} y_i < \frac{r' + \ell'}{2}\right]$$

$$\leq \Pr\left[\sum_{i \in [\ell']} y_i < (6/10)\ell'\right]$$

$$\leq \exp(-\ell'/20),$$

where the next-to-last inequality is due to the premise $r' \leq p\ell/10 = \ell'/5$, and the last by a Chernoff bound. Thus, if we weren't done within $\ell' - 1$ effective steps, we must be done within ℓ' with only this error.

Using union bound and plugging in $\ell' = p\ell/2$, the total error probability is at most $2\exp(-p\ell/40)$.

We state known lower bounds for the worst-case (non-smoothed) setting, with $\hat{t_u}$ and $\hat{t_q}$ update and query times.

Lemma 5.2. There is no dynamic algorithm for counting s k-cycles for $k \geq 3$ odd (and hence s-triangle detection), counting s 4-cycles, counting s-t 3-paths, counting s-t 4-paths, bipartite perfect matching, bipartite maximum matching and bipartite minimum vertex cover with error probability at most 1/3, polynomial preprocessing time, $\hat{t}_u(\hat{n}) = O(\hat{n}^{1-\epsilon})$ and $\hat{t}_q(\hat{n}) = O(\hat{n}^{2-\epsilon})$ (both amortized) for any $\epsilon > 0$ unless the OMv conjecture is false.

The proofs of all these claims use worst-case dynamic graphs of a similar structure: either one or \hat{n} phases, where each phase consists of \hat{n} updates to different edges, followed by a single query.

These claims and their proofs appear in: [18, Theorem 26] for s k-cycles; [21, Lemma 7.6] for s t-cycles; [21, Lemma 7.5] for t-cycles; and [20, Corollary 3.4 and footnote 9] for t-cycles t-cycles counting, bipartite matchings and bipartite vertex cover.

We next augment this lemma to get lower bounds for the p-smoothed setting with an adaptive adversary.

Theorem 5.3. Fix $0 \le p \le 1$. There is no dynamic algorithm for p-smoothed dynamic graphs with an adaptive adversary for counting s k-cycles for $k \ge 3$ odd (and hence s-triangle detection), counting s-t 3-paths, counting s-t 4-paths, bipartite perfect matching, bipartite maximum matching and bipartite minimum vertex cover with error probability at most 1/3, polynomial preprocessing time, $t_{\rm u}(n) = O(p^{2-\epsilon}n^{1-\epsilon})$ and $t_{\rm q}(n) = O((pn)^{2-\epsilon})$ (both amortized) for any $\epsilon > 0$, unless the OMV conjecture is false.

In the next section, we show that counting s-triangles, s 4-cycles, s-t 3-paths and s-t 4-paths need even higher running times (in terms of p), i.e. one cannot have $t_{\rm u}(n) = O(pn^{1-\epsilon})$ and $t_{\rm q}(n) = O(pn^{2-\epsilon})$. Moreover, these lower bounds hold for the weaker, oblivious adversaries.

Proof. When p = o(1/n), the assertion is trivial. Assume for contradiction that in the p-smoothed setting with adaptive adversary, one of the above problems has an algorithm with better running

⁵The statement of [21, Lemma 7.6] regards $t_q = O(1)$ only, but the proof clearly extends to a general t_q .

times than claimed on an n-node graph. For the bipartite problems, we will show the lower bound on a bipartite balanced graph. We devise a fast dynamic algorithm for the same problem in a worst-case dynamic graph \mathcal{H} on $\hat{n} = pn/20$ nodes by creating a p-smoothed n-node dynamic graph \mathcal{G} and running the claimed fast algorithm on it. The new graph \mathcal{G} is created by simulating a carefully chosen adversary that makes sure \mathcal{G} has \mathcal{H} as a disconnected subgraph; hence, the answers to the queries on \mathcal{G} and \mathcal{H} are correlated. This allows us to solve the problem on \mathcal{H} with running times better than stated in Lemma 5.2, a contradiction.

Consider an instance of the problem on a fully-dynamic non-smoothed graph \mathcal{H} on $\hat{n} = pn/20$ nodes, with an initial graph H_0 . Recall that the worst-case dynamic graph is composed of phases, where each phase consists of \hat{n} updates followed by a query. We simulate the first phase, where H_1 is the final graph on which the query is made; the rest pf the phases, if exist, are similar.

Choose a random n-node graph G_0 where each edge exists w.p. 1/2, which is a p-smoothed initial graph by Observation 1. Note that there is no need to initialize the graph, and instead, we decide on the existence of each edge at the first time it is read or changed. Chooses an arbitrary set \hat{V} of pn/20 nodes, which includes the designated nodes (s and sometimes t) if those exist in the problem, and is a balanced bipartite graph if the problem is on bipartite graphs. Let $R = \hat{V} \times V$ be the set of all potential edges inside \hat{V} and connecting \hat{V} to the rest of the graph. Set r = |R|, so $\frac{p}{21}n^2 \le r \le \frac{p}{20}n^2$ for large enough n. The set R is fixed for the rest of this proof.

By Lemma 5.1 with $R' \subseteq R$ and $r' \le r$, the adversary can make sure the subgraph on \hat{V} is isomorphic to the initial lower bound graph H_0 , and is disconnected from the rest of the graph, in $\ell = 40 \frac{r}{p} \log n$ steps w.h.p. Start the simulation by these ℓ steps: chooses each step to be adversarial w.p. p and otherwise random, choose random changes uniformly at random, and the adversarial changes as in the lemma.

The *n*-node graph (of \mathcal{G}) now has a subgraph on \hat{V} isomorphic to H_0 and disconnected from the rest of the graph. We now simulate the first phase. Let R' be the set of edges in $\hat{V} \times \hat{V}$ that are isomorphic to $H_0 \triangle H_1$, and set r' = |R'|; the structure of the lower bound graph \mathcal{H} guarantees $r' = \hat{n} = pn/20$. The adversary uses Lemma 5.1 to perform $\ell = 40 \frac{r'}{p} \log n$ updates such that after them the subgraph on \hat{V} is isomorphic to H_1 and disconnected from the rest of the graph w.h.p. It then perform the first query, and returns the same value or a function of it, as follows.

For all the subgraph counting problems, there is the same number of subgraphs through s (or s and t) in H_1 and in the disconnected subgraph on \hat{V} isomorphic to it, so the algorithm returns the same. For maximum bipartite matching, the graph on $V \setminus \hat{V}$ is a random balanced bipartite graph on $n - \hat{n}$ nodes with each edge w.p. 1/2. Hence, it has a matching of $\frac{n-\hat{n}}{2}$ edges w.h.p. [13], and this quantity should be added to the output on H_1 . The same output describes the minimum vertex cover size, by Kőnig's Theorem [25]. Finally, the graph has a perfect matching iff h_1 has a perfect matching, so the outputs are the same.

For complexity, note that the preprocessing takes $\tilde{O}(n^2)$ edge updates (in the graph \mathcal{G}), i.e. $\tilde{O}(n^2t_{\rm u}(n))$ operations (and the other operations are negligible). By assumption, this is $O(p^{2-\epsilon}n^{3-\epsilon})$ which is polynomial in \hat{n} . Each phase of \hat{n} updates of the original graph involves $O(n \log n)$ updates and a single query. Hence, each update of the original graph takes amortized

$$\hat{t_{u}}(\hat{n}) = \frac{1}{\hat{n}} n \log n \cdot t_{u}(n)$$
$$= \frac{1}{\hat{n}} O((pn)^{2-\epsilon})$$
$$= O(\hat{n}^{1-\epsilon})$$

and each query takes

$$\hat{t_q}(\hat{n}) = t_q(n) = O((pn)^{2-\epsilon}) = O(\hat{n}^{2-\epsilon})$$

contradicting Lemma 5.2.

5.2 Connectivity lower bound

In Lemma 5.1, if r' is small, and we take the minimal number of steps $\ell = \Theta(r'/p)$, the error term might grow too large to apply this simulation for, say, poly(n) times. One way to circumvent that is by making each simulation longer by a factor of $\log n$, reducing its error to $1/\operatorname{poly}(n)$, which allows us to union bound over all simulations. However, in some cases (specifically, connectivity) this is too costly. We show a better result utilizing the amortization over all simulations.

Lemma 5.4. Consider an n-node graph G_0 , a set $R \subseteq {V \choose 2}$ of r = |R| potential edges, and a sequence $R'_1, \ldots R'_k$ of k subsets $R'_i \subseteq R$ of size $r'_i = |R'_i|$ each, and $1 \le r_i \le \hat{r}$. An adaptive adversary can guarantee k phases, each starting with G_{i-1} and ends with G_i such that $(G_i \triangle G_{i-1}) \cap R = R_i$, using at most $\ell = \frac{12k\hat{r}}{p}$ steps, with error probability $1/k^c$ for any constant c and sufficiently large k, provided $r \le \frac{pn^2}{18}$.

Proof. Using the guarantee on r, we can apply Lemma 5.1, using $\ell_{\max} = \frac{40(c+2)\hat{r}\log k}{p}$, with some constant c>0. The error is upper bounded by k^{-c-2} , and k^{-c-1} for all iterations using union bound. We therefore assume the adversary "gives up" on a simulation after ℓ_{\max} steps, which happens with a small probability. This allows us to show concentration over enough phases.

Consider a single phase i. As per the analysis from the proof of Lemma 5.1, denote $\ell_i := \operatorname{argmin}(f(j) = G_i)$, the first time that G_i is reached.

Applying Lemma 5.1 with different values gives $\Pr[\ell_i > t] \leq \exp(-pt/40)$. By Fubini's theorem,

$$\mathbb{E}[\ell_i] = \sum_{t=0}^{\infty} \Pr[\ell_i \ge t] \le \sum_{t=0}^{\infty} \exp(-pt/40).$$

But this last sequences converges (having a fixed fractional quotient, $e^{-p/40} < 1$), proving the expectation is finite $\mathbb{E}[\ell_i] < \infty$, which is crucial to invoke Wald's equation for stopping times later on.

Similarly to the proof of Lemma 5.1, we start at $f(0) = r_i$ but this time we count all steps, not only effective ones. We continue until $f(\ell_i) = 0$, the first time we reach 0. In between f(i) > 0, and so each difference $X_j = f(j) - f(j-1)$ has the same distribution, X. With probability at least p, this step hit an edge of R, and conditioned on that, we have expected decrease of at least 4/5. Overall, we write $\mathbb{E}[X] \leq -4p/5$. The sum of all differences is exactly r_i , and so by Wald's equation:

$$r_i = \mathbb{E}\left[\sum_{i \in [\ell']} X_i\right] = \mathbb{E}[\ell_i] \cdot \mathbb{E}[X].$$

Plugging in the values and bounds, we finally get $\mathbb{E}[\ell_i] \leq 5r_i/(4p)$. Recall the fact that $\ell_i \in [0, \ell_{\max}]$, and note that each ℓ_i is independent of the others.

Use $\ell := \sum_{i \in [k]} \ell_i$ with $\mathbb{E}[\ell] \leq 5\hat{r}k/(4p)$, and apply Hoeffding's inequality:

$$\Pr\bigg[\ell \geq \frac{12\hat{r}k}{p}\bigg] \leq \Pr\bigg[\ell \geq \mathbb{E}[\ell] + \frac{10\hat{r}k}{p}\bigg] \leq \exp\bigg(-\frac{200\hat{r}^2k^2/p^2}{k\ell_{\max}^2}\bigg) = \exp\bigg(-\frac{k}{1600c^2\log^2(k)}\bigg) \leq \frac{1}{k^{c+1}}.$$

Where the last inequality holds for large enough k. Applying union bound over the two errors, the adversary succeeds within $12k\hat{r}/p$ steps with error probability at most $1/k^c$.

We apply Lemma 5.4 with $\hat{r} = 4$ and $k = \sqrt{n}$. This gives error that is $o_n(1)$, and overall $\Theta(\sqrt{n}/p)$ steps (with no extra logarithmic factors, as desired).

We state a known lower bound for connectivity in the worst-case (non-smoothed) setting.

Lemma 5.5 ([34, Section 9.1]). There is no dynamic algorithm for **connectivity** with error probability $\hat{n}^{-\Omega(1)}$, constant query time, and $\hat{t_u}(\hat{n}) = o(\log \hat{n})$ amortized time.

The proof uses a sequence of phases, which must take $\Omega(\sqrt{\hat{n}} \log \hat{n})$ steps each on average. Each phase is composed of $2\sqrt{\hat{n}}$ edge flips and then $\sqrt{\hat{n}}$ subsequences, each composed of O(1) edge flips and a query.

The connectivity lower bound is from [34], where the result appears in Section 9.1 and the details of the proof are in Theorem 2.4 and Section 6.1.

The proof uses a sequence of $\sqrt[3]{\hat{n}}$ phases, each composed of two macro-operations: a macro-updates, followed by verify-sum. A macro-update can be implemented by $2\sqrt{\hat{n}}$ edge flips, and a verify-sum can be implemented by $\sqrt{\hat{n}}$ phases, each consists of O(1) edge flips and a single query. The paper shows that each phase must take $\Omega(\sqrt{\hat{n}}\log\hat{n})$ cell-prob steps in average, and hence the same number of operations, which yields the lower bound.

Combining this result with Lemma 5.1 and Lemma 5.4, we can give a lower bound for connectivity in the *p*-smoothed setting with an adaptive adversary.

Theorem 5.6. Fix $\log^2 n/n . There is no dynamic algorithm for p-smoothed dynamic graphs with an adaptive adversary for$ **connectivity** $with error probability <math>(pn)^{-\Omega(1)}$, constant query time, and $t_u(n) = o(\log(pn))$ amortized time.

Proof. Assume for contradiction that in the *p*-smoothed setting with adaptive adversary, there is a connectivity algorithm as above with $t_{\rm u} = o(\log(pn))$ on an *n*-node graph. We follow a simulation argument as in the proof of Theorem 5.3.

Consider an instance of the problem on a fully-dynamic non-smoothed graph \mathcal{H} on $\hat{n} = pn/20$ nodes, with an initial graph H_0 . Recall that the worst-case dynamic graph is composed of phases, where each phase consists of $2\hat{n}$ updates, followed by \hat{n} sub-phases of O(1) updates and a query. We simulate the first phase, where the first part of $2\sqrt{\hat{n}}$ edge flips ends with a graph H', and the queries in the next $k = \sqrt{\hat{n}}$ sub-phases are on the graphs H_1, \ldots, H_k . The rest of the phases are similarly simulated.

Choose a random n-node graph G_0 where each edge exists w.p. 1/2 (a p-smoothed input by Observation 1), and an arbitrary set \hat{V} of pn/20 nodes. Let $R = \hat{V} \times V$ be the set of all potential edges inside \hat{V} and connecting \hat{V} to the rest of the graph. Set r = |R|, so $\frac{p}{21}n^2 \le r \le \frac{p}{20}n^2$ for large enough n. Fix an edge \hat{e} connecting \hat{V} and the rest of the graph.

By Lemma 5.1 with $R' \subseteq R$ and $r' \le r$, the adversary can make sure the subgraph on \hat{V} is isomorphic to the initial lower bound graph H_0 , and the only edge connecting \hat{V} and the rest of the graph is \hat{e} , in $\ell = 40 \frac{r}{p} \log n$ steps w.h.p. Start the simulation with these ℓ steps.

The *n*-node graph (of \mathcal{G}) now has a subgraph on \hat{V} isomorphic to H_0 and connected to the rest of the graph by a single edge \hat{e} . We now simulate the first part of the phase, which ends with H'. Let R' be the set of edges in $\hat{V} \times \hat{V}$ that are isomorphic to $H_0 \triangle H'$, and set r' = |R'|; the structure of the lower bound graph \mathcal{H} guarantees $r' = 2\sqrt{\hat{n}} = \sqrt{pn/5}$. The adversary uses Lemma 5.1 to perform $\ell = 10\frac{r'}{n} = O(\sqrt{n/p})$ steps (note that ℓ is large enough due to the range of p) such that

after them the subgraph on \hat{V} is isomorphic to H' and is connected to the rest of the graph only by \hat{e} w.h.p.

We now turn to simulate the sub-phases. Recall that there are $k = \sqrt{\hat{n}}$ sub-phases, each with O(1) edge changes an a single query. By Lemma 5.4, this can by be simulated in $\ell = O(\sqrt{\hat{n}}/p) = O(\sqrt{n/p})$ steps with failure probability $\hat{n}^{-c/2} = O((pn)^{-c/2})$ for any constant c.

In each query, we return the same output on the simulated H_i as on the n-node graph. This is because the subgraph on \hat{V} is connected iff H_i is connected, by the isomorphism: the rest of the graph is a random graph with each edge existing w.p. 1/2, so it is connected w.h.p. if n is large enough (see Claim 4.4); and the edge \hat{e} connects \hat{V} to the rest of the graph. Hence, the whole graph is connected iff H_i is connected.

For complexity, note that each query is answered in a constant time, and each phase is simulated using $O(\sqrt{\hat{n}})$ updates, which are $O(\sqrt{\hat{n}})t_{\rm u}(n)$ steps. As a step must take $\Omega(\sqrt{\hat{n}}\log(\hat{n}))$ steps on average, we cannot have $t_{\rm u}(n) = o(\log(\hat{n})) = o(\log(pn))$ amortized time.

5.3 Lower bounds with oblivious add/remove adversary

In this section we focus on oblivious adversaries, but with the power to add/remove edges, which allows it to "insist" on the embedding. A weaker version of Lemma 5.1 can be shown for this adversary, one that only holds with p = 1 - o(1), but it is enough to leverage to a non-trivial lower bound, showing separation between the oblivious add/remove and the oblivious flip adversaries (along with the upper bounds of Lemma 4.6).

For convenience, write q = 1 - p, the probability that a random edge is flipped at a given step. We show:

Lemma 5.7 (Embedding a graph, oblivious add/remove adversary). Consider an n-node graph G, a set $R \subseteq \binom{V}{2}$ of r = |R| potential edges, and a subset $R' \subseteq R$ of size |R'| = r'. An adaptive adversary can reach a new graph G' satisfying $(G \triangle G') \cap R = R'$ within at most ℓ steps with failure probability at most

$$r' \cdot q^{\frac{\ell}{r'}} + \frac{q\ell r}{n^2}.$$

The oblivious adversary uses all its changes for R', trying to enforce the desired arrangement (using add and remove operations). This can be done even with no knowledge about the current graph, assuming the previous embedding worked well.

Interestingly, even without knowledge of G_0 , the oblivious adversary that can insist on certain changes, which allows it to control its own piece of the graph, albeit quite small. The flip adversary cannot do even this.

Proof. Consider a sequence of ℓ changes. The adversary equally divides its attention among the r' edges, repeatedly adding or removing them accordingly. Each adversarial change is swapped by a random one with probability q, and so for each edge, with probability $q^{\ell/r'}$ none of the q add/remove operations on it follow through and the change does not happen. Using a union bound over all edges of R', we have probability of $r' \cdot q^{\ell/r'}$ that some edge was not changed by the adversary.

On the other hand, each step involves a probability of $q \cdot (r/n^2)$ for a random edge to be chosen, and then hit the set R. Using a union bound over ℓ changes, we have an upper bound of $q\ell r/n^2$ that any random change in the entire sequence touched R.

We use this lemma to prove lower bounds for the p-smoothed setting with an oblivious add/remove adversary, in a way similar to the proof of Theorem 5.3.

Theorem 5.8. Fix a constant $0 < \delta < 1$ and $1 - n^{-\delta} \le p \le 1$. There is no dynamic algorithm for p-smoothed dynamic graphs with an oblivious add/remove adversary for counting s k-cycles for $k \ge 3$ odd (and hence s-triangle detection), counting s-t 3-paths, counting s-t 4-paths, bipartite perfect matching, bipartite maximum matching and bipartite minimum vertex cover with error probability at most 1/3, polynomial preprocessing time, $t_{\rm u}(n) = O(n^{\frac{\delta}{3}-\epsilon})$ and $t_{\rm q}(n) = O(n^{\frac{2\delta}{3}-\epsilon})$ (both amortized) for any $\epsilon > 0$ unless the OMv conjecture is false.

This result is clearly much weaker than Theorem 5.3, which is not surprising as the oblivious add/remove adversary is weaker than the oblivious one. Yet, it is interesting as it shows a separation between the oblivious add/remove adversary and the oblivious flip adversary for the problem of bipartite perfect matching — compare to Lemma 4.6.

Proof. Assume for contradiction that in the p-smoothed setting with oblivious add/remove adversary, one of the above problems has an algorithm with better running times than claimed on an n-node graph. We devise a fast dynamic algorithm for the same problem in a worst-case dynamic graph \mathcal{H} on $\hat{n} = n^{\delta/3}$ nodes by creating a p-smoothed n-node dynamic graph \mathcal{G} and running the claimed fast algorithm on it, as in the proof of Theorem 5.3.

Consider an instance of the problem on a fully-dynamic non-smoothed graph \mathcal{H} on \hat{n} nodes, with an initial graph H_0 . Choose a random n-node graph G_0 where each edge exists w.p. 1/2 as the initial graph (Observation 1). Chooses an arbitrary set \hat{V} of \hat{n} nodes, which includes the designated nodes (s and sometimes t) if those exist in the problem, and is a balanced bipartite graph if the problem is on bipartite graphs. Let $R = \hat{V} \times V$ be the set of all potential edges inside \hat{V} and connecting \hat{V} to the rest of the graph. Set $r = |R| = n^{1+\delta/3}$.

By Lemma 5.7 with $R' \subseteq R$ (and $r' \le r$), the adversary can make sure the subgraph on \hat{V} is isomorphic to the initial lower bound graph H_0 , and is disconnected from the rest of the graph, in $\ell = 3r \log n$ steps with error probability

$$r' \cdot q^{\ell/r'} + \frac{q\ell r}{n^2} \le r(1/2)^{3\log n} + 3\log n \cdot n^{-\delta/3} = o(1).$$

The *n*-node graph (of \mathcal{G}) now has a subgraph on \hat{V} isomorphic to H_0 and disconnected from the rest of the graph. We now simulate the first phase. Let R' be the set of edges in $\hat{V} \times \hat{V}$ that are isomorphic to $H_0 \triangle H_1$, and set r' = |R'|; the structure of the lower bound graph \mathcal{H} guarantees $r' = \hat{n}$. The adversary uses Lemma 5.7 to perform $\ell = 3r' \log n$ updates such that after them the subgraph on \hat{V} is isomorphic to H_1 and disconnected from the rest of the graph with error probability at most

$$r' \cdot q^{\ell/r'} + \frac{q\ell r}{n^2} \le n^{-2\delta \log n} + 3n^{-1-\delta/3} \log n = o(1).$$

Note that even repeating this for \hat{n} phases leaves the error probability in o(1). It then performs the first query, and returns the same value or a function of it, as in the proof of Theorem 5.3.

For complexity, note that the preprocessing takes $O(n^2)$ edge updates (in the graph \mathcal{G}), i.e. $O(n^2t_{\rm u}(n))$ operations. By assumption, this is $O(n^{2+\delta/3-\epsilon})$ which is polynomial in \hat{n} . Each phase of \hat{n} updates of the original graph involves $3r'\log n = O(\hat{n}\log n)$ updates and a single query. Hence, each update of the original graph takes amortized

$$\begin{split} \hat{t_{\mathbf{u}}}(\hat{n}) &= \frac{1}{\hat{n}} O(\hat{n} \log n) \cdot t_{\mathbf{u}}(n) \\ &= O(t_{\mathbf{u}}(n) \log n) \\ &= O(n^{\frac{\delta}{3} - \epsilon} \log n) \\ &= O(\hat{n}^{1 - \epsilon}) \end{split}$$

and each query takes

$$\begin{split} \hat{t_{\mathbf{q}}}(\hat{n}) &= t_{\mathbf{q}}(n) \\ &= O(n^{\frac{2\delta}{3} - \epsilon}) \\ &= O(\hat{n}^{2 - \epsilon}) \end{split}$$

contradicting Lemma 5.2.

6 Lower bounds for counting small subgraphs

The main focus of this section is proving a conditional lower bound for counting s-t 3-paths in a p-smoothed model, even with the weakest adversary—an oblivious flip adversary. We then extend this lower bound to other subgraph counting problems.

Given the OMv conjecture, the average-case parity OuMv problem is also hard (Lemma 1.1). The proof then goes through the problem of counting s-t 3-paths in a specific family of well-structured graphs, called P_3 -partite graph, similarly to the outline of [21]. We show:

- Average-case parity OuMv can be solved using an algorithm that counts s-t 3-paths in P_3 -partite p-smoothed dynamic graphs (Section 6.1).
- An algorithm that counts s-t 3-paths in a (general) p-smoothed dynamic graph, can be used to count the specific type of such paths in a P_3 -partite p-smoothed dynamic graph (Section 6.2).
- From these reductions, we conclude the conditional hardness of counting s-t 3-paths in a general p-smoothed dynamic graph (Section 6.3).

Overall, we get an (almost) tight lower bound for counting s-t 3-paths, for (almost) any value of p. This is extended to other subgraph counting problems in Section 6.4. Throughout this section, we use adversaries that pick the initial random graphs at random (with each edge existing with an independent probability of 1/2), which assures that the smoothed initial graphs are also random, by Observation 1.

6.1 Solving OuMv by counting s-t 3-paths on P_3 -partite graphs

A graph is called P_3 -partite if it is composed of 4 node sets, $V = \{s\} \sqcup A \sqcup B \sqcup \{t\}$, and all its edges are of type sA, AB, or Bt. Formally, it is a (H, Π) -partite graph with H being a path (v_1, v_2, v_3, v_4) and Π mapping s to v_1 , all the nodes of A to v_2 , all the nodes of B to v_3 , and t to v_4 . In this section we prove the following lemma.

Lemma 6.1. If there exists a data structure for counting s-t 3-paths on a p-smoothed P_3 -partite dynamic n'-node graph initialized at random, with an oblivious flip adversary and preprocessing, update and query times $t_{pre}(n')$, $t_q(n')$ and error probability at most 1/100, then there exists an algorithm solving the average-case parity OuMv problem of dimension n in

$$3t_{\mathrm{pre}}(n') + O\left(\frac{n\log n}{p}\right) \cdot t_{\mathrm{u}}(n') + O(n) \cdot t_{\mathrm{q}}(n') + O\left(\frac{n^2\log n\log(n/p)}{p}\right)$$

steps of computation, where n' = 2n + 2.

6.1.1 The setting

OuMv and s-t 3-paths. Following the connection established in [20], we associate the OuMv problem of dimension n with P_3 -partite graphs on n' = 2n + 2 nodes $V = \{s\} \sqcup A \sqcup B \sqcup \{t\}$, where |A| = |B| = n. A P_3 -partite graph has edges of types sA, AB, Bt, and they can be naturally represented algebraically: u (resp., v) represent all neighbors of s in A (resp., of t in B), and M is the adjacency matrix between A and B. It is easy to verify that the multiplication $u^T M v$ (over \mathbb{R}) is the number of 3-paths of type sABt.

Indexing. With the aforementioned connection in mind, it is more convenient to index the vectors u, v and the matrix M with graph edges. That is, we use $u(s, a_i)$ for u(i), use $v(b_j, t)$ for v(j) and use $M(a_i, b_j)$ for M(i, j). The indices of the three objects are disjoint and together they correspond to all n(n+2) allowed edges in the P_3 -partite graph above. We often use u(e), v(e), M(e), with an appropriate edge e.

The adversarial strategy. We fix a (randomized) oblivious flip adversary throughout the section, an adversary that flips an edge of sA or Bt chosen u.a.r, and never flips an edge of AB:

$$\mathcal{D}_{\text{adv}}(e) := \begin{cases} \frac{1}{2n}, & \text{if } e \text{ is of type } sA \text{ or } Bt \\ 0, & \text{if } e \text{ is of type } AB \end{cases}$$

In the p-smoothed model, each change samples an edge independently and according to the distribution $\mathcal{D}_{\text{adv}}^p = p \cdot \mathcal{D}_{\text{adv}} + (1-p) \cdot U_{P_3}$, where U_{P_3} is the uniform distribution over all n(n+2) allowed edges in the P_3 -partite graph. More specifically:

$$\mathcal{D}_{\text{adv}}^{p}(e) := \begin{cases} \frac{p}{2n} + \frac{1-p}{n(n+2)}, & \text{if } e \text{ is of type } sA \text{ or } Bt \\ \frac{1-p}{n(n+2)}, & \text{if } e \text{ is of type } AB \end{cases}$$

Random sequence of updates. Our reduction needs to create an authentic sequence of changes, as if it came from the p-smoothed dynamic graph with the adversarial strategy above. Such a sequence is simply a (randomly ordered) sample set from $\mathcal{D}_{\text{adv}}^p$, which can be represented by its histogram H, a vector of occurrences where H(e) is the number of times an edge e appears.

In actuality, we require *conditional* sampling: for new OuMv inputs u_i, v_i , we wish to create a sequence of changes that ends with certain edge sets for types sA, Bt (dictated by u_i, v_i). This means that (the parity of) H(e) for all these edges is fixed ahead, which can potentially skew other things in our sample set (e.g., since the sum of all H(e) is the total number of changes performed).

Poissonization. In order to overcome such complications, we use Poissonization (Fact 2): we sample Pois(t) changes from $\mathcal{D}^p_{\text{adv}}$ by independently sampling a histogram: the number of occurrences of edge e, denoted H(e), is sampled from Pois $(\mathcal{D}^p_{\text{adv}}(e) \cdot t)$. This way, conditioning on the parity of H(e) does not affect the distribution of H(e') for any $e' \neq e$.

Changing edges of type AB. Generating a sequence of changes that complies with the target vectors (u_i, v_i) might change edges that correspond to M along the way (unless p = 1). This means a query for the number of s-t 3-paths is answered with $u_i^T M' v_i$ for some M' instead of the original M. Similarly to [21], we circumvent this problem by creating 3 correlated sequences of changes that contain the same changes on edges of u_i, v_i and otherwise their changes correspond to matrices

 M_1, M_2, M_3 that sum (modulo 2) exactly to M. Thus, one can retrieve the desired (parity) value by distributivity over \mathbb{F}_2 :

$$u_i^T M v_i = u_i^T M_1 v_i + u_i^T M_2 v_i + u_i^T M_3 v_i.$$

6.1.2 The reduction

Assume a data structure that runs on graphs with n' = 2n + 2 nodes with preprocessing time $t_{\text{pre}}(n')$, update time $t_{\text{u}}(n')$ and query time $t_{\text{q}}(n')$. We show how to use these operations in order to solve the parity OuMv problem (of dimension n).

The initial inputs for OuMv are M and u_0, v_0 , where all entries have random Boolean value. These correspond perfectly to an initial random P_3 -partite graph on n' nodes and random initialization. One can construct the graph, apply the preprocessing and use a single query to retrieve the parity of $u_0^T M v_0$.

Next we describe how, given new inputs u_i, v_i , one can utilize the data structure to efficiently compute the parity of $u_i^T M v_i$. This is done using three data structures, for each we generate a sequence of changes. We can then query once each data structure, and combine the three answers to finish up.

Algorithm 1: Algorithm Sol for solving OuMv by counting paths in P_3 -partite graphs

- 1 $t \leftarrow 5n \log(n)/p$
- 2 Initialize S^1, S^2, S^3 to be empty sequences of changes
- **3** Compute the differences $u_{\text{dif}} \leftarrow u_i u_{i-1}$ and $v_{\text{dif}} \leftarrow v_i v_{i-1}$ (modulo 2)
- 4 For each sA edge e, draw $z_e \sim \text{Pois}(\mathcal{D}^p_{\text{adv}}(e) \cdot t)|_{z_e \equiv_2 u_{\text{dif}}(e)}$. Add z_e copies of e to \mathcal{S}^j for $j \in [3]$
- 5 For each Bt edge e, draw $z_e \sim \text{Pois}(\mathcal{D}^p_{\text{adv}}(e) \cdot t)|_{z_e \equiv_2 v_{\text{dif}}(e)}$. Add z_e copies of e to \mathcal{S}^j for $j \in [3]$
- **6** for For edges of type AB, for each $j \in [3]$ do

Draw
$$z^j \sim \operatorname{Pois}\left(\frac{(1-p)n}{n+2} \cdot (t/2)\right)$$

Draw z^j uniformly random matrix entries e (with repetitions), and add each such edge to the two sequences $\mathcal{S}^{j'}$ for which $j' \neq j$.

- 7 Randomize the order of each of the three sequences S^1, S^2, S^3 . Feed each sequence to a different data structure instance, applying the update procedure with each change.
- 8 Query all 3 instances for the number of s-t 3-paths, sum the answers and output the parity of the sum.

Call the procedure above SOL and run it for each new pair u_i, v_i to produce the one-bit answer. In particular, we refer to steps (4)–(7) as the reduction steps, taking two difference vectors $u_{\text{dif}}, v_{\text{dif}}$ and returning three sequences of changes $\mathcal{S}^1, \mathcal{S}^2, \mathcal{S}^3$.

6.1.3 Analysis

For the sake of clarity, in this section we use $\mathcal{S}_{\text{red}}^{j}$ to refer to the sequences \mathcal{S}^{j} output by the reduction (for any $j \in [3]$), and denote the histogram of such a sequence by $\mathcal{H}_{\text{red}}^{j}$. For AB-type edges, drawn in step (6), denote the amount of times the edge e was chosen for some $j \in [3]$ by z_{e}^{j} . Thus, for every AB-type edge e we have

$$\mathcal{H}^{j}_{\mathrm{red}}(e) = \sum_{j' \in [3] \setminus \{j\}} z_e^{j'},$$

since edges were added to each sequence from the two iteration with different index.

Note that the first inputs M, (u_0, v_0) are simply mirrored perfectly in the initial graphs of all three copies. We next prove the correctness of SoL for any fixed sequence of vector pairs. That is, we show that the reduction maintains the connection between OuMv and s-t 3-paths.

Claim 6.2 (Correctness for a fixed input). Fix inputs M and $(u_0, v_0), \ldots, (u_n, v_n)$. For any $i = 0, \ldots, n$, if all the queries to the 3 data structures are answered correctly, then the online algorithm correctly outputs $u_i^T M v_i$.

Proof. We prove that the following invariant holds for i = 0, ..., n: at the end of iteration i the edges of types sA and Bt in all three copies correspond to (u_i, v_i) , and edges of type AB in the three copies correspond to three matrices M^1, M^2, M^3 such that $M^1 + M^2 + M^3 \equiv_2 M$.

This suffices for proving the claim, since for each iteration i = 0, ..., n we have

$$u_i^T M^1 v_i + u_i^T M^2 v_i + u_i^T M^3 v_i = u_i^T M v_i.$$

The invariant is proved by induction. For the base case i=0, all three initial graphs mirror M and u_0, v_0 perfectly, and indeed $M+M+M=3M\equiv_2 M$.

For the inductive step, focus on a single iteration of Sol. Edges of types sA are shared by all 3 copies and undergo the same changes. By induction hypothesis, at the beginning of the i^{th} iteration, the existence of any sA edge e corresponds to $u_{i-1}(e)$. By the conditional sampling process, at the end of the iteration its existence $\mathbb{1}(e)$ satisfies:

$$1(e) \equiv_2 u_{i-1}(e) + z_e \equiv_2 u_{i-1}(e) + u_{\text{dif}}(e) \equiv_2 u_i(e).$$

The congruence implies equality, since values are Boolean on both ends. The same holds for Bt edges and the vectors v_{i-1}, v_i .

Edges of type AB are added differently to each copy. For each $j \in [3]$, let M^j and $\mathbb{1}^j(e)$ denote the edges in the j^{th} copy at the beginning and the end of the iteration, respectively. By the induction hypothesis we have $M^1 + M^2 + M^3 \equiv_2 M$. For each AB edge e and copy j we have $\mathbb{1}^j(e) \equiv_2 M^j(e) + \mathcal{H}^j_{\text{red}}(e)$. Altogether:

$$\mathbb{1}^{1}(e) + \mathbb{1}^{2}(e) + \mathbb{1}^{3}(e) \equiv_{2} \left((M^{1}(e) + M^{2}(e) + M^{3}(e)) \right) + \left(\mathcal{H}^{1}_{\text{red}}(e) + \mathcal{H}^{2}_{\text{red}}(e) + \mathcal{H}^{3}_{\text{red}}(e) \right) \\
\equiv_{2} M(e) + 2\left(z_{e}^{1} + z_{e}^{2} + z_{e}^{3}\right) \\
\equiv_{3} M(e),$$

where the second congruence uses the induction hypothesis for the first summand and the reduction construction for the second. Equality is implied as both ends are Boolean values. \Box

Next, we analyze a distributional OuMv input, but not yet a uniformly random one. Instead, we focus on a distribution μ_{adv} over pairs $(u_{\text{dif}}, v_{\text{dif}})$ that naturally arises from a sequences of changes.

Consider a p-smoothed sequence of Pois(t) updates, with the adversarial strategy \mathcal{D}_{adv} . Denote this sequence by \mathcal{S}_{adv} , and its histogram by \mathcal{H}_{adv} . Their distribution is exactly:

$$\mathcal{S}_{\mathrm{adv}} \sim (\mathcal{D}_{\mathrm{adv}}^p)^{\mathrm{Pois}(t)}; \quad \mathcal{H}_{\mathrm{adv}}(e) \sim \mathrm{Pois}(\mathcal{D}_{\mathrm{adv}}^p(e) \cdot t).$$

Define the projection of \mathcal{H}_{adv} to parities for edges of types sA, Bt:

$$u_{\text{adv}}(e) := \mathcal{H}_{\text{adv}}(e) \pmod{2}$$
 $v_{\text{adv}}(e) := \mathcal{H}_{\text{adv}}(e) \pmod{2}$

Our desired distribution μ_{adv} is the projection of the random histogram \mathcal{H}_{adv} to these vectors:

$$\mu_{\text{adv}}(u, v) := \Pr_{\mathcal{H}_{\text{adv}}}[u_{\text{adv}} = u \land v_{\text{adv}} = v]$$

Our next claim is that on a pair sampled from μ_{adv} , the resulting histograms from the reduction $\mathcal{H}^{j}_{\text{red}}$ (for $j \in [3]$) have the same distribution as \mathcal{H}_{adv} :

Claim 6.3 (reduction output on distributional input). If the reduction (steps (4)-(7)) is fed with distributional inputs ($u_{\text{dif}}, v_{\text{dif}}$) $\sim \mu_{\text{adv}}$, then the three output sequences satisfy, for all $j \in [3]$:

$$\mathcal{S}_{\mathrm{red}}^{j} \stackrel{\mathrm{d}}{=} \mathcal{S}_{\mathrm{adv}}$$

Proof. As both adversarial and the reduction sequences are randomly order, it suffices to prove the statement for the histograms. Fix two Boolean vectors (u, v), and denote by $\mathcal{H}^{j}_{\text{red}}[u, v]$ the histograms created when $(u_{\text{dif}}, v_{\text{dif}}) = (u, v)$.

By Poissonization, the distribution of \mathcal{H}_{adv} is a product of independent distributions $\mathcal{H}_{adv}(e)$. Thus, for any pair of Boolean vectors u, v, the conditioning on $\{u_{adv} = u \land v_{adv} = v\}$ affects each entry $\mathcal{H}_{adv}(e)$ separately, for e of types sA, Bt. This is mirrored by steps (4), (5) which determine $\mathcal{H}^{j}_{red}[u, v]$ for $j \in [3]$.

For edges of type AB, step (6) takes a Poisson number of samples, each is uniform over the n^2 edges of type AB. Thus, again by Poissonization:

$$z_e^j \sim \text{Pois}(\mathcal{D}_{adv}(e) \cdot (t/2)),$$

where we used $\mathcal{D}_{adv}(e) = \frac{1-p}{n(n+2)}$ for AB-type edges. By additivity of Poisson distributions, for $j \in [3]$ and any AB-type edge e, the following distributions are equal:

$$\mathcal{H}_{\mathrm{red}}^{j}(e) \stackrel{\mathrm{d}}{=} \sum_{j' \in [3] \setminus \{j\}} z_{e}^{j'} \sim \mathrm{Pois}(\mathcal{D}_{\mathrm{adv}}(e) \cdot t).$$

Overall, conditioned on $(u_{\text{dif}}, v_{\text{dif}}) = (u, v)$, we get:

$$\mathcal{H}_{\text{red}}^{j}[u,v] \stackrel{\text{d}}{=} \mathcal{H}_{\text{adv}}|(u_{\text{adv}} = u \land v_{\text{adv}} = v). \tag{4}$$

By the law of total probability, with $(u_{\text{dif}}, v_{\text{dif}}) \sim \mu_{\text{adv}}$ and for $j \in [3]$, we get the following:

$$\mathcal{H}_{\mathrm{red}}^{j} \stackrel{\mathrm{d}}{=} \sum_{(u,v)} \mu_{\mathrm{adv}}(u,v) \cdot \mathcal{H}_{\mathrm{red}}^{j}[u,v] \stackrel{\mathrm{d}}{=} \sum_{(u,v)} \mu_{\mathrm{adv}}(u,v) \cdot \mathcal{H}_{\mathrm{adv}} | (u_{\mathrm{adv}} = u \wedge v_{\mathrm{adv}} = v) \stackrel{\mathrm{d}}{=} \mathcal{H}_{\mathrm{adv}}. \qquad \Box$$

To relate the last claim to the average-case OuMv, note that instead of drawing a uniformly random pair (u_i, v_i) , one can uniformly draw the difference vectors $(u_{\text{dif}}, v_{\text{dif}})$ at each iteration. We thus relate n independent draws from μ_{adv} (denoted μ_{adv}^n) to a uniform choice of n pairs of difference vectors.

By a slight abuse of notation, we think of both distributions as over $2n^2$ bits (2n vectors of n bits), and use U_m for the uniform distribution on m bits. Our closeness measure is statistical distance, which is a normalized ℓ_1 distance (with factor 1/2).

Claim 6.4. It holds that

$$SD(\mu_{\text{adv}}^n, U_{2n^2}) \le 2n^{-3}$$

Proof. Focus on a single iteration. All edges of type sA, Bt have the same probability under $\mathcal{D}_{\text{adv}}^p$, call it p_1 . By Poissonization, each of the 2n bits in μ_{adv} is i.i.d according to the parity of $\text{Pois}(p_1 \cdot t)$, which by Fact 2 is simply $\text{Ber}((1 + e^{-p_1 \cdot t})/2)$. Thus, for the j^{th} bit of μ_{adv} :

$$SD((\mu_{adv})_j, U_1) \le e^{-2p_1 \cdot t} \le e^{-5\log n} = n^{-5},$$

where the inequality is due to $p_1 \ge p/(2n)$ and $t = 5n \log n/p$.

Finally, due to independence of the 2n edges within the same iteration, and independence among the n iterations, we can use subadditivity of statistical distance (Item 1 of Fact 3):

$$SD(\mu_{adv}^n, U_{2n^2}) \le n \sum_{j=1}^{2n} SD((\mu_{adv})_j, U_1) \le 2n^2 \cdot n^{-5} = 2n^{-3}.$$

Combining Claim 6.2 and Claim 6.3, and Claim 6.4 proves the correctness of Sol:

Claim 6.5 (Correctness). Assume there exists a data structure for counting s-t 3-paths which answers all queries correctly with probability at least 0.99 on a p-smoothed sequence of changes, then algorithm Sol solves average-case OuMv w.p. at least 0.96.

Proof. An average-case input for OuMv can be constructed as follows. Draw uniformly random M, u_0, v_0 and then for n iterations, draw uniformly random difference vectors $u_{\text{dif}}, v_{\text{dif}}$. By setting u_i as the xor of u_{i-1} and the i-th difference vector u_{dif} , we get a sequence of uniformly random vectors u_0, \ldots, u_n ; the vectors v_0, \ldots, v_n are defined similarly.

First, we consider a similar input, where the difference vectors $u_{\rm dif}$, $v_{\rm dif}$ are drawn from $\mu_{\rm adv}$ instead of uniformly (each iteration independently). By Claim 6.3, the sequences $\mathcal{S}_{\rm red}^j$ at each iteration are distributed according to the adversarial strategy $\mathcal{D}_{\rm adv}$, and so each copy of the data structure has error probability at most 0.01. By union bound, the probability of any error within all executions is at most 0.03.

By Claim 6.2, whenever there are no errors, SoL correctly outputs $u_i^T M v_i$. Thus, over the distributional inputs it succeeds with probability at least 0.97.

Lastly, by Claim 6.4, the entire input has statistical distance at most $2n^{-3}$ from the one of the average-case. By Item 2 of Fact 3, the error of an algorithm can only differ by the statistical distance between the input distributions. Thus, on an average-case input, Sol succeeds at least with probability $0.97 - 2n^{-3} > 0.96$.

Claim 6.6 (Running time). If there exists a data structure for counting s-t 3-paths with preprocessing time $t_{\rm pre}$, update time $t_{\rm u}$ and query time $t_{\rm q}$, then with probability 1-o(1) the running time of our algorithm is

$$3t_{\mathrm{pre}}(n') + O\left(\frac{n^2 \log n}{p}\right) \cdot t_{\mathrm{u}}(n') + O(n) \cdot t_{\mathrm{q}}(n') + O\left(\frac{n^2 \log n \log(n/p)}{p}\right).$$

Proof. We pre-process three separate executions, and query 3(n+1) times overall. The rest of the proof analyzes the number of update operations, and the time for all other computations.

Focus on one iteration. As we successfully simulate a sequence of Pois(t) changes, by Fact 2:

$$\Pr[\text{Pois}(t) > et] \le \frac{(et)^{et}e^{-t}}{(et)^{et}} = e^{-t} \le e^{-n}.$$

That is, there is a negligible probability we exceed O(t) changes (where we can abort). Assume henceforth this is not the case. In particular, the reduction uses a total of O(t) update operations.

The nitty-gritty is in generating Poisson values. By Fact 2, one can generate a sample of Pois(λ) with running time O(m) where m is the output value and $\mathbb{E}[m] = \lambda$. When generating multiple values, we can rely on their sum for running time, but it is important to take into account the number of values generated.⁶ Differently put, the running time is actually $O(m_i + 1)$ for each generated value m_i , and for k such values the running time is O(m + k) where $m = \sum_{i \in [k]} m_i$.

In steps (4) and (5) we need conditional samples, which can be obtained by rejection sampling. This is true since the parity of each such value is almost a fair random coin (as seen in the proof of Claim 6.4). Thus, any desired parity value has probability at least 1/3, and using $O(\log n)$ repetitions suffices to get a polynomially small error probability (even after union bounding the 2n conditional samples). Generating 2n such values, with sum O(t) (with probability 1 - o(1)) takes $O((t+2n) \cdot \log n)$ time.

In step (6) we generate all AB-type changes. Generating each value separately would result in n^2 values (most of which are 0), which is too costly. Instead, we sample the total amount of changes (which is O(t) w.h.p), and assign random edge to each one, using $\log n$ random bits. The overall running time of this step is $O(t \log n)$. In step (7) we randomly order the sequences, each of length O(t), which takes $O(t \log t)$ steps.

Overall, for a single iteration we require $O(t \log t)$ steps of computation, on top of O(t) update operations and one query. The numbers for n iterations follow once we plug $t = O\left(\frac{n \log n}{p}\right)$.

Finally we can deduce the lemma.

Proof of Lemma 6.1. union bounding the error probabilities from Claim 6.5 and Claim 6.6, the existence of an algorithm for counting s-t 3-paths with running times $t_{\rm pre}$, $t_{\rm u}$, $t_{\rm q}$ implies an algorithm that correctly solves average-case parity OuMv using Sol, with error probability at most 0.04 + o(1) < 0.05 and the running time from Claim 6.6.

6.2 Reducing P_3 -partite graphs to general graphs

In this section, we reduce counting s-t 3-paths in p-smoothed P_3 -partite dynamic graphs to counting all s-t 3-paths in a general p'-smoothed dynamic graph, for some $p' = \Theta(p)$. We prove the following lemma.

Lemma 6.7. If there exists a data structure for counting s-t 3-paths on (general) p'-smoothed dynamic graphs initialized at random that fails with probability at most 1/1600, then there exists a data structure for counting s-t 3-paths on p-smoothed P_3 -partite dynamic graphs initialized at random, where $p \in [p', 2p']$, with the same asymptotic running times and error probability at most 1/100.

To prove this lemma, we take a p-smoothed P_3 -partite dynamic graph, and create 16 p'-smoothed unrestricted dynamic graphs, each with the same set of nodes as the original graph. Moreover, the 16 dynamic graphs differ only by their initial graphs, while they all consist of the same sequence of changes; hence, we only introduce one sequence of changes. Later on, we show that from the number of s-t 3-paths in the 16 unrestricted graphs, one can deduce the number of s-t 3-paths in the P_3 -partite graph.

The reduction works for any value of $p \in [0, 1]$, and coincides with the relevant models in the cases of p = 0 and p = 1: when p = 0, the original sequence is uniform on the edges allowed in a P_3 -partite graph and the new sequence is uniform on all the graph edges, as in [21]; when p = 1, both sequences are the same, and the reduction shows an adversarial strategy in the general model that focuses only on edges that can appear in a P_3 -partite graph.

⁶Indeed, consider generating k samples from Pois(1/k). Their sum is w.h.p. O(1), but the overall running time is $\Theta(k)$, as we pay additional $\Theta(1)$ for each value.

6.2.1 Simulating a single step

We transform a sequence of changes in the p-smoothed restricted model (hereafter: restricted sequence) into a longer sequence of changes in a p'-smoothed general model (hereafter: general sequence). Recall that R_H is the set of allowed edges in the restricted model. In order to construct the general sequence we use a parameter $0 \le \alpha \le 1$. For each change of an edge e in the restricted sequence, w.p. α we add to the general sequence the same edge e, and w.p. $1 - \alpha$ we add to the general sequence a uniformly random edge from the set $\overline{R_H} = \binom{V}{2} \setminus R_H$ of edges not allowed in the restricted sequence.

Let

$$\alpha = \frac{|R_H|}{|R_H| + (1-p)|\overline{R_H}|}$$

and note that $\alpha \in [|R_H|/{|V| \choose 2}, 1]$ since $p \in [0, 1]$. Given a flip of an edge e in the restricted sequence, we choose the change in the general sequence as follows.

- 1. Flip a coin C with probability α .
- 2. If C=1, make an *interior* update, i.e. flip the edge e in the general sequence. Note that the choice of e in the restricted model implies that, conditioned on C=1, w.p. p the edge e is chosen by the adversary (from R_H), and w.p. 1-p it is chosen uniformly at random from R_H .
- 3. If C = 0, make an exterior update: add to the general sequence an edge chosen uniformly at random from $\overline{R_H}$.

To see that the update sequences created by this process is indeed a p'-smoothed sequence on general graphs, note that the updates of the following types happen with the prescribed probabilities.

- An adversarial interior update w.p. $p' = \alpha \cdot p$;
- A random interior update w.p. $(1-p')|R_H|/\binom{|V|}{2} = \alpha \cdot (1-p)$;
- A random exterior update w.p. $(1-p')|\overline{R_H}|/\binom{|V|}{2}=1-\alpha$.

That is, an adversarial update happens w.p. p', and otherwise the update is chosen uniformly at random from all of $\binom{V}{2}$. This corresponds to a p'-smoothed sequence with an adversary that only chooses edges from R_H .

Recall that a P_3 -partite graph consists of 4 parts, denoted $V = \{s\} \cup A \cup B \cup \{t\}$, where s and t are single nodes and |A| = |B| = n. Its set of interior edges is thus

$$R_{P_3} = (\{s\} \times A) \cup (A \times B) \cup (B \times \{t\}).$$

Since |V| = 2n + 2 and $|R_{P_3}| = n(n+2)$, we have $\alpha \in [\frac{1}{2}, 1]$ (for any value of p), and therefore $p' \in [p/2, p]$.

The complement set $\overline{R_{P_3}}$ consists of edges of exterior types: sB, At, AA, BB, st. As the edge (s,t) cannot participate in any simple 3-path from s to t, we are interested in the four other edge types.

6.2.2 The actual reduction

In order to reduce the problem of counting s-t 3-paths in general (smoothed dynamic) graphs into the same problem in P_3 -partite (smoothed dynamic) graphs we use the inclusion-edgclusion idea (which appeared in [21], and earlier in [4,7]).

We start with a p-smoothed sequence of edge flips in the P_3 -partite graph \mathcal{G} , with node set $V = \{s\} \cup A \cup B \cup \{t\}$. The dynamic graph \mathcal{G} consists of an initial graph G_0 chosen uniformly at random with edges only from R_{P_3} , and a sequence e_1, \ldots, e_T of T updates in the form of edges from R_{P_3} being flipped.

From this, we define 16 (highly correlated) dynamic graphs $\mathcal{G}^{(0)}, \ldots, \mathcal{G}^{(15)}$, each of which being a p'-smoothed dynamic graph with the same node set V. The 16 initial graphs are constructed by randomly partitioning each of the 4 sets of exterior edges (sB, At, AA, BB) into two subsets, and allocate these subsets to the $16 = 2^4$ graphs such that each combination of subsets occurs once. The 16 dynamic graph are composed of these 16 different initial graphs, while their sequences of changes are identical.

Formally, the reduction does the following:

Algorithm 2: Algorithm P3TOGENERAL for counting paths in a general graph by counting them in P_3 -partite graphs

```
1 Partition sB into sB = E_{sB}^1 \sqcup E_{sB}^0, where each edge goes to one of the sets u.a.r.
2 Similarly, partition At, AA, BB into E_{At}^1, E_{AA}^1, E_{BB}^1 and their complements E_{At}^0, E_{AA}^0, E_{BB}^0
                                                // partition each set of external edges into two
3 Initialize E' = \emptyset, and w.p. 1/2 assign E' \leftarrow \{s, t\}
4 for i \leftarrow 0, \dots, 15 do
      Let b_{i1}, b_{i2}, b_{i3}, b_{i4} be the binary representation of i
E_0^{(i)} \leftarrow E_0 \sqcup E_{sB}^{b_{i1}} \sqcup E_{At}^{b_{i2}} \sqcup E_{AA}^{b_{i3}} \sqcup E_{BB}^{b_{i4}} \sqcup E'
                                                                  // define the 16 initial graphs
5 Initialize c \leftarrow 1, c' \leftarrow 1.
6 while c \leq T do
       if next operation is query then query all graphs, compute the output and continue
                   // compute output as described in Eq. (5) and end the current loop
         iteration
       Flip a coin C \sim Bin(\alpha)
                                                                                                        // \alpha as above
       if C = 1 then f \leftarrow e_c
       else draw f uniformly from \overline{R_H}
       for i \leftarrow 0, \dots, 15 do
        update E_{c'}^{(i)} \leftarrow E_{c'-1}^{(i)} \triangle \{f\}
       Update counters: c' \leftarrow c' + 1, and if C = 1 then c \leftarrow c + 1
```

Let T' be the number of edge flips in the dynamic graph $\mathcal{G}^{(i)}$ (the last value of c'), which is identical for all i. Since $\alpha \geq 1/2$, by a simple Chernoff bound the probability that $T' \geq 3T$ is at most $e^{-\Omega(T)}$. This completes the construction of the dynamic graphs $\mathcal{G}^{(0)}, \ldots, \mathcal{G}^{(15)}$.

6.2.3 Analysis

We now show that the reduction indeed creates p'-smoothed dynamic (general) graphs, and that by counting s-t 3-paths on them we can deduce the number of s-t 3-paths in the p-smoothed P_3 -partite dynamic graph \mathcal{G} , thus proving Lemma 6.7.

Claim 6.8. For each $i \in \{0,\ldots,15\}$, the dynamic graph $\mathcal{G}^{(i)} = (G_0^{(i)},\ldots,G_{T'}^{(i)})$ is a p'-smoothed dynamic graph.

Proof. Fix $i \in \{0, ..., 15\}$. The probability of any node pair (x, y) to be an edge in $G_0^{(i)}$ is exactly 1/2. Indeed, if (x, y) is interior (of types sA, AB, Bt) then it is taken from G_0 , where it is chosen w.p. 1/2. If (x,y) is an exterior edges, then it is either (s,t) which is taken w.p. 1/2 (see the choice of E'). Otherwise, it is of one of four types sB, At, AA, BB. Say (x, y) is of type sB (similar argument works for At, AA, BB), then it has probability exactly 1/2 to be chosen in $E_{sB}^{b_{i,1}}$ (either if $b_{i,1} = 0$ or $b_{i,1} = 1$). Overall, any $(x, y) \in \binom{V}{2}$ is chosen to G_0^i w.p. 1/2. By Observation 1, this is a p'-smoothed initial graph, for an adversary that picks a random initial graph.

Finally, by our choice of α and p', the edge f is drawn in a p'-smoothed manner, as seen before.

The following definition will be useful.

Definition 6 (properly partitioning multiple edge set). Let F_1, \ldots, F_m be m pairwise disjoint edge

sets on the same node set V, and $F = \bigsqcup_{i=1}^m F_i$. A set of 2^m graphs $G^{(0)}, \ldots, G^{(2^m-1)}$, with $G^{(i)} = (V, E^{(i)})$, is said to properly partition the edge sets F_1, \ldots, F_m if there exist m partitions $F_\ell = F'_\ell \sqcup F''_\ell$ such that:

• Each graph chooses a part of each edge set:

$$\forall i \in \{0, \dots, 2^m - 1\}, \ell \in [m] : E^{(i)} \cap F_{\ell} \in \{F'_{\ell}, F''_{\ell}\}\$$

• Each graph chooses a different combination of parts:

$$\forall i \neq j \in \{0, \dots, 2^m - 1\}, F \in \{F_0, \dots, F_{15}\} : E^{(i)} \cap F \neq E^{(j)} \cap F$$

In the context of P_3 -partite graphs, m=4, and each F_i corresponds to an exterior edge type (e.g., $F_1 = \{s\} \times B$). If the conditions above hold for a set of 16 graphs, we say they properly partition the exterior edges.

The next claim shows that the 16 graphs are correlated at all times.

Claim 6.9. For any $c' \in [T']$, the 16 graphs $G_{c'}^{(0)}, \ldots, G_{c'}^{(15)}$ properly partition the exterior edge types $F_1 = \{s\} \times B, F_2 = A \times \{t\}, F_3 = \binom{A}{2}, F_4 = \binom{B}{2}$.

Proof. We prove the claim by induction. At time c'=0, this holds by definition, with F'_0 and F''_0 being the randomly chosen sets E_{sB}^1, E_{sB}^0 , and similarly for all other exterior edge types.

For the step, assume that at time c' the graphs $G_{c'}^{(0)}, \ldots, G_{c'}^{(15)}$ properly partition the exterior edge types using $\{F'_j, F''_j\}_{j=1}^4$. If at time c'+1 an interior edge f was updated (that is, $f=e_c$ for some $c \in [T]$), then the same partition trivially works.

If at time c'+1 an exterior edge f was updated, then $f \in F_{\ell}$ for some $\ell \in [4]$. Recall that f is flipped in all 16 graphs. Take the partition where $(F'_{\ell}, F''_{\ell}) \leftarrow (F'_{\ell}(\text{old}) \triangle \{f\}, F''_{\ell}(\text{old}) \triangle \{f\})$ and all other (F_i', F_i'') for $i \neq \ell$ remain unchanged.

Formally, the first condition holds for the edge set F_{ℓ} since

$$\forall i: \ E_{c'+1}^{(i)} \cap F_{\ell} = \left(E_{c'}^{(i)} \cap F_{\ell} \right) \triangle \{f\} \in \{F_{\ell}'(\text{old}) \triangle \{f\}, F_{\ell}''(\text{old}) \triangle \{f\}\} = \{F_{\ell}', F_{\ell}''\},$$

where the containment is due to induction hypothesis. Similarly, the second condition holds trivially for each partition (F'_i, F''_i) that was left unchanged, and

$$\forall i \neq j: \ E_{c'+1}^{(i)} \cap F = \left(E_{c'}^{(i)} \cap F_{\ell} \right) \triangle \{f\} \neq \left(E_{c'}^{(j)} \cap F_{\ell} \right) \triangle \{f\} = E_{c'+1}^{(j)} \cap F$$

for (F'_{ℓ}, F''_{ℓ}) that was changed. The inequality uses the induction hypothesis and the fact that $S \mapsto S \triangle \{f\}$ is a bijection.

Thus,
$$G_{c'+1}^{(0)}, \ldots, G_{c'+1}^{(15)}$$
 properly partition F_1, F_2, F_3, F_4 , with partitions $F_\ell = F'_\ell \sqcup F''_\ell$.

Claim 6.10. Fix $c \in [T]$ and $c' \in [T']$ such that in $G_{c'}^{(i)}$ (for all i) the interior edge change e_c already occurred while the change e_{c+1} did not. For each choice of $X, Y \in \{A, B\}$ (where X and Y may be identical), denote by C_{sXYt}^i the count of all s-t 3-paths of type sXYt in graph $G_{c'}^{(i)}$, and by C the number of paths of type sABt in the original graph G_c (recall it has no s-t 3-paths of other types). Then

$$\sum_{i=0}^{15} C_{sABt}^{i} = 16 \cdot C; \qquad \sum_{i=0}^{15} C_{sBAt}^{i} = 4 \cdot |E_{c} \cap (A \times B)|$$

$$\sum_{i=0}^{15} C_{sAAt}^{i} = 4(n-1) \cdot |E_{c} \cap (\{s\} \times A)|; \qquad \sum_{i=0}^{15} C_{sBBt}^{i} = 4(n-1) \cdot |E_{c} \cap (B \times \{t\})|$$

Proof. The proof takes into account the different path types.

Paths of type sABt. Note that the interior edges are the same for all 16 graphs as well as the original graph. Paths of type sABt consist only of interior edges, and therefore appear in all 16 graphs as well as the original graph, which proves the first equality.

The other equalities use paths with exterior edges as well. Due to Lemma 6.9, the 16 graphs properly partition the exterior edge with $\{(F'_{\ell}, F''_{\ell})\}_{\ell \in [4]}$.

Paths of type sBAt. For each choice of $i \in \{0, ..., 15\}, a \in A, b \in B$, denote by $\mathbb{1}_{a,b}^i$ the event that $\{(s,b),(b,a),(a,t)\} \subseteq E_c^{(i)}$. Switching the order of summation we have:

$$\sum_{i=0}^{15} C^i_{sBAt} = \sum_{i=0}^{15} \sum_{\substack{a \in A \\ b \in B}} \mathbb{1}^i_{a,b} = \sum_{\substack{a \in A \\ b \in B}} \sum_{i=0}^{15} \mathbb{1}^i_{a,b}.$$

Any interior edge $(x,y) \in E_c$ in the original graph at time c also exists in all other graphs at time c'. For each choice where $(x,y) \in E_c$, we add 1 to the count if and only if (s,b) and (a,t) are both in graph $G_{c'}^{(i)}$. Since (s,b) and (a,t) each belong to a specific part (either F'_{ℓ} or F''_{ℓ} for some ℓ), there is a unique combination for both, and exactly 4 graphs of the 16 are aligned with it. Thus

$$\sum_{\substack{a \in A \\ b \in B}} \sum_{i=0}^{15} \mathbb{1}_{a,b}^{i} = \sum_{\substack{a \in A \\ b \in B}} 4 \cdot \mathbb{1}_{(a,b) \in E_c} = 4 \cdot |E_c \cap (A \times B)|,$$

which proves the second equality.

Paths of type sAAt. For any $i \in \{0, ..., 15\}$ and $a, a' \in A$ $a \neq a'$, denote by $\mathbb{1}_{a,a'}^i$ the event that $\{(s, a), (a, a'), (a', t)\} \subseteq E^{(i)}$. Switching the order of summation we have:

$$\sum_{i=0}^{15} C^i_{sAAt} = \sum_{i=0}^{15} \sum_{\substack{a \in A \\ a \neq a' \in A}} \mathbb{1}^i_{a,a'} = \sum_{\substack{a \in A \\ a \neq a' \in A}} \sum_{i=0}^{15} \mathbb{1}^i_{a,a'}.$$

Here, the interior edges are (s, a) of type sA, which appear in all graphs. Each choice $a \in A$ such that $(s, a) \in E_t$, and any choice $a' \in A \setminus \{a\}$ induce two exterior edges (a, a') and (a', t) that belong to certain parts, causing a unique combination that aligns with exactly 4 graphs. Thus

$$\sum_{\substack{a \in A \\ a \neq a' \in A}} \sum_{i=0}^{15} \mathbb{1}_{a,a'}^{i} = \sum_{\substack{a \in A \\ a \neq a' \in A}} 4 \cdot \mathbb{1}_{(s,a) \in E_c} = \sum_{a \in A} 4(n-1) \cdot \mathbb{1}_{(s,a) \in E_c} = 4(n-1) \cdot |E_c \cap (\{s\} \times A)|,$$

proving the third equality. The fourth equality is analogous.

The above claim allows us to easily deduce Lemma 6.7. By maintaining the 16 p'-smoothed general graphs and making queries to the number C^i of s-t 3-paths in each graph $G_{c'}^{(i)}$ at time c', we can find the total number

$$C_{\text{all}} = \sum_{i=0}^{15} C^i$$

of s-t 3-paths in these graphs (with repetitions). Note that this number is also the sum of the sums above, that is,

$$C_{\text{all}} = \sum_{i=0}^{15} \left(C_{sABt}^{i} + C_{sBAt}^{i} + C_{sAAt}^{i} + C_{sBBt}^{i} \right).$$

The addends from the above claim are all easy to compute, except for C. Define 3 counters

$$C_{AB} = |E_c \cap (A \times B)|; \quad C_{sA} = |E_c \cap (\{s\} \times A)|; \quad C_{Bt} = |E_c \cap (B \times \{t\})|$$

and update them whenever an edge of AB, sA or Bt is added or removed. The sum of the above equalities thus gives

$$\sum_{i=0}^{15} C^i = 16 \cdot C + 4 \cdot |E_c \cap (A \times B)| + 4(n-1) \cdot |E_c \cap (\{s\} \times A)| + 4(n-1) \cdot |E_c \cap (B \times \{t\})|$$

which simplifies to

$$\sum_{i=0}^{15} C^i = 16 \cdot C + 4 \cdot C_{AB} + 4(n-1) \cdot C_{sA} + 4(n-1) \cdot C_{Bt}$$
 (5)

and it is trivial to compute the number C of s-t 3-paths in G_c , as desired.

6.3 Putting it all together

Our main theorem is derived by Lemma 1.1, Lemma 6.1 and Lemma 6.7.

We remark that for ease of presentation throughout the proof we used n for dimension of the algebraic objects (matrix M and vectors u_i, v_i), and $n' = 2n + 2 = \Theta(n)$ for the number of nodes in the graphs. The polynomial factors can use either. For the sake of clarity, in the statement below we revert back to n being the number of nodes in the input dynamic graph.

Theorem 6.11. Fix 0 . Conditioned on the OMv conjecture, any dynamic graph algorithm for counting s-t 3-paths with error probability at most <math>1/1600 on p-smoothed dynamic graphs, with preprocessing, update and query times $t_{\text{Dre}}(n), t_{\text{u}}(n), t_{\text{g}}(n)$, must satisfy

$$t_{\text{pre}}(n) + \frac{n \log n}{p} \cdot t_{\text{u}}(n) + n \cdot t_{\text{q}}(n) + \frac{n^2 \log n \log(n/p)}{p} = \Omega(n^{3-\epsilon}),$$

for any $\epsilon > 0$. Specifically, no algorithm for counting s-t 3-paths can simultaneously have $t_{\rm pre} = O(n^{3-\epsilon})$, $t_{\rm u} = O(pn^{1-\epsilon})$, and $t_{\rm q} = O(n^{2-\epsilon})$, for any $\epsilon > 0$.

Proof. For any small $\epsilon > 0$, we note the inequality holds trivially for $p = O(n^{\epsilon-1})$ (indeed, the rightmost term on the l.h.s. exceeds the r.h.s. on its own right). We may therefore assume that $p = \omega(n^{\epsilon-1})$ for some arbitrarily small ϵ .

Assume a dynamic graph algorithm exists with the mentioned running times, then by Lemma 6.7 there exists an algorithm that counts sABt paths on a P_3 -partite p'-smoothed dynamic graph. We have $p' = \Theta(p)$ which leaves the same polynomial dependency.

By Lemma 6.1, the average-case parity OuMv problems can be solved with running time as stated.

Finally, by Lemma 1.1, this running time must exceed $\Omega(n^{3-\epsilon})$, the r.h.s in the statement of the theorem.

For the conclusion at the end, note that all logarithmic factors cancel out by the conjecture, as $\log n = o(n^{\epsilon})$ for any fixed $\epsilon > 0$.

6.4 Lower bounds for other counting problems

Our lower bound method can be altered in order to attain lower bounds for several other subgraph-counting problems. Our focus naturally lies with problems that present a gap between average-case and worst-case complexities (see [21]). Along with the results in Section 4, we obtain near tight bounds for all these problems (up to an ϵ in the exponent that stems from the OMv conjecture).

Theorem 6.12. Fix 0 . Conditioned on the OMv conjecture, any dynamic graph algorithm for counting either (a) s-t 4-paths; (b) 3-cycles through s; or (c) 4-cycles through s, with error probability at most <math>1/1600 on p-smoothed dynamic graphs, with preprocessing, update and query times $t_{pre}(n), t_u(n), t_q(n)$, must satisfy the equation from Theorem 6.11.

Specifically, no algorithm for any of the above problems can simultaneously have $t_{pre} = O(n^{3-\epsilon})$, $t_{\rm u} = O(pn^{1-\epsilon})$, and $t_{\rm q} = O(n^{2-\epsilon})$, for any $\epsilon > 0$.

Proof. The proof for s-4-cycles is by reduction to counting s-t 3-paths, which is detailed next. For counting s-3-cycles and or s-t 4-paths, the proof goes by alternating the proof of Theorem 6.11. We refrain from presenting the full proofs again (which will require repeating sections 6.1, 6.2 and 6.3 with minimal adjustments) and instead only present the necessary changes.

Counting 4-cycles through s. Here, we get the lower bound directly from Theorem 6.11, be showing how to use an algorithm for counting 4-cycles through s in order to count s-t 3-paths.

Given a dynamic graph with two designate nodes s, t, create two copies of it, one with the edge (s,t) and one without this edge; apply all the changes on both graphs throughout the execution, including flipping the edge (s,t) in both if it is flipped in the original dynamic graph. Assuming an algorithm for counting s-t 3-paths, apply it to both graphs, and whenever there is a query (for the number of s-t 3-paths) regarding the original graph, make queries (for the number of 4-cycles through s) to the algorithm to both copies, and return the difference.

To see the answer is correct, note that any 4-cycles through s that does not go through the edge (s,t) appears in both graphs and hence is subtracted, while any such 4-cycle that uses (s,t) appears only on the graph where the edge (s,t) exists, and the number of such paths is the difference returned by the reduction. Finally, note that there is a trivial one-to-one correspondence between 4-cycles through s in which (s,t) appears in the new graph that contains (s,t), and s-t 3-paths in the original graph. Hence, the reduction returns the correct value.

To conclude, note that the model (and p) are exactly the same, i.e. the simulation of the original p-smoothed dynamic graph indeed creates two p-smoothed dynamic graphs with the same number of nodes and the same parameter p.

Counting s-t 4-paths. A straightforward attempt to reduce this problem to counting s-t 3-paths is by adding a single node t' that is connected only to t, and counting s-t' 4-paths. While this reduction would easily go through in the worst-case setting, in the smoothed case it will not: random edges might connect t' to other nodes in the graph, breaking the tight connection between the numbers of s-t 3-paths and s-t' 4-paths.

Instead, one can perform the reduction at an earlier stage, and obtain a variant of Lemma 6.1, where no edges from t' are allowed, except for (t',t). Formally, this will imply a lower bound for the restricted case of P_4 -partite graphs. Then one must apply the inclusion-edgeclusion technique, similar to Lemma 6.7, invoking a larger multiplicative error factor due to the added parts in the graph partition (and potential edge types to exclude).

Counting 3-cycles through s. Here again one should focus on the two parts and fix each one of them separately. For the parallel of Lemma 6.1, note the graph only has 3 parts: s, A, B. We wish to count triangles sABs, once each. Such a count would represent the OuMv multiplication just as in the case of 3-paths (as we only use one orientation).

Lastly, the parallel to Lemma 6.7 would need to exclude counts of sAAs and sBBs. Such counts would simply add up to $\binom{\deg_A(s)}{2} + \binom{\deg_B(s)}{2}$, where $\deg_A(s)$ and $\deg_B(s)$ represent the number of neighbors s has in A and in B, respectively. The degrees can be easily maintained and counted out by the algorithm. The error factor is even smaller in this case, as there are less parts in the partition of V.

Appendix

A Tools from probability theory

We use the following versions of Chernoff/Hoeffding bounds:

Fact 1. For independent random variables X_1, \ldots, X_n with sum $X = \sum_{i \in [n]} X_i$ and expectation $\mathbb{E}[X] = \mu$, the following hold:

• If all X_i are supported on [a,b], then for any t > 0:

$$\Pr[X \ge \mu + t] \le \exp\left(\frac{-2t^2}{n(b-a)^2}\right).$$

• If all X_i are Bernoulli variables, then for any $0 < \delta < 1$:

$$\Pr[X \le (1 - \delta)\mu] \le \exp\left(\frac{-\delta^2 \mu}{2}\right).$$

We extensively use the Poisson distribution, and the Poissonization technique. The Poisson distribution is supported on $\mathbb{N}_{\geq 0}$ and parameterized by a single parameter λ , where the probability of the outcome to be $k \in \mathbb{N}_{\geq 0}$ is defined by:

$$\Pr_{x \sim \text{Pois}(\lambda)}[x = k] := \frac{\lambda^k \cdot e^{-\lambda}}{k!}.$$

In our proofs we extensively use the following well-known facts about the Poisson distribution (see, e.g., [32, Section 8.4]).

Fact 2.

1. Additivity of Poisson: for two parameters λ_1, λ_2 , it holds that:

$$Pois(\lambda_1 + \lambda_2) \sim Pois(\lambda_1) + Pois(\lambda_2).$$

2. Parity of Poisson: for any parameter $\lambda > 0$, and random variable $R \sim \text{Pois}(\lambda)$, we have

$$\Pr[R \equiv_2 0] = \frac{1 + e^{-2\lambda}}{2}.$$

- 3. "Poissonization": consider a set of Pois(t) samples from a distribution \mathcal{D} with support [n], and denote by H(i) the number of times element i was sampled. Then H(i) distributes according to Pois($\mathcal{D}(i) \cdot t$), and $H(1), \ldots, H(n)$ are independent from one another.
- 4. Concentration: let $z \sim \text{Pois}(\lambda)$. For any $a < \lambda$ and $b > \lambda$, we have:

$$\Pr[z < a] \le \frac{(e\lambda)^a e^{-\lambda}}{a^a} \quad \Pr[z > b] \le \frac{(e\lambda)^b e^{-\lambda}}{b^b}.$$

5. Efficient sampling (e.g., [26]): a value from Pois(t) can be generated with running time O(k) where k is the eventual output value (recall $\mathbb{E}[k] = t$).

Finally, to compare random variables (or distributions), we use the well-known notion of statistical distance, which equals exactly half the ℓ_1 distance (also known as total variation distance). We write P(x) for the probability of element x in P. For two distributions P, Q over the same domain \mathcal{X} the statistical distance between them is defined by

$$\mathrm{SD}(P,Q) := \frac{1}{2} \sum_{X \in \mathcal{X}} |P(X) - Q(X)|.$$

This is simply the ℓ_1 distance of the two distributions laid out as probability vectors. We use the following known facts about statistical distance (see, e.g., [6]):

Fact 3.

1. Sub-additivity of statistical distance for product distributions: given distribution P_i, Q_i supported on \mathcal{X}_i for i = 1, ..., k, the product distributions satisfy:

$$SD(P_1 \times \cdots \times P_k , Q_1 \times \cdots \times Q_k) \le \sum_{i=1}^k SD(P_i, Q_i).$$

2. An equivalent definition of the statistical distance over finite spaces uses the best distinguisher function $D: \mathcal{X} \to \{0,1\}$. That is

$$SD(P,Q) = \max_{D} \left| \Pr_{x \sim P}[D(x) = 1] - \Pr_{x \sim Q}[D(x) = 1] \right|.$$

This is in particular useful when P,Q are distributions over inputs to some algorithm A: one can define the distinguisher E that outputs 1 if A erred on the input X (note that A can be a randomized algorithm, making E a randomized distinguisher). Thus, when P and Q are statistically close, so is the probability that A errs on them. Indeed, by the equality above

$$\left| \Pr_{x \sim P} [E(x) = 1] - \Pr_{x \sim Q} [E(x) = 1] \right| \le \operatorname{SD}(P, Q).$$

B Massaging the OMv problem

Most of our lower bounds are conditioned no the OMv conjecture which can be seen as a variant of the informal claim "there is no $O(n^{3-\epsilon})$ combinatorial matrix multiplication algorithm". See [20] for further discussion of this connection.

Definition 7 (the OMv problem). The OMv problem is the following online problem. Fix an integer n. The initial inputs are a Boolean n-vector v_0 and a Boolean $n \times n$ matrix M. Then, Boolean vectors v_i of dimension n arrive online, for $i = 1, \ldots, n$. An algorithm solves the OMv problem correctly if it outputs the Boolean vector Mv_i before the arrival of v_{i+1} , for $i = 0, 1, \ldots, n-1$, and outputs Mv_n .

Definition 8 (the OuMv problem). The OuMv problem is the following online problem. Fix an integer n. The initial inputs are two Boolean n-vectors u_0, v_0 , and a Boolean $n \times n$ matrix M. Then, pairs of Boolean vectors (u_i, v_i) of dimension n arrive online, for $i = 1, \ldots, n$. An algorithm solves the OuMv problem correctly if it outputs the Boolean value $u_i^T M v_i$ before the arrival of (u_{i+1}, v_{i+1}) , for $i = 0, 1, \ldots, n-1$, and outputs $u_n^T M v_n$.

Conjecture 1 (OMv conjecture [20, Conjecture 1.1]). For any constant $\epsilon > 0$, there is no $O(n^{3-\epsilon})$ -time algorithm that solves OMv with error probability at most 1/3 in the word-RAM model with $O(\log n)$ -bit words.

This conjecture is a standard starting point of many lower bounds for dynamic graph algorithms since its introduction. It is also common to reduce this conjecture to other conjectures that are easier to work with, most notably the OuMv conjecture. We follow the footsteps of [21] and make further reductions, reaching the parity OuMv problem:

Definition 2 (parity OuMv problem). The parity OuMv problem is the following online problem. Fix an integer n. The initial inputs are two Boolean n-vectors u_0, v_0 , and a Boolean $n \times n$ matrix M. Then, pairs of Boolean vectors (u_i, v_i) of dimension n arrive online, for $i = 1, \ldots, n$. An algorithm solves the parity OuMv problem correctly if it outputs the Boolean value $u_i^T M v_i$ (over \mathbb{F}_2) before the arrival of (u_{i+1}, v_{i+1}) , for $i = 0, 1, \ldots, n-1$, and outputs $u_n^T M v_n$.

An algorithm solves the average-case parity OuMv problem if it answers correctly, with error probability at most 1/20, over inputs that are chosen independent and uniformly at random (that is, all entries of the matrix M and the vectors u_i, v_i are i.i.d. random bits).

The following result is established in [21].

Lemma 1.1 ([21, Section 2]). For any constant $\epsilon > 0$, there is no $O(n^{3-\epsilon})$ -time algorithm that solves the average-case parity OuMv problem, unless the OMv conjecture fails.

We overview the proof here for completeness. It consists of two reductions.

Reduction #1: existence to parity. The first reduction works on worst-case inputs, and replaces the matrix-vector Boolean multiplication (using operators (\vee, \wedge)) by a multiplication over the field \mathbb{F}_2 (using $(+, \cdot)$), i.e., it asks to output $uMv \mod 2$. In graph notations, we replace the question "is there a path" with the question "is the number of paths even or odd"?

The reduction is done by replacing some 1 entries in M with 0. Such replacement occurs at each entry independently with probability 1/2, and the outcome of this process is a random matrix M'. The core of the proof is in the following insights.

- If $u^T M v = 0$ over \mathbb{Z} , then $u^T M' v = 0$ as well
- If $u^T M v = \ell > 0$, then $u^T M' v$ is even or odd with equal probability.

To see the second item, consider an entry $M_{ij}=1$ that affects the value u^TMv (that is, $u_i=v_j=1$). First draw all other entries of M', to obtain temporarily M'' with $M''_{ij}=1$ and integer value $a=u^TM''v$. Still, $M'_{ij}\sim Ber(1/2)$, and thus the value of $u^TM'v$ will be a or a-1 with equal probability. Overall, we get that the parity of $u^TM'v$ is even or odd with equal probability.

Formally, the reduction gets u, M, v and runs the process above independently for $\Theta((\log n)$ times, using a solver for parity OuMv for each resulting matrix M'. If at least once the answer was 1 ('odd'), it returns 1 for the existence question. Otherwise it returns 0.

Note that in fact, matrix multiplication is usually studied over a field and not using Boolean operations. It is thus only natural to conjecture a variant of the OMv conjecture over \mathbb{F}_2 . A reduction of this conjecture to OuMv would give exactly the parity OuMv conjecture discussed here, without using randomization. We nevertheless stick to the standard OMv conjecture.

Reduction #2: worst-case parity to average-case parity. For the question of existence (using operators (\vee, \wedge)), the average-case is much easier than the worst case: the answer is 1 w.h.p. Interestingly, this is not at all the case for the parity question (that is, using $(+, \cdot)$ over the field \mathbb{F}_2). We show that the parity version of OuMv discussed above is hard also on average, establishing Lemma 1.1.

The proof follows an elegant idea (that dates back to [3]): given a Boolean matrix M, draw a uniformly random Boolean matrix M^1 , and define another matrix M^2 by $M^2 = M + M^1$ (over \mathbb{F}_2); note that M^2 is also a random matrix. Thus we can write $M = M^1 + M^2$ where both M^1 and M^2 are uniformly random. Similarly, split each vector u_i (or v_i) as $u_i^1 + u_i^2$ where both are uniformly random. By distributivity of matrix multiplication over \mathbb{F}_2 , we can write:

$$u_i^T M v_i = \sum_{b_1, b_2, b_3 \in \{0,1\}} (u_i^{b_1})^T M^{b_2} v_i^{b_3}.$$

Each of the eight addends on the right is a multiplication of uniformly random Boolean matrix and vectors.

Formally, assume we have an algorithm solving parity OuMv in the average-case, with error ϵ . An algorithm for the worst-case gets an arbitrary input, and does the following: split the matrix (and vectors) randomly as in the above equation, and apply the average-case algorithm on all 8 parts. It can only fail if (at least) one of the 8 average-case executions failed, which by union bound occurs with probability at most 8ϵ .

References

- [1] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*, pages 434–443. IEEE Computer Society, 2014.
- [2] David Alberts and Monika Rauch Henzinger. Average-case analysis of dynamic graph algorithms. *Algorithmica*, 20(1):31–60, 1998.
- [3] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of computer and system sciences*, 47(3):549–595, 1993.
- [4] Enric Boix-Adserà, Matthew S. Brennan, and Guy Bresler. The average-case complexity of counting cliques in erdős-rényi hypergraphs. In *FOCS*, pages 1256–1280. IEEE Computer Society, 2019.
- [5] Raphaël Clifford, Allan Grønlund, and Kasper Green Larsen. New unconditional hardness results for dynamic and online problems. In FOCS, pages 1089–1107. IEEE Computer Society, 2015.
- [6] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, 2001.
- [7] Mina Dalirrooyfard, Andrea Lincoln, and Virginia Vassilevska Williams. New techniques for proving fine-grained average-case hardness. In *FOCS*, pages 774–785. IEEE, 2020.
- [8] Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. volume 51, pages 968–992, 2004.
- [9] Michael Dinitz, Jeremy Fineman, Seth Gilbert, and Calvin Newport. Smoothed analysis of information spreading in dynamic networks. In 36th International Symposium on Distributed Computing, page 1, 2022.

- [10] Michael Dinitz, Jeremy T Fineman, Seth Gilbert, and Calvin Newport. Smoothed analysis of dynamic networks. *Distributed Computing*, 31(4):273–287, 2018.
- [11] Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. SIAM J. Comput., 29(5):1740–1759, 2000.
- [12] Paul Erdős and Alfréd Rényi. On random graphs I. Publ. math. debrecen, 6(290-297):18, 1959.
- [13] Paul Erdős and Alfréd Rényi. On random matrices. Publ. Math. Inst. Hangar. Acad. Sci., 8:455–461, 1964.
- [14] Tobias Friedrich, Thomas Sauerwald, and Dan Vilenchik. Smoothed analysis of balancing networks. Random Structures & Algorithms, 39(1):115–138, 2011.
- [15] Seth Gilbert, Uri Meir, Ami Paz, and Gregory Schwartzman. On the complexity of load balancing in dynamic networks. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 254–264, 2021.
- [16] Nika Haghtalab, Tim Roughgarden, and Abhishek Shetty. Smoothed analysis of online and differentially private learning. Advances in Neural Information Processing Systems, 33:9203– 9215, 2020.
- [17] Nika Haghtalab, Tim Roughgarden, and Abhishek Shetty. Smoothed analysis with adaptive adversaries. In *FOCS*, pages 942–953. IEEE, 2021.
- [18] Kathrin Hanauer, Monika Henzinger, and Qi Cheng Hua. Fully dynamic four-vertex subgraph counting. In *SAND*, volume 221 of *LIPIcs*, pages 18:1–18:17. Schloss Dagstuhl Leibniz-Zentrum für Informatik, 2022.
- [19] Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms A quick reference guide. *ACM J. Exp. Algorithmics*, 27:1.11:1–1.11:45, 2022.
- [20] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In STOC, pages 21–30. ACM, 2015.
- [21] Monika Henzinger, Andrea Lincoln, and Barna Saha. The complexity of average-case dynamic subgraph counting. In *SODA*, pages 459–498. SIAM, 2022.
- [22] Shang-En Huang, Dawei Huang, Tsvi Kopelowitz, Seth Pettie, and Mikkel Thorup. Fully dynamic connectivity in $O(\log \log \log n)^2$) amortized expected time. Theoretics, 2, 2023.
- [23] Svante Janson, Tomasz Luczak, and Andrzej Rucinski. *Random graphs*. John Wiley & Sons, 2011.
- [24] 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.
- [25] Dénes Kőnig. Graphok és matrixok. Matematikai és Fizikai Lapok, 38:116–119, 1931.
- [26] Donald Ervin Knuth. The art of computer programming, volume 3. Pearson Education, 1997.

- [27] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In SODA, pages 1272–1287. SIAM, 2016.
- [28] Michael Krivelevich, Daniel Reichman, and Wojciech Samotij. Smoothed analysis on connected graphs. SIAM Journal on Discrete Mathematics, 29(3):1654–1669, 2015.
- [29] Michael Krivelevich, Benny Sudakov, and Prasad Tetali. On smoothed analysis in dense graphs and formulas. *Random Structures & Algorithms*, 29(2):180–193, 2006.
- [30] Kasper Green Larsen and Huacheng Yu. Super-logarithmic lower bounds for dynamic graph problems. In *FOCS*, pages 1589–1604. IEEE, 2023.
- [31] Uri Meir, Ami Paz, and Gregory Schwartzman. Models of smoothing in dynamic networks. In *DISC*, volume 179 of *LIPIcs*, pages 36:1–36:16. Schloss Dagstuhl Leibniz-Zentrum für Informatik, 2020.
- [32] Michael Mitzenmacher and Eli Upfal. Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis. Cambridge university press, 2017.
- [33] Sotiris E. Nikoletseas, John H. Reif, Paul G. Spirakis, and Moti Yung. Stochastic graphs have short memory: Fully dynamic connectivity in poly-log expected time. In *ICALP*, volume 944 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 1995.
- [34] Mihai Pătrașcu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. SIAM J. Comput., 35(4):932–963, 2006.
- [35] Mihai Puatracscu. Towards polynomial lower bounds for dynamic problems. In *STOC*, pages 603–610. ACM, 2010.
- [36] Mihai Puatracscu and Mikkel Thorup. Don't rush into a union: take time to find your roots. In *STOC*, pages 559–568. ACM, 2011.
- [37] Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011.
- [38] Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In *SODA*, pages 118–126. SIAM, 2007.
- [39] Gregory Schwartzman and Yuichi Sudo. Smoothed Analysis of Population Protocols. In 35th International Symposium on Distributed Computing (DISC 2021), volume 209 of Leibniz International Proceedings in Informatics (LIPIcs), pages 34:1–34:19. Schloss Dagstuhl Leibniz-Zentrum für Informatik, 2021.
- [40] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51(3):385–463, 2004.
- [41] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis: an attempt to explain the behavior of algorithms in practice. *Commun. ACM*, 52(10):76–84, 2009.