Smoothed Analysis of Dynamic Graph Algorithms

Uri Meir
Blavatnik School of Computer Science, Tel Aviv University, Israel

Ami Paz
LISN, CNRS & Paris-Saclay University, France

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.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Dynamic graph algorithms, Smoothed analysis, Shortest paths

Digital Object Identifier 10.4230/LIPIcs.ITCS.2026.102

Related Version *Full Version*: `https://arxiv.org/abs/2502.13007` [25]

Funding *Ami Paz*: Supported by ANR project DIDYA (ANR-25-CE48-0897).

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 [4, 24, 29, 31]; 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 [29]. Another celebrated line of work proves polynomial conditional lower bounds [1, 9, 16, 21, 30, 32]; 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 update[1] unless queries take roughly quadratic time [16], assuming the OMv conjecture (see Section 4). Similarly to other algorithmic models, it is natural to ask whether known results in worst-case analysis[2] 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 [28], 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 [17] 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 $\Omega\big(n^{1-\epsilon}\big)$ 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 [35, 36]. 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 probability [10, 22, 23]. 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 [7, 8, 26].

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

[2] In 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.

Table 1 : A robust versus a fragile lower bound. The table shows the update time complexities of s-t 3-paths and connectivity, with a non-adaptive flip adversary and constant $0 < p < 1$. The update times are for $t_{\mathrm{q}} = O(1)$ and $t_{\mathrm{pre}} = O(n^{3-\epsilon})$. The s-t 3-paths lower bounds are conditioned on the OMv conjecture.

Problem	Average case ($p = 0$)	Smoothed case ($0 < p < 1$ const.)	Worst case ($p = 1$)
s-t 3-paths	$O(1)$ [17]	$\Omega(n^{1-\epsilon})$, $O(n)$ (This work)	$\Omega(n^{1-\epsilon})$, $O(n)$ [16], [14]
Connectivity	$O(1)$ (This work)	$O(1)$ (This work)	$\Omega(\log n)$, $\tilde{O}(\log n)$ [29], [18]

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_{pre} it needs for processing the initial graph, the time t_{u} for processing an edge change, and the time t_{q} for answering a query.

We next define a p-smoothed input, for a parameter $0 \leq p \leq 1$, 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.

A smooth transition. A recent work on average-case complexity [17] 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 $\Omega\big(pn^{1-\epsilon}\big)$ 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 Figure 3. In Figure 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 the full version, we show that up to constant factors the average-case complexity of any problem is also independent of the operations' encoding.

Table 2 : Bridging the gaps between average-case and worst-case in small subgraph counting. The table shows the update complexities of short paths counting, for $t_q = O(1)$ and $t_{pre} = O(n^{3-\epsilon})$, as a function of p. For these problems the smoothed complexities with all adversaries coincide. The lower bounds are conditioned on the OMv conjecture.

Problem	Average case	Smoothed case	Worst case
s-t 2-paths	$O(1)$ [17]		
s-t 3-paths	$O(1)$ [17]	$\Omega(pn^{1-\epsilon})$, $O(pn)$ (This work)	$\Omega(n^{1-\epsilon})$, $O(n)$ [16], [14]
s-t 4-paths	$O(1)$ [17]	$\Omega(pn^{1-\epsilon})$, $O(pn)$ (This work)	$\Omega(n^{1-\epsilon})$, $O(n)$ [16]
s-t 5-paths	$\Omega(n^{1-\epsilon})$ [17]		

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*. That is, can an adversary disrupt the algorithm even more if it knows the noisy input that was actually fed to the algorithm in previous rounds, rather than only its intended input?[3] Similar distinctions were recently made in smoothed analysis of dynamic networks [26] and online learning [13].

In our case, at each round the *adaptive adversary* chooses a new edge to change, with the full knowledge of previous changes made, including the noisy ones, and thus the current state of the graph. 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.

[3] One could consider an even stronger, "fully adaptive" adversary, that also sees the inner state of the algorithm (which includes the noise from previous rounds). Trivially, all our lower bounds apply to such an adversary as well. As all our upper bounds for the adaptive model are deterministic, they also apply against such an adversary.

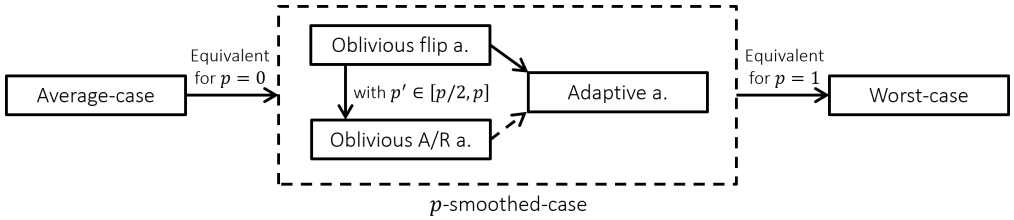


Figure 3 : Relations between models (Section 1.2 and Section 1.3.3). An arrow from model A to model B indicates that model B is at least as hard as model A (that is, the complexity of a problem can only increase). A solid arrow represents that this relation is strict, i.e., for some problem and range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

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 [29], 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.

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 the full version 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. 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.

Table 4 : New results for various problems in various models. The table shows the update times t_u of different problems assuming $t_q = O(1)$ and $t_{pre} = o(n^{3-\epsilon})$. A cell with no reference means the result is new. The worst-case upper bounds for s-t 4-paths and s-t 4-cycles are new as well. 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 $1 - n^{-\delta} \leq p \leq 1$. All the polynomial lower bounds are conditioned on the OMv conjecture.

Problem	Average case	Oblivious flip a.	Oblivious AR a.	Adaptive adv.	Worst case model
Small subgraph counting					
s-t 3-paths	$O(1)$ [17]	$\Omega(pn^{1-\epsilon})$, $O(pn)$			$\Omega(n^{1-\epsilon})$, $O(n)$ [16], [14]
s-t 4-paths	$O(1)$ [17]	$\Omega(pn^{1-\epsilon})$, $O(pn)$			$\Omega(n^{1-\epsilon})$, $O(n)$ [16]
s triangles	$O(1)$ [17]	$\Omega(pn^{1-\epsilon})$, $O(pn)$			$\Omega(n^{1-\epsilon})$, $O(n)$ [16]
s 4-cycles	$O(1)$ [17]	$\Omega(pn^{1-\epsilon})$, $O(pn)$			$\Omega(n^{1-\epsilon})$, $O(n)$ [17]
Other key problems					
Connectivity	$O(1)$		–	$\Omega(\log pn)$	$\Omega(\log n)$, $\tilde{O}(\log n)$ [29], [18]
Perfect matching	$O(1)$		$\Omega(n^{\delta/3-\epsilon})$	$\Omega(p^{2-\epsilon}n^{1-\epsilon})$	$\Omega(n^{1-\epsilon})$, $O(n^{1.495})$ [16], [33]

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 and prove lower bounds using a new embedding technique, a type of worst-case to smoothed-case reduction, 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 a polynomial lower bound on the update time of bipartite perfect matching.

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, but establishes 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 $t_u(n) = O(p^{2-\epsilon}n^{1-\epsilon})$ and $t_q(n) = O((pn)^{2-\epsilon})$ 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 $t_u(n) = O(\min\left\{\frac{1}{1-p}, n\right\}^{1/3-\epsilon})$ and $t_q(n) = O(\min\left\{\frac{1}{1-p}, n\right\}^{2/3-\epsilon})$.

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 [17], 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 (Figure 5a). In the full version we present simple algorithms (a la [17]) 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).

▶ **Theorem C** (informal). *There is an algorithm for counting **s-t 3-paths**, **s-t 4-paths**, **s triangles** and **s 4-cycles**, with $t_{\mathrm{u}} = O(pn + 1)$ (expected) and $t_{\mathrm{q}} = O(1)$ 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 [6] and s 4-cycles [14].

In the full version we complement these upper bounds by almost-matching conditional lower bounds on the update time (see Section 4 for an overview). 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 4.

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

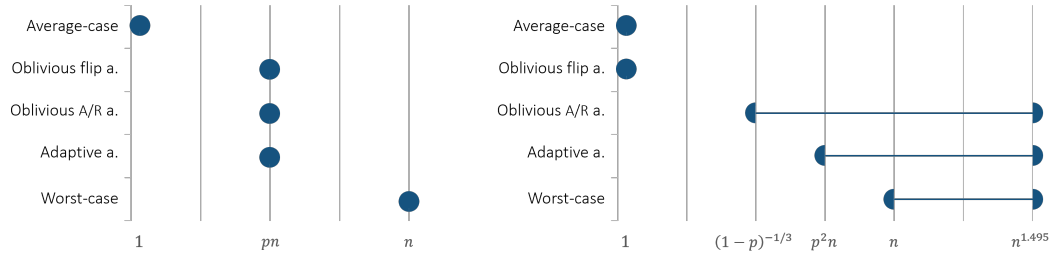
$$t_{\mathrm{pre}}(n) + \frac{n}{p} \cdot t_{\mathrm{u}}(n) + n \cdot t_{\mathrm{q}}(n) + \frac{n^2}{p} = \tilde{\Omega}(n^{3-\epsilon}),$$

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

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 $p = 0$, we show a hierarchy among the smoothing models for different regimes of $0 < p < 1$. The adaptive adversary can be defined with either operation type, and can disregard the additional 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 $p' = p/(2 - p) \in [p/2, p]$). This induces a (non-strict) hierarchy between the models (Figure 3). We next discuss separation results for these models.



(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 Figure 3. Separations between smoothed and non-smoothed models arise from Figure 5a using, e.g., $p = 1/\sqrt{n}$. Figure 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 ($p = 1 - 1/\operatorname{poly}(n)$).

For some problems, such as counting small subgraphs, the three smoothing models coincide with one another (Figure 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 $0 \leq p < n^{-\delta}$ and from the average case for $n^{-\delta} < p \leq 1$, for any constant $\delta > 0$.

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 (Figure 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, average-case, and adaptive smoothed model). Using our embedding technique, Theorem B gives a super-constant 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.

2 Related work

2.1 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 [15]. The most recent dynamic algorithm for connectivity is by Huang, Huang, Kopelowitz, Pettie and Thorup [18], and has $t_{\mathrm{u}} = O(\log n \log^2 \log n)$ amortized update time and $t_{\mathrm{q}} = O(\log n / \log \log \log n)$ time per query. Small subgraph counting was recently studied by Hanauer, Henzinger and Hua [14], who showed, e.g., that counting s-t 3-paths and s-triangles (triangles through a given node s) can be done with $t_{\mathrm{u}} = O(n)$ and constant query time. Kara, Nikolic, Ngo, Olteanu and

Zhang [19] studied databases that support enumeration of triangles in trinary relations, and their work implies triangle counting in dynamic graphs with $t_{\text{pre}} = O(m_0^{3/2}), t_{\text{u}} = O(m^{1/2})$, and $t_{\text{q}} = O(1)$.

Pătraşcu and Demaine [29] 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 $t_{\text{u}} = o(\log n)$ amortized time, a result that remains the best known to date. Some follow-up works [4, 24, 31] 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, 9, 21, 30, 32], which used several different conjectures, Henzinger, Krinninger, Nanongkai and Saranurak [16] 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, $t_{\text{u}} = O(n^{1-\epsilon})$ and $t_{\text{q}} = O(n^{2-\epsilon})$, 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.

2.2 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 [28]. 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 [17, 28]). 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 [17] 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 [16], and specifically lower bounds for average-case complexity in this setting [3,5]. 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 [17]; 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.

2.3 Smoothed analysis in different models

Smoothed analysis was introduced by Spielman and Teng [35] 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 [10, 22, 23] and dynamic networks [7, 8, 11, 26], 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 [34] and online learning [12, 13]. The effect of the adversary's adaptivity was first studied with regard to information spreading in dynamic networks [26], 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 [13], 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.

3 Preliminaries

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

3.1 Defining smoothed dynamic graphs

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 1** (p-smoothed dynamic graph, adaptive)**.** *A p-smoothed dynamic graph with an adaptive adversary is a dynamic graph $\mathcal{G} = (G_0, \ldots, G_T)$ 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 $e \in \binom{V}{2}$ 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 $\binom{V}{2}$. 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 2** (*p*-smoothed dynamic graph, oblivious)**.** *A p-smoothed dynamic graph with an oblivious adversary is a dynamic graph $\mathcal{G} = (G_0, \ldots, G_T)$ 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 $c_i \sim \mathrm{Ber}(p)$ 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 3.** *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 4, we consider a variant where the graph edges are restricted to a set $R_H \subseteq \binom{V}{2}$ 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 $\binom{V}{2}$.

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

▶ **Definition 4** ((H,Π)-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 $\mathcal{G}$ 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 4 we discuss P_3-partite graphs where $\mathcal{G}$ uses node set $V = \{s\} \bigsqcup A \bigsqcup B \bigsqcup \{t\}$. 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 $sABt$.

3.2 Properties of the Poisson distribution

Our proofs use probabilistic tools extensively. For the distance between two distribution P, Q over the same domain we use the well-known *statistical distance* (sometimes called total-variation distance). We make an extensive use of the *Poisson distribution* and more importantly the *Poissonization* technique (see, e.g., [27, Section 8.4]). We use $\mathrm{Pois}(t)$ for the Poisson distribution with parameter t and sometimes for a random variable drawn from this distribution. Some helpful properties of the Poisson distribution are summarized in the following fact.

▶ **Fact 5.**

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

$$\mathrm{Pois}(\lambda_1 + \lambda_2) \sim \mathrm{Pois}(\lambda_1) + \mathrm{Pois}(\lambda_2).$$

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

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

3. *"Poissonization": consider a set of $\mathrm{Pois}(t)$ 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 $\mathrm{Pois}(\mathcal{D}(i) \cdot t)$, and $H(1), \ldots, H(n)$ are independent from one another.*
4. *Concentration: let $z \sim \mathrm{Pois}(\lambda)$. For any $a < \lambda$ and $b > \lambda$, we have:*

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

5. *Efficient sampling (e.g., [20]): a value from $\mathrm{Pois}(t)$ can be generated with running time $O(k)$ where k is the eventual output value (recall $\mathbb{E}[k] = t$).*

4 A technical overview of the subgraph counting lower bounds

The most technically involved proofs are those of our tight lower bounds for counting small subgraphs. These apply in the weakest model (oblivious flip adversary) while the matching upper bounds are in the strongest model (adaptive adversary), thus resolving the smoothed complexity of these problems for all three smoothed input models.

As discussed before, we achieve some lower bounds using an embedding technique. However, for counting small subgraphs, these bounds are far from being tight, and a stronger technique is necessary. To obtain tight lower bounds, we instead employ a sequence of reductions, expanding on previous techniques used to show lower bounds for average-case complexities [3, 5, 17].

In a nutshell, [17] presents a sequence of reductions which solves the OMv problem (discussed below) by aggregating solutions from several executions of a dynamic counting algorithm, run on several different (well-correlated) uniformly random dynamic graphs. These instances are randomly generated from a single OMv input in a way that: (i) the noise is correlated and cancels out, allowing to solve the OMv problem; and (ii) each instance marginally resembles (in statistical distance) to a uniformly random input.

The outline of our lower bounds is similar, but condition (ii) above crucially changes: the instances produced by the reduction should marginally resemble p-smoothed inputs, rather than uniformly random (average-case) inputs. In contrast to the average case, proving a lower bound for the smoothed case requires a choice of a *specific adversarial strategy* that "imposes hardness".[4] Applying p-smoothing to this strategy yields a certain distribution over dynamic graphs that replaces the uniformly random input in condition (ii) above.

In addition to the above, the sequence of fine-grained reductions must be (iii) highly efficient. While this was a non-issue in the average case, it requires a careful treatment in our setting, where we apply conditional sampling from non-uniform distributions. In our argument we repeatedly use the *Poissonization* technique, described below, to control the dependencies between different instances, obtaining (i), (ii) and (iii). 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* $\mathrm{Pois}(\lambda)$ is supported on $\mathbb{N}_{\geq 0}$ and parameterized by its mean value λ (see [27, Section 8.4]). In a nutshell, $\mathrm{Pois}(\lambda)$ 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 Section 3.2.

The Poissonization technique. A particularly useful property of the Poisson distribution is known as *Poissonization* (Item 3 of Fact 5), 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 $\mathcal{H} : E \to \mathbb{N}_{\geq 0}$, 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 $\mathrm{Pois}(r \cdot \mathcal{D}(e_i))$, where $\mathcal{D}(e_i)$ represents the probability of e_i in $\mathcal{D}$. This yields a total of $t = \mathrm{Pois}(r)$ 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 [17] and correct inaccuracies. More crucially, Poissonization helps us overcome the additional challenges that arise due to the adversarial presence.

[4] Indeed, not all adversarial strategies would produce a nontrivial lower bound. For example, if the adversary chooses a uniformly random input, then after p-smoothing the input remains uniformly random – for which there are efficient algorithms for the problems we consider.

Sections 4.1, 4.2 and 4.3 together prove a lower bound for counting s-t 3-paths, which is extended to other problems in the full version of the paper. The bounds are conditioned on the OMv conjecture [16], which formalizes the common belief that a truly sub-cubic combinatorial algorithm for matrix multiplication does not exist.

4.1 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 [16] and is standard by now, and two reductions that were presented in [17].

▶ **Definition 6** (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.*

▶ **Conjecture 7** (OMv conjecture [16, 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 8** (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 9** ([17, 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 10. At this point, [17, 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.

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

We next 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 [17] 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:

$$u_i^T M v_i \equiv_2 u_i^T M_1 v_i + u_i^T M_2 v_i + u_i^T M_3 v_i. \tag{1}$$

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 whose strategy is not deterministic, but instead chooses an sA or Bt edge u.a.r. That is, each change is distributed on the graph edges by

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

After applying p-smoothing, each change has the following distribution over the graph edges

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

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 $\mathcal{D}^p_{\text{adv}}$, 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., $z_e \sim \text{Pois}(\mathcal{D}^p_{\text{adv}}(e) \cdot t)|_{z_e \equiv_2 u_{\text{dif}}(e)}$. 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 $\text{Pois}(t_{AB}/2)$ 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 $\text{Pois}(t)$ edge changes drawn from $\mathcal{D}^p_{\text{adv}}$. 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 11. 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 [17] 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 proves our claims, as well as theirs.

4.3 Step 3: From restricted graphs to general graphs

Finally, we show 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 P_3-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.

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. doi:10.1109/FOCS.2014.53.

2 David Alberts and Monika Rauch Henzinger. Average-case analysis of dynamic graph algorithms. *Algorithmica*, 20(1):31–60, 1998. doi:10.1007/PL00009186.

3 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. doi:10.1109/FOCS.2019.00078.

4 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. doi:10.1109/FOCS.2015.71.

5 Mina Dalirrooyfard, Andrea Lincoln, and Virginia Vassilevska Williams. New techniques for proving fine-grained average-case hardness. In *FOCS*, pages 774–785. IEEE, 2020. doi:10.1109/FOCS46700.2020.00077.

6 Camil Demetrescu and Giuseppe F Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM (JACM)*, 51(6):968–992, 2004. doi:10.1145/1039488.1039492.

7 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.

8 Michael Dinitz, Jeremy T Fineman, Seth Gilbert, and Calvin Newport. Smoothed analysis of dynamic networks. *Distributed Computing*, 31(4):273–287, 2018. doi:10.1007/S00446-017-0300-8.

9 Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM J. Comput.*, 29(5):1740–1759, 2000. doi:10.1137/S0097539797327908.

10 Tobias Friedrich, Thomas Sauerwald, and Dan Vilenchik. Smoothed analysis of balancing networks. *Random Structures & Algorithms*, 39(1):115–138, 2011. doi:10.1002/RSA.20341.

11 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. doi:10.1145/3409964.3461808.

12 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.

13 Nika Haghtalab, Tim Roughgarden, and Abhishek Shetty. Smoothed analysis with adaptive adversaries. In *FOCS*, pages 942–953. IEEE, 2021. doi:10.1109/FOCS52979.2021.00095.

14 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. doi:10.4230/LIPIcs.SAND.2022.18.

15 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. doi:10.1145/3555806.

16 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. doi:10.1145/2746539.2746609.

17 Monika Henzinger, Andrea Lincoln, and Barna Saha. The complexity of average-case dynamic subgraph counting. In *SODA*, pages 459–498. SIAM, 2022. doi:10.1137/1.9781611977073.23.

18 Shang-En Huang, Dawei Huang, Tsvi Kopelowitz, Seth Pettie, and Mikkel Thorup. Fully dynamic connectivity in $O(\log n(\log\log n)^2)$ amortized expected time. *TheoretiCS*, 2, 2023. doi:10.46298/THEORETICS.23.6.

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

20 Donald Ervin Knuth. *The art of computer programming*, volume 3. Pearson Education, 1997.

21 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *SODA*, pages 1272–1287. SIAM, 2016. doi:10.1137/1.9781611974331.CH89.

22 Michael Krivelevich, Daniel Reichman, and Wojciech Samotij. Smoothed analysis on connected graphs. *SIAM Journal on Discrete Mathematics*, 29(3):1654–1669, 2015. doi:10.1137/151002496.

23 Michael Krivelevich, Benny Sudakov, and Prasad Tetali. On smoothed analysis in dense graphs and formulas. *Random Structures & Algorithms*, 29(2):180–193, 2006. doi:10.1002/RSA.20097.

24 Kasper Green Larsen and Huacheng Yu. Super-logarithmic lower bounds for dynamic graph problems. In *FOCS*, pages 1589–1604. IEEE, 2023. doi:10.1109/FOCS57990.2023.00096.

25 Uri Meir and Ami Paz. Smoothed analysis of dynamic graph algorithms, 2025. doi:10.48550/arXiv.2502.13007.

26 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. doi:10.4230/LIPIcs.DISC.2020.36.

27 Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

28 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. doi:10.1007/3-540-60084-1_71.

29 Mihai Pătraşcu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM J. Comput.*, 35(4):932–963, 2006. doi:10.1137/S0097539705447256.

30 Mihai Puatracscu. Towards polynomial lower bounds for dynamic problems. In *STOC*, pages 603–610. ACM, 2010. doi:10.1145/1806689.1806772.

31 Mihai Puatracscu and Mikkel Thorup. Don't rush into a union: take time to find your roots. In *STOC*, pages 559–568. ACM, 2011. doi:10.1145/1993636.1993711.

32 Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011. doi:10.1007/S00453-010-9401-5.

33 Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In *SODA*, pages 118–126. SIAM, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283397.

34 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. doi:10.4230/LIPIcs.DISC.2021.34.

35 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. doi:10.1145/990308.990310.

36 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. doi:10.1145/1562764.1562785.