Dynamic Algorithms against an Adaptive Adversary: Generic Constructions and Lower Bounds

Amos Beimel*
Ben-Gurion University
Beer-Sheva, Israel
amos.beimel@gmail.com

Haim Kaplan†
Tel Aviv University
Google Research
Tel Aviv, Israel
haimk@tau.ac.il

Yishay Mansour‡
Tel Aviv University
Google Research
Tel Aviv, Israel
mansour.yishay@gmail.com

Kobbi Nissim§
Georgetown University
Washington, D.C., USA
kobbi.nissim@georgetown.edu

Thatchaphol Saranurak
University of Michigan
Ann Arbor, Michigan, USA
thsa@umich.edu

Uri Stemmer¶
Tel Aviv University
Google Research
Tel Aviv, Israel
u@uri.co.il

ABSTRACT

Given an input that undergoes a sequence of updates, a dynamic algorithm maintains a valid solution to some predefined problem at any point in time; the goal is to design an algorithm in which computing a solution to the updated input is done more efficiently than computing the solution from scratch. A dynamic algorithm against an adaptive adversary is required to be correct when the adversary chooses the next update after seeing the previous outputs of the algorithm. We obtain faster dynamic algorithms against an adaptive adversary and separation results between what is achievable in the oblivious vs. adaptive settings. To get these results we exploit techniques from differential privacy, cryptography, and adaptive data analysis. Our results are as follows.

We give a general reduction transforming a dynamic algorithm against an oblivious adversary to a dynamic algorithm robust against an adaptive adversary. This reduction maintains several copies of the oblivious algorithm and uses differential privacy to protect their random bits. Using this reduction we obtain dynamic algorithms against an adaptive adversary with improved update and query times for global minimum cut, all pairs distances, and all pairs effective resistance.

We further improve our update and query times by showing how to maintain a sparsifier over an expander decomposition that can be refreshed fast. This fast refresh enables it to be robust against what we call a blinking adversary that can observe the output of the algorithm only following refreshes. We believe that these techniques will prove useful for additional problems.

On the flip side, we specify dynamic problems that, assuming a random oracle, every dynamic algorithm that solves them against an adaptive adversary must be polynomially slower than a rather straightforward dynamic algorithm that solves them against an oblivious adversary. We first show a separation result for a search problem and then show a separation result for an estimation problem. In the latter case our separation result draws from lower bounds in adaptive data analysis.

CCS CONCEPTS

• **Theory of computation → Dynamic graph algorithms**.

KEYWORDS

Dynamic algorithms, adaptive adversaries, differential privacy

ACM Reference Format:
Amos Beimel, Haim Kaplan, Yishay Mansour, Kobbi Nissim, Thatchaphol Saranurak, and Uri Stemmer. 2022. Dynamic Algorithms against an Adaptive Adversary: Generic Constructions and Lower Bounds. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC '22), June 20–24, 2022, Rome, Italy.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3519935.3520064

1 INTRODUCTION

Randomized algorithms are often analyzed under the assumption that their internal randomness is independent of their inputs. This is a reasonable assumption for *offline* algorithms, which get all their inputs at once, process it, and spit out the results. However, in *online* or *interactive* settings, this assumption is not always reasonable. For example, consider a *dynamic* setting where the input comes in gradually (e.g., a graph undergoing a sequence of edge updates), and the algorithm continuously reports some value of interest (e.g., the size

*Work partially funded by ERC grant 742754 (project NTSC), by the Israel Science Foundation grant no. 391/21, and by the Cyber Security Research Center at Ben-Gurion University
†Partially supported by Israel Science Foundation (grant 1595/19), and the Blavatnik Family Foundation.
‡Work partially funded from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 882396), by the Israel Science Foundation (grant number 993/17), Tel Aviv University Center for AI and Data Science (TAD), and the Yandex Initiative for Machine Learning at Tel Aviv University.
§Work partially funded by NSF grant No. 2001041 and by a gift to Georgetown University.
¶Partially supported by the Israel Science Foundation (grant 1871/19) and by Len Blavatnik and the Blavatnik Family foundation.

STOC '22, June 20–24, 2022, Rome, Italy

ACM ISBN 978-1-4503-9264-8/22/06…$15.00
https://doi.org/10.1145/3519935.3520064

of the minimal cut in the current graph). In such a dynamic setting, it might be the case that future inputs to the algorithm depend on its previous outputs, and hence, depend on its internal randomness. For example, consider a large system in which a dynamic algorithm is used to analyze data coming from one part of the system while answering queries generated by another part of the system, but these (supposedly) different parts of the system are connected via a feedback loop. In such a case, it is no longer true that the inputs of the algorithm are independent of its internal randomness.

Nevertheless, classical algorithms, even for interactive settings, are typically analyzed under the (not always reasonable) assumption that their inputs are independent of their internal randomness. (The reason is that taking these dependencies into account often makes the analysis significantly more challenging.) One approach for avoiding the problem is to make the system deterministic. This, however, is very limiting as randomness is essential for good performance in streaming algorithms, online algorithms, and dynamic algorithms—basically in every algorithmic area in which the algorithm runs interactively. This calls for the design of algorithms providing (provable) utility guarantees even when their inputs are chosen adaptively. Indeed, this motivated an exciting line of work, spanning different communities in theoretical computer science, all focused on this question. This includes works from the streaming community [2, 3, 14, 15, 26, 29, 47, 51, 51, 66], learning community [5, 29, 37, 42, 46, 62], and dynamic algorithms [16, 18, 19, 21–25, 32–35, 43–45, 49, 55, 56, 65, 67]. We continue the study of this question for dynamic algorithms.

Before presenting our new results, we make our setting more precise. Given an input x that undergoes a sequence of updates, our goal is to maintain a valid solution to some predefined problem P at any point in time. We consider both *estimation* and *search* problems. In an *estimation* problem the goal is to provide a $(1 \pm \epsilon)$ approximation of a numeric quantity that is a function of the current input. An example is the global min-cut problem, where an update inserts or deletes an edge and the goal is to output a $(1 \pm \epsilon)$ approximation of the size of the global min-cut. In a *search* problem, the response to the query is a non-numeric value. For example, in the search version of the global min-cut problem the goal is to output a cut whose size is not much larger than the size of the smallest cut.

Formally, given a problem P (over a domain X) and an initial input $x_0 \in X$, we consider a sequence of m input updates $u_1, u_2, \ldots, u_m$, where every u_i is a function $u_i : X \to X$. After every such update u_i, the (current) input is replaced with $x_i \leftarrow u_i(x_{i-1})$, and our goal is to respond with a valid solution for $P(x_i)$. We refer to the case where the sequence of input updates is fixed in advance as the *oblivious* setting. In this work, we focus on the *adaptive* setting, where the sequence of input updates may be chosen adaptively. We think of the entity that generates the inputs as an "adversary" whose goal is to force the algorithm to misbehave (either to err or to have a large runtime). Specifically, the adaptive setting is modeled by a two-player game between a (randomized) `Algorithm` and an `Adversary`. At the beginning, we fix a problem P, and the `Adversary` chooses the initial input x_0. Then the game proceeds in rounds, where in the ith round:

(1) The `Adversary` chooses an update u_i, thereby modifying the current input to $x_i \leftarrow u_i(x_{i-1})$. Note that u_i may depend on all previous updates and outputs of the `Algorithm`.
(2) The `Algorithm` processes the new update u_i and outputs its current response z_i.

Remark 1.1. For simplicity, in the above two-player game we focused on the case where the algorithm outputs a response after every update. Actually, in later sections of the paper, we will make the distinction between an *update* and a *query*. Specifically, in every time step the `Adversary` poses either an *update* (after which the `Algorithm` updates its data structure and outputs nothing) or a *query* (after which the `Algorithm` outputs a response). Moreover, in some of our results, we will separate between the preprocessing time (the period of time after the algorithm obtains its initial input x_0, and before it obtains its first update) and the update time and query time.

We say that the `Algorithm` solves P in the adaptive setting with amortized update (and query) time t if for any `Adversary`, with high probability, in the above two-player game,

(1) For every i we have that z_i is a valid solution for $P(x_i)$.
(2) The `Algorithm` runs in amortized time t per update.

Notation. We refer to algorithms in the adaptive setting as *algorithms that work against an adaptive adversary*, or *adaptive algorithms* in short. Analogously, we refer to algorithms in the oblivious setting as *algorithms that work against an oblivious adversary*, or *oblivious algorithms* in short.

1.1 Our Contributions

In this paper we take a general perspective into studying dynamic algorithms against an adaptive adversary. On the positive side, we develop a general technique for transforming an oblivious algorithm into an adaptive algorithm. We show that our general technique results in more efficient adaptive algorithms for several graph problems of interest. On the negative side, we prove separation results. Specifically, we present dynamic problems that can be trivially solved against oblivious adversaries, but, under certain assumptions, require a significantly higher computation time when the adversary is adaptive. This is the first separation between the oblivious setting and the adversarial setting for dynamic algorithms. In particular, this is the first separation between randomized and deterministic dynamic algorithms, since deterministic algorithms always work against an adaptive adversary.

1.1.1 Our Positive Results. We describe a generic black-box reduction to obtain a dynamic algorithm against an adaptive adversary from an oblivious one. This reduction can be applied to any oblivious dynamic algorithm for an estimation problem. We then apply our generic reduction to obtain adaptive algorithms for several well-studied graph problems. To speed up the adaptive algorithms that we get from our generic reduction, we construct a sparsifier over a dynamic expander decomposition with fast initialization time. By combining our reduction technique with this sparsifier we substantially improve the amortized update time. We obtain the following theorem.

Theorem 1.2. *Given a graph G with n vertices undergoing edge insertions and deletions, there are dynamic algorithms against an adaptive adversary for the following problems:*

- ***Global min cuts:*** *$(1+\epsilon)$-approximation in $\tilde{O}(m^{1/2}n^{1/4})$ amortized update and query time (see Corollary 4.1).*
- ***All-pairs effective resistance:*** *$(1+\epsilon)$-approximation using $n^{3/4+o(1)}$ amortized update and query time (see Corollary 4.13).*
- ***All-pairs distances:*** *$\log^{3i+O(1)} n$-approximation using $n^{1/2+1/(2i)+o(1)}$ amortized update and query time, for any integer i (see Corollary 4.7).*
- ***All-pairs distances with better approximation:*** *$\log n \cdot \mathrm{poly}(\log\log n)$-approximation using $\tilde{O}(m^{4/5})$ amortized update and query time, where m is the current number of edges (see Corollary 4.2).*

In Table 1, we compare our results to previously known results. For all problems we consider, essentially the only known technique against an adaptive adversary is to maintain a sparsifier of the graph [17] and simply query on top of the sparsifier.

1.1.2 Our Negative Results. We present two separation results, one for a search problem and one for an estimation problem. Our separation results rely on (unproven) computational assumptions, which hold in the random oracle model.[1] Assuming a random oracle is a common assumption in cryptography, starting in the seminal work of Bellare and Rogaway [13]. Many practical constructions are first designed and proved assuming a random oracle and then implemented using a cryptographic hash function replacing the random oracle. This is known as the random oracle methodology.[2] Thus, heuristically, the computational assumptions we make hold for cryptographic hash functions. We obtain the following theorem.

Theorem 1.3 (informal). *Under some computational assumptions (or, alternatively, in the random oracle model):*

1. *For any constant $c > 1$, there is a dynamic search problem P^n_{search}, where n is a parameter controlling the instance size, that can be solved in the oblivious setting with amortized update time $O(n)$, but requires amortized update time $\Omega(n^c)$ in the adversarial setting, even if the algorithm is allowed time $2^{0.5n}$ in the preprocessing stage.*
2. *There is a dynamic estimation problem P^n_{est}, where n is a parameter controlling the instance size, that can be solved in the oblivious setting with total time $O(n^6)$ over $O(n^2)$ updates, but requires total time $\Omega(n^7)$ over $O(n^2)$ updates in the adversarial setting.*

We note that our lower bound for the estimation problem P^n_{est} matches what we would get by applying our positive result (our generic reduction) to the oblivious algorithm that solves P^n_{est}.

[1] A random oracle is an infinite random string $\mathcal{R}$ such that the algorithm can read the i-th bit in $\mathcal{R}$, denoted $\mathcal{R}[i]$, at the cost of one time unit. We assume that each bit of $\mathcal{R}$ is uniformly distributed and independent of all other bits of $\mathcal{R}$, thus, the only way to get any information on $\mathcal{R}[i]$ is to read this bit.

[2] We remark that the random oracle methodology is a heuristic. Canetti, Halevi, and Goldreich [28] provide specifically tailored constructions of cryptographic schemes such that they become insecure under any instantiation of the random oracle with a computable function.

1.2 Technical Overview

We give a technical overview of our results. Any informalities made herein will be removed in the sections that follow.

1.2.1 Generic Transformation Using Differential Privacy. Differential privacy [38] is a mathematical definition for privacy that aims to enable statistical analyses of datasets while providing strong guarantees that individual level information does not leak. Over the last few years, differential privacy has proven itself to be an important algorithmic notion (even when data privacy is not of concern), and has found itself useful in many other fields, such as machine learning, mechanism design, secure computation, probability theory, secure storage, and more [5, 10, 53, 54, 57].

Recall that the difficulty in the adaptive setting arises from potential dependencies between the inputs of the algorithm and its internal randomness. Our transformation uses a technique, introduced by [47] (in the context of streaming algorithms), for using *differential privacy* to protect not the input data, but rather the internal randomness of algorithm. As [47] showed, this can be used to limit (in a precise way) the dependencies between the internal randomness of the algorithm and its inputs, thereby allowing to argue more easily about the utility guarantees of the algorithm in the adaptive setting. Following [47], this technique was also used by [3, 14] for streaming algorithms and by [42] for machine unlearning. We adapt this technique and connect it to the setting of dynamic algorithms in general and dynamic algorithms for graph problems in particular.

Informally, our generic transformation can be described as follows.

1. Let T be a parameter, and let $\mathcal{A}$ be an oblivious algorithm.
2. Before the first update arrives, we initialize $c = \tilde{O}(\sqrt{T})$ independent copies of $\mathcal{A}$, with the initial input x_0.
3. For T time step $i = 1, 2, 3, \ldots, T$:
   (a) Obtain an update u_i.
   (b) Feed the update u_i to *all* of the copies of $\mathcal{A}$.
   (c) Sample $\tilde{O}(1)$ of the copies of $\mathcal{A}$, query them, aggregate their responses with differential privacy, and output the aggregated value.
4. Reset all of the copies of $\mathcal{A}$, i.e., re-initialize each copy on the current input with fresh randomness, and goto step 3.

It can be shown that this construction satisfies differential privacy w.r.t. the internal randomnesses of each of the copies of algorithm $\mathcal{A}$. The intuition is that by instantiating $c = \tilde{O}(\sqrt{T})$ copies of $\mathcal{A}$ we have enough "privacy budget" to aggregate T values privately (this follows from advanced composition theorems for differential privacy [39]). After exhausting our privacy budget, we reset all our data structures, and hence, reset our privacy budget. The total time we need for T steps is

$$\tilde{O}\left(\sqrt{T} \cdot t_{\text{total}} + T \cdot t_q\right),$$

where t_{total} is the total time needed to conduct T updates to algorithm $\mathcal{A}$, and t_q is the query time of algorithm $\mathcal{A}$. Instantiating this generic construction for the graph problems we study already gives new (and non trivial) results, but does not yet obtain the results stated in Theorem 1.2 and in Table 1. Specifically, this obtains all the results in Theorem 1.2 except that the update and query

Table 1: A comparison between previous oblivious and adaptive algorithms, and our new adaptive algorithm. The bound shown is the maximum of update time and query time. We omit $\mathrm{polylog}(n)$ factors. By *static* algorithms, we mean algorithms that compute the answer from scratch.

Problems	Approx.	Previous oblivious algo.	Previous adaptive algo.	Our new adaptive algo.
Global min cut	$(1+\epsilon)$	$n^{1/2}$ [63]	m (use static algo. [52])	$m^{1/2}n^{1/4}$
All-pairs effective resistance	$(1+\epsilon)$	$n^{2/3+o(1)}$ [31]	m (use static algo.)	$n^{3/4+o(1)}$
All-pairs distances	$\log^{3i+O(1)} n$	$n^{1/i+o(1)}$ [41]	n (implicit in [17])	$n^{1/2+1/(2i)+o(1)}$
	$\log n \cdot$ poly log log n	$n^{2/3+o(1)}$ [31]	m (use static algo.)	$m^{4/5}$

time bounds depend on m, the number of edges, rather than n, the number of vertices.

1.2.2 The Blinking Adversary Model. We observe that for the graph problems we are interested in, we can improve over the above generic construction as follows. We design oblivious algorithms with an extra property that allows us to "refresh" them in Step 4 of the above generic transformation faster than the time needed to initialize them from scratch. Formally, the "refresh" properties that we need are:

(1) The algorithm maintains utility against a "semi-adaptive" adversary (which we call a *blinking adversary*), defined as follows. Every time we hit the refresh button, say in time i_{refresh}, the adversary gets to see all of the outputs given by the algorithm *before* time i_{refresh}. The adversary may use this in order to determine the next updates and queries. From that point on, until the next time we hit the refresh button, the adversary is oblivious in the sense that it does not get to see additional outputs of the algorithm.[3]
(2) Hitting the refresh button is *faster* than instantiating the algorithm from scratch.

We show that if we have an algorithm $\mathcal{A}$ with a (hopefully fast) refresh button, then when applying our generic construction to $\mathcal{A}$, in Step 4 of the generic construction it suffices to *refresh* all the copies of algorithm $\mathcal{A}$, instead of completely resetting them. Assuming that refresh is indeed faster than reset, then we get a speedup to our resulting construction. We then show how to construct dynamic algorithms (for the graph problems we are interested in) with a fast refresh button.

Designing algorithms with fast refresh. Let $\mathcal{A}$ be an oblivious algorithm for one of our graph problems. In order to speedup $\mathcal{A}$'s reset time, we ideally would have used an appropriate sparsifier (i.e., a graph with few edges that approximates the properties of the original graph), and run our oblivious algorithm $\mathcal{A}$ on top of the sparsifier. This would make sure that the time needed to restart $\mathcal{A}$ (without restarting the sparsifier) depends on n rather than m. Unfortunately, known sparsifiers that well approximate the functions that we estimate, do not work against an adaptive adversary. Consequently, if we use such a sparsifier (and never reset it) then the adversary may learn about the sparsifier's randomness through the estimates spitted out by the algorithm and use this knowledge to fool the algorithm. On the other hand, resetting the sparsifier would cost us $O(m)$ time, which would be too much.

To overcome this challenge, we design a sparsifier with a fast $\tilde{O}(n)$ time *refresh* button. Our construction of a sparsifier has two parts. We first use an adaptive dynamic algorithm of Bernstein et al. [17] that maintains a decomposition of G into expanders. This algorithm can handle insertion and deletion of edges in polylog amortized time; we execute the update step of this algorithm in each update to the graph. The second part is a construction of a sparsifier from the decomposition to expanders. As we do not know how to construct an adaptive dynamic algorithm for this task, we only execute it in the refresh.

1.2.3 Negative Result for a Search Problem. In the full version of this paper [11], we prove that there is a search problem that is much easier for an oblivious algorithm than for an adaptive algorithm. We next describe the ideas of this proof. To simplify the presentation in the introduction, we present a simplified problem; our separation in [11] is stronger.

Assume that $H_n : \{0,1\}^n \to \{0,1\}^n$ is a function such that $H_n(x)$ can be computed in time $O(n)$ for every $x \in \{0,1\}^n$. Consider a dynamic problem in which an adversary maintains a set $X \subset \{0,1\}^n$ of *excluded* strings (where $|X| < 2^{0.5n}$), initialized as the empty set. In each update the adversary adds an element to X and the algorithm has to output the pair $\big(x, H_n(x)\big)$ for an element $x \notin X$. An oblivious dynamic algorithm picks a random x in the preprocessing stage, computes $w = H_n(x)$, and outputs the pair (x, w). No matter how an oblivious adversary chooses its updates to X, the probability that in a sequence of at most $2^{0.5n}$ updates the adversary adds to X the random x chosen by the algorithm is negligible (as $x \in_R \{0,1\}^n$ and the size of X is at most $2^{0.5n}$), and the algorithm can use the same output (x, w) after each update, thus not paying at all for an update. However, an adaptive adversary can see the output x_{i-1} of the algorithm after the $(i-1)$-th update and add x_{i-1} to X. Thus, after each update the algorithm has to compute $H_n(x_i)$ for a new x_i. We want to argue that an adaptive algorithm has to spend $\Omega(n)$ amortized time per update. For this we need to

[3] In our application, we hit the refresh button after every T outputs of the algorithm, which means that the adversary gets to see the outputs of the algorithm in bulks of size T.

assume that H_n is moderately hard, e.g., computing $H_n(x)$ requires time $\Omega(n)$. However, this assumption does not suffice; we need to assume that computing $H_n(x_1), \ldots, H_n(x_\ell)$ for some ℓ requires time $\Omega(n\ell)$ for any sequence of inputs $x_1, \ldots, x_\ell$ chosen by the algorithm. That is, we need to assume that computing H_n on many inputs cannot be done substantially more efficiently than computing each $H_n(x_i)$ independently; such assumption is known as a direct-sum assumption. Thus, assuming that there is a function that has a direct-sum property we get a separation.

In the simple separation described above, the algorithm can in the preprocessing stage choose a sequence $x_1, x_2, \ldots, x_\ell$ and compute the values $H_n(x_1), H_n(x_2), \ldots, H_n(x_\ell)$. Thereby the algorithm does not need to spend any time after each update. To force the algorithm to work during the updates, in [11] we define a more complicated dynamic problem and prove that the amortized update time of an adaptive algorithm for this problem is high even if the preprocessing time of the algorithm is $2^{0.5n}$.

The problem discussed above actually captures the very well-known technique in dynamic graph algorithms for exploiting an oblivious adversary (e.g. how dynamic reachability and shortest paths algorithms choose a random root for an ES-tree [20, 48, 58], or how dynamic maximal matching algorithms choose a random edge to match [7, 60], and similarly for dynamic independent set [9, 30] and dynamic spanner algorithms [8]). Roughly speaking, the set X above corresponds to a set of deleted edges in the graph. These algorithms need to "commit" to some x, but whenever x is deleted/excluded from the graph, they need to recommit to a new x' and spend a lot of time. By choosing a random x, the algorithm would not recommit so often against an oblivious adversary, hence obtain small update time. Our result, therefore, formalizes the intuition that this general approach does not extend (at least as is) to the adaptive setting.

1.2.4 Negative Result for an Estimation Problem. In the full version of this paper [11], we present a separation result for an estimation problem. Our result uses techniques from the recent line of work on *adaptive data analysis* [37]. We remark that a similar connection to adaptive data analysis was utilized by [51], in order to show impossibility results for adaptive streaming algorithms. However, our analysis differs significantly as our focus is on runtime lower bounds, while the focus of [51] was on space lower bounds.

To obtain our negative result, we introduce (and assume the existence of) a primitive, which we call *boxes scheme*, that allows a dealer to insert m plaintext inputs into "closed boxes" such that:

(1) A closed box can be opened, retrieving the plaintext input, in time T (for some parameter T).
(2) Any algorithm that runs in time $b \cdot T$ cannot learn "anything" about the content of more than $O(b)$ of the boxes.

Given this primitive (which we define precisely in [11]), we consider the following problem.

Definition 1.4 (The average of boxes problem, informal)**.** The initial input consists of m closed boxes. On every time step, the algorithm gets a predicate (mapping *plaintexts* to $\{0,1\}$), and the algorithm needs to estimate the average of this predicate over the *content* of the m boxes.

This is an easy problem in the oblivious setting, because the algorithm can sample $\tilde{O}(1)$ of the boxes, open them, and use their content in order to estimate the average of all the predicates given throughout the execution. However, as we show, this is a hard problem in the adaptive setting. Specifically, every adaptive algorithm for this problem essentially must open $\Omega(m)$ of the m boxes it gets as input. Intuitively, as opening boxes takes time, we get a separation between the oblivious and the adaptive settings, thereby proving item 2 of Theorem 1.3.

2 PRELIMINARIES ON DIFFERENTIAL PRIVACY

Roughly speaking, an algorithm is differentially private if its output distribution is "stable" w.r.t. a change to a single input element. To formalize this, let X be a domain. A *database* $S \in X^n$ is a list of elements from domain X. The *i-th row* of S is the i-th element in S.

Definition 2.1 (Differential Privacy)**.** A randomized algorithm $\mathcal{A}$ is (ϵ, δ)-*differentially private* (in short (ϵ, δ)-*DP*) if for any two databases S and S' that differ on one row and any subset of outputs T, it holds that

$$\Pr[\mathcal{A}(S) \in T] \leq e^{\epsilon} \cdot \Pr[\mathcal{A}(S') \in T] + \delta,$$

where the probability is over the randomness of $\mathcal{A}$. The parameter ϵ is referred to as the *privacy parameter*. When $\delta = 0$ we omit it and write ϵ-DP.

Composition. A crucial property of differential privacy is that it is preserved under *adaptive composition*. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be algorithms. The adaptive composition $\mathcal{A} = \mathcal{A}_2 \circ \mathcal{A}_1$ is such that, given a database S, $\mathcal{A}$ invokes $a_1 = \mathcal{A}_1(S)$, then $a_2 = \mathcal{A}_2(a_1, S)$, and finally outputs (a_1, a_2). The *basic composition theorem* guarantees that, if $\mathcal{A}_1, \ldots, \mathcal{A}_k$ are each (ϵ, δ)-DP algorithms, then the composition $\mathcal{A}_k \circ \cdots \circ \mathcal{A}_1$ is $(\epsilon' = k\epsilon, k\delta)$-DP. The *advanced composition theorem* shows that the privacy parameter ϵ' need not grow linearly in k, but instead only in $\approx \sqrt{k}$.

Theorem 2.2 (Advanced Composition [39]). *Let $\epsilon, \delta' \in (0,1]$ and $\delta \in [0,1]$. If $\mathcal{A}_1, \ldots, \mathcal{A}_k$ are each (ϵ, δ)-DP algorithms, then $\mathcal{A}_k \circ \cdots \circ \mathcal{A}_1$ is $(\epsilon', \delta' + k\delta)$-DP where*

$$\epsilon' = \sqrt{2k \ln(1/\delta')} \cdot \epsilon + 2k\epsilon^2.$$

In our applications, the second term (which is linear in k) will be dominated by the first term. The saving in ϵ' from k to $\sqrt{k}$ enables a polynomial speed up in our applications.

Amplification via sampling. *Secrecy-of-the-sample* is a technique for "amplifying" privacy by subsampling. Informally, if a γ-fraction of the input database rows are sampled, and only those are given as input to a differentially private algorithm then the privacy parameter is reduced (i.e., improved) by a factor proportional to $\approx \gamma$.

Theorem 2.3 (Amplicafication via sampling ([27, Lemma 4.12])). *Let $\mathcal{A}$ be an ϵ-DP algorithm where $\epsilon \leq 1$. Let $\mathcal{A}'$ be the algorithm that, given a database S of size n, first constructs a database $T \subset S$ by sub-sampling with repetition $k \leq n/2$ rows from S and then returns $\mathcal{A}(T)$. Then, $\mathcal{A}'$ is $(\frac{6k}{n} \cdot \epsilon)$-DP.*[4]

[4]The statement in [27] is more general and allows $\mathcal{A}$ to be (ϵ, δ)-DP.

Generalization. Our analysis relies on the *generalization property* of differential privacy. Let $\mathcal{D}$ be a distribution over a domain X and let $h : X \to \{0,1\}$ be a predicate. Suppose that the goal is to estimate $h(\mathcal{D}) = \mathbb{E}_{x\sim\mathcal{D}}[h(x)]$. A simple solution is to sample a set S consisting of few elements from X independently from $\mathcal{D}$, and then compute the empirical average $h(S) = \frac{1}{|S|}\sum_{x\in S} h(x)$. By standard concentration bounds, we have $h(\mathcal{D}) \approx h(S)$. That is, the empirical estimate on a small sample S *generalizes* to the estimate over the underlying distribution $\mathcal{D}$.

The argument above, however, fails if S is sampled first and h is chosen adaptively, because h can "overfit" S. The theorem below says that, as long as the predicate h is generated from a differentially private algorithm $\mathcal{A}$, we can still guarantee generalization of h (even when the choice of h is a function of S). As shown in [47], this key property will link differentially privacy to accuracy of algorithms against an adaptive adversary.

Theorem 2.4 (Generalization of DP [5, 37]). *Let $\epsilon \in (0, 1/3)$, $\delta \in (0, \epsilon/4)$ and $t \geq \frac{1}{\epsilon^2}\log(\frac{2\epsilon}{\delta})$. Let $\mathcal{D}$ be a distribution on a domain X. Let $S \sim \mathcal{D}^t$ be a database containing t elements sampled independently from $\mathcal{D}$. Let $\mathcal{A}$ be an algorithm that, given any database S of size t, outputs a predicate $h : X \to \{0,1\}$. (We emphasize that h may depend on S.)*

If $\mathcal{A}$ is (ϵ, δ)-DP, then the empirical average of h on sample S, i.e., $h(S) = \frac{1}{|S|}\sum_{x\in S} h(x)$, and h's expectation over the underlying distribution $\mathcal{D}$, i.e., $h(\mathcal{D}) = \mathbb{E}_{x\sim\mathcal{D}}[h(x)]$, are within 10ϵ with probability at least $1 - \frac{\delta}{\epsilon}$. In other words, we have

$$\Pr_{\substack{S\sim\mathcal{D}^t \\ h\leftarrow\mathcal{A}(S)}}\left[\left|\frac{1}{|S|}\sum_{x\in S} h(x) - \mathbb{E}_{x\sim\mathcal{D}}[h(x)]\right| \geq 10\epsilon\right] < \frac{\delta}{\epsilon}.$$

The only differentially private subroutine we need in this paper is a very simple algorithm for computing an approximate median of elements in databases.

Theorem 2.5 (Private Median (folklore)). *Let X be a finite domain with total order. For every $\epsilon, \beta \in (0,1)$, there is $\Gamma = O(\frac{1}{\epsilon}\log(\frac{|X|}{\beta}))$ such that the following holds. There exists an $(\epsilon, 0)$-DP algorithm $\mathtt{pMedian}_{\epsilon,\beta}$ that, given a database $S \in X^*$, in $O(|S| \cdot \frac{1}{\epsilon}\log^3(\frac{|X|}{\beta}) \cdot \mathrm{polylog}(|S|))$ time outputs an element $x \in X$ (possibly $x \notin S$) such that, with probability at least $1-\beta$, there are at least $|S|/2-\Gamma$ elements in S that are bigger or equal to x and at least $|S|/2 - \Gamma$ elements in S that are smaller or equal to x.*

The algorithm is based on binary search. If we assume that we can sample a real number from the Laplace distribution in constant time, then the running time would be $O(|S|\log|X|)$. Here, we do not assume that and use the bound from [4]. We remark that there are several advanced constructions for private median with error that grows very slowly as a function of the domain size $|X|$ (only polynomially with $\log^*|X|$) [12, 27, 50]. In our application, however, the domain size is already small, and hence, we can use the simpler construction stated above.

3 A GENERIC REDUCTION

In this section, we present a simple black-box transformation of dynamic algorithms against an oblivious adversary to ones against an adaptive adversary. The approach builds on the work of [47] for transforming streaming algorithms, which focus on space instead of update time. A key difference is that we apply subsampling for speeding up. This is crucial to ensure that our applications in Section 4 have non-trivial update and query time. We give the formal statement and its proof below.

Theorem 3.1. *Let $\delta_{\mathrm{fail}}, \alpha > 0$ be parameters. Let g be a function that maps elements in some domain X to a number in $[-U, -\frac{1}{U}] \cup \{0\} \cup [\frac{1}{U}, U]$ where $U > 1$. Suppose there is a dynamic algorithm $\mathcal{A}$ for estimating g against an* oblivious *adversary that, given an initial input $x_0 \in X$ undergoing a sequence of T updates, guarantees the following:*

- *The total preprocessing and update time for handling T updates is t_{total}.*
- *The query time is t_q and, with probability $\geq 9/10$, the answer is a γ-approximation of $g(x)$.*

Then, there is a dynamic algorithm $\mathcal{A}'$ against an adaptive *adversary that, with probability at least $1 - \delta_{\mathrm{fail}}$, maintains a $\gamma(1+\alpha)$-approximation of a function $g(x)$ when the input undergoes T updates in $\tilde{O}(\sqrt{T}\cdot t_{\mathrm{total}} + T\cdot t_q)$ total update time. By restarting the algorithm every T steps we can run $\mathcal{A}'$ on a sequence of updates of any length in $\tilde{O}(t_{\mathrm{total}}/\sqrt{T} + t_q)$ amortized update time. The $\tilde{O}$ in this theorem hides $\mathrm{polylog}(\frac{T\log U}{\alpha\delta_{\mathrm{fail}}})$ factors.*

Algorithm $\mathcal{A}'$ description.

(1) Before the first update arrives, initialize $c = \tilde{O}(\sqrt{T})$ copies of $\mathcal{A}$, denoted by $\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(c)}$, with the input x_0.
(2) For each step $i \in [1, T]$, given an update u_i,
  (a) Feed the update u_i to every copy of $\mathcal{A}$.
  (b) To compute a $\gamma(1+\alpha)$-approximation of $g(x_i)$, do the following:
    (i) Independently and uniformly sample $s = \tilde{O}(1)$ indices $j_1, \ldots, j_s \in [1, c]$.[5] We emphasize the these sampled indices are not revealed to the adversary.
    (ii) For each $1 \leq k \leq s$, query $\mathcal{A}^{(j_k)}$ and let $\mathtt{out}_i^{(j_k)}$ denote its estimate of $g(x_i)$, round up $\mathtt{out}_i^{(j_k)}$ to the nearest power of $(1+\alpha)$ and denote it by $\widetilde{\mathtt{out}}_i^{(j_k)}$. If $\mathtt{out}_i^{(j_k)} = 0$, set $\widetilde{\mathtt{out}}_i^{(j_k)} = 0$.
    (iii) Algorithm $\mathcal{A}'$ outputs the estimate of $g(x_i)$ as
    $$\mathtt{out}'_i = \mathtt{pMedian}_{\epsilon_{\mathrm{med}},\beta}(\widetilde{\mathtt{out}}_i^{(j_1)}, \ldots, \widetilde{\mathtt{out}}_i^{(j_s)})$$
    where $\mathtt{pMedian}_{\epsilon_{\mathrm{med}},\beta}$ is the algorithm for estimating the median with differential privacy (see Theorem 2.5), where $\epsilon_{\mathrm{med}} = 1/2$ and $\beta = \delta_{\mathrm{fail}}/2T$.

Specifying parameters. Here, we specify the parameters of the algorithm. This is needed for precisely stating the total update time. Let X_{med} denote the total ordered domain for $\mathtt{pMedian}_{\epsilon_{\mathrm{med}},\beta}$. In our setting, X_{med} is simply the range of $\widetilde{\mathtt{out}}_i^{(j_k)}$ which is $\{0\} \cup \{\pm(1+\alpha)^a \mid -\log_{(1+\alpha)} U \leq a \leq \log_{(1+\alpha)} U\}$. So $|X_{\mathrm{med}}| = O(\frac{\log U}{\alpha})$. The parameter Γ from Theorem 2.5 is such that

$$\Gamma = O(\frac{1}{\epsilon_{\mathrm{med}}}\log(\frac{|X_{\mathrm{med}}|}{\beta})) = O(\log(\frac{\log U}{\alpha\beta})) = \tilde{O}(1).$$

[5] This is unlike in [47] where all instances of $\mathcal{A}$ are used.

Now, we set $s = 100\Gamma$, so that when $\mathtt{pMedian}_{\epsilon_{\text{med}},\beta}$ is given s numbers, it returns a number whose rank is in $s/2 \pm \Gamma = (1/2 \pm 1/100)s$, a good enough approximation of the median. Lastly, we set the number of copies as

$$c = 200 \cdot 6s\epsilon_{\text{med}} \cdot \sqrt{2T \ln(100/\beta)} = \tilde{O}(\sqrt{T}).$$

This choice of c is such that after subsampling and composition the entire algorithm would be private for the appropriate parameters with respect to its random bits. See Corollary 3.4.

Total update time. The total time c copies of $\mathcal{A}$ preprocess the initial input x_0 and handle all T updates is clearly $c \cdot t_{\text{total}}$ by definition of t_{total}. For each step i, we only query s many copies of $\mathcal{A}$ to obtain $\mathtt{out}_i^{(j_1)}, \ldots, \mathtt{out}_i^{(j_s)}$. Rounding $\mathtt{out}_i^{(j_k)}$ to its nearest power of $(1+\alpha)$ takes $O(\log \frac{\log U}{\alpha})$ time by binary search. Computing $\mathtt{out}_i'$ is to evaluate $\mathtt{pMedian}_{\epsilon_{\text{med}},\beta}$, which takes $t_{\text{med}} = O(s \cdot \frac{1}{\epsilon_{\text{med}}} \log(\frac{|X_{\text{med}}|}{\beta}) \cdot \text{polylog}(s)) = O(s \cdot \frac{1}{\epsilon_{\text{med}}} \log(\frac{\log U}{\alpha\beta}) \cdot \text{polylog}(s))$ time by Theorem 2.5. Therefore, we can conclude:

Proposition 3.2. *The total update time of $\mathcal{A}'$ is at most*

$$\begin{aligned}
&c \cdot t_{\text{total}} + T \times O\left((t_q + \log \frac{\log U}{\alpha}) \cdot s + t_{\text{med}}\right) \\
&= O\left(t_{\text{total}} \cdot \sqrt{T} \cdot \log\left(\frac{T \log U}{\alpha \delta_{\text{fail}}}\right) \cdot \sqrt{\log \frac{T}{\delta_{\text{fail}}}}\right) \\
&\quad + O\left(t_q \cdot T \cdot \log\left(\frac{T \log U}{\alpha \delta_{\text{fail}}}\right) + T \cdot \text{polylog}\left(\frac{T \log U}{\alpha \delta_{\text{fail}}}\right)\right) \\
&= \tilde{O}(t_{\text{total}}\sqrt{T} + t_q T)
\end{aligned}$$

The second line is obtained by simply plugging in the definitions of $\beta = \delta_{\text{fail}}/2T$, $\Gamma = O(\log(\frac{\log U}{\alpha\beta})) = O(\log(\frac{T \log U}{\alpha\delta_{\text{fail}}}))$, $s = 100\Gamma = O(\log(\frac{T \log U}{\alpha\delta_{\text{fail}}}))$ and, hence, we have that $c = O(\sqrt{T} \cdot \log(\frac{T \log U}{\alpha\delta_{\text{fail}}}) \cdot \sqrt{\log \frac{T}{\delta_{\text{fail}}}})$ and so the second line follows.

Accuracy against an adaptive adversary. It only remains to argue that $\mathcal{A}'$ maintains accurate approximation of $g(x)$ against an adaptive adversary. Let $r^{(1)}, \ldots, r^{(c)} \in \{0,1\}^*$ denote the random strings used by the oblivious algorithms $\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(c)}$ during the T updates. We view the collection of random string $R = \{r^{(1)}, \ldots, r^{(c)}\}$ as a database where each $r^{(j)}$ is its row. We will show that the transcript of the interaction between the adversary and algorithm $\mathcal{A}'$ is differentially private *with respect to R*. (This is perhaps the most important conceptual idea from [47].) Then, we will exploit this fact to argue that the answers of $\mathcal{A}'$ are accurate. Let us formalize this plan below.

For any time step i, let $\mathtt{out}_i'(R)$ denote the output of $\mathcal{A}'$ at time step i when the collection R is fixed. Note that $\mathtt{out}_i'(R)$ is still a random variable because $\mathcal{A}'$ uses some additional random strings for subsampling and computing a private median. Now, we define $\mathcal{T}_i(R) = (u_i, \mathtt{out}_i'(R))$ as the transcript between $\mathtt{Adversary}$ and algorithm $\mathcal{A}'$ at step i. Let

$$\mathcal{T}(R) = x_0, \mathcal{T}_1(R), \ldots, \mathcal{T}_T(R)$$

denote the transcript. We also prepend the transcript with the input x_0 before the first update arrives. Since R is freshly sampled at the beginning, it is completely independent from x_0. We view $\mathcal{T}_i$ and $\mathcal{T}$ as algorithms that, given a database R, return the transcripts. From this view, we can prove that they are differentially private with respect to R.

Lemma 3.3. *For a fixed step i, $\mathcal{T}_i$ is $(\frac{6s}{c} \cdot \epsilon_{\text{med}}, 0)$-DP with respect to R.*

Proof. Given a transcript $\mathcal{T}_i(R) = (u_i, \mathtt{out}_i'(R))$ of only a single step i, the update u_i does not give any (new) information about R. So it suffices to consider $\mathtt{out}_i'(R)$, which is set to

$$\mathtt{pMedian}_{\epsilon_{\text{med}},\beta}(\widetilde{\mathtt{out}}_i^{(j_1)}, \ldots, \widetilde{\mathtt{out}}_i^{(j_s)}).$$

By Theorem 2.5, we have that $\mathtt{pMedian}_{\epsilon_{\text{med}},\beta}$ is $(\epsilon_{\text{med}}, 0)$-DP. Its inputs are $\widetilde{\mathtt{out}}_i^{(j_1)}, \ldots, \widetilde{\mathtt{out}}_i^{(j_s)}$, which are determined by the subset $\{r^{(j_1)}, \ldots, r^{(j_s)}\} \subset R$, which in turn are obtained by sub-sampling from R. By invoking Theorem 2.3 (and note that $s \le c/2$), subsampling boosts the privacy parameter and so $\mathtt{out}_i'(R)$ is $(\frac{6s}{c} \cdot \epsilon_{\text{med}}, 0)$-DP with respect to R as claimed. □

Corollary 3.4. *$\mathcal{T}$ is $(\frac{1}{100}, \frac{\beta}{100})$-DP with respect to R.*

Proof. Observe that $\mathcal{T}$ is an adaptive composition $\mathcal{T}_T \circ \cdots \circ \mathcal{T}_1$ (except that we prepend x_0 which is independent from R). Since each $\mathcal{T}_i$ is $(\frac{6s}{c} \cdot \epsilon_{\text{med}}, 0)$-DP as shown in Lemma 3.3, by applying the advanced composition theorem (Theorem 2.2) with parameters $\epsilon = \frac{6s}{c}\epsilon_{\text{med}}$, $\delta = 0$, and $\delta' = \beta/100$, we have that that $\mathcal{T}$ is $(\epsilon', \delta k + \delta')$-DP where

$$\begin{aligned}
\epsilon' &= \sqrt{2T \ln(100/\beta)} \cdot \left(\frac{6s}{c}\epsilon_{\text{med}}\right) + 2T \cdot \left(\frac{6s}{c}\epsilon_{\text{med}}\right)^2 \\
&\le \frac{1}{200} + \frac{1}{200} = \frac{1}{100}
\end{aligned}$$

because $c = 200 \cdot 6s\epsilon_{\text{med}} \cdot \sqrt{2T \ln(100/\beta)}$. Also, $\delta k + \delta' = \beta/100$. Therefore, $\mathcal{T}$ is $(\frac{1}{100}, \frac{\beta}{100})$-DP. □

Next, we exploit differential privacy for accuracy against an adaptive adversary. Let $\vec{x}_i = (x_0, u_1, \ldots, u_i)$ denote the whole input sequence up to time i. Let $\mathcal{A}(r, \vec{x}_i)$ denote the output of the oblivious algorithm $\mathcal{A}$ on input sequence $\vec{x}_i$, given a random string r. Let

$$\text{acc}_{\vec{x}_i}(r) = \mathbf{1}\{g(x_i) \le \mathcal{A}(r, \vec{x}_i) \le \gamma g(x_i)\}$$

be the indicator function deciding if $\mathcal{A}(r, \vec{x}_i)$ is γ-accurate. Note that $\text{acc}_{\vec{x}_i}(r^{(j)})$ indicates precisely whether the instance $\mathcal{A}^{(j)}$ is accurate at time i. Now, we show that at all times, most instances of the oblivious algorithm are γ-accurate.

Lemma 3.5. *For each fixed $i \in [1, T]$, $\sum_{j=1}^{c} \text{acc}_{\vec{x}_i}(r^{(j)}) \ge \frac{4}{5}c$ with probability at least $1 - \beta$.*

Proof. Observe that the function $\text{acc}_{\vec{x}_i}(\cdot)$ is determined by the transcript $\mathcal{T}$. This is because the input sequence $\vec{x}_i$ is just a substring of the transcript $\mathcal{T}$ and $\vec{x}_i$ determines the predicate $\text{acc}_{\vec{x}_i}$.

Now, we have the following (1) each row of R is a string drawn independently from the uniform distribution $\mathcal{U}$, (2) $\text{acc}_{\vec{x}_i}$ is a predicate on strings and is determined by $\mathcal{T}$ as argued above, and (3) $\mathcal{T}$ can be viewed as a $(\frac{1}{100}, \frac{\beta}{100})$-DP algorithm with respect to R by Corollary 3.4. By the generalization property of differential privacy (Theorem 2.4), we have that the empirical average of $\text{acc}_{\vec{x}_i}$ on

R, i.e., $\frac{1}{c}\sum_{j=1}^{c}\mathrm{acc}_{\vec{x}_i}(r^{(j)})$, and $\mathrm{acc}_{\vec{x}_i}$'s expectation over the underlying distribution $\mathcal{U}$, i.e., $\mathbb{E}_{r\sim\mathcal{U}}[\mathrm{acc}_{\vec{x}}(r)]$ should be close to each other. More formally, by invoking Theorem 2.4 where $\epsilon = 1/100$, $\delta = \beta/100$ and $t = c \gg \frac{1}{\epsilon^2}\log(\frac{2\epsilon}{\delta})$, we have that

$$\Pr_{\substack{R\sim\mathcal{U}^c \\ \mathrm{acc}_{\vec{x}_i}\leftarrow\mathcal{T}(R)}}\left[\left|\frac{1}{c}\sum_{j=1}^{c}\mathrm{acc}_{\vec{x}_i}(r^{(j)}) - \mathbb{E}_{r\sim\mathcal{U}}[\mathrm{acc}_{\vec{x}_i}(r)]\right| \geq \frac{1}{10}\right] \leq \beta.$$

Since the oblivious algorithm $\mathcal{A}$ returns accurate answers with probability at least 9/10 as long as its random choice is independent from the input, for any arbitrary input sequence $\vec{x}$, we have $\mathbb{E}_{r\sim\mathcal{U}}[\mathrm{acc}_{\vec{x}}(r)] \geq 9/10$. Therefore, we have that with probability at least $1-\beta$,

$$\frac{1}{c}\sum_{j=1}^{c}\mathrm{acc}_{\vec{x}_i}(r^{(j)}) \geq \frac{9}{10} - \frac{1}{10} \geq \frac{4}{5}$$

as desired. □

Given that most instances of the oblivious algorithm are always accurate, it is intuitively immediate that $\mathcal{A}'$ is always accurate too. This is because $\mathcal{A}'$ returns the median of the sub-sampled answers from oblivious algorithms. Below, we verify this.

Corollary 3.6. *For all $i \in [1,T]$, $g(x_i) \leq \mathtt{out}'_i \leq \gamma(1+\alpha)g(x_i)$ with probability at least $1-\delta_{\mathrm{fail}}$.*

Proof. Consider a fixed step i. Recall that $\mathcal{A}'$ independently samples s indices $j_1,\ldots,j_s$ and queries $\mathcal{A}^{(j_k)}$ for $1 \leq k \leq s$. Let $\mathrm{acc}_k = \mathrm{acc}_{\vec{x}_i}(r^{(j_k)})$ indicate whether $\mathcal{A}^{(j_k)}$ is accurate at time i. Lemma 3.5 implies that $\mathbb{E}[\sum_{k=1}^{s}\mathrm{acc}_k] \geq \frac{4}{5}s$. By Hoeffding's bound, $\sum_{k=1}^{s}\mathrm{acc}_k \geq \frac{3}{4}s$ with probability at least $1-\exp(-\Theta(s)) \geq 1-\beta$ by making sure that the constant in the definition of s (actually Γ) is large enough.

If $\mathrm{acc}_k = 1$, we have that $g(x_i) \leq \mathtt{out}_i^{(j_k)} \leq \gamma g(x_i)$ and so $g(x_i) \leq \widetilde{\mathtt{out}}_i^{(j_k)} \leq \gamma(1+\alpha)g(x_i)$. So at least $\frac{3}{4}$-fraction of $\widetilde{\mathtt{out}}_i^{(j_1)},\ldots,\widetilde{\mathtt{out}}_i^{(j_s)}$ are $\gamma(1+\alpha)$-approximation of $g(x_i)$. With probability at least $1-\beta$, $\mathtt{pMedian}_{\epsilon_{\mathrm{med}},\beta}$ returns $\mathtt{out}'_i$ such that there are $\frac{1}{2}-\frac{\Gamma}{s} \geq \frac{49}{100}$ fraction of $\widetilde{\mathtt{out}}_i^{(j_1)},\ldots,\widetilde{\mathtt{out}}_i^{(j_s)}$ that are at least $\mathtt{out}'_i$ and the same holds for those that are at most $\mathtt{out}'_i$. Therefore, $\mathtt{out}'_i$ is a $\gamma(1+\alpha)$-approximation of $g(x_i)$ with probability at least $1-2\beta$. By union bound, this holds over all time steps with probability at least $1-2T\beta = 1-\delta_{\mathrm{fail}}$. □

Via the accuracy guarantee from Corollary 3.6 together with the total update time bound from Proposition 3.2, we now conclude the proof of Theorem 3.1.

Extensions of Theorem 3.1. Before we conclude this section, we discuss several possible ways to extend the reduction from Theorem 3.1.

Worst-case update time. First of all, although the reduction is stated for amortized update time, it can be made worst-case if the given oblivious algorithm guarantee worst-case update time. More formally, we have the following.

Theorem 3.7. *Let $g : X \to [-U,-\frac{1}{U}]\cup\{0\}\cup[\frac{1}{U},U]$ be a function that maps elements in some domain X to a number in $[-U,-\frac{1}{U}]\cup\{0\}\cup[\frac{1}{U},U]$ where $U > 1$. Suppose there is a dynamic algorithm $\mathcal{A}$ against an* oblivious *adversary that, given an initial input x_0 undergoing a sequence of T updates, guarantees the following:*

- *The preprocessing time on x_0 is t_p*
- *The worst-case update time for each update is t_u.*
- *The query time is t_q and, with probability $\geq 9/10$, the answer is a γ-approximation of $g(x)$.*

Then, there is a dynamic algorithm $\mathcal{A}''$ against an adaptive *adversary that, with probability at least $1-\delta_{\mathrm{fail}}$, maintains a $\gamma(1+\alpha)$-approximation of a function $g(x)$ when x undergoes any sequence of update using*

$$\tilde{O}\left(\frac{t_p}{\sqrt{T}} + \sqrt{T}t_u + t_q\right)$$

worst-case update time.

Proof sketch. Using exactly the same algorithm from Theorem 3.1, we obtain the adaptive algorithm $\mathcal{A}'$ can handle T updates whose preprocessing time is $\tilde{O}(t_p\sqrt{T})$ and the worst-case update/query time is $\tilde{O}(t_u\sqrt{T}+t_q)$. To get an algorithm with $\tilde{O}(\frac{t_p}{\sqrt{T}} + \sqrt{T}t_u + t_q)$ worst-case update time, we create two instances $\mathcal{A}'_{odd}$ and $\mathcal{A}'_{even}$ of $\mathcal{A}'$ and proceed in phases. Each phase has $\Theta(T)$ updates. We only need to show how to avoid spending a large preprocessing time of $\tilde{O}(t_p\sqrt{T})$ in a single time step. During the odd phases, we use $\mathcal{A}'_{odd}$ to handle the queries and distribute the work for preprocessing of $\mathcal{A}'_{even}$ equally on each time step in this phase. During the even phases, we do the opposite. So the preprocessing time is "spread" over $\Theta(T)$ updates, which consequently contributes $\tilde{O}(\frac{t_p}{\sqrt{T}})$ worst-case update time. This a very standard technique in the dynamic algorithm literature (see e.g. Lemma 8.1 of [6]). □

Speed up for stable answers. Suppose that we know that during T updates, the $(1+\epsilon)$-approximate answers to the queries can change only λ times for some $\lambda \ll T$. Then, using the same idea from [47], the total update time of Theorem 3.1 can be improved to $\tilde{O}(t_{\mathrm{total}}\sqrt{\lambda} + t_q)$.

This idea could be useful for several problems. For examples, suppose that we want to maintain $(1+\epsilon)$-approximation of global minimum cut, or (s,t)-minimum cuts, or maximum matching in unweighted graphs. If the current answer k, we know that it must takes at least ϵk updates before the answer changes by a $(1+\epsilon)$ factor. Suppose that, somehow, the answer is always at least k, then we have $\lambda \leq T/\epsilon k$. It is also possible to remove the assumption that the answer is at least k: if there is a separated adaptive algorithm that can take care of the problem when the answer is less than k, then we can run both algorithms in parallel. This idea of combining the two algorithms, one for small answers and another for large answers, was explicitly used in [14] in the streaming setting.

Speed up via batch updates. In the algorithm for Theorem 3.1, recall that we create $c = \tilde{O}(\sqrt{T})$ copies of $\mathcal{A}$, denoted by $\mathcal{A}^{(1)},\ldots,\mathcal{A}^{(c)}$. For each copy $\mathcal{A}^{(j)}$, we feed an update u_i at every time step i one by one. Consider what if we are lazy in feeding the update to $\mathcal{A}^{(j)}$. That is, we wait until $\mathcal{A}^{(j)}$ is sampled and we want to query $\mathcal{A}^{(j)}$.

Only then we feed a batch of updates containing all updates that we have not feed to $\mathcal{A}^{(j)}$ until the current update. The batch will be of size $O(c/s) = \tilde{O}(\sqrt{T})$ in expectation. That is, through out the sequence of T updates, each $\mathcal{A}^{(j)}$ is expected to handles only $\tilde{O}(\sqrt{T})$ batches containing $\tilde{O}(\sqrt{T})$ updates.

This implementation would not change the correctness. But the whole algorithm can possibly be faster if $\mathcal{A}$ is a dynamic algorithm that can handle a batch update of size $\tilde{O}(\sqrt{T})$ faster than a sequence of $\tilde{O}(\sqrt{T})$ updates. This is a property that is quite natural to expect. Indeed, there are some dynamic algorithms such that the larger the batch the faster the update time on average, including dynamic matrix inverse and its applications such that dynamic reachability [59, 64]. However, these algorithms are not for approximating functions and so we cannot exploit them in this paper.

4 APPLICATIONS TO DYNAMIC GRAPH ALGORITHMS

In this section, we show new dynamic approximation algorithms against an adaptive adversary for four graph problems including, global minimum cut, all-pairs distances, effective resistances, and minimum cuts. Given a graph G undergoing edge updates, let n denote a number of vertices and m denote a *current* number of edges in G. In Section 4.1, we show how the generic reduction from Theorem 3.1 immediately transforms known algorithms against an oblivious adversary to work against an adaptive adversary with $o(m)$ update and query time. In Section 4.2, we show how to speed up the update time using sparsifiers and in Section 4.3 we discuss limitations of adaptive dynamic algorithms maintaining sparsifiers. Then, in Section 4.4, we show how to avoid these limitations and obtain a sparsification technique that allows us to assume that $m = \tilde{O}(n)$ all the time and speed up our algorithms.

Throughout this section, $\tilde{O}$ hides a polylog(n) factor. We also assume edge weights are integers of size at most poly(n). Also, to simplify the calculation, we will assume $\epsilon = \Omega(1)$ in all of our dynamic $(1+\epsilon)$-approximation algorithms.

4.1 Applying the Generic Reduction

The update time of our dynamic algorithms in this subsection depends on m. We assume that m never changes by more than a constant factor, because otherwise, we can restart the algorithm from scratch which would increase the amortized update time by at most a constant factor.

We start with the first dynamic algorithm against an adaptive adversary for $(1+\epsilon)$-approximate global mincut.

Corollary 4.1 (Global minimum cuts)**.** *For every constant $\epsilon > 0$, there is a dynamic algorithm against an adaptive adversary that, given an unweighted graph G undergoing edge insertions and deletions, with probability $1-1/\text{poly}(n)$,[6] maintains a $(1+\epsilon)$-approximate value of the global mincut in $\tilde{O}(m^{1/2}n^{1/4}) = \tilde{O}(m^{3/4})$ amortized update time.*

Proof. We simply apply Theorem 3.1 to the dynamic algorithm against an oblivious adversary by Thorup [63, Theorem 11]. When the graph initially has m edges, his algorithm takes $\tilde{O}(m)$ preprocessing time[7] and $\tilde{O}(\sqrt{n})$ worst-case update time. So the total update time for handling T updates (for any T) is

$$t_{\text{total}} = \tilde{O}(m + T\sqrt{n}).$$

Thorup's algorithm maintains the $(1+\epsilon/3)$-approximation of the global mincut explicitly, so we can query it in $t_q = O(1)$ time. By plugging this into Theorem 3.1 where $\alpha = \epsilon/3$, since $(1+\epsilon/3)\cdot(1+\alpha) \le (1+\epsilon)$, we obtain an $(1+\epsilon)$-approximation algorithm against an adaptive adversary with amortized update time

$$\tilde{O}\left(\frac{m + T\sqrt{n}}{\sqrt{T}}\right).$$

This amortized update time is minimized for $T \approx m/\sqrt{n}$. Therefore if we rebuild our data structure after $T \approx m/\sqrt{n}$ updates we get an amortized update time of $\tilde{O}(m^{1/2}n^{1/4})$. □

Corollary 4.2 (All-pairs distances)**.** *There is a dynamic algorithm against an adaptive adversary that, given a weighted graph G, handles the following operations in $\tilde{O}(m^{4/5})$ amortized update time:*

- *Insert or delete an edge from the graph, and*
- *Given $s, t \in V(G)$, with probability at least $1 - 1/\text{poly}(n)$, return a $(\log(n)\cdot\text{poly}(\log\log n))$-approximation of the distance between s and t.*

Proof. Chen et al. [31, Lemma 7.15 and the proof of Theorem 7.1] give a dynamic algorithm $\mathcal{A}$ against an oblivious adversary with the following guarantee. Given a weighted graph G with n vertices and m edges and any parameter j, the algorithm preprocesses G and with probability $1 - 1/\text{poly}(n)$ handles at most $T = O(j)$ operations in $t_{\text{total}} = \tilde{O}(m^2/j)$ total update time. The operations that $\mathcal{A}$ can handle include:

- edge insertions and deletions, and
- given (s,t), return a $(\log(n)\cdot\text{poly}(\log\log n))$ approximation of the (s,t)-distance in $t_q = \tilde{O}(j)$ time.

We want to apply the transformation from Theorem 3.1 to $\mathcal{A}$, but there is a small technical issue. In Theorem 3.1, we only consider algorithms that maintain one single number, but $\mathcal{A}$ can return an answer for any pair (s,t). So we instead assume that $\mathcal{A}$ also maintains a pair of variables $(\texttt{src}, \texttt{snk})$. Each (s,t)-query to $\mathcal{A}$ contains two sub-steps: first we update $(\texttt{src}, \texttt{snk}) \leftarrow (s,t)$ and then $\mathcal{A}$ returns the answer for $(\texttt{src}, \texttt{snk})$, which is now the only single number that $\mathcal{A}$ maintains. So we can indeed apply Theorem 3.1 to $\mathcal{A}$.

Applying Theorem 3.1 to $\mathcal{A}$, we obtain an algorithm against an adaptive adversary with amortized update time of

$$\tilde{O}\left(\frac{m^2/j}{\sqrt{j}} + j\right).$$

By choosing $j = m^{4/5}$ (and restarting after every j updates) we get an amortized update time of $\tilde{O}(m^{4/5})$. □

Next, we show that by plugging another oblivious algorithm into the reduction, we can speed up the above result.

[6] In this paper, when we say $1/\text{poly}(n)$, we mean that it stands for every polynomial that is greater than 1.

[7] The preprocessing time is not explicitly stated in [63]. This preprocessing includes, graph sparsification via uniform sampling, greedily packing $\tilde{O}(1)$ trees, and initializing information inside each tree. All of these takes near-linear time.

Corollary 4.3 (All-pairs distances with cruder approximation)**.** *For any integer $i \geq 2$, there is a dynamic algorithm against an adaptive adversary that, given a weighted graph G, handles the following operations in $m^{1/2+1/(2i)+o(1)}$ amortized update time:*

- *Insert or delete an edge from the graph, and*
- *Given $s, t \in V(G)$, with probability at least $1 - 1/\mathrm{poly}(n)$, return a $O(\log^{3i-2} n)$-approximation of the distance between s and t.*

Proof. Forster, Goranci, and Henzinger [41, Theorem 5.1] show a dynamic algorithm $\mathcal{A}$ against an oblivious adversary that can handle edge insertions and deletions in $m^{1/i+o(1)}$ amortized update time *when an initial graph is an empty graph* and, given $s, t \in V(G)$, can return a $O(\log^{3i-2} n)$-approximate (s,t)-distance in $\mathrm{polylog}(n)$ time. Again, we can use the same small modification as in Corollary 4.2 to view as $\mathcal{A}$ as an algorithm that maintain only one number.

When an initial graph is not empty but has m edges, algorithm $\mathcal{A}$ can handle T operations of edge updates and queries in at most $t_{\text{total}} = (m+T) \cdot m^{1/i+o(1)}$ time with query time is $t_q = \mathrm{polylog}(n)$. Therefore, via Theorem 3.1 we obtain an algorithm $\mathcal{A}'$ against an adaptive adversary with amortized update time of

$$\tilde{O}\left(\frac{(m+T) \cdot m^{1/i+o(1)}}{\sqrt{T}} + \mathrm{polylog}(n)\right).$$

This is $\tilde{O}\left(m^{1/2+1/(2i)+o(1)}\right)$ if we restart the structure every $T = m^{1-1/i}$ updates. □

Corollary 4.4 (All-pairs effective resistance)**.** *There is a dynamic algorithm against an adaptive adversary that, given a weighted graph G, handles the following operations in amortized update time $m^{1/4}n^{1/2+o(1)}$:*

- *Insert or delete an edge from the graph, and*
- *Given $s, t \in V(G)$, with probability at least $1 - 1/\mathrm{poly}(n)$, return a $(1+\epsilon)$-approximation of the effective resistance between s and t.*

Proof. Chen et al. [31, Proof of Theorem 8.1] give a dynamic algorithm $\mathcal{A}$ against an oblivious adversary with the following guarantee. Given a weighted graph G with n vertices and m edges and any parameters β and d, the algorithm preprocesses G in $t_p = O(\frac{m}{\beta^5} \cdot (\frac{\log n}{\epsilon})^{O(1)})$ time and then handles the following operations in $t_u = O(\beta^{-2d+3}(\frac{\log n}{\epsilon})^{O(d)})$ amortized time:

- edge insertions and deletions, and
- given (s,t), update $(\mathtt{src}, \mathtt{snk}) \leftarrow (s,t)$.

That is, given T operations above, the total update time is $t_{\text{total}} = t_p + T \cdot t_u$. The algorithm also supports queries for a $(1+\epsilon)$-approximation of the $(\mathtt{src}, \mathtt{snk})$-effective resistance in

$$t_q = O\left(n\beta^d \left(\frac{\log n}{\epsilon}\right)^{O(d)}\right)$$

time. We set $d = \omega(1)$, say $d = O(\log \log n)$, and will set $1/\beta = (n^2/m)^{\Theta(1/d)}$ meaning that $1/\beta = n^{o(1)}$. This implies that

$$\left(\frac{\log n}{\epsilon}\right)^{O(d)} / \beta^{O(1)} = n^{o(1)}$$

(recall that we assume ϵ to be a constant), which will help us simplifying several factors in the update time below.

By plugging $\mathcal{A}$ into Theorem 3.1, we obtain an algorithm against an adaptive adversary with amortized update time of

$$\tilde{O}\left(\frac{t_p + T \cdot t_u}{\sqrt{T}} + t_q\right) = \left(\frac{m + T \cdot \beta^{-2d}}{\sqrt{T}} + n\beta^d\right) \cdot n^{o(1)}$$

Now, to balance the first two terms in the bound, we set $T = m\beta^{2d}$. This gives the bound of

$$(\sqrt{m}\beta^{-d} + n\beta^d) \cdot n^{o(1)} = m^{1/4}n^{1/2+o(1)}$$

by setting $\beta^d = m^{1/4}/n^{1/2}$ to balance the terms. Indeed, $1/\beta = (n^2/m)^{\Theta(1/d)}$ as promised. □

4.2 Speed up via Sparsification against an Adaptive Adversary

In the rest of this section, we show how to speed up the update time of the algorithms from Section 4.1. The idea is simple. Given a dynamic graph G with n vertices, we want to use a dynamic algorithm against an adaptive adversary for maintaining a *sparsifier* H of G where H preserves a certain structure of G (e.g. cuts, distances, or effective resistances) but H contains at most $\tilde{O}(n)$ edges, and then apply the algorithms Section 4.1 on H. So the final update time only depends on n and not m.

Preliminaries on graph sparsification. To formally use this idea, we recall some definitions related to sparsification of graphs. Given a (weighted) graph G, we say that H is an *α-spanner* of G if H is a subgraph of G and the distances between all pairs of vertices $s, t \in G$ are preserved with in a factor of α, i.e.

$$\mathrm{dist}_G(u,v) \leq \mathrm{dist}_H(u,v) \leq \alpha \cdot \mathrm{dist}_G(u,v).$$

We say that H is an *α-cut sparsifier* of G if, for any cut $(S, V(G) \setminus S)$, we have that

$$\delta_G(S) \leq \delta_H(S) \leq \alpha \cdot \delta_G(S)$$

where $\delta_G(S)$ and $\delta_H(S)$ denote the cut size of S in G and H, respectively. Also, we say that H is an *α-spectral sparsifier* of G, if for any vector $x \in \mathbb{R}^V$, we have that

$$x^\top L_G x \leq x^\top L_H x \leq \alpha \cdot x^\top L_G x$$

where L_G and L_H are the Laplacian matrices of G and H, respectively.

Fact 4.5. *Suppose that H is an α-spectral sparsifier of G.*

- *H is an α-cut sparsifier of G (see e.g. [61]).*
- *The effective resistance in G between all pairs (s,t) are approximately preserved in H up to a factor of α. That is,*

$$R_G(u,v) \leq R_H(u,v) \leq \alpha \cdot R_G(u,v)$$

for all $u, v \in V$ where $R_G(u,v)$ and $R_H(u,v)$ denote the effective resistance between u and v in G and H, respectively (see [36, Lemma 2.5]).

Using dynamic spanners as a blackbox. Bernstein et al. [17] showed how to maintain a polylog(n)-spanner against an adaptive adversary efficiently.

Theorem 4.6 ([17, Theorem 1.1]). *There is a randomized dynamic algorithm against an adaptive adversary that, given an n-vertex graph G undergoing edge insertions and deletions, with high probability, explicitly maintains a* polylog(n)*-spanner H of size $\tilde{O}(n)$ using* polylog(n) *amortized update time.*

With the above theorem, we can immediately speed up Corollary 4.3 as follows:

Corollary 4.7 (All-pairs distances)**.** *For any integer $i \geq 2$, there is a dynamic algorithm against an adaptive adversary that, given a weighted graph G, handles the following operations in $n^{1/2+1/(2i)+o(1)}$ amortized update time:*

- *Insert or delete an edge from the graph, and*
- *Given $s, t \in V(G)$, with probability at least $1 - 1/\text{poly}(n)$, return a $O(\log^{3i+O(1)} n)$-approximation of the distance between s and t.*

Proof. Using Theorem 4.6, we maintain a polylog(n)-spanner H of G containing $\tilde{O}(n)$ edges in polylog(n) amortized update time. Since H can be maintained against an adaptive adversary, we think of H as our input graph and pay an additional polylog(n) approximation factor and update time. The argument now proceeds exactly in the same way as in the proof of Corollary 4.3 except that now the input graph always contains at most $\tilde{O}(n)$ edges. Since the update time of the algorithm of Theorem 4.6 is polylog(n), the number of updates to H is at most polylog(n) for every update to G. □

4.3 Current Limitation of Sparsification against an Adaptive Adversary

Here, we discuss the current limitation of dynamic sparsifiers against an adaptive adversary, which explains why we could not apply the same idea to get $(1+\epsilon)$-approximation algorithms against adaptive adversary.

Spanners. Observe that as long as we work on top of a polylog(n)-spanner, we will need to pay additional polylog(n) factor in the approximation factor. We can hope to improve this factor because, against an oblivious adversary, there actually exists an algorithm by Forster and Goranci [40] for maintaining a $(2k-1)$-spanner of size at most $\tilde{O}(n^{1+1/k})$ edges using only $O(k \log^2 n)$ amortized update time for any integer $k \geq 1$. This approximation-size trade-off is tight assuming the Erdos conjecture.

If there was a dynamic spanner algorithm with similar guarantees that works against an adaptive adversary, we would be able to reduce the additional approximation factor, due to sparsification, from polylog(n) to $2k-1$, while the update time would be slightly slower because the sparsifiers have size $\tilde{O}(n^{1+1/k})$ instead of $\tilde{O}(n)$. Unfortunately, it is an open problem if such a algorithm exists. This is the main reason why we could not state the sparsified version of Corollary 4.2 where the approximation ratio remains $\log(n) \cdot \text{poly}(\log \log n)$.

Open Question 4.8. *Is there a dynamic algorithm against an adaptive adversary for maintaining a $(2k-1)$-spanner of size $\tilde{O}(n^{1+1/k})$ using* polylog(n) *update time?*

Spectral sparsifiers. The situation is similar for spectral sparsifiers. Against an oblivious adversary, there exists a dynamic algorithm by Abraham et al. [1] for maintaining $(1+\epsilon)$-spectral sparsifier containing only $\tilde{O}(n)$ edges with polylog(n) amortized update time. If there was an algorithm against an adaptive adversary with the same guarantees, then we could immediately use it in the same manner as in Corollary 4.7 to speed up Corollary 4.4 by replacing m by n in their update time, while paying only an extra $(1+\epsilon)$-approximation ratio in the query. Unfortunately, whether such a dynamic algorithm for $(1+\epsilon)$-spectral sparsifier exists still remains a fascinating open problem.

Open Question 4.9. *Is there a dynamic algorithm against an adaptive adversary for maintaining a $(1+\epsilon)$-spectral sparsifier of size $\tilde{O}(n)$ using* polylog(n) *update time?*

The current start-of-the-art of dynamic spectral sparsifiser algorithms against an adaptive adversary still have large approximation ratio. In particular, Bernstein et al. [17] show how to maintain a polylog(n)-spectral sparsifier in polylog(n) update time, and also a $O(k)$-cut sparsifiers in $\tilde{O}(n^{1/k})$ update time. We could apply these algorithms to speed up Corollary 4.4, but then we must pay a large additional approximation factor.

Fortunately, in Section 4.4, we are able to show a way to work around this issue.

4.4 Speed up via Sparsification against a Blinking Adversary

In this section, we show that even if dynamic $(1+\epsilon)$-spectral sparsifiers against an adaptive adversary are not known, we can still apply the sparsification idea to the whole reduction. Below, (1) we describe the sparsification lemma in Lemma 4.12 and discuss why we can view it as an algorithm for an intermediate model between oblivious and adaptive adversaries which we call a *blinking adversary*, defined in Section 1.2.2 and then (2) we apply Lemma 4.12 to speed up our applications.

The algorithm is based on dynamic expander decomposition. We recall the definition of expanders here.

Definition 4.10. Given a weighted graph $G = (V, E, w)$ and a vertex set $S \subseteq V$, the *volume* of S is $\text{vol}_G(S) = \sum_{u \in S} \deg_G(u)$ where $\deg_G(u) = \sum_{(u,v) \in E} w(u,v)$ is the weighted degree of u in G. The size of a cut $(S, V \setminus S)$ in G is $\delta_G(S) = \sum_{u \in S, v \in V \setminus S} w(u,v)$. We say that G is a *ϕ-expander* if for any cut $(S, V \setminus S)$, its *conductance* is defined as $\Phi(G) = \min_{(S, V \setminus S)} \frac{\delta_G(S)}{\min\{\text{vol}_G(S), \text{vol}_G(V \setminus S)\}} \geq \phi$.

The following lemma says that for any graph $G = (V, E)$ with n vertices and m edges, there exists a partition/decomposition of edges of G into ϕ-expanders where each vertex appears in at most $O(\log^3 n)$ expanders on average. Moreover, this decomposition can be maintained against an adaptive adversary.

Theorem 4.11 (Dynamic expander decomposition [17, Theorem 4.3]). *For any $\phi = O(1/\log^4 m)$ there exists a dynamic algorithm against an adaptive adversary that preprocesses a weighted*

graph G with n vertices and m edges in $O(\phi^{-1}m\log^6 n)$ time. The algorithm maintains with probability $1-1/\text{poly}(n)$ a decomposition of G into ϕ-expanders $G_1,\ldots,G_z$ that is, every edge of G is exactly one G_i. The graphs $(G_i)_{1\le i\le z}$ are edge disjoint and we have $\sum_{i=1}^{z}|V(G_i)| = O(n\log^3 n)$. The algorithm supports edge deletions and insertions in $O(\phi^{-2}\log^7 n)$ amortized time. After each update, the output consists of a list of changes to the decomposition. The changes consist of (i) edge deletions or deletions of isolated vertices to some graphs G_i, (ii) removing some graphs G_i from the decomposition, and (iii) new graphs G_i are added to the decomposition.[8]

Actually the algorithm can be made deterministic when $\phi = O(1/2^{\log^{4/5} n}) = 1/n^{o(1)}$ by using the deterministic expander decomposition from [33]. This would only add an extra $n^{o(1)}$ factor in the update time of our applications, but in a first read, the reader may assume for simplicity that Theorem 4.11 is deterministic.

Given the dynamic expander decomposition from Theorem 4.11, we show that we can generate a $(1+\epsilon)$-spectral sparsifier in $\tilde{O}(n)$ time and even maintain it dynamically if the adversary is oblivious to the sparsifier. We emphasize that the time is independent of m and depends only on n.

Lemma 4.12. *Let* `cnt` *be the variable that counts the total number of edge changes in all ϕ-expanders $G_1,\ldots,G_z$ of G from Theorem 4.11 during a sequence of insertions and deletions. We can extend the algorithm from Theorem 4.11 so that it handles the following additional operation:*

- *Sparsify(t): return a graph H and continue maintaining H until* `cnt` *has increased by t using $\tilde{O}(n+t)$ total update time. During the sequence of edge updates before* `cnt` *has increased by t, H contains $\tilde{O}(n+t)$ edges and there are at most $\tilde{O}(t)$ edge changes in H. If these edge updates are fixed at the time Sparsify(t) is called (i.e., if the adversary is oblivious), then with high probability, H is a $(1+\epsilon)$-spectral sparsifier throughout the update sequence. We emphasize that each call to Sparsify(t) uses fresh random bits to initialize and maintain H in this call.*

The proof of Lemma 4.12 combines technical ingredients from [17] and is given in the full version of this paper [11].

Discussion: Blinking adversary. Assuming an underlying adaptive dynamic expander decomposition, we can think of the sparsifier of Lemma 4.12 as a sparsifier with a fast refresh capability: Each call to *sparsify* refreshes the sparsifier and makes it accurate independently of the past updates.

Suppose we always call Sparsify(t) whenever `cnt` increases by t. Then we, in fact, maintain using $\tilde{O}(n/t)$ amortized update time a $(1+\epsilon)$-spectral sparsifier H of G against a blinking adversary as define in Section 1.2.2. This is because, Lemma 4.12 guarantees accuracy of H for a fixed sequence of updates (before `cnt` increases by t), and each call to Sparsify(t) uses fresh random bits. So although the adversary observes H following the last call to Sparsify(t), this does not give it any useful information to fool the next call to Sparsify(t).

[8] Theorem 4.3 in [17] is stated only for unweighted graphs. However, it is straightforward to extend the algorithm to weighted graphs by grouping edges of G by weights. As there are $O(\log n)$ groups assuming that edge weights are at most $\text{poly}(n)$, this only add an extra factor of $O(\log n)$ to all the bounds.

This model of an adversary lies between an oblivious adversary that cannot observe the algorithm's answers at all and an adaptive adversary that can observe the algorithm's answers after every update. Interestingly, we show that algorithms against this intermediate model of adversary can be used for speeding up some of the applications obtained by our generic reduction. Specifically, we show the following Corollary.

Corollary 4.13 (All-pairs effective resistance (sparsified))**.** *There is a dynamic algorithm against an adaptive adversary that, given a weighted graph G, handles the following operations in $n^{3/4+o(1)}$ amortized update time:*

- *Insert or delete an edge from the graph, and*
- *Given $s,t\in V(G)$, with probability at least $1-1/\text{poly}(n)$, return a $(1+\epsilon)$-approximation of the distance between s and t.*

Proof. First of all, we maintain the expander decomposition of G in the background using the algorithm from Theorem 4.11 with $\tilde{O}(1)$ amortized update time when $\phi = \Theta(1/\log^4 m)$. Let `cnt` be the variable that counts the total number of edge changes in all ϕ-expanders from the decomposition.

We proceed in phases and restart the phase whenever `cnt` has increased by T since the beginning of the phase (we will later set $T=\sqrt{n}$).[9] For each phase, our goal is to show a data structure for $(1+O(\epsilon))$-approximating effective resistance between vertices of G against an *oblivious* adversary whose update time is independent of m. Once this is done, we can apply Theorem 3.1 to make it work against an adaptive adversary. Let $c=\tilde{O}(\sqrt{T})$ be the number of copies of the oblivious algorithms in the reduction of Theorem 3.1.

To achieve this goal, at the beginning of the phase, we compute $H^{(1)},\ldots,H^{(c)}$ using Lemma 4.12 where each $H^{(j)}=$ Sparsify(T) are independently generated. For each instance $\mathcal{A}^{(j)}$ of the oblivious dynamic algorithm for effective resistance by Chen [31, Proof of Theorem 8.1], we treat $H^{(j)}$ as its input graph. As in the proof of Corollary 4.4, we recall that the guarantee of this algorithm here:

Given parameters β and d, the algorithm preprocesses $H^{(j)}$ in $t_p = O(\frac{|E(H^{(j)})|}{\beta^5}\cdot(\frac{\log n}{\epsilon})^{O(1)})$ time. Then, the algorithm supports edge-update operations and $(\mathtt{src},\mathtt{snk})$-update operations in $t_u = O(\beta^{-2d+3}(\frac{\log n}{\epsilon})^{O(d)})$ amortized update time. At any time, the algorithm supports queries for a $(1+\epsilon)$-approximation of the $(\mathtt{src},\mathtt{snk})$-effective resistance in $t_q = O(n\beta^d(\frac{\log n}{\epsilon})^{O(d)})$ time. Similar to Corollary 4.4, we will set $d=\omega(1)$, say $d=O(\log\log n)$, and will set $\beta^d = 1/n^{1/4}$ and so $1/\beta = n^{1/4d} = n^{o(1)}$. This implies that $(\frac{\log n}{\epsilon})^{O(d)}/\beta^{O(1)} = n^{o(1)}$ (recall that we assume ϵ to be a constant).

During the phase, `cnt` can increase by at most T and so by Lemma 4.12 there are at most $T_H=\tilde{O}(T)$ updates to $H^{(j)}$. By substituting parameters, during the phase, the total update time of $\mathcal{A}^{(j)}$ for handling T_H updates in H is $t_{\text{total}} = \tilde{O}(t_p + T_H t_u) = (n+T\sqrt{n})n^{o(1)}$, and the query time is $t_q = n^{3/4+o(1)}$. Since we can assume that the adversary for $\mathcal{A}^{(j)}$ is oblivious, Lemma 4.12 guarantees that $H^{(j)}$ remains a $(1+\epsilon)$-spectral sparsifier of G

[9] It is possible that after one update to G, `cnt` increases by more than T. This means that within one update to G, there can be more than one phase.

throughout the phase. Therefore, each $\mathcal{A}^{(j)}$ indeed answers $(1+O(\epsilon))$-approximation of $(\mathtt{src}, \mathtt{snk})$-effective resistance in G.

By applying Theorem 3.1, we can maintain $(1+O(\epsilon))$-approximate solution of G against an adaptive adversary where the amortized update time for each phase is

$$\tilde{O}\left(\frac{t_{\text{total}}}{\sqrt{T_H}} + t_q\right) = \left(\frac{n + T\cdot\sqrt{n}}{\sqrt{T}} + n^{3/4}\right)\cdot n^{o(1)}.$$

Now, to balance the first two terms in the bound, we set $T = \sqrt{n}$. This gives the bound of $n^{3/4+o(1)}$. The additional time for maintaining $H^{(1)}, \ldots, H^{(c)}$ is $\frac{c\times\tilde{O}(n+T)}{T} = \frac{\tilde{O}(\sqrt{T}\cdot(n+T))}{T} = \tilde{O}(n^{3/4})$ amortized update time. Hence, the total amortized update time is $n^{3/4+o(1)}$. □

REFERENCES

[1] Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. 2016. On fully dynamic graph sparsifiers. In *Proceedings of the 57th IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 335–344.

[2] Noga Alon, Omri Ben-Eliezer, Yuval Dagan, Shay Moran, Moni Naor, and Eylon Yogev. 2021. Adversarial laws of large numbers and optimal regret in online classification. In *Proceedings of the 53rd ACM Symposium on Theory of Computing (STOC)*. ACM, 447–455.

[3] Idan Attias, Edith Cohen, Moshe Shechner, and Uri Stemmer. 2021. A Framework for Adversarial Streaming via Differential Privacy and Difference Estimators. *CoRR* abs/2107.14527 (2021).

[4] Victor Balcer and Salil P. Vadhan. 2019. Differential Privacy on Finite Computers. *J. Priv. Confidentiality* 9, 2 (2019). https://doi.org/10.29012/jpc.679

[5] Raef Bassily, Kobbi Nissim, Adam Smith, Thomas Steinke, Uri Stemmer, and Jonathan Ullman. 2021. Algorithmic stability for adaptive data analysis. *SIAM J. Comput.* 50, 3 (2021), 77–405.

[6] Surender Baswana, Shreejit Ray Chaudhury, Keerti Choudhary, and Shahbaz Khan. 2016. Dynamic DFS in undirected graphs: Breaking the $O(m)$ barrier. In *Proceedings of the 27th ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 730–739.

[7] Surender Baswana, Manoj Gupta, and Sandeep Sen. 2018. Fully Dynamic Maximal Matching in $O(\log n)$ Update Time (Corrected Version). *SIAM J. Comput.* 47, 3 (2018), 617–650. https://doi.org/10.1137/16M1106158

[8] Surender Baswana, Sumeet Khurana, and Soumojit Sarkar. 2012. Fully dynamic randomized algorithms for graph spanners. *ACM Trans. Algorithms* 8, 4 (2012), 35:1–35:51. https://doi.org/10.1145/2344422.2344425 Announced at SODA'08.

[9] Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. 2019. Fully Dynamic Maximal Independent Set with Polylogarithmic Update Time. In *Proceedings of the 60th IEEE Symposium on Foundations of Computer Science (FOCS)*. 382–405. https://doi.org/10.1109/FOCS.2019.00032

[10] Amos Beimel, Iftach Haitner, Nikolaos Makriyannis, and Eran Omri. 2018. Tighter Bounds on Multi-Party Coin Flipping via Augmented Weak Martingales and Differentially Private Sampling. In *Proceedings of the 59th IEEE Symposium on Foundations of Computer Science (FOCS)*. 838–849.

[11] Amos Beimel, Haim Kaplan, Yishay Mansour, Kobbi Nissim, Thatchaphol Saranurak, and Uri Stemmer. 2021. Dynamic Algorithms Against an Adaptive Adversary: Generic Constructions and Lower Bounds. *CoRR* abs/2111.03980 (2021). arXiv:2111.03980 https://arxiv.org/abs/2111.03980

[12] Amos Beimel, Kobbi Nissim, and Uri Stemmer. 2016. Private Learning and Sanitization: Pure vs. Approximate Differential Privacy. *Theory Comput.* 12, 1 (2016), 1–61.

[13] Mihir Bellare and Phillip Rogaway. 1993. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS)*. 62–73.

[14] Omri Ben-Eliezer, Talya Eden, and Krzysztof Onak. 2021. Adversarially Robust Streaming via Dense-Sparse Trade-offs. *CoRR* abs/2109.03785 (2021).

[15] Omri Ben-Eliezer, Rajesh Jayaram, David P Woodruff, and Eylon Yogev. 2020. A framework for adversarially robust streaming algorithms. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems*. 63–80.

[16] Aaron Bernstein. 2017. Deterministic Partially Dynamic Single Source Shortest Paths in Weighted Graphs. In *Proceedings of the 44th International Colloquium on Automata, Languages and Programming (ICALP) (LIPIcs, Vol. 80)*. 44:1–44:14.

[17] Aaron Bernstein, Jan van den Brand, Maximilian Probst Gutenberg, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, and He Sun. 2020. Fully-dynamic graph sparsifiers against an adaptive adversary. *arXiv preprint arXiv:2004.08432* (2020).

[18] Aaron Bernstein and Shiri Chechik. 2016. Deterministic decremental single source shortest paths: beyond the $O(mn)$ bound. In *Proceedings of the 48th ACM Symposium on Theory of Computing (STOC)*. 389–397.

[19] Aaron Bernstein and Shiri Chechik. 2017. Deterministic partially dynamic single source shortest paths for sparse graphs. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 453–469.

[20] Aaron Bernstein, Maximilian Probst, and Christian Wulff-Nilsen. 2019. Decremental strongly-connected components and single-source reachability in near-linear time. In *Proceedings of the 51st ACM Symposium on Theory of Computing (STOC)*. 365–376.

[21] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. 2015. Deterministic Fully Dynamic Data Structures for Vertex Cover and Matching. In *Proceedings of the 26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

[22] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. 2016. New deterministic approximation algorithms for fully dynamic matching. In *Proceedings of the 48th ACM Symposium on Theory of Computing (STOC)*. 398–411.

[23] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. 2017. Fully Dynamic Approximate Maximum Matching and Minimum Vertex Cover in $O(\log^3 n)$ Worst Case Update Time. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 470–489.

[24] Sayan Bhattacharya and Peter Kiss. 2021. Deterministic Rounding of Dynamic Fractional Matchings. *arXiv preprint arXiv:2105.01615* (2021).

[25] Sayan Bhattacharya and Janardhan Kulkarni. 2019. Deterministically Maintaining a $(2+\epsilon)$-Approximate Minimum Vertex Cover in $O(1/\epsilon^2)$ Amortized Update Time. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

[26] Vladimir Braverman, Avinatan Hassidim, Yossi Matias, Mariano Schain, Sandeep Silwal, and Samson Zhou. 2021. Adversarial Robustness of Streaming Algorithms through Importance Sampling. *arXiv preprint arXiv:2106.14952* (2021).

[27] Mark Bun, Kobbi Nissim, Uri Stemmer, and Salil Vadhan. 2015. Differentially private release and learning of threshold functions. In *Proceedings of the 56th IEEE Symposium on Foundations of Computer Science (FOCS)*. 634–649.

[28] Ran Canetti, Oded Goldreich, and Shai Halevi. 1998. The Random Oracle Methodology, Revisited. In *Proceedings of the 30th ACM Symposium on Theory of Computing (STOC)*. 209–218.

[29] Amit Chakrabarti, Prantar Ghosh, and Manuel Stoeckl. 2022. Adversarially Robust Coloring for Graph Streams. In *Proceedings of the 13th Innovations in Theoretical Computer Science Conference, (ITCS) (LIPIcs, Vol. 215)*. 37:1–37:23.

[30] Shiri Chechik and Tianyi Zhang. 2019. Fully Dynamic Maximal Independent Set in Expected Poly-Log Update Time. In *Proceedings of the 60th IEEE Symposium on Foundations of Computer Science (FOCS)*. 370–381. https://doi.org/10.1109/FOCS.2019.00031

[31] Li Chen, Gramoz Goranci, Monika Henzinger, Richard Peng, and Thatchaphol Saranurak. 2020. Fast dynamic cuts, distances and effective resistances via vertex sparsifiers. In *Proceedings of the 61st IEEE Symposium on Foundations of Computer Science (FOCS)*. 1135–1146.

[32] Julia Chuzhoy. 2021. Decremental all-pairs shortest paths in deterministic near-linear time. In *Proceedings of the 53rd ACM Symposium on Theory of Computing (STOC)*. 626–639. https://doi.org/10.1145/3406325.3451025

[33] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. 2020. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *Proceedings of the 61st IEEE Symposium on Foundations of Computer Science (FOCS)*. 1158–1167.

[34] Julia Chuzhoy and Sanjeev Khanna. 2019. A new algorithm for decremental single-source shortest paths with applications to vertex-capacitated flow and cut problems. In *Proceedings of the 51st ACM Symposium on Theory of Computing (STOC)*. 389–400.

[35] Julia Chuzhoy and Thatchaphol Saranurak. 2021. Deterministic Algorithms for Decremental Shortest Paths via Layered Core Decomposition. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2478–2496.

[36] David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng. 2019. Fully dynamic spectral vertex sparsifiers and applications. In *Proceedings of the 51st ACM Symposium on Theory of Computing (STOC)*. 914–925.

[37] Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Leon Roth. 2015. Preserving statistical validity in adaptive data analysis. In *Proceedings of the 47th ACM Symposium on Theory of Computing (STOC)*. 117–126.

[38] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the 3rd Theory of Cryptography Conference, (TCC) (Lecture Notes in Computer Science, Vol. 3876)*. Springer, 265–284.

[39] Cynthia Dwork, Guy N. Rothblum, and Salil P. Vadhan. 2010. Boosting and Differential Privacy. In *Proceedings of the 51st IEEE Symposium on Foundations of Computer Science (FOCS)*. 51–60. https://doi.org/10.1109/FOCS.2010.12

[40] Sebastian Forster and Gramoz Goranci. 2019. Dynamic low-stretch trees via dynamic low-diameter decompositions. In *Proceedings of the 51st ACM Symposium on Theory of Computing (STOC)*. 377–388.

[41] Sebastian Forster, Gramoz Goranci, and Monika Henzinger. 2021. Dynamic maintenance of low-stretch probabilistic tree embeddings with applications. In

Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA). 1226–1245.

[42] Varun Gupta, Christopher Jung, Seth Neel, Aaron Roth, Saeed Sharifi-Malvajerdi, and Chris Waites. 2021. Adaptive Machine Unlearning. *arXiv preprint arXiv:2106.04378* (2021).

[43] Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. 2020. New Algorithms and Hardness for Incremental Single-Source Shortest Paths in Directed Graphs. In *Proceedings of the 52nd ACM Symposium on Theory of Computing (STOC)*. 153–166. https://arxiv.org/abs/2001.10751

[44] Maximilian Probst Gutenberg and Christian Wulff-Nilsen. 2020. Decremental SSSP in weighted digraphs: Faster and against an adaptive adversary. In *Proceedings of the 31st ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2542–2561.

[45] Maximilian Probst Gutenberg and Christian Wulff-Nilsen. 2020. Deterministic algorithms for decremental approximate shortest paths: Faster and simpler. In *Proceedings of the 31st ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2522–2541.

[46] Moritz Hardt and Jonathan R. Ullman. 2014. Preventing False Discovery in Interactive Data Analysis Is Hard. In *Proceedings of the 55th IEEE Symposium on Foundations of Computer Science (FOCS)*. 454–463.

[47] Avinatan Hassidim, Haim Kaplan, Yishay Mansour, Yossi Matias, and Uri Stemmer. 2020. Adversarially Robust Streaming Algorithms via Differential Privacy. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020*. https://proceedings.neurips.cc/paper/2020/hash/0172d289da48c48de8c5ebf3de9f7ee1-Abstract.html

[48] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. 2014. Decremental single-source shortest paths on undirected graphs in near-linear total update time. In *Proceedings of the 55th IEEE Symposium on Foundations of Computer Science (FOCS)*. 146–155.

[49] Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. 2001. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM (JACM)* 48, 4 (2001), 723–760.

[50] Haim Kaplan, Katrina Ligett, Yishay Mansour, Moni Naor, and Uri Stemmer. 2020. Privately Learning Thresholds: Closing the Exponential Gap. In *Proceedings of the 33rd Conference on Learning Theory, COLT (Proceedings of Machine Learning Research, Vol. 125)*. 2263–2285.

[51] Haim Kaplan, Yishay Mansour, Kobbi Nissim, and Uri Stemmer. 2021. Separating Adaptive Streaming from Oblivious Streaming Using the Bounded Storage Model. In *Proceedings of Advances in Cryptology – CRYPTO 2021 (Lecture Notes in Computer Science, Vol. 12827)*. 94–121.

[52] David R. Karger. 2000. Minimum cuts in near-linear time. *J. ACM* 47, 1 (2000), 46–76. https://doi.org/10.1145/331605.331608

[53] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. 2017. Accessing Data while Preserving Privacy. *CoRR* abs/1706.01552 (2017).

[54] Frank McSherry and Kunal Talwar. 2007. Mechanism Design via Differential Privacy. In *Proceedings of the 48th IEEE Symposium on Foundations of Computer Science (FOCS)*. 94–103.

[55] Danupon Nanongkai and Thatchaphol Saranurak. 2017. Dynamic spanning forest with worst-case update time: adaptive, Las Vegas, and $O(n^{1/2-\epsilon})$-time. In *Proceedings of the 49th ACM Symposium on Theory of Computing (STOC)*. 1122–1129. https://doi.org/10.1145/3055399.3055447

[56] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. 2017. Dynamic Minimum Spanning Forest with Subpolynomial Worst-Case Update Time. In *Proceedings of the 58th IEEE Symposium on Foundations of Computer Science (FOCS)*. 950–961.

[57] Kobbi Nissim and Uri Stemmer. 2019. Concentration Bounds for High Sensitivity Functions Through Differential Privacy. *J. Priv. Confidentiality* 9, 1 (2019).

[58] Liam Roditty and Uri Zwick. 2008. Improved dynamic reachability algorithms for directed graphs. *SIAM J. Comput.* 37, 5 (2008), 1455–1471.

[59] Piotr Sankowski and Marcin Mucha. 2010. Fast dynamic transitive closure with lookahead. *Algorithmica* 56, 2 (2010), 180–197.

[60] Shay Solomon. [n. d.]. Fully Dynamic Maximal Matching in Constant Update Time. In *Proceedings of the 57th IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE press, 325–334.

[61] Daniel A Spielman and Shang-Hua Teng. 2011. Spectral sparsification of graphs. *SIAM J. Comput.* 40, 4 (2011), 981–1025.

[62] Thomas Steinke and Jonathan Ullman. 2017. Tight lower bounds for differentially private selection. In *Proceedings of the 58th IEEE Symposium on Foundations of Computer Science (FOCS)*. 552–563.

[63] Mikkel Thorup. 2007. Fully-dynamic min-cut. *Combinatorica* 27, 1 (2007), 91–127.

[64] Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. 2019. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *Proceedings of the 60th IEEE Symposium on Foundations of Computer Science (FOCS)*. 456–480.

[65] David Wajc. 2020. Rounding Dynamic Matchings Against an Adaptive Adversary. In *Proceedings of the 52nd ACM Symposium on Theory of Computing (STOC)*. 194–207. http://arxiv.org/abs/1911.05545

[66] David P Woodruff and Samson Zhou. 2020. Tight bounds for adversarially robust streams and sliding windows via difference estimators. *arXiv preprint arXiv:2011.07471* (2020).

[67] Christian Wulff-Nilsen. 2017. Fully-dynamic minimum spanning forest with improved worst-case update time. In *Proceedings of the 49th ACM Symposium on Theory of Computing (STOC)*. 1130–1143. https://doi.org/10.1145/3055399.3055415