Deterministic Decremental Reachability, SCC, and Shortest Paths via Directed Expanders and Congestion Balancing

Aaron Bernstein
Department of Computer Science
Rutgers University New Brunswick
New Brunswick, US
bernstei@gmail.com

Maximilian Probst Gutenberg
Department of Computer Science
University of Copenhagen and BARC
Copenhagen, Denmark
maximilian.probst@outlook.com

Thatchaphol Saranurak
Toyota Technological Institute at Chicago
Chicago, US
saranurak@ttic.edu

***Abstract*—Let $G = (V, E, w)$ be a weighted, directed graph subject to a sequence of adversarial edge deletions. In the decremental single-source reachability problem (SSR), we are given a fixed source s and the goal is to maintain a data structure that can answer path-queries $s \rightarrowtail v$ for any $v \in V$. In the more general single-source shortest paths (SSSP) problem the goal is to return an approximate shortest path to v, and in the SCC problem the goal is to maintain strongly connected components of G and to answer path queries within each component. All of these problems have been very actively studied over the past two decades, but all the fast algorithms are randomized and, more significantly, they can only answer path queries if they assume a weaker model: they assume an *oblivious* adversary which is not adaptive and must fix the update sequence in advance. This assumption significantly limits the use of these data structures, most notably preventing them from being used as subroutines in static algorithms.**

All the above problems are notoriously difficult in the adaptive setting. In fact, the state-of-the-art is still the Even and Shiloach tree, which dates back all the way to 1981 [1] and achieves total update time $O(mn)$. We present the first algorithms to break through this barrier.

- ***deterministic* decremental SSR/SCC with total update time $mn^{2/3+o(1)}$**
- ***deterministic* decremental SSSP with total update time $n^{2+2/3+o(1)}$**

To achieve these results, we develop two general techniques for working with dynamic graphs. The first generalizes expander-based tools to dynamic directed graphs. While these tools have already proven very successful in undirected graphs, the underlying expander decomposition they rely on does not exist in directed graphs. We thus need to develop an efficient framework for using expanders in directed graphs, as well as overcome several technical challenges in processing directed expanders. We establish several powerful primitives that we hope will pave the way for other expander-based algorithms in directed graphs.

The second technique, which we call *congestion balancing*, provides a new method for maintaining flow under adversarial deletions. The results above use this technique to maintain an embedding of an expander. The technique is quite general, and to highlight its power, we use it to achieve the following additional result:

- **The first near-optimal algorithm for decremental bipartite matching**

***Keywords*-dynamic algorithm, single-source shortest paths, single-source reachability, strongly-connected components**

I. Introduction

Let $G = (V, E, w)$ be a weighted, directed graph that is subject to dynamic updates that change the edges of G. We consider three closely related problems. In single-source reachability (SSR), we are given a fixed source s, and the goal is to maintain a data structure that can answer path queries $s \rightarrowtail v$ for any $v \in V$. The single-source shortest path problem (SSSP) is a generalization of SSR where the goal is return an approximate shortest path from s to v. Finally, in dynamic strongly-connected components (SCC), the goal is to maintain a data structure such that given any two vertices $u, v \in V$, it can determine whether they are in the same SCC, i.e. whether u and v are on a common cycle in G, and if yes, can report a path between them in either direction.

All three of the above problems have received an enormous amount of attention in the dynamic setting. The most general model is the *fully dynamic* one, where each adversarial update can either insert or delete an edge from G. But in this model there are very strong conditional lower bounds for all the above problems [2], [3].

For this reason, much of the work on these problems focuses on the weaker *decremental* model, where the algorithm is given some input graph $G = (V, E, w)$, and the adversary deletes one edge at a time until the graph is empty. Here, results are typically expressed in terms of the *total* update time over the entire sequence of deletions. Let n be the number of vertices in the original input graph, m the number of edges. The first algorithm for these problems is the Even and Shiloach tree [1] from 1981, which achieves total update time $O(mn)$ (amortized $O(n)$); See [4] for a simple extension to directed graphs. A long line of work has since led to near-optimal algorithms for these problems in *undirected* graphs, including some in the fully dynamic model [5], [6], [7], [8], [9], [10], [11], [12], [13]. The directed version is more difficult, but a series of results culminated in a near-optimal total update time $\tilde{O}(m)$ for decremental SSR/SCC [14], [15], [16], [17], [18] and moderate improvements for decremental SSSP: for example, total update time $\tilde{O}(mn^{3/4})$ in [19] and an extremely recent $\tilde{O}(n^2)$ result [20].

But all of the above $o(mn)$ algorithms for directed graphs suffer from a crucial drawback: they are randomized, and more significantly, they are only able to return paths if they assume an *oblivious adversary*. Such an adversary cannot change its updates based on the algorithm's answers to path-queries: put otherwise, the adversary must fix its entire update sequence in advance. Much of the recent work in the field of dynamic graphs as a whole has focused on developing so-called adaptive algorithms that do not assume an oblivious adversary. This is important for two reasons. Firstly, adaptive algorithms work in a less restrictive model. Secondly, several recent papers have used dynamic graph algorithms as subroutines within the multiplicative-weight update method to speed up *static* algorithms; for example, decremental shortest paths to speed up various (static) flow algorithms [21], [22], [23], or incremental min-cut to speed up a TSP algorithm [24]. These applications to static algorithms all require *adaptive* dynamic algorithms.

Despite all the progress for non-adaptive algorithms, the fastest adaptive algorithm for all the directed problems mentioned above remains the Even and Shiloach tree from 1981, which has total update time $O(mn)$. In this paper, we present the first algorithms to break through this barrier.

Theorem I.1. *Let G be a directed graph. There exists an algorithm for decremental single-source reachability and decremental strongly connected components (SCC) with total update time $mn^{2/3+o(1)}$. The SCC algorithm not only explicitly maintains SCCs, but can answer path queries within an SCC. The algorithms can, respectively, determine whether a vertex v is reachable from s, or whether two vertices are in the same SCC, in $O(1)$ time. The time to answer a path query is $P \cdot n^{o(1)}$, where P is the length of the (simple) output path.*

Theorem I.2. *Let G be a directed graph with positive weights and let W be the ratio of the largest to smallest weight. There exists an algorithm for decremental $(1+\epsilon)$-approximate single-source shortest paths with total update time $n^{2+2/3+o(1)}\log(W)/\epsilon$. (An update can delete an edge or increase an edge weight.) The query time is $O(1)$ for returning an approximate distance and $|P| \cdot n^{o(1)}$ for an approximate path, where $|P|$ is the length of the (simple) output path.*

Related Work: Probst Gutenberg and Wulff-Nilsen considered a relaxed version of decremental SSSP that can only return *distance estimates*, not an actual path. They showed an adaptive (randomized) algorithm for this problem with total update time $\tilde{O}(m^{2/3}n^{4/3}) = \tilde{O}(n^{2+2/3})$ [19]. The adaptivity of this result crucially depends on the assumption that the adversary cannot see the paths used by the algorithm, so these results cannot be extended to the problems we are solving in this paper. Secondly, there are several results (both adaptive and oblivious) on dynamic SSC/SSSP in the *incremental* setting, where the algorithm starts with an empty graph and edges are *inserted* one at a time (see e.g. [25], [26], [27], [28]). These incremental-only results use a very different set of techniques that do not transfer to the decremental setting.

Directed expanders, key objects in this paper, are closely related to the notion of *directed tree-width* introduced in [29], [30], which is a key concept in deep structural statements, including the directed grid-minor theorem [31], [32] and the directed Erdos-Posa theorem [33], [34], [35].[1] The approximation algorithm for a variant of the disjoint paths problem by [36] exploits the *directed well-linked decomposition* which is related to directed expander decomposition stated in this paper. However, their technique is static and not concerned with time-efficiency beyond polynomial time.

A. Techniques

Our techniques are mostly very different from those of the earlier randomized algorithms, because those crucially relied on "hiding" their choices from an oblivious adversary. Our algorithms instead rely on expander-based tools. While these have previously been used to break long-standing barriers for adaptive algorithms in dynamic *undirected* graphs [10], [11], [12], [22], [23], our paper is to first to successfully apply them to dynamic algorithms for directed graphs. Our results require a large number of new techniques; we highlight the most significant ones below.

An efficient framework for directed expanders (Section V): Expander-based algorithms in undirected graphs rely on the following basic decomposition: given any graph $G = (V, E)$, it is possible to partition E into sets X and R, such that X is the union of disconnected expanders, and $|R| \ll |E|$. The idea is then to use expander-tools on X and deal with the small set R separately. Unfortunately, such a guarantee is not possible for directed graphs: if G is a dense DAG, then R must contain all the edges of G.

This paper explicitly shows the following decomposition for directed graphs: E can be partitioned into three sets X, D, R such that X is the union of disconnected (directed) expanders, D is acyclic, and $|R| \ll |X|$. (We actually use an analogous decomposition for vertex expanders.) We then use this decomposition as the crux of our new framework, which weaves together new fast algorithms for directed expanders with existing fast algorithms for DAGs. We hope that this framework will pave the way for future work that applies expander-tools to directed graphs.

Congestion-balancing flow (Section VI): One of our main technical contributions is a new approach to maintaining a large flow in the presence of adversarial edge

[1]In particular, directed expanders are graphs that contain a large *well-linked set* [36], [37] and directed tree-width of a graph is approximated, up to a constant, by the maximum size over all well-linked sets [29].

deletions (it is new to undirected graphs as well). Intuitively, a flow solution is more robust if it spreads out the congestion among all the edges of the graph. There are, however, two main challenges to formalizing this intuition. The first is that some edges may be more "crucial" than others, so will necessarily have a higher congestion. The second is that these crucial edges might change over time, whereupon the flow must be rebalanced. We introduce a general approach for efficiently computing the "right" congestion of each edge. We then show that a potential function based on minimum-cost flow allows us to cleanly analyze the total amount of rebalancing necessary.

In our decremental SSR/SCC/SSSP results, we use congestion-balancing flow to maintain an embedding of an expander. But the technique is quite general, and to highlight its power, we use it achieve significantly improved bounds for the seemingly unrelated problem of decremental bipartite matching (see below).

New Primitives for Directed Expanders: Our new framework requires generalizing the essential expander primitives to directed graphs. While some of the primitives transfer almost automatically (e.g. unit flow), others pose significant technical challenge. We highlight two in particular.

In *expander pruning* we are given an expander $G = (V, E)$ subject to adversarial edge deletions. The goal of pruning is to dynamically maintain a set of pruned vertices $P \subseteq V$ such that the induced graph $G[V \setminus P]$ remains an expander. There are two known approaches to pruning in undirected graphs [12], [38], but both break down in directed graphs because a sparse cut in one direction may not be sparse in the other. Our approach takes inspiration from [12], but requires a different key subroutine to work in directed graphs. In addition to generalizing the result of [12], our approach also ends up being simpler and cleaner.

The *cut-matching game* is the well-known tool for certifying expansion of graphs and was first introduced in [39]. There are two state-of-the-art variants: one is randomized but works in directed graphs [40], while the second recent variant is deterministic but limited to undirected graphs [13]. In this paper, we develop a cut-matching game that achieves the best of both worlds: it is deterministic and works in directed graphs. To do this, we generalize several of the lemmas in [41] to bound a more complex entropy-based potential function, and generalize the key subroutine for the cut player in [13] to work directed graphs.

Both our pruning result and our new cut-matching game are stated as black-box results that can easily be plugged into other algorithms. Given how essential these tools have proven in undirected graphs, we think it is likely our contributions will prove useful for future work on directed expanders.

B. An Additional Result: Decremental Bipartite Mathing

As mentioned above, along the way to our main results we develop improved algorithms for dynamic matching. Consider the problem of maintaining a $(1-\epsilon)$-approximate maximum matching in an unweighted dynamic graph. In the fully dynamic setting, although there is a wide literature on faster update times for larger approximations, the best known update time for a $(1-\epsilon)$ approximation is $O(\sqrt{m})$ [42], and there is evidence that $O(\sqrt{m})$ is a hard barrier to break through [3], [43]. For this reason, there has been a series of upper and lower bounds in the more relaxed incremental model, where the algorithm starts with an empty graph and edges are only inserted [44], [45], [46], [47]. Most relevantly to our result, there is an incremental $(1-\epsilon)$-approximation with amortized $O(\log^2 n)$ update time in bipartite graphs [46], later improved to $O(1)$ update time in general graphs [47]. But the techniques of both papers are restricted to the incremental setting, and nothing analogous is known for decremental graphs; in fact, here $O(\sqrt{m})$ remained the best-known.

We show that a simple application of our congestion-balancing flow technique yields a near-optimal algorithm for $(1-\epsilon)$-approximate matching in decremental *bipartite* graphs; achieving a similar result for non-bipartite graphs remains an open problem. See Section VI-A for details.

Theorem I.3. *Let G be an unweighted bipartite graph. There exists a decremental algorithm with total update time $O(m\log^3(n)/\epsilon^4)$ (amortized $O(\log^3(n)/\epsilon^4)$) that maintains an integral matching M of value at least $\mu(G)(1-\epsilon)$, where G always refers to the current version of the graph. The algorithm is randomized, but works against an adaptive adversary; if we allow the algorithm to return a fractional matching instead of an integral one, then it is deterministic.*

II. Organisation

In the next section, we define notation for the rest of the article. We then provide an overview over our approach (Section IV) and then elaborate on the description of our main components (Section VI-A). Finally, we provide a sketch of result Theorem I.3 and provide some intuition how the main ingredient behind the result can be used to maintain expanders. We point out that this article is an extended abstract and due to the limitations in space, formal descriptions and full proofs are often deferred to the full version of the paper.

III. Preliminaries

We usually refer to n as the number of vertices in a graph. We use $\tilde{O}(\cdot)$ and $\tilde{\Omega}(\cdot)$ to hide $\mathrm{poly}\log n$ factors in the big-oh notations. Similarly, we use $\widehat{O}(\cdot)$ and $\widehat{\Omega}(\cdot)$ to hide $n^{o(1)}$ factors.

Graphs in this paper are directed. Given a graph G, the *reverse graph* $G^{(\mathrm{rev})}$ of G is obtained from G by reversing the direction of every edge in G. For any subset $S, T \subseteq V$, $E(S,T)$ is a set of directed edges (u,v) where $u \in S$ and $v \in T$. Let $G[S]$ denote the induced subgraph on S. Let $w : E \to \mathbb{R}$ be an edge weight function of G. Given $F \subseteq E$, let $w(F) = \sum_{e \in F} w(e)$ be the total weight of F; more generally, for any function g on the edges $g(F) = \sum_{e \in E} g(e)$. The weighted in-degree and out-degree of a vertex u are $\deg^{in}(u) = w(E(V,u))$ and $\deg^{out}(u) = w(E(u,V))$, respectively. The weighted degree of u is $\deg(u) = \deg^{in}(u) + \deg^{out}(u)$. The volume of a set S is $\text{vol}(S) = \sum_{u \in S} \deg(u)$. Several of our subroutines on expanders will use small fractional weights.

For any S with $\text{vol}(S) \le \text{vol}(V \setminus S)$ we refer to $(S, V \setminus S)$ as a *cut* in G. Let $\delta^{out}(S) = w(E(S, V \setminus S))$ and $\delta^{in}(S) = w(E(V \setminus S, S))$ denote the total weight of edges going out and coming in to S, respectively. We say that cut $(S, V \setminus S)$ is *ϵ-balanced* if $\text{vol}(S) \ge \epsilon \text{vol}(V)$, and it is ϕ-sparse if $\min\{\delta^{in}(S), \delta^{out}(S)\} < \phi \text{vol}(S)$. We say that (L, S, R) is a *vertex-cut* of G if L,S, and R partition the vertex set V, and either $E(L,R) = \emptyset$ or $E(R,L) = \emptyset$. Assuming that $|L| \le |R|$, (L,S,R) is *ϵ-vertex-balanced* if $|L| \ge \epsilon|V|$, and it is *ϕ-vertex-sparse* if $|S| < \phi|L|$. We add the subscript G to the notations whenever it is not clear which graph we are referring to.

We say that a data structure supports *SCC path-queries* in G, if given vertices u and v, it either correctly reports that u and v are not strongly connected in G in $O(1)$ time, or returns a directed simple path P_{uv} from u to v and a directed simple P_{vu} from v to u. We say that the data structure has *almost path-length query time* if, whenever a path P is returned, the data structure takes only $\widehat{O}(|P|)$ time to output the path. We emphasize that the returned path must be simple.[2]

A *decremental* graph G is a graph undergoing a sequence of deletions of edges and of isolated vertices. There is an easy reduction from decremental SSR from s to decremental SCC: just add an edge from every $v \in V$ to s.

IV. High-level Overview

We start with the definition of *expanders* which are the central object of this paper.

Definition IV.1 (Expanders)**.** A directed graph G is a *ϕ-vertex expander* if it has no ϕ-vertex-sparse vertex-cut. Similarly, G is *ϕ-(edge) expander* if it has no ϕ-sparse cut.[3]

Intuitively, expanders are graphs that are "robustly connected" and, in particular, they are strongly connected. It is well-known that many problems become much easier on expanders. So, given a problem on general graph, we would like to reduce the problem to expanders.

It turns out that every *undirected* graph admits the following *expander decomposition*: for any $\phi > 0$, a $\tilde{O}(\phi)$-fraction of vertices/edges can be removed so that the remaining is a set of vertex-disjoint ϕ-vertex/edge expander. Unfortunately, this is impossible in directed graphs. Consider, for example, a DAG. However, a DAG is the only obstacle; for any $\phi > 0$, we can remove $\tilde{O}(\phi)$-fraction of vertices/edges, so that the remaining part is a DAG of ϕ-vertex/edge expanders. This observation can be made precise as follows.[4]

Fact IV.2 (Directed Expander Decomposition)**.** *Let $G = (V,E)$ be any directed n-vertex graph and $\phi > 0$ be a parameter. There is a partition $\{R, X_1, \ldots, X_k\}$ of V such that*

1) *$|R| \le O(\phi n \log n)$;*
2) *$G[X_i]$ is a ϕ-vertex expander for each i;*
3) *Let D be obtained from G by deleting R and contracting each X_i. Then, D is a DAG.*

The edge version of Fact IV.2 can be stated as follows: for any unweighted m-edge graph $G = (V,E)$, there is a partition $\{X_1, \ldots, X_k\}$ of V and $R \subset E$ where $|R| \le O(\phi m \log m)$, each $G[X_i]$ is a ϕ-expander, and D is a DAG (where D is defined as above). It can be generalized to weighted graphs as well.

This decomposition motivates the framework of our algorithm, although for the sake of efficiency we only maintain an approximate version (see Invariant V.2 below.) The decomposition suggests that we need four main ingredients: **1)** a dynamic expander decomposition in directed graphs, **2)** a fast algorithm on vertex-expanders, **3)** a fast algorithm on DAGs, and **4)** a way to deal with the small remaining part R. Our algorithm will run in time $\widehat{O}(m|R|) = \widehat{O}(mn^{2/3})$, as we choose $\phi = n^{-1/3}$. Note that we do not work with edge-expanders because then R would have size $|R| = \tilde{O}(\phi m)$, which is too big for us. See Section V for how all components fit together.

Here, let us focus on fast algorithms on expanders. One of our main tasks is to certify that a given (sub)-graph G is a vertex-expander. This leads us to the notion of embedding:

Definition IV.3 (Embedding and Embedded Graph)**.** Let $G = (V,E)$ be a directed graph. An *embedding* $\mathcal{P}$ in G is a collection of simple directed paths in G where each path $P \in \mathcal{P}$ has associated *value* $\text{val}(P) > 0$. We say that $\mathcal{P}$ has *length* len if every path $P \in \mathcal{P}$ contains at most len edges. We say that $\mathcal{P}$ has *vertex-congestion* cong if, for every vertex $v \in V$, $\sum_{P \in \mathcal{P}_v} \text{val}(P) \le \mathsf{cong}$ where $\mathcal{P}_v$ is the set of paths in $\mathcal{P}$ containing v. We say that $\mathcal{P}$ has *edge-congestion* cong

[2]Otherwise one can arbitrarily increase the length of the returned path through cycles and hence it can be trivial to achieve almost path-length query time.

[3]Note that an isolated vertex is an expander (in both edge and vertex versions).

[4]Although this decomposition is easy to prove by simply recursively cutting a ϕ-sparse cut, it is never explicitly stated before to our best knowledge.

if, for every edge $e \in E$, $\sum_{P \in \mathcal{P}_e} \mathrm{val}(P) \leq \mathsf{cong}$ where $\mathcal{P}_e$ is the set of paths in $\mathcal{P}$ containing e.

Given an embedding $\mathcal{P}$, there is a corresponding weighted directed graph W where, for each path $P \in \mathcal{P}$ from u to v, there is a directed edge (u, v) with weight $\mathrm{val}(P)$. We call W an *embedded graph* corresponding to $\mathcal{P}$ and say that $\mathcal{P}$ embeds W into G.

The following fact shows that, to certify that G is a vertex expander, it is enough to *embed* an (edge)-expander W into G with small congestion.

Fact IV.4. *Let $G = (V, E)$ be a graph. Let $W = (V, E', w)$ be a ϕ-expander with minimum weighted degree 1. If W can be embedded into G with vertex congestion* cong*, then G is a (ϕ/cong)-vertex expander.*

Proof: Consider a vertex cut (L, S, R) in G where $|L| \leq |R|$. Suppose that $E(L, R) = \emptyset$, otherwise $E(R, L) = \emptyset$ and the proof is symmetric. Observe that each edge $e \in E_W(L, V \setminus L)$ in W corresponds to a path in G that goes out of L and, hence, must contain some vertex from S. So the total weight of these edges in W can be at most $\delta_W^{out}(L) \leq |S| \cdot \mathsf{cong}$. At the same time, $\delta_W^{out}(L) \geq \phi \mathrm{vol}_W(L) \geq \phi|L|$ as W is a ϕ-expander with minimum weighted degree 1. So $|S| \geq \frac{\phi}{\mathsf{cong}}|L|$ as desired. ∎

In our actual algorithm, instead of certifying that G is a vertex expander (i.e. G has no sparse vertex-cut), we relax to the task to only certifying that G has no balanced sparse vertex-cut. This motivates the definition of ϕ-witness which is used throughout the paper:

Definition IV.5 (Witness)**.** We say that W is a *ϕ-witness* of G if $V(W) \subseteq V(G)$, W is a $\widehat{\Omega}(1)$-(edge)-expander where $9/10$-fraction of vertices have weighted degree at least $1/2$, and there is an embedding of W into G with vertex-congestion $1/\phi$. (Note that $E(W)$ does not have to be a subset of $E(G)$.) We say that W is a ϕ-short-witness if it is a ϕ-witness and the embedding has length $\widehat{O}(1/\phi)$. We say that W is a *large* witness if $|V(W)| \geq 9|V(G)|/10$.[5]

We sometimes informally refer to a graph that contains a large witness as an *almost vertex-expander*. This is because of the below fact whose proof is similar to Fact IV.4.

Fact IV.6. *Let $G = (V, E)$ be a graph that contains a large ϕ-witness W. Then G has no $1/3$-vertex-balanced $(\phi/n^{o(1)})$-vertex-sparse vertex cut.*

Now, we have reduced the problem of certifying an almost vertex-expander to maintaining a large witness. Although finding a low congestion embedding in vertex expanders can be done very efficiently in the static setting (using the well known cut-matching game), there is one crucial obstacle in the dynamic setting.

[5]The constant $9/10$ is somewhat arbitrary.

Consider the following simple scenario. We start with a complete graph G and parameter $\phi = \widehat{\Omega}(1)$. A standard (static) construction of a large ϕ-witness runs in $\widehat{O}(m)$ time and gives an *unweighted* $\widehat{\Omega}(1)$-expander W where all vertex degrees are $\Theta(\log(n))$. Let $\mathcal{P}$ be the embedding of W. Observe that each path from $\mathcal{P}$ has value 1 and $|\mathcal{P}| = O(n \log n)$.

Unfortunately, once the adversary knows $\mathcal{P}$, he can destroy each embedding path $P \in \mathcal{P}$ by deleting any edge in P. In total, he can delete only $O(n \log n)$ edges in G to destroy the whole embedding of W. The algorithm would then have to construct a new witness, which the adversary could again destroy with $O(n \log n)$ deletions. This process continues until G has a balanced, sparse vertex-cut, which might not happen until $\Omega(n^2)$ deletions. That is, this standard approach requires the algorithm to re-embed a new witness $\tilde{\Omega}(n)$ times, which is not only slow, but requires too many changes to the witness.

To overcome this obstable, we use the idea called *congestion balancing* to maintain a witness W that only needs to be re-embedded $\tilde{O}(1/\phi)$ times throughout the entire sequence of deletions (formally stated in Theorem V.3). As a warm-up to the proof of Theorem V.3, we show in Section VI-A how to apply this idea to the simpler bipartite matching problem.

V. The Main Components

In this section, we state all the algorithmic components formally and show how to combine them to prove Theorem I.1. As we mentioned in Section IV, our framework needs **1)** A dynamic expander decomposition **2)** a fast algorithm on vertex expanders, **3)** a fast algorithm on DAGs, and **4)** a way to deal with the small remaining part $\hat{S}$.

It turns out that the existing algorithm of Lacki (unrelated to expanders) for separating out any small set of vertices [48] is a handy tool for taking care of the DAG part and the small remaining part, and allows us to focus on almost vertex expanders. This algorithm has previously used in a similar way in [16]. We state the algorithm as a reduction below and defer the proof to the full version of the paper.

Proposition V.1 (see [48], [16])**.** *Let $G = (V, E)$ be a decremental graph. Let $\mathcal{A}$ be a data structure that* **1)** *maintains a monotonically growing set $S \subseteq V$ and after every adversarial update reports any additions made to S and* **2)** *maintains the SCCs in $G \setminus S$ explicitly in total update time $T(m, n)$ and supports SCC path queries in $G \setminus S$ in almost-path-length query time.*

Then, there exists a data structure $\mathcal{B}$ that maintains the SCCs of G explicitly and supports SCC path-queries in G (in almost-path-length query time). The total update time is $O(T(m, n) + m|S| \log n)$, where $|S|$ refers to the final size of the set S.

As we usually use G to denote an input graph to each subroutine. We denote the input to the top-level algorithm by $G^* = (V^*, E^*)$. Motivated by the directed expander decomposition from Fact IV.2 and Lacki's reduction above, we maintain the following invariant:

Invariant V.2. *Our decremental SCC algorithm will maintain an incremental set $\hat{S}$ such that $|\hat{S}| = \widehat{O}(n^{2/3})$ and at the end of processing any update, if the (non-singleton) SCCs of $G \setminus S$ are $C_1, ..., C_k$, then each C_i contains a large $\widehat{\Omega}(1/n^{1/3})$-short-witness. To ensure that $\hat{S}$ remains small, the algorithm will only add set S to $\hat{S}$ if S corresponds to some* sparse *vertex cut (L, S, R).*

Robust Witness via Congestion-Balancing: Let G be some SCC in $G^* \setminus \hat{S}$ at some point during the update sequence. To preserve Invariant V.2, we need a subroutine that maintains a large ϕ-witness of G where $\phi = \widehat{\Omega}(1/n^{1/3})$. If the subroutine fails to find such a witness, it returns a $\Omega(1/n^{o(1)})$-balanced, ϕ-sparse vertex-cut (L, S, R); that is, it certifies that G is far from being a vertex expander, and must be further decomposed. (In particular, the top-level algorithm will add S to the boundary set $\hat{S}$ and recurse on both L and R.) Our new technique congestion-balancing flow will allow us to construct a *robust* witness that is suitable to the dynamic setting; see Section VI for more details.

Theorem V.3 (Robust Witness Maintenance)**.** *There is a deterministic algorithm* ROBUST-WITNESS(G, ϕ) *that takes as input a directed decremental n-vertex graph G and a parameter $\phi \in (0, 1/\log^2(n)]$. The algorithm maintains a large (weighted) ϕ-short-witness W of G using $\widehat{O}(m/\phi^2)$ total update time such that every edge weight in W is a positive multiple of $1/d$, for some number $d \leq 2d_{avg}$, where d_{avg} is the initial average degree of G. The total edge weight in W is $O(n \log n)$. After every edge deletion, the the algorithm either updates W or outputs a $(\phi n^{o(1)})$-vertex-sparse $(1/n^{o(1)})$-vertex-balanced vertex-cut and terminates.*

Let $W^{(i)}$ be W after the i-th update. There exists a set R of reset indices *where $|R| = \widehat{O}(\phi^{-1})$, such that for each $i \notin R$, $W^{(i)} \supseteq W^{(i+1)}$. That is, the algorithm has $\widehat{O}(\phi^{-1})$ phases such that, within each phase, W is a decremental graph. The algorithm reports when each phase begins. It explicitly maintains the embedding $\mathcal{P}$ of W into G and reports all changes made to W and $\mathcal{P}$.*

The reason that W only shrinks between each phase is as follows. Whenever the adversary deletes some edge e in an embedded path P that corresponds to an edge e' in W, we will delete e' from W. To guarantee that W remains an expander after edge deletions, we run our new expander pruning algorithm in directed graphs on W that further removes a small part from W and guarantees that the remaining is still an expander. Nevertheless, after too many deletions, W will be too small and we need to re-embed W.

To highlight the strength of this result, the above theorem shows we only needs to re-embed a witness $\widehat{O}(\phi^{-1})$ times throughout the entire sequence of deletions, whereas the standard technique might require $\tilde{\Omega}(n)$ re-embeddings in the worst case as mentioned in Section IV.

Maintaining Short Distances from a Witness: Consider some SCC G of $G^*[V^* \setminus \hat{S}]$ with a large ϕ-witness W. We build two separate data structures on G. The first, given any vertex $u \in V(G) \setminus V(W)$, returns a path between u and some $w \in V(W)$. The second can answer path queries for any $w_1, w_2 \in V(W)$. It is easy to see that the two combined can answer SCC path-queries in G. The statement of the first data structure is a bit subtle; we give a formal theorem, followed by some intuition for what the theorem statement means.

Theorem V.4. *There is a data structure* FOREST-FROM-WITNESS(G, W, ϕ) *that takes as input an n-vertex m-edge graph $G = (V, E)$, a set $W \subseteq V$ with $|W| \geq |V|/2$ and a parameter $\phi > 0$. The algorithm must process two kinds of updates. The first deletes any edge e from E; the second removes a vertex from W (but the vertex remains in V), while always obeying the promise that $|W| \geq |V|/2$. The data structure must maintain a forest of trees $\mathcal{F}_{out}$ such that every tree $T \in \mathcal{F}_{out}$ has the following properties: all edges of T are in $E(G)$; T is rooted at a vertex of W; every edge in T is directed away from the root; and T has depth $\widehat{O}(1/\phi)$. The data structure also maintains a forest $\mathcal{F}_{in}$ with the same properties, except each edge in T is directed towards the root.*

At any time, the data structure may perform the following operation: it finds a $\widehat{O}(\phi)$-sparse vertex cut (L, S, R) with $W \cap (L \cup S) = \emptyset$ and replace G with $G[R]$. (This operation is NOT an adversarial update, but is rather the responsibility of the data structure.) The data structure maintains the invariant that every $v \in V$ is present in exactly one tree from $\mathcal{F}_{out}$ and exactly one from $\mathcal{F}_{in}$; given any v, the data structure can report the roots of these trees in $O(\log(n))$ time. (Note that as V may shrink over time, this property only needs to hold for vertex v in the current *set V.) The total time spent processing updates and performing sparse-cut operations is $\widehat{O}(m/\phi)$.*

Although the data structure works for *any* set W, W will always correspond to a ϕ-witness in the higher-level algorithm. The adversarial update that removes a vertex from W corresponds to the event that the witness shrinks in the higher-level algorithm. The forests $\mathcal{F}_{in}$ and $\mathcal{F}_{out}$ allow the algorithm to return paths of length $\widehat{O}(1/\phi)$ from any $v \in V(G)$ to/from W: find the tree that contains v and follow the path to the root, which is always in W. The requirement that each tree has low-depth will be necessary to reduce the update time. But once we add this requirement, we encounter the issue that some vertices may be very far from W, so we need to give the data structure a way to remove them from $V(G)$. This is the role of the sparse-cut operation: we will show in the proof that if v is far from W, it is always possible to find a sparse vertex cut (L, S, R) such that v is in L and hence removed from G. (The higher-level algorithm will process this operation by adding S to $\hat{S}$, so that L becomes part of a different SCC in $G^*[V^* \setminus \hat{S}]$.)

Maintaining Paths Inside the Witness: The second data structure shows how to maintain short paths between all pairs of vertices in an (edge) expander. The input W will always correspond to a large ϕ-witness, and will thus have expansion $1/n^{o(1)}$. This data structure is not new to our paper, as it is essentially identical to an analogous structure for undirected graphs in [23]. The only major difference is that we need to plug in our new expander pruning algorithm for directed graphs. Note that the theorem below will only allow us to find paths in $E(W)$, not $E(G)$; we show later how to use the embedding of W to convert them to paths in $E(G)$.

Theorem V.5. *There is a deterministic data structure* PATH-INSIDE-EXPANDER(W) *that takes as input an n-vertex m-edge $1/n^{o(1)}$-expander W subject to decremental updates. Each update can delete an arbitrary batch of vertices and edges from W, but must obey the promise that the resulting graph remains a ϕ-expander. Given any query $u, v \in V(W)$, the algorithm returns in $n^{o(1)}$ time a directed simple path P_{uv} from u to v and a directed simple path P_{vu} of v to u, both of length at most $n^{o(1)}$. The total update time of the data structure is $\widehat{O}(m)$.*

A. The Algorithm

The proof of Theorem I.1 combines all the above ingredients. See Algorithm 1 for pseudocode.

Analysis Sketch: The argument has three main parts. The first is that each call SCC-HELPER(G) re-initializes data structure in Line 14 only $\widehat{O}(1/\phi^*)$ times, since that is the number of phases in ROBUST-WITNESS (Theorem V.3). The second is that every time a vertex v participates in a new call SCC-HELPER(G), $|V(G)|$ must have decreased by a $(1 - 1/n^{o(1)})$ factor, so v participates in $\widehat{O}(1)$ calls. The third is that we always have $|\hat{S}| = \widehat{O}(n\phi^*) = \widehat{O}(n^{2/3})$, because vertices added to $\hat{S}$ always correspond to a ϕ^*-sparse cut.

The basic idea for the query is that given any u, v in some SCC $C \in \mathcal{C}$ with Witness W, we use FOREST-FROM-WITNESS to find paths from u and v to W and use PATH-INSIDE-EXPANDER to complete the path inside W. The complication is that the resulting path P might not be simple. We can always extract a simple path $P' \subseteq P$, but the query time would be proportional to $|P|$, not $|P'|$. We thus need to use a more clever query procedure.

Comparison to Previous Work: Our framework combines many old and new techniques, so we briefly categorize them. Proposition V.1 and Theorem V.4 follow from ideas in two earlier papers [48], [16] that are unrelated to expanders.

Algorithm 1: Maintaining an SCC-oracle for the main graph G^* (Theorem I.1)

```
1  Initialize $\hat{S} \leftarrow \emptyset$, $\phi^* \leftarrow n^{-1/3}$, $\mathcal{C} \leftarrow \{V^*\}$ // $\mathcal{C}$ is
     the collection of SCCs in $G^* \setminus \hat{S}$
2  Initialize the framework of Proposition V.1
3  Run SCC-HELPER($G^*$)  // Will always run
     SCC-HELPER($C$) for every SCC $C \in \mathcal{C}$
4  Procedure Setup for SCC-HELPER($G$)
5      Initialize ROBUST-WITNESS($G, \phi^*$). Let $W$ be
         the large $\phi^*$-witness maintained
6      Initialize PATH-INSIDE-EXPANDER($W$)
7      Initialize FOREST-FROM-WITNESS($G, W, \phi^*$)
8  Procedure Updating the data structures in
     SCC-HELPER($G$)
9      All adversarial edge deletions are fed to
         ROBUST-WITNESS and
         FOREST-FROM-WITNESS
10     if ROBUST-WITNESS in Line 5 terminates with a
         cut $(L, S, R)$ then
11         $\hat{S} \leftarrow \hat{S} \cup S$; remove $V(G)$ from $\mathcal{C}$; add $L, R$
             to $\mathcal{C}$
12         Initialize SCC-HELPER($G[L]$) and
             SCC-HELPER($G[R]$)
13         Terminate call SCC-HELPER($G$) // $V(G)$
             is decomposed into $L$ and $R$
14     if ROBUST-WITNESS in Line 5 starts a new
         phase and hence creates a new $W$ then
15         Initialize new data structures
             PATH-INSIDE-EXPANDER($W$) and
             FOREST-FROM-WITNESS($G, W, \phi^*$) and
             terminate existing ones in Lines 6 and 7
16     if ROBUST-WITNESS deletes vertices/edges from
         $W$ within a phase then
17         Feed these deletions as a batch deletion to
             PATH-INSIDE-EXPANDER($W$)
18         if vertex $v$ is deleted from $W$ then feed to
             FOREST-FROM-WITNESS($G, W, \phi^*$) an
             update that removes $v$ from $W$
19     if FOREST-FROM-WITNESS returns a
         $\widehat{O}(\phi^*)$-sparse vertex cut $(L, S, R)$ and replaces
         $G$ with $G[R]$ then
20         $\hat{S} \leftarrow \hat{S} \cup S$; add $L$ to $\mathcal{C}$; replace $G \in \mathcal{C}$ with
             $G[R]$ // $L$ is removed from SCC $G$
21         Initialize SCC-HELPER($G[L]$) // L is a
             new SCC in $G^*[V^* \setminus \hat{S}]$
```

Theorem V.5 easily generalizes from an existing result for undirected graphs [23], but only once our new directed primitives are in place.

Our primary *new contributions* are threefold: **1)** A new framework which integrates dynamic expander decomposition with earlier tools for directed graphs in [48], [16], **2)** Robust witness maintenance and congestion-balancing flow, and **3)** New primitives for directed expanders – especially directed expander pruning and cut-matching game – which are crucial for Theorems V.3 and V.5 in this section.

VI. Maintaining a Witness via Congestion-Balancing Flow

In this section, we present Algorithm Robust-Witness from Theorem V.3. The algorithm has several components, but the main innovation is a new approach we call congestion-balancing flow. To highlight this approach, we first show how it can be used to yield new results for the simpler problem of decremental bipartite matching (Theorem I.3).

A. Warmup: Decremental Bipartite Matching

Informal Overview:: We focus on the following problem: say that we are given a bipartite graph $G_0 = (L_0 \cup R_0, E)$ with $|L_0| = n$ and $|E_0| = m$, and say that the graph has a perfect matching (i.e. $\mu(G_0) = n$). We assume that n is a power of 2. Let $\epsilon < 1$ be some fixed constant. Now, consider any adversarial sequence of edge deletions, and let G always refer to the current version of the graph. The algorithm must maintain a *fractional* matching in G of size $\geq (1-5\epsilon)n$ OR certify that $\mu(G) \leq (1-\epsilon)n$, at which point it can terminate. In other words, the algorithm must maintain a matching until $\mu(G)$ decreases by a $(1-\epsilon)$ factor. The total update time should be $\tilde{O}(m)$. This algorithm gets us most of the way to proving Theorem I.3. (The conversion from fractional to integral matching is done via the black-box of Wajc [49].)

Consider the following lazy approach. Start by computing a matching M of size $(1-\epsilon)n$ in $O(m)$ time (using e.g. Hopcroft-Karp [50]). The adversary must now delete $\Omega(\epsilon n)$ edges before M has size $< (1-5\epsilon)n$, at which point we compute a new matching. This algorithm is too slow: we spend $O(m)$ time to compute a matching that survives for $O(n)$ deletions, for a total update time of $O(m^2/n)$.

We would like to construct a robust matching that can survive for more than $\Omega(n)$ deletions. We will construct a fractional matching M that attempts to put low value on each edge; this way, the adversary must delete many edges to remove ϵn value from M. It may not be possible to put low value on *all* edges, as some edges may be "crucial" for any matching, but we present a technique for efficiently balancing the edge-congestion. We will then show that over the entire sequence of deletions there can only be a small number of crucial edges, so the adversary cannot profit too often from deleting them.

Our algorithm will run in phases. Each edge is given capacity $\kappa(e)$, which intuitively captures how crucial e is. The algorithm initially sets $\kappa(e) = 1/n$, but $\kappa(e)$ can increase over time; these capacities transfer between phases. At the beginning of each phase, we first run Hopcroft-Karp to ensure that $\mu(G) \sim (1-\epsilon)n$; if not, we can terminate. So we can assume that we always have $\mu(G) \geq (1-2\epsilon)n$. We now try to compute a fractional matching M such that $\mathrm{val}(M) \geq (1-4\epsilon)n$ and $\mathrm{val}(e) \leq \kappa(e)\ \forall e \in E$. If we find such an M, we use the lazy approach from before: we wait until the adversary deletes ϵn value from M, and then we initiate a new phase. If the algorithm fails to find such an M, it instead returns a cut C where the edge-capacities are too small. The algorithm then doubles $\kappa(e)$ for all $e \in C$, and again tries to compute a matching M. This process will eventually terminate because we know that $\mu(G) \geq (1-2\epsilon)n$; thus, once the edge-capacities are high enough, there will certainly be a matching M with $\mathrm{val}(M) \geq (1-4\epsilon)n$. (Note that we never increase $\kappa(e)$ beyond 1, because a matching already has vertex capacity 1, so any edges with capacity ≥ 1 effectively have infinite capacity.)

The crux of our algorithm is showing that the *total* number of doubling steps, across all phases, is only $O(\log(n))$. Assuming this fact, let $K = \sum_{e \in G_0} \kappa(e)$. We will show that each doubling step only doubles κ along a low-capacity cut, so K only increases by $O(n)$. Since the number of doubling steps is $O(\log(n))$, we always have $K = O(n \log(n))$. This upper bound on K in turn implies that there are only $O(\log(n))$ phases, because each phase must delete $\Omega(n)$ value from the matching M, which clearly involves deleting at least $\Omega(n)$ edge-capacity.

To show that the number of doubling steps is $O(\log(n))$, we introduce the following potential function $\Pi(G,\kappa)$. Let the *cost* of each e be $c(e) = \log(n\kappa(e))$. Now, let $\mathcal{M}$ be the set of all integral matchings M (ignoring edge capacities) of size at least $(1-2\epsilon)n$; recall from above that we can assume $\mathcal{M} \neq \emptyset$. Define $\Pi(G,\kappa)$ to be the minimum cost among all matchings from $\mathcal{M}$. It is easy to see that each $\Pi(G,\kappa)$ is initially zero and is non-decreasing. Moreover, since every edge has $\kappa(e) \leq 1$ and $c(e) \leq \log(n)$, we also have $\Pi(G,\kappa) = O(n \log(n))$ at all times. We now argue (at a high level) that each doubling step increases $\Pi(G,\kappa)$ by $\Omega(n)$. Let C be the cut that prevented the algorithm from finding a fractional matching M with $\mathrm{val}(M) \geq (1-4\epsilon)n$. Any integral matching $M \in \mathcal{M}$ has $\mathrm{val}(M) \geq (1-2\epsilon)n$, so it must have $\geq 2\epsilon n$ edges that cross C. Moreover, since the cut-capacity is small, $\Omega(\epsilon n)$ of these crossing edges must have capacity < 1. The doubling step then doubles $\kappa(e)$ for each such edge, increasing each $c(e)$ by 1, and thus increasing $c(M)$ by $\Omega(\epsilon n) = \Omega(n)$, as desired.

B. *Overview of Algorithm* Robust-Witness

The algorithm for maintaining a witness follows the same congestion-balancing approach as the decremental matching algorithm, but the details are significantly more involved.

The algorithm will again run in phases. Just as algorithm Robust-Matching began each phase by checking that the graph contains a large matching, now the algorithm checks that the graph contains a very large ϕ-witness; if not, the algorithm is able to find a sparse, balanced cut and terminate. From now on we assume such a witness exists.

As described in Section IV, an arbitrary embedding $\mathcal{P}$ might not be robust to adversarial deletions, because a small number of edges might have most of the flow. To balance the edge-congestion, we introduce a capacity $\kappa(e)$ on each edge. Initially we set $\kappa(e) = 1/d$, where d is the average degree in the input graph. At each step, the algorithms uses approximate flows and the cut-matching game to try to find a witness with vertex congestion $\tilde{O}(1/\phi)$ and edge-congestions $\kappa(e)$. If it fails, the subroutine finds a low-capacity cut C; it then doubles capacities in C and tries again. Since we assume a witness does exist, the algorithm will eventually find a witness once the edge-capacities are high enough.

Once we have a witness W with embedding $\mathcal{P}$, we use the lazy approach. Say the adversary deletes an edge (u, v). Because our embedding obeyed capacity constraints, this can remove at most edges from W of total weight at most $\kappa(u, v)$. To maintain expansion, we feed these deletions into our expander pruning algorithm to yield a pruned set P, and shrink our witness to $W[V(W) - P]$. To guarantee that W remains a large witness, we end the phase once the pruned set P it too large. We will show that we end a phase only after the adversary deletes $\widehat{\Omega}(n)$ edge-capacity from the graph.

As with Robust-Matching, the crux of our analysis will be to show that the total of number of doubling steps is $\widehat{O}(1/\phi)$. To do so, we again use costs $c(e) = \log(d\kappa(e))$ and use a potential function $\Pi(G, \kappa)$ which measures the *min-cost embedding* in G among all *very large* ϕ-witness. As the vertex congestion is $1/\phi$, this potential $\Pi(G, \kappa)$ is at most n/ϕ. Also, we are able to show that each doubling step increases the potential by $\widehat{\Omega}(n)$ using an argument that is more involved than the one for matching. Therefore, there are at most $\widehat{O}(1/\phi)$ doubling steps as desired.

Given this bound, we can bound the total number of phases: each doubling step adds at most n to the total capacity κ, and the initial capacity is at most $1/d \cdot m = n$. So the final total capacity is at most $\widehat{O}(n/\phi)$. As each phase must delete $\widehat{\Omega}(n)$ capacity, there are at most $\widehat{O}(1/\phi)$ phases.

VII. Conclusion

In this extended abstract, we provided an overview of algorithms to obtain three new algorithms for decremental graphs: 1) a deterministic algorithm with running time $mn^{2/3+o(1)}$ that can answer SCC and SSR queries, 2) a deterministic algorithm with running time $n^{2+2/3+o(1)}$ that maintains SSSP and 3) a randomized (but adaptive) algorithm that maintains matchings with near optimal running time $\tilde{O}(m)$.

Each of these algorithms is a significant improvement for the problem at hand, and especially the former two algorithms improve on the long-standing upper bound of $O(mn)$ by Even and Shiloach [1].

Our progress motivates the following related open questions:

- Can we find *deterministic* algorithms for SSR, SCC and directed SSSP that run in near-linear time? For SSR and SCC such an algorithm is known when randomization is allowed [18]. For directed SSSP, even obtaining a *randomized* (non-adaptive) algorithm with near-linear update time is a major open question (although this goal has been achieved for very dense graphs [20]). We also point out that while a randomized near-linear update time algorithm exists for undirected SSSP [51], even in this setting, the current best deterministic algorithms have running time $mn^{1/2+o(1)}$ and $\tilde{O}(n^2)$ [52], [53], [54], [55].
- Can we obtain deterministic algorithms for the directed decremental $(1+\epsilon)$-approximate All-Pairs Shortest-Path problem with near-optimal total running time $\tilde{O}(mn)$? Such an algorithm is currently only known in the randomized setting [56], however, the best deterministic algorithm runs in total update time $\tilde{O}(mn^2)$ [57].

Acknowledgment

Aaron Bernstein is supported by NSF CAREER Grant 1942010 and the Simons Group for Algorithms & Geometry. Maximilian Probst Gutenberg is supported by Basic Algorithms Research Copenhagen (BARC), supported by Thorup's Investigator Grant from the Villum Foundation under Grant No. 16582.

We are very grateful to Mira Bernstein for showing us how to lower-bound the increase of the entropy potential function in the directed cut-matching game, which is a crucial step in our framework. The first author would like to thank David Wajc for helping him work through the black-box in [49], which allows us to convert our dynamic algorithm for fractional matching into one for integral matching. We are grateful to Julia Chuzhoy for allowing us to apply the short-path oracle on expanders to our framework, which is crucial to obtain almost-path-length query time. This result is by directly translating the same subroutine for undirected graphs shown in [23] to directed graphs using our new primitives for directed graphs.

References

[1] S. Even and Y. Shiloach, "An on-line edge-deletion problem," *Journal of the ACM (JACM)*, vol. 28, no. 1, pp. 1–4, 1981.

[2] A. Abboud and V. V. Williams, "Popular conjectures imply strong lower bounds for dynamic problems," in *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*. IEEE Computer Society, 2014, pp. 434–443. [Online]. Available: https://doi.org/10.1109/FOCS.2014.53

[3] M. Henzinger, S. Krinninger, D. Nanongkai, and T. Saranurak, "Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture," in *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, 2015, pp. 21–30.

[4] M. R. Henzinger and V. King, "Fully dynamic biconnectivity and transitive closure," in *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, 1995, pp. 664–672.

[5] G. N. Frederickson, "Data structures for on-line updating of minimum spanning trees, with applications," *SIAM J. Comput.*, vol. 14, no. 4, pp. 781–798, 1985, announced at STOC'83. [Online]. Available: http://dx.doi.org/10.1137/0214055

[6] M. R. Henzinger and V. King, "Randomized fully dynamic graph algorithms with polylogarithmic time per operation," *J. ACM*, vol. 46, no. 4, pp. 502–516, 1999. [Online]. Available: https://doi.org/10.1145/320211.320215

[7] J. Holm, K. de Lichtenberg, and M. Thorup, "Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity," *J. ACM*, vol. 48, no. 4, pp. 723–760, 2001. [Online]. Available: https://doi.org/10.1145/502090.502095

[8] M. Thorup, "Near-optimal fully-dynamic graph connectivity," in *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, 2000, pp. 343–350. [Online]. Available: https://doi.org/10.1145/335305.335345

[9] M. Patrascu and E. D. Demaine, "Lower bounds for dynamic connectivity," in *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, 2004, pp. 546–553. [Online]. Available: https://doi.org/10.1145/1007352.1007435

[10] D. Nanongkai and T. Saranurak, "Dynamic spanning forest with worst-case update time: adaptive, Las Vegas, and $O(n^{1/2-\epsilon})$-time," in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, 2017, pp. 1122–1129. [Online]. Available: https://doi.org/10.1145/3055399.3055447

[11] C. Wulff-Nilsen, "Fully-dynamic minimum spanning forest with improved worst-case update time," in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, 2017, pp. 1130–1143. [Online]. Available: https://doi.org/10.1145/3055399.3055415

[12] D. Nanongkai, T. Saranurak, and C. Wulff-Nilsen, "Dynamic minimum spanning forest with subpolynomial worst-case update time," in *FOCS*. IEEE Computer Society, 2017, pp. 950–961.

[13] J. Chuzhoy, Y. Gao, J. Li, D. Nanongkai, R. Peng, and T. Saranurak, "A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond," in *2019 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2020.

[14] M. Henzinger, S. Krinninger, and D. Nanongkai, "Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs," in *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, 2014, pp. 674–683.

[15] ——, "Improved algorithms for decremental single-source reachability on directed graphs," in *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, 2015, pp. 725–736.

[16] S. Chechik, T. D. Hansen, G. F. Italiano, J. Lacki, and N. Parotsidis, "Decremental single-source reachability and strongly connected components in õ(m$\sqrt{}$n) total update time," in *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, 2016, pp. 315–324.

[17] G. F. Italiano, A. Karczmarz, J. Lacki, and P. Sankowski, "Decremental single-source reachability in planar digraphs," in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, H. Hatami, P. McKenzie, and V. King, Eds. ACM, 2017, pp. 1108–1121.

[18] A. Bernstein, M. Probst, and C. Wulff-Nilsen, "Decremental strongly-connected components and single-source reachability in near-linear time," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, 2019, pp. 365–376.

[19] M. P. Gutenberg and C. Wulff-Nilsen, "Decremental SSSP in weighted digraphs: Faster and against an adaptive adversary," in *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, S. Chawla, Ed. SIAM, 2020, pp. 2542–2561.

[20] A. Bernstein, M. P. Gutenberg, and C. Wulff-Nilsen, "Near-optimal decremental sssp in dense weighted digraphs," in *2019 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2020.

[21] A. Madry, "Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms," in *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, 2010, pp. 121–130. [Online]. Available: https://doi.org/10.1145/1806689.1806708

[22] J. Chuzhoy and S. Khanna, "A new algorithm for decremental single-source shortest paths with applications to vertex-capacitated flow and cut problems," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, 2019, pp. 389–400.

[23] J. Chuzhoy and T. Saranurak, "Deterministic decremental shortest path algorithms via nearly optimal layered core decomposition," *Unpublished*, 2020.

[24] C. Chekuri and K. Quanrud, "Approximating the held-karp bound for metric TSP in nearly-linear time," in *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, 2017, pp. 789–800.

[25] B. Haeupler, T. Kavitha, R. Mathew, S. Sen, and R. E. Tarjan, "Incremental cycle detection, topological ordering, and strong component maintenance," *ACM Trans. Algorithms*, vol. 8, no. 1, pp. 3:1–3:33, Jan. 2012.

[26] M. A. Bender, J. T. Fineman, S. Gilbert, and R. E. Tarjan, "A new approach to incremental cycle detection and related problems," *ACM Trans. Algorithms*, vol. 12, no. 2, Dec. 2015.

[27] A. Bernstein and S. Chechik, "Incremental topological sort and cycle detection in expected total time," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, 2018, pp. 21–34.

[28] M. P. Gutenberg, V. V. Williams, and N. Wein, "New algorithms and hardness for incremental single-source shortest paths in directed graphs," in *Symposium on Theory of Computing*, 2020. [Online]. Available: https://arxiv.org/abs/2001.10751

[29] B. A. Reed, "Introducing directed tree width," *Electron. Notes Discret. Math.*, vol. 3, pp. 222–229, 1999. [Online]. Available: https://doi.org/10.1016/S1571-0653(05)80061-7

[30] T. Johnson, N. Robertson, P. D. Seymour, and R. Thomas, "Directed tree-width," *J. Comb. Theory, Ser. B*, vol. 82, no. 1, pp. 138–154, 2001. [Online]. Available: https://doi.org/10.1006/jctb.2000.2031

[31] K. Kawarabayashi and S. Kreutzer, "The directed grid theorem," in *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, 2015, pp. 655–664. [Online]. Available: https://doi.org/10.1145/2746539.2746586

[32] M. Hatzel, K. Kawarabayashi, and S. Kreutzer, "Polynomial planar directed grid theorem," in *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, 2019, pp. 1465–1484. [Online]. Available: https://doi.org/10.1137/1.9781611975482.89

[33] B. A. Reed, N. Robertson, P. D. Seymour, and R. Thomas, "Packing directed circuits," *Combinatorica*, vol. 16, no. 4, pp. 535–554, 1996. [Online]. Available: https://doi.org/10.1007/BF01271272

[34] S. A. Amiri, K. Kawarabayashi, S. Kreutzer, and P. Wollan, "The erdos-posa property for directed graphs," *CoRR*, vol. abs/1603.02504, 2016. [Online]. Available: http://arxiv.org/abs/1603.02504

[35] T. Masarík, I. Muzi, M. Pilipczuk, P. Rzażewski, and M. Sorge, "Packing directed circuits quarter-integrally," in *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, 2019, pp. 72:1–72:13. [Online]. Available: https://doi.org/10.4230/LIPIcs.ESA.2019.72

[36] C. Chekuri and A. Ene, "The all-or-nothing flow problem in directed graphs with symmetric demand pairs," *Math. Program.*, vol. 154, no. 1-2, pp. 249–272, 2015. [Online]. Available: https://doi.org/10.1007/s10107-014-0856-z

[37] C. Chekuri, A. Ene, and M. Pilipczuk, "Constant congestion routing of symmetric demands in planar directed graphs," *SIAM J. Discrete Math.*, vol. 32, no. 3, pp. 2134–2160, 2018. [Online]. Available: https://doi.org/10.1137/17M1150694

[38] T. Saranurak and D. Wang, "Expander decomposition and pruning: Faster, stronger, and simpler," in *SODA*. SIAM, 2019, pp. 2616–2635.

[39] R. Khandekar, S. Rao, and U. V. Vazirani, "Graph partitioning using single commodity flows," *J. ACM*, vol. 56, no. 4, pp. 19:1–19:15, 2009. [Online]. Available: https://doi.org/10.1145/1538902.1538903

[40] A. Louis, "Cut-matching games on directed graphs," *CoRR*, vol. abs/1010.1047, 2010. [Online]. Available: http://arxiv.org/abs/1010.1047

[41] R. Khandekar, S. Khot, L. Orecchia, and N. K. Vishnoi, "On a cut-matching game for the sparsest cut problem," *Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2007-177*, 2007.

[42] M. Gupta and R. Peng, "Fully dynamic (1+ e)-approximate matchings," in *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*. IEEE Computer Society, 2013, pp. 548–557.

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

[44] S. Dahlgaard, "On the hardness of partially dynamic graph problems and connections to diameter," in *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, and D. Sangiorgi, Eds., vol. 55, 2016, pp. 48:1–48:14.

[45] B. Bosek, D. Leniowski, P. Sankowski, and A. Zych, "Online bipartite matching in offline time," in *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, 2014, pp. 384–393. [Online]. Available: https://doi.org/10.1109/FOCS.2014.48

[46] M. Gupta, "Maintaining approximate maximum matching in an incremental bipartite graph in polylogarithmic update time," in *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, 2014, pp. 227–239. [Online]. Available: https://doi.org/10.4230/LIPIcs.FSTTCS.2014.227

[47] F. Grandoni, S. Leonardi, P. Sankowski, C. Schwiegelshohn, and S. Solomon, "$(1 + \epsilon)$-approximate incremental matching in constant deterministic amortized time," in *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, 2019, pp. 1886–1898.

[48] J. Lacki, "Improved deterministic algorithms for decremental transitive closure and strongly connected components," in *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, D. Randall, Ed. SIAM, 2011, pp. 1438–1445. [Online]. Available: https://doi.org/10.1137/1.9781611973082.111

[49] D. Wajc, "Rounding dynamic matchings against an adaptive adversary," in *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, 2020, pp. 194–207.

[50] J. E. Hopcroft and R. M. Karp, "An nˆ5/2 algorithm for maximum matchings in bipartite graphs," *SIAM Journal on computing*, vol. 2, no. 4, pp. 225–231, 1973.

[51] M. Henzinger, S. Krinninger, and D. Nanongkai, "Decremental single-source shortest paths on undirected graphs in near-linear total update time," in *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, 2014, pp. 146–155.

[52] A. Bernstein and S. Chechik, "Deterministic decremental single source shortest paths: beyond the o(mn) bound," in *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, 2016, pp. 389–397. [Online]. Available: https://doi.org/10.1145/2897518.2897521

[53] ——, "Deterministic partially dynamic single source shortest paths for sparse graphs," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2017, pp. 453–469.

[54] M. P. Gutenberg and C. Wulff-Nilsen, "Deterministic algorithms for decremental approximate shortest paths: Faster and simpler," in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2020, pp. 2522–2541.

[55] A. Bernstein, J. v. d. Brand, M. P. Gutenberg, D. Nanongkai, T. Saranurak, A. Sidford, and H. Sun, "Fully-dynamic graph sparsifiers against an adaptive adversary," *arXiv preprint arXiv:2004.08432*, 2020.

[56] A. Bernstein, "Maintaining shortest paths under deletions in weighted directed graphs," *SIAM Journal on Computing*, vol. 45, no. 2, pp. 548–574, 2016.

[57] C. Demetrescu and G. F. Italiano, "A new approach to dynamic all pairs shortest paths," *Journal of the ACM (JACM)*, vol. 51, no. 6, pp. 968–992, 2004.