Near-Optimal Deterministic Vertex-Failure Connectivity Oracles

Yaowei Long  
Computer Science and Engineering Division  
University of Michigan  
Ann Arbor, USA  
yaowei@umich.edu

Thatchaphol Saranurak  
Computer Science and Engineering Division  
University of Michigan  
Ann Arbor, USA  
ths@umich.edu

Abstract—We revisit the vertex-failure connectivity oracle problem. This is one of the most basic graph data structure problems under vertex updates, yet its complexity is still not well-understood. We essentially settle the complexity of this problem by showing a new data structure whose space, preprocessing time, update time, and query time are simultaneously optimal up to sub-polynomial factors assuming popular conjectures. Moreover, the data structure is deterministic.

More precisely, for any integer $d_\star$, the data structure preprocesses a graph $G$ with $n$ vertices and $m$ edges in $\tilde{O}(md_\star)$ time and uses $O(\min(m, nd_\star))$ space. Then, given the vertex set $D$ to be deleted where $|D| = d \leq d_\star$, it takes $\tilde{O}(d^2)$ updates time. Finally, given any vertex pair $(u, v)$, it checks if $u$ and $v$ are connected in $G \setminus D$ in $O(d)$ time. This improves the previously best deterministic algorithm by Duan and Pettie [SICOMP 2020] in both space and update time by a factor of $d$. It also significantly speeds up the $\Omega(\min(mn, n^2))$ preprocessing time of all known (even randomized) algorithms with update time at most $\tilde{O}(d^5)$.

I. INTRODUCTION

We revisit the vertex-failure connectivity oracle problem [2]–[4]. This is one of the most basic graph data structure problems under vertex updates, yet its complexity is still not well-understood. In this paper, we essentially settle the complexity of this problem by showing a new data structure whose space, preprocessing time, update time, and query time are simultaneously optimal up to sub-polynomial factors assuming popular conjectures. Moreover, the data structure is deterministic.

More precisely, in the vertex-failure connectivity oracle problem, there are three phases. First, we preprocess an undirected graph $G = (V, E)$ with $n$ vertices and $m$ edges and an integer $d_\star$. Second, given a vertex set $D \subseteq V$ of size $|D| = d \leq d_\star$, we update the data structure. Finally, given a vertex pair $(u, v)$, we return whether $u$ and $v$ are connected in the updated graph $G \setminus D$. The preprocessing, update, and query time are the time used by the data structure in the first, second, and third phases respectively. To put it another way, this problem is just the well-known decremental connectivity problem when all vertex deletions are given in one batch, instead of an online sequence.

The full version of this paper is available as [1] at https://arxiv.org/abs/2205.03930.

To see the context, we also mention its sister problem, the edge-failure connectivity oracle problem, where we are given $d$ edges to be deleted instead of vertices. Since 1997, Pastrascu and Thorup [5] already showed a near-optimal deterministic edge-failure oracle, except that it has large preprocessing time. This is later improved to a near-optimal oracle with $\tilde{O}(m)$ space, $\tilde{O}(m)$ preprocessing, $\tilde{O}(d)$ update, and $O(\log \log n)$ query time using the linear sketching techniques (see Theorem 7.9 of [4] and also [6], [7]).

The state of the art for vertex-failure connectivity oracles is much worse. The high-level reason why vertex failures are more challenging is, while $d$ edge deletions cause at most $d$ new connected components, just a single vertex deletion may create a lot of new connect components. For very small $d_\star \leq 2$, classical graph decompositions such that block trees and SQRT trees yield optimal vertex-failure oracles. When $d_\star = 3$, the data structure by Kanevsky et al. [8] implies a near-optimal oracle with $O(n)$ space, $O(m)$ preprocessing, $O(1)$ update and query time. It was until 2010 when Duan and Pettie [2] gave the first solution for general $d_\star$ by showing a deterministic data structure with $\tilde{O}(md_\star \cdot n^{1/2} \cdot \log(2d^\star) \cdot \log \log(1/d))$ space and preprocessing time, $\tilde{O}(d^2 + 4)$ update and $O(d)$ query time. Later in 2017, they [4] showed two improved algorithms: (1) a deterministic data structure with $\tilde{O}(md_\star)$ space, $\tilde{O}(mn)$ preprocessing, $\tilde{O}(d^2)$ update, and $O(d)$ query time, and (2) a randomized Monte Carlo data structure with $\tilde{O}(m)$ space, $\tilde{O}(mn)$ preprocessing, $\tilde{O}(d^2)$ update, and $O(d)$ query time. More recently, oracles whose update and query time depend only on $d_\star$ or $d$ were shown [9], [10], but the bounds are significantly slower than the ones in [4] when polylog($n$) factors are ignored.

Given the big discrepancy between the edge-failure and vertex-failure cases in all time bounds, a natural question is whether we can match them. Unfortunately, the conditional

The algorithm uses $\tilde{O}(n)$ space outside storing the graph. In [4], they claimed $\tilde{O}(n)$ total space by assuming that $D \subseteq E$ and each edge in $D$ is specified by its two endpoints. Note that, in general, $\Omega(m)$ space is unavoidable as the oracles can be used to reconstruct any connected graph: we can query if $(u, v) \in E$, by deleting $\{\} \cup (u, v)$ and querying if $u$ and $v$ are connected.

Throughout the paper, we use $\tilde{O}(\cdot)$ to hide a polylog($n$) factor, use $\tilde{O}(\cdot)$ to hide a $n^{1/2}$ factor and use $O(\cdot)$ to hide a polyloglog($n$) factor.
lower bounds [11] show that query time must be $\tilde{\Omega}(d)$ as long as the algorithm has poly($n$) preprocessing and poly($dn^{o(1)}$) update time. This separates the vertex-failure case from the edge-failure case. The next question is then: what is the right complexity for vertex-failure connectivity oracles?

Before our paper, the answer of this question was not well-understood. For example, three questions are posted by Duan and Pettie [4] where they wrote “the following open problems are quite challenging”. The first question is about improving preprocessing time. Second, can we have both $\hat{O}(d)$ update and query time? Third, is there a deterministic data structure with $\hat{O}(m)$ space? We resolve all these questions in this paper. In fact, we essentially settle the complexity of all parameters of vertex-failure connectivity oracles up to sub-polynomial factors assuming popular conjectures. Our near-optimal deterministic oracle is stated below.

**Theorem 1.** There exists a deterministic vertex-failure connectivity oracle that uses $O(m \log^* n)$ space, $\hat{O}(m) + \hat{O}(md)$ preprocessing time, $\hat{O}(d^2)$ update time, and $O(d)$ query time.

All the $m$ factors can be replaced by $m = \min\{m, n, d(n+1)\}$ by using the standard sparsification algorithm by Nagamochi and Ibaraki [12] to first preprocess the graph in $O(m)$ time. (This also works for all known oracles discussed above.) So the bounds can be improved to $O(m \log^* n)$ space and $\hat{O}(m) + \hat{O}(d^2) + \hat{O}(md)$ preprocessing time. Also, all the $n^{o(1)}$ factors in Theorem 1 come solely from the overhead factor of the current deterministic vertex expander decomposition. It is believable that there exists a vertex expander decomposition algorithm with a polylog($n$) overhead factor, which would automatically improve all the $n^{o(1)}$ factors to polylog($n$).

Comparing with known results (see Table I), Theorem 1 improves the previous best deterministic algorithm by [4] by a factor of $d$ in both space and update time. It also significantly speeds up the $\Omega(\min\{mn, n^2\})$ preprocessing time of all known (randomized) algorithms with update time at most $\hat{O}(d^3)$. Lastly, our space complexity also beats the state-of-the-art of $O(m \log^6 n)$ in the randomized oracle by [4].

Next, we state conditional lower bounds that is tight with Theorem 1.

**Theorem 2.** Let $A$ be any vertex-failure connectivity oracle with $t_p$ preprocessing time, $t_u$ update time, and $t_q$ query time bound. Assuming popular conjectures, we have the following:

1. $A$ must take $\Omega(\min\{m, nd_s\})$ space. (See [4])
2. If $t_p = \text{poly}(n)$, then $t_u + t_q = \hat{O}(d^2)$.
3. If $t_p = \text{poly}(n)$ and $t_u = \text{poly}(dn^{o(1)})$, then $t_q = \hat{O}(d)$, (see Corollary 3.12 of [11])
4. If $t_u, t_q = \text{poly}(dn^{o(1)})$ and $A$ is a combinatorial algorithm$, then $t_p = \hat{O}(md)$.

The above result basically says that (1) we cannot improve the space bound, (2) either update or query time must be at least $\Omega(d^2)$, (3) query time must be at least $\hat{O}(d)$, and (4) the preprocessing time must be at least $\Omega(md)$. This means that all parameters from Theorem 1 are simultaneously tight up to sub-polynomial factors. Note, however, that the preprocessing time is tight only for combinatorial algorithms.

As a side result, we also study a related problem of vertex-failure global connectivity oracles. This oracle is similar to a vertex-failure connectivity oracle, except that in the second phase, after we are given $D \subset V$, we only need to return if $G \setminus D$ is connected and there is no third phase. Surprisingly, although this problem might seem easier because there is only one binary number to maintain instead of $G$ possible queries, we show that there is no oracle with reasonable update time for this problem.

**Theorem 3.** Let $A$ be a vertex-failure global connectivity oracle with poly($n$) preprocessing time. Assuming SETH, $A$ must spend $\hat{O}(n)$ update time even when the deletion set $D$ has size $|D| = n^{o(1)}$.

This is in stark contrast with the edge version; edge-failure connectivity oracles of [5]–[7] with $O(d)$ update time can also answer global connectivity and even count the number of connected components.

### A. Technical Highlights

The technical novelty of this paper lies in the new oracle from Theorem 1 with two parts to be highlighted.

First, we give a $O(m)$-time algorithm for computing the low-degree hierarchy of [4]. This step previously requires $O(m \log n)$ time in the preprocessing algorithm by Duan and Pettie [4] and it was their bottleneck. This new algorithm then leads to faster preprocessing time. Actually, similar hierarchies are the key structures behind most vertex-failure oracles (for the connectivity problem [2] and for the shortest path problem [14]). Our construction is based on expander decomposition, which is very versatile, and so we expect that the approach can be adapted to solve more problems under vertex failures.

Towards our construction, we give the first almost-linear-time algorithm for computing vertex-expander decomposition. Given that in the last decade numerous fast graph algorithms are based on edge-expander decomposition (see e.g. [15]–[19]), our fast vertex-expander decomposition will be likely very useful. To facilitate future applications, we show a general decomposition that works even for general vertex weights and demands. We even generalize the technique to obtain hypergraph-expander decomposition. See Lemma 3 for the basic version.

Second, given a near-optimal low degree hierarchy, the main challenge is designing an oracle with small space and update time based on it. There were two previous solutions by Duan and Pettie [4]: (1) a randomized Monte-Carlo near-optimal algorithm with $\hat{O}(m)$ space and $O(d^2)$ update time, and (2) a deterministic algorithm with $\hat{O}(md)$ space and $O(d^3)$ update time. We show how to achieve the best of both,
near-optimal and deterministic. The techniques we developed can be interpreted as a deterministic repairable hypergraph-to-graph connectivity-sparseifier which transforms a hypergraph to a graph with the same vertex set and almost preserves pairwise connectivity in the vertex-failure setting. Actually, the sparseifier is somewhat stronger because it preserves the connectivity of each hyperedge individually. Thus we expect that this tool can have applications on other vertex-failures problem.

B. Related work

In [20], Henzinger and Neumann studied the same problem but in the fully dynamic setting where the updates can both turn-on or turn-off vertices (not just turning-off as in our setting). Their result is a reduction to the deletion-only version and so our new result implies an improved solution in their setting too. When updates are given as an online sequence and fully dynamic, this is exactly the well-studied dynamic subgraph connectivity problem [21]–[25] whose complexity was just settled for combinatorial algorithms by the recent conditional lower bounds by [26]. In planar graphs, a near-optimal vertex-failure connectivity oracle was given by Borradaile et al. [3].

II. LOW DEGREE HIERARCHY VIA VERTEX EXPANDER DECOMPOSITION

The low degree hierarchy of Duan and Pettie [4] is the crucial underlying structure of their oracle and also ours. Although its definition is quite complicated, its purpose is easy to describe. Informally, this hierarchy allows us to assume that the input graph is of the following format:

1) \( G = (L, R, E) \) is a semi-bipartite graph (i.e. there is no edge between \( R \)).

2) \( L \) is spanned by a path \( \tau \) and \( \tau \) is given to us.

We will see why this graph can be useful in Section III. For now, we focus on constructing the hierarchy itself.

In this section, we give an overview of an almost-linear-time construction algorithm of the low degree hierarchy. Duan and Pettie [4] showed that constructing the hierarchy reduces to \( O(\log n) \) calls to a decomposition related to Steiner forests.

For any terminal set \( T \subseteq V \) in a graph \( G \), we say that a forest \( F \subseteq E \) is a \( T \)-Steiner forest if, for every terminal pair \( u, v \in T \), \( u \) and \( v \) are connected in \( F \) iff there are connected in \( G \). The decomposition says that, for any terminal set \( T \), one can delete \( |T|/2 \) vertices from the graph, so that the resulting graph contains a low-degree Steiner forest for the remaining terminals. This is formalized as follows:

**Lemma 1** (Low-degree Steiner Forest Decomposition [4]). Let \( G = (V, E) \) be a graph with terminal set \( T \subseteq V \). There is an \( O(\log |T|) \)-time algorithm that computes

- a vertex set \( X \) of size at most \(|T|/2 \), and
- a \((T \setminus X)\)-Steiner forest \( F \) in \( G \setminus X \) with maximum degree 4.

Lemma 1 is based on the \( (+1) \)-additive-approximation algorithm for computing minimum degree spanning trees by Fuhrer and Raghavachari [27]. Calling Lemma 1 with \( T = V \) leads to \( \Omega(mn \log n) \) preprocessing time in [4].

Our key contribution is showing that, when the maximum degree of \( F \) is relaxed from 4 to \( n^{c(1)} \), the running time of Lemma 1 can be improved from \( O(m|T| \log |T|) \) to \( O(mn) \). We completely bypass [27] and exploit vertex expanders instead.

Recall that a vertex cut \((L, S, R)\) is a vertex partition such that \( L, R \neq \emptyset \) and there is no edge between \( L \) and \( R \). We say that a terminal set \( T \) is \( \phi \)-linked in \( G \) if, for any vertex cut \((L, S, R)\) in \( G \), we have that \( |S| \geq \phi \min\{|(L \cup S) \cap T|, |(R \cup S) \cap T|\} \). If the whole set \( V \) is \( \phi \)-linked in \( G \), then we say that \( G \) is a \( \phi \)-vertex-expander. When \( G \) is a \( 1/n^{c(1)} \)-vertex-expander, we usually just say that \( G \) is a vertex expander. When \( T \) is \( 1/n^{c(1)} \)-linked in \( G \), then we say that \( T \) is well-linked in \( G \).

Our starting point is the observation that any vertex expander contains a low degree spanning tree. More generally, if \( T \) is well-linked in \( G \), then there is a low degree \( T \)-Steiner tree. Chekuri et al. [28] showed how to compute such tree in polynomial time. As this is too slow for us, we show that how...
to compute it in almost-linear time:

**Lemma 2.** There is a deterministic algorithm that, given any $\phi$-linked set $T$ in $G$, compute a $T$-Steiner tree with maximum degree $O(\log^2 n)$ in $O(m)$ time.

**Proof sketch.** We run in $O(m)$ time the cut-matching game between vertices in $T$ where the cut player is from [29] and the matching player is the deterministic almost-linear time vertex max flow derived from [30]. Since $T$ is $\phi$-linked, the game must return an embedding $P$ of an expander $W$ where $V(W) = T$ and $P$ has vertex congestion at most $O(\log^2 n)$. The union of all paths in $P$ will span $T$ and have maximum degree $O(\log^2 n)$, which spans our desired $T$-Steiner tree. 

Of course, we cannot assume that $T$ itself is well-linked. However, via a standard expander decomposition framework, the problem can be reduced to the well-linked case modulo removing some vertices. To formally carry out this approach, however, we will need a fast vertex expander decomposition algorithm, and, indeed, we show how to achieve that in almost-linear time:

**Lemma 3 (Vertex Expander Decomposition with Terminals).** There is a deterministic algorithm that, given a graph $G = (V, E)$, a terminal set $T \subseteq V$ and a parameter $\phi$, in $O(n)$ time computes a separator set $X$ of size $|X| \leq \phi |T| n^{o(1)}$ such that, for each connected component $G_i = G[V_i]$ in $G \setminus X$, $T \cap V_i$ is $\phi$-linked in $G_i$. In particular, if $|T| = n$, then $G_i$ is a $\phi$-vertex expander.

**Speed up the Low-degree Steiner Forest Decomposition.**

Now, we are ready to speed up Lemma 1. Given a terminal set $T$, we call Lemma 3 with $\phi = 1/n^{o(1)}$ and obtain the vertex set $X$ of size at most $|T|/2$ such that, for each connected component $G_i = G[V_i]$ in $G \setminus X$, we have that $T \cap V_i$ is well-linked in $G_i$. Then, we just apply Lemma 2 on each $G_i$ with terminal $T \cap V_i$. The union of Steiner tree on each $G_i$ gives us the $(T \setminus X)$-Steiner forest $F$ with maximum degree $n^{o(1)}$ as desired.

To compute the low-degree hierarchy itself, we call the low-degree Steiner forest decomposition $O(\log n)$ times in the same way as in [4]. The idea is simple. Set $T_1 \leftarrow V$ as an initial terminal set. For any $i > 1$, we invoke Lemma 1 with terminal set $T \leftarrow T_{i-1}$ and obtain $T_i \leftarrow X$. As $|T_i| \leq |T_{i-1}|/2$, the recursion depth is $O(\log n)$. Based on these terminal sets $T_1, T_2, \ldots$, the low-degree hierarchy is naturally defined.

### III. Optimal Deterministic Oracles

In this section, we give an overview of how to obtain a vertex-failure oracle with near-optimal space and update time. (Achieving near optimal preprocessing and query time is relatively simpler.) We will start by describing the previous deterministic construction with $O(md)$ space and $O(d^3)$ update time of [4]. Then, we show how to improve either space or update time individually. Obtaining both improvement simultaneously turn out to be challenging. This is the most technically involved step of this part and we will sketch the idea at the end of the overview.

Throughout this overview, we will assume that the input graph $G = (L, R, E)$ is a semi-bipartite graph (i.e. there is no edge between $R$). Moreover, $L$ is spanned by a path $\tau$ and $\tau$ is given to us. This restricted structure of $G$ will allow us to explain the key ideas more clearly. Actually, as mentioned in Section II, the low-degree hierarchy "almost" allows us to reduce the problem to this case, so this assumption is almost without lost of generality.

To simplify notations, assume further that $|L| = |R| = n$ and the vertex set $D$ is deleted always contains $d$ vertices from $L$, denoted by $D_L$, and $d$ vertices from $R$, denoted by $D_R$. In the updated graph $G \setminus D$, the path $\tau$ will be split into $d + 1$ connected intervals, denoted by $I = I_1, \ldots, I_{d+1}$. We call each interval $I \in I$ a left interval (or simply an interval) and each vertex $v \in R$ a right vertex. Sometimes, we use $\gamma$ to denote a right vertex, to make the notation consistent with the main body of the paper. Let $A_L \subseteq L$ denote the ordered set of vertices adjacent to $\gamma$ with order consistent with the path $\tau$.

To support connectivity queries in $O(d)$ time, given the set $D$, it suffices (as shown in [2], [4]) to compute connectivity between the left intervals $I$ in $G \setminus D$, i.e., whether $I$ and $I'$ are in the same connected components of $G \setminus D$ for every $I, I' \in I$. Suppose the query is $(u, v)$. In the easy case when both $u, v \in L$, we just identify the intervals $I$ containing $u$ and $I'$ containing $v$ in $O(1)$ time and answer if $I$ and $I'$ are connected in $G \setminus D$ in $O(1)$. Now, even if $u \in R$, we can identify a non-deleted neighbor $u'$ of $u$ in $O(d)$ time as there are only $O(d)$ deleted vertices. Note that $u' \in L$ and, say, $u' \in I$. If $v \in R$, then we similarly find a neighbor $v' \in L$ of $v$ in $O(d)$ time, and say $v' \in I'$. Finally, we answer if $I$ and $I'$ are connected in $G \setminus D$. The total query time is $O(d)$ time.

From now, we discuss how to preprocess $G$ using small space so that, given $D$, we can compute the connectivity between the left intervals $I$ in $G \setminus D$ fast. Below, for any set $L', L'' \subseteq L$ on the left, we say that $L'$ and $L''$ are adjacent in $G$ if there exists an edge $(u, v) \in L \times L''$ or there exists a right vertex $\gamma$ adjacent to both vertices in $L'$ and $L''$. We say that $L'$ and $L''$ are adjacent via $(u, v)$ in the former case and adjacent via $\gamma$ in the latter case.

The bounds discussed below will be slightly better than our final bounds for general graphs. More precisely, we will pay an additional $n^{o(1)}$ factor in update time and $O(1)$ factor in space when apply the ideas together with the low-degree hierarchy.

#### A. Warming-up: No right vertices

To motivate the idea, consider the extremely simple case when $R = \emptyset$. We will show a data structure with $O(m)$ space and $O(d^2)$ update time. In this case, the whole edge set $E(G)$ can be represented in a 2-dimensional table $T = [L] \times [L]$ where the entry $(u, v)$ is the number of multi-edges between $u$ and $v$. Hence, during preprocessing we can build a range counting data structure on this table, which takes $O(|E(G)|) = O(m)$ space. Given the deletion set $D$, the left intervals $I$ are
defined, and we can check if \( I \) and \( I' \) are adjacent in \( G \setminus D \) by querying the range counting data structure in \( O(1) \) time if
\[
|E(G) \cap (I \times I')| > 0. \tag{1}
\]

By querying by a pair \( I, I' \in I \), we deduce the connected components of \( I \in O(d^2) \) time. We note that in our complete algorithm, we will treat undirected edges in such counting problem as ordered pairs, so strictly speaking, we should check if \(|E(G) \cap (I \times I')| + |E(G) \cap (I' \times I)| > 0\). For simplicity, we will only consider a half of each counting problem throughout the overview, and the other half is almost symmetric.

\[ B. \text{ Deterministic Construction of [4]: } O(md) \text{ space and } O(d^2) \text{ update time.} \]

We first describe the deterministic construction of [4] that uses \( O(md) \) space and \( O(d^2) \) update time. We view the high-level idea of [4] for handling right vertices as follows: Construct a restricted (multi-)graph \( \tilde{G} \) whose vertex set only contains \( V(G) = L \), but \( \tilde{G} \) should capture connectivity between left vertices of \( G \) even after vertex deletions. Since \( \tilde{G} \) contains only left vertices, we can use the range counting idea on \( G \).

Here is a natural way to construct \( \tilde{G} \). Starting with \( \tilde{G} = G[L] \), for each right vertex \( \gamma \in R \), add a \((d + 1)\)-vertex connected graph on its neighbors \( A_{\gamma} \subseteq L \) (or a clique if \(|A_{\gamma}| \leq d\), in which the edges are called the artificial edges of \( \gamma \), denoted by \( \tilde{E}_{\gamma} \)). Specially, we define \( \tilde{E}_{\gamma} = A_{\gamma} \times B_{\gamma} \), where \( B_{\gamma} \subseteq A_{\gamma} \) is an arbitrary subset of size \( \min\{|A_{\gamma}|, d + 1\} \).

Observe that \( \tilde{E}_{\gamma} \) spans \( A_{\gamma} \) with the following fault-tolerant property: for any \( D_L \subseteq L \) with \(|D_L| = d, A_{\gamma} \setminus D_L \) (which are obliviously connected in \( G \setminus D_L \) via \( \gamma \)) are still connected in \( G \setminus D_L \). It follows that, for any deleted set \( D = D_L \cup D_R \), the graph \( \tilde{G}_D := \tilde{G} \setminus (D_L \cup \bigcup_{\gamma \in D_R} \tilde{E}_{\gamma}) \) preserves connectivity between all left vertices of \( G \setminus D \) and, in particular, between all left intervals \( I \). So it suffices to compute components of \( \tilde{G} \) in \( G_D \). We can query if \( I \) and \( I' \) are adjacent in \( G_D \) by checking if
\[
|E(\tilde{G}) \cap (I \times I')| - \sum_{\gamma \in D_R} |\tilde{E}_{\gamma} \cap (I \times I')| > 0, \tag{2}
\]
which can be computed using \( 1 + |D_R| = O(d) \) range queries. By querying all pairs, the connected components of \( \tilde{G} \) in \( G_D \) can be deduced in \( O(d^3) \) time as desired. The total space for range counting structures on \( E(\tilde{G}) \) and all \( \tilde{E}_{\gamma} \) is \( O(|E(\tilde{G})| + \sum_{\gamma} |\tilde{E}_{\gamma}|) = O(|E(G)| + \sum_{\gamma} |A_{\gamma}|d) = O(md). \)

\[ C. \text{ Update-Time Improvement: } O(md) \text{ space and } O(d^2) \text{ update time.} \]

Then we will show how to deterministically improve the update time to \( O(d^2) \). The high-level idea is that we want to avoid querying for all \( O(d^2) \) pairs of left intervals \( I, I' \in I \).

\[ ^6 \text{We note that the deterministic construction in [4] did not define } \tilde{E}_{\gamma} \text{ as a product set } A_{\gamma} \times B_{\gamma}, \text{ but their } \tilde{E}_{\gamma} \text{ still forms a } \min\{|A_{\gamma}|, d + 1\}-\text{connected graph on } A_{\gamma}. \] Our product-set construction will be crucial for faster update time as will be explained in Section III-C.

We will use a Borůvka's algorithm instead, which can reduce the number of queries to \( O(d) \). However, the queries become more complicated.

Recall that Borůvka's algorithm uses the "hook and merge" approach. There are \( O(\log |I|) \) phases. In each phase there has already been a partition of intervals \( I \) into groups \( Z_1, ..., Z_z \) such that intervals in a same group are in a same connected component. In this phase, these groups will be further merged by finding, for each \( Z_k \), an adjacent group \( Z' \) such that some \( I \in Z' \) and some \( I \in Z_k \) are adjacent. This can be reduced (via a binary search) to some batched-adjacency queries: given \( Z_k \) and a batch of groups \( Z_i, Z_{i+1}, ..., Z_r, \) there is any group in this batch adjacent to \( Z_k \). Observe that this query can be answered by checking if
\[
\sum_{I \in Z_k, I' \in Z_{i,r}} |E(\tilde{G}) \cap (I \times I')| - \sum_{\gamma \in D_R} \sum_{I \in Z_k, I' \in Z_{i,r}} |\tilde{E}_{\gamma} \cap (I \times I')| > 0, \tag{3}
\]
where \( Z_{i,r} = \bigcup_{k \leq l \leq r} Z_k' \). By further exploiting the fact that each \( \tilde{E}_{\gamma} \) is a product set \( A_{\gamma} \times B_{\gamma} \), we can write (3) in the following equivalent form:
\[
\sum_{I \in Z_k, I' \in Z_{i,r}} |E(\tilde{G}) \cap (I \times I')| - \sum_{\gamma \in D_R} \left( \sum_{I \in Z_k} |A_{\gamma} \cap I| \cdot \sum_{I' \in Z_{i,r}} |B_{\gamma} \cap I'| \right) > 0. \tag{4}
\]

The key to fast update time is the following. We arrange intervals in an ordered list \( I_t \) (denotes the phase number) such that \( I_t \) is the concatenation of \( Z_1, ..., Z_z \). Now, by preprocessing a 2D-counting table \( \text{Table}_t = \{(E(\tilde{G}) \cap (I \times I'))_{I \times I' \in I_t \times I_t} \} \) with size \( |I|^2 \), the first term can be answered by a single 2D-range counting query on \( \text{Table}_t \). Next, by preprocessing, for each \( \gamma \in D_R \), a 1D counting array \( \text{A-Array}_{\gamma,t} = \{\sum_{I \in I_t} |A_{\gamma} \cap I|\} \) and another 1D array \( \text{B-Array}_{\gamma,t} = \{\sum_{I \in I_t} |B_{\gamma} \cap I|\} \). The second term can be answered using \( 2|D_R| \) queries to these arrays \( \text{A-Array}_{\gamma,t} \) and \( \text{B-Array}_{\gamma,t} \). Therefore, the query time for each batched-adjacency query is \( O(d) \). For each phase, we make \( O(d) \) such queries which take \( O(d^2) \) total query time. Notice that since the ordered list \( I_t \) will be shuffled between phases, \( \text{Table}_t, \text{A-Array}_{\gamma,t} \) and \( \text{B-Array}_{\gamma,t} \) must be rebuilt at the beginning of each phase, but this takes only \( O(d^2) \) total time. Over all \( O(\log |I|) \) phases, the total update time is \( O(d^3) \).

We emphasize that this approach highly relies on that each \( \tilde{E}_{\gamma} \) is a product set \( A_{\gamma} \times B_{\gamma} \), which enables us to consider two dimensions separately. If \( \tilde{E}_{\gamma} \) is less structured, a 2D counting table is required for each \( \gamma \in D_R \) in this approach, and we cannot afford to construct them in \( O(d^2) \) update time. The space is still \( O(md) \) as shown above. Note that this is an inherent limitation of this approach. As long as \( \tilde{E}_{\gamma} \) forms a \( \min\{|A_{\gamma}|, d + 1\}-\text{connected graph on } A_{\gamma} \), we will have space \( \Omega(md) \) in the worst case.
D. Space Improvement: $O(m)$ space and $O(d^3)$ update time.

Now, we will improve the space to $O(m)$, but it will bring the update time back to $O(d^3)$. This step already gives us the first deterministic oracle with near-linear space. The high-level idea is to relax the fault-tolerant property in order to get sparser artificial edges. Concretely, it is acceptable that, for some right vertices $\gamma$, its artificial edges $\hat{E}_\gamma$ do not satisfy the fault-tolerant property, but we want the number of such vertices to be small so we can afford to “repair” their $\hat{E}_\gamma$ in an additional repairing phase of the update algorithm. We note that the Monte Carlo construction in [4] also takes advantage of a similar relaxation.

Towards this goal, let $\hat{E} = \bigcup_{\gamma \in R} \hat{E}_\gamma$ contains all artificial edges. We will redefine the artificial edges $\hat{E}_\gamma$ of $\gamma$ so that the total number of artificial edges not counting multiplicity, denoted by $|\hat{E}|_0$, is linear to $\sum_{\gamma \in R} |A_\gamma| = O(m)$. The construction works by scanning through right vertices $\gamma \in R$ one by one. Initialize $\hat{E} = \emptyset$. Then, for each $\gamma \in R$, if $(u,v) \in \hat{E}$ for all $(u,v) \in A_\gamma \times B_\gamma$ (as before $B_\gamma \subseteq A_\gamma$ is an arbitrary subset with size $\min\{|A_\gamma|, d+1\}$), then define $\hat{E}_\gamma = A_\gamma \times B_\gamma$. Otherwise, there exists $(u_\gamma, v_\gamma) \notin \hat{E}$ where $u_\gamma, v_\gamma \in A_\gamma$, and we set $\hat{E}_\gamma = A_\gamma \times \{u_\gamma, v_\gamma\}$. Lastly, we insert $\hat{E}_\gamma$ into $\hat{E}$ and proceed to the next right vertex.

Note that although $\sum_{\gamma \in R} |E_\gamma|$ can be large, $|\hat{E}|_0$ is bounded by the number of new artificial edges added in each round. Since there is no new edge in the former case, it follows that $|\hat{E}|_0 \leq \sum_{\gamma \in R} 2|A_\gamma| = O(m)$.

We then show there are only a few $\hat{E}_\gamma$ without the fault-tolerant property. Let $R_{\text{fault}}$ contain all right vertices $\gamma$ where $D_L$ contains both $u_\gamma$ and $v_\gamma$. Observe that $A_\gamma \setminus D_L$ is not connected by $\hat{E}_\gamma \setminus D_L$ only if $\gamma \in R_{\text{fault}}$. As $D_L$ contains at most $|D_L|^2$ different pairs of vertices, so $|R_{\text{fault}}| \leq |D_L|^2 = O(d^2)$. Therefore, we have that the graph $G_D := G \setminus (D_L \cup \bigcup_{\gamma \in R} \hat{E}_\gamma)$ still preserves connectivity between all left intervals of $G \setminus D$, except that it might miss the connectivity information from each $\gamma \in R_{\text{fault}}$.

To compute the connected components of $\mathcal{I}$ in $G \setminus D$, we fist compute connected components of $\mathcal{I}$ in $G_D$. This can be done in $O(d^2)$ time as in Section III-C. But then some connected components $\mathcal{T}$ in $G_D$ may be further connected via $\gamma \in R_{\text{fault}}$. Therefore, we add a repairing phase in the update algorithm. For every $I \in \mathcal{I}$ and $\gamma \in R_{\text{fault}}$, we simply check $I \cap A_\gamma \neq \emptyset$. If there is some $I, I' \in \mathcal{I}$ and $\gamma \in R_{\text{fault}}$ such that $I \cap A_\gamma \neq \emptyset$ and $I' \cap A_\gamma \neq \emptyset$, then the connected components of $I$ and $I'$ can be merged as $I$ and $I'$ are adjacent via $\gamma$. After these queries, we can finally deduce connected components of $\mathcal{T}$ in $G \setminus D$. We can check if $I \cap A_\gamma \neq \emptyset$ in $O(1)$ time. However, there are $O(d^3)$ choices of $I, I' \in \mathcal{I}$ and $\gamma \in R_{\text{fault}}$, the repairing phase needs $O(d^3)$ time.

Therefore, we obtain an oracle with $O(m)$ space and $O(d^3)$ update time.

E. Repairing-Time Improvement: $O(d^3)$ update time with $O(d^2)$ repairing time

As a prerequisite to our final construction in Section III-F, here we will improve the running time of the repairing phase from $O(d^3)$ to $O(d^2)$. However, the total update time would still be $O(d^3)$, because our artificial edges $\hat{E}_\gamma$ in this step will not be well-structured and so we cannot apply the Borůvka's algorithm for interval connectivity as we did in Section III-C. We will sketch how to handle this issue in Section III-F.

The high-level idea is to redesign the artificial edges $\hat{E}_\gamma$ for each $\gamma \in R$, such that for any failure set $D$, vertices in $A_\gamma \setminus D_L$ may not be connected by $\hat{E}_\gamma \setminus D_L$, but they will be connected by $(\hat{E}_\gamma \cup \hat{F}_\gamma) \setminus D_L$, where $\hat{F}_\gamma$ is a small set of $D$-repairing edges generated after $D$ is deleted. In fact, in our following construction, the total number of $D$-repairing edges for all $\gamma \in R$ is bounded by $O(d^2)$. Therefore, the repairing phase only needs to add the connectivity provided by these repairing edges and it takes $O(d^2)$ time.

We introduce a new structure called the segmentation hierarchy. For each $\gamma \in R$, the segmentation hierarchy $\mathcal{S}_\gamma$ is simply a recursive division of $A_\gamma$ like a segment tree. The elements of $\mathcal{S}_\gamma$ are segments, which are consecutive sublists of $A_\gamma \setminus D_L$. $\mathcal{S}_\gamma$ is partitioned into $r = O(\log |A_\gamma|)$ levels $\mathcal{S}_{\gamma,1}, \ldots, \mathcal{S}_{\gamma,r}$, such that for each level $j$, segments in $\mathcal{S}_{\gamma,j}$ form a partition the list $A_\gamma$. In particular, the top level has the unique segment $A_\gamma$ and each segment at the bottom level is a singleton list of each vertex in $A_\gamma$. Furthermore, for each level $j \geq 2$, each segment $S \in \mathcal{S}_{\gamma,j}$ is the union of two child-segments $S_{1,2} \in \mathcal{S}_{\gamma,j-1}$.

The construction of $\hat{E}_\gamma$ is similar to the previous step. We process $\gamma \in R$ one by one and initialize $\hat{E} = \emptyset$. Let $B_\gamma$ be an arbitrary subset of $A_\gamma$, of size $\min\{d+1, |A_\gamma|\}$. For each $\gamma \in R$ if there are less than $|S_{\gamma,j}|$ pairs in $(A_\gamma \times B_\gamma) \setminus \hat{E}$, we set $\hat{E}_\gamma = A_\gamma \times B_\gamma$ as usual. Otherwise we assign a distinct pair $(u_\gamma, v_\gamma) \in (A_\gamma \times B_\gamma) \setminus \hat{E}$ to each segment $S \in \mathcal{S}_\gamma$ and let $\hat{E}_\gamma = \sum_{S \in \mathcal{S}_\gamma} S \times \{u_\gamma, v_\gamma\}$. Lastly, insert $\hat{E}_\gamma$ to $\hat{E}$ and proceed to the next $\gamma$. Now, we claim that the total number of artificial edges (without multiplicity) is $|\hat{E}|_0 = O(m)$ since the former case will bring at most $|S_{\gamma,j}|$ new edges and in the latter case the number of new edges is at most $|\hat{E}_\gamma| = O(\sum_{S \in \mathcal{S}_\gamma} |S|) = O(|A_\gamma|)$. So $|\hat{E}|_0 = \sum_{\gamma \in R} O(|A_\gamma|) = O(m)$ and so the total space is $O(|\hat{E}|_0) = O(m)$.

The repairing algorithm is based on the segmentation hierarchy. Let $\mathcal{F}_{\text{fault}}$ contain all segments $S$ (in $\mathcal{S}_{\gamma}$ for all $\gamma \in R$) such that $u_\gamma, v_\gamma \in D_L$. By the same argument, $S \setminus D_L$ is not connected by $\hat{E}_\gamma \setminus D_L$ only if $S \in \mathcal{F}_{\text{fault}}$, and we have $|\mathcal{F}_{\text{fault}}| = O(d^2)$. For each $\gamma \in R$ with nonempty $\mathcal{F}_{\text{fault}} \cap \mathcal{S}_\gamma$, suppose that we are given a non-deleted vertex $v_\gamma \in A_\gamma \setminus D_L$. Then we construct $D$-repairing edges $\hat{F}_\gamma$, by, for each $S \in \mathcal{F}_{\text{fault}} \cap \mathcal{S}_\gamma$, connecting its two child-segments $S_{1,2}$ to $v_\gamma$. Concretely, we add to $\hat{F}_\gamma$ four $D$-repairing edges connecting $v_\gamma$ to each of $u_{S_1}, v_{S_1}, u_{S_2}, v_{S_2}$ for each $v_\gamma$. In the complete algorithm, the $v_\gamma$ for each $\gamma \in R$ with $\mathcal{F}_{\text{fault}} \cap \mathcal{S}_\gamma \neq \emptyset$ can be computed in totally $O(d^2)$ time.
S ∈ S_{\text{fault}} ∩ S_{\gamma}. Thus there are totally \(O(|S_{\text{fault}}|) = O(d^2)\) D-repairing edges and the repairing time is \(\hat{O}(d^2)\).

To see the correctness, we say that a segment \(S ∈ S_{\gamma}\) is maximal if \(S ∈ S_{\gamma} \setminus S_{\text{fault}}\) but its parent-segment is in \(S_{\gamma} ∩ S_{\text{fault}}\). Observe that maximal segments cover \(A_{\gamma} \setminus D\), the whole non-deleted part of \(A_{\gamma}\). Now, for each maximal segment \(S\), we have that either \(uS \notin D\) or \(vS \notin D\). Say \(uS \notin D\), so \(uS\) is connected to \(v\) via \(F\), and the whole \(S\) is connected to \(uS\) via \(E_{\gamma}\). Therefore, indeed we have that vertices in \(A_{\gamma} \setminus D\) will be connected via \((E_{\gamma} ∩ F) \setminus D\) as promised.

\(F. \text{Final Algorithm: } \hat{O}(m) \text{ space and } \hat{O}(d^2) \text{ update time.}\)

The last step to our final algorithm is to make \(E_{\gamma}\) in Section III-E well-structured. We cannot hope each \(E_{\gamma}\) forms a perfect product set because of the segmentation hierarchy. Yet, we can still design it to have some good structural property so that it is compatible with the Borůvka’s based algorithm in Section III-C, which finally leads us to \(\hat{O}(d^2)\) update time and \(\hat{O}(m)\) space simultaneously.

The construction algorithm of artificial edges is relatively simple and we only summarize the outputs here. For each right vertex \(\gamma ∈ R\), we still assign a pair \((uS, vS) ∈ A_{\gamma} \times A_{\gamma}\) to each segment \(S ∈ S_{\gamma}\), and the actual \(E_{\gamma}\) will be

\[
E_{\gamma} = A_{\gamma} × B_{\gamma} + \sum_{S ∈ S_{\gamma}} S × \{uS\},
\]

(5)

where the add operations denote the multiset union operations. The set \(B_{\gamma}\) for each \(\gamma\) is designed to ensure that there will be \(\|E\|_0 = \hat{O}(m)\) distinct artificial edges using similar argument as in Section III-E, so the space bound is \(\hat{O}(m)\).

Now, we discuss how this structure enables \(\hat{O}(d^2)\) update time. For each \(S ∈ S_{\gamma}\), we call \(uS\) the witness of \(S\). Furthermore, let \(f : S_{\gamma} → A_{\gamma}\) be the witness function defined by \(f(S) = uS\) for each \(S ∈ S_{\gamma}\). Our construction ensures that \(f\) has the following monotonically increasing property on each level \(S_{\gamma,j}\); for each \(S, S' ∈ S_{\gamma,j}\) such that \(S\) is located before \(S'\) on \(A_{\gamma}\), \(f(S)\) is located before \(f(S')\) according the order of vertices in \(A_{\gamma}\).

Answering a batched-adjacency query in Section III-C given \(Z_{l,r}\) now becomes more involved. We sketch the ideas below. The product term \(A_{\gamma} × Z_{l,r}\) is easy to deal with using the idea in Section III-C. To handle the term \(\sum_{S ∈ S_{\gamma}} S × f(S)\), we first write down its contribution\(^8\) to the second term of the left hand side of (3) for fixed \(\gamma ∈ D_{R_{\gamma}}\), level \(j\) and \(I ∈ Z_{l,r}\) as

\[
Q(\gamma, j, I) = \left(\sum_{S ∈ S_{\gamma,j}} S × f(S)\right) \cap \left(\bigcup_{I' ∈ Z_{l,r}} I' × I\right).
\]

(6)

In fact, there will be totally at most \(\hat{O}(d^2)\) queries on \(Q(\gamma, j, I)\), summing over different \(\gamma, j, I\) in all Borůvka’s phases. In what follows, we discuss how to compute \(Q(\gamma, j, I)\) for fixed \(\gamma, j, I\) in \(\hat{O}(1)\) time.

Recall that \(I\) is an interval on the path \(\tau\). Consider the set of segments \(\{S ∈ S_{\gamma,j} \mid f(S) ∈ I\}\), i.e., the set of level-j segments whose witnesses are in \(I\). By the monotonically increasing property of \(f\), observe that the union of these segments form a consecutive subset of \(A_{\gamma}\), denoted by \(A_{\gamma}(v_L, v_R)\). Then the contribution can be rewritten as

\[
Q(\gamma, j, I) = A_{\gamma}(v_L, v_R) ∩ \bigcup_{I' ∈ Z_{l,r}} I'.
\]

(7)

This query can be further reduced to one 3D range counting query by exploiting that intervals \(I'\) in \(Z_{l,r}\) locate consecutively in the order \(I_l\) for the \(l\)-th Borůvka’s phase. Therefore, \(Q(\gamma, j, I)\) can be indeed queried in \(\hat{O}(1)\) time. Moreover, the required data structures can be constructed in \(\hat{O}(d^2)\) time at the beginning of each Borůvka’s phase.

\(G. \text{Perspective: Repairable Hypergraph-to-Graph Connectivity-Sparsifiers}\)

Lastly we discuss our techniques in Section III in a different perspective. If we corresponds left vertices and right vertices in the semi-bipartite graph to vertices and hyperedges in a hypergraph respectively, the techniques on constructing artificial edges and D-repairing edges can be viewed as some kind of sparsifier which transform a hypergraph \(H\) to a graph \(G\). Concretely, the transformation will keep the vertex set unchanged, and for each hyperedge \(e ∈ E(H)\), add an edge set \(\hat{E}(e)\) to \(E(G)\). Namely we will have \(V(G) = V(H)\) and \(E(G) = \bigcup_{e ∈ E(H)} \hat{E}(e)\). Moreover, the sparsifier has the following properties.

- (Vertex Restricted) The vertex set is unchanged.
- (Sparse) The number of edges in \(G\) is nearly linear to the total size of edges in \(H\). Namely \(|E(G)| = \hat{O}(\sum_{e ∈ E(H)} |e|)\), where we use the same notation \(e\) to denote the set of \(e\)’s endpoints.
- (Fault-Tolerant/Repairable) Given any failure set \(D ⊆ V(G)\) with \(|D| = d\), by augmenting each \(\hat{E}(e)\) of a set \(\hat{F}(e)\) of repairing edges, it will be guaranteed that for each hyperedge \(e ∈ E(H)\), vertices in \(e\) are still connected by \(\hat{E}(e) \cup \hat{F}(e)\). Moreover, the total size of \(\hat{F}(e)\) summing over all \(e ∈ E(H)\) is \(\hat{O}(d^2)\), and all nonempty \(\hat{F}(e)\) can be computed in totally \(\hat{O}(d^2)\) time.

We apply this tool to obtain an nearly optimal vertex-failure connectivity oracle. Furthermore, we note that this sparsifier is somewhat stronger than maintaining just pairwise connectivity because the connectivity of each hyperedge \(e\) is preserved by \(\hat{E}(e) \cup \hat{F}(e)\) individually. Thus we expect that this tool or its variant can have applications on other vertex-failure problems.

\(IV. \text{Open Problems}\)

While Theorems 1 and 2 together have settled the complexity of the connectivity oracles with vertex failures up to sub-polynomial factors, they still leave interesting and questions for this problem.
1) If the update and query time can depend only on $d_\epsilon$, what is the best bound we can hope for? van den Brand and Saranurak [9] showed a randomized Monte-Carlo oracle with $O(n^2)$ space, $O(n^{1.5})$ preprocessing, $O(d^2)$ update, and $O(d^2)$ query time. Note that both update and query time are both constant for all $d = O(1)$. Very recently, Pilipczuk et al. [10] gave the first deterministic solution for this setting though the bounds seem far from being optimal.

2) Our $O(nd_\epsilon)$ preprocessing time is tight only under the combinatorial BMM conjecture. Can we improve the bound using fast matrix multiplication?

The following questions about generalization of this problem seem challenging:

1) Suppose the vertices to be deleted are given one by one and queries are given between each deletion, can we still handle all $d$ deletions in total $O(d^2)$ update time?

2) Is there a vertex-failure $c$-edge/vertex)-connectivity oracle for $c > 1$? The dynamic $c$-edge connectivity algorithm by Jin and Sun [42] implies a near-optimal edge-failure $c$-edge connectivity oracle for $c = \log^{O(1)} n$, but all other variants of this problems remain open.

REFERENCES

[1] Y. Long and T. Saranurak, “Near-Optimal Deterministic Vertex-Failure Connectivity Oracles,” arXiv preprint arXiv:2205.03930, 2022.

[2] R. Duan and S. Pettie, “Connectivity oracles for failure prone graphs,” in Proceedings of the forty-second ACM Symposium on Theory of Computing, 2010, pp. 465–474.

[3] G. Boroudadi, S. Pettie, and C. Wulff-Nilsen, “Connectivity oracles for planar graphs,” in Scandinavian Workshop on Algorithm Theory. Springer, 2012, pp. 316–327.

[4] R. Duan and S. Pettie, “Connectivity oracles for graphs subject to vertex failures,” SIAM Journal on Computing, vol. 49, no. 6, pp. 1363–1396, 2020, announced at SODA 17.

[5] M. Patrascu and M. Thorup, “Planning for fast connectivity updates,” in 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS’07). IEEE, 2007, pp. 263–271.

[6] B. M. Kapron, V. King, and B. M. Kapron, “Dynamic graph connectivity in polylogarithmic worst case time,” in Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms. SIAM, 2013, pp. 1131–1142.

[7] D. Gibb, B. Kapron, V. King, and N. Thornton, “Dynamic graph connectivity with improved worst case update time and sublinear space,” arXiv preprint arXiv:1509.06464, 2015.

[8] A. Kanevsky, R. Tamassia, G. Di Battista, and J. Chen, “On-line maintenance of the four-connected components of a graph,” in [1991] Proceedings 32nd Annual Symposium on Foundations of Computer Science. IEEE Computer Society, 1991, pp. 793–801.

[9] J. van den Brand and T. Saranurak, “Sensitive distance and reachability oracles for large batch updates,” in 2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS). IEEE, 2019, pp. 424–435.

[10] M. Pilipczuk, N. Schirrmacher, S. Siebertz, S. Todisco, and A. Vigny, “Algorithms and data structures for first-order logic with connectivity under vertex failures,” arXiv preprint arXiv:2111.03725, 2021.

[11] M. Henzinger, S. Krimmering, 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 symposium on Theory of computing, 2015, pp. 21–30.

[12] H. Nagamochi and T. Ibaraki, “A linear-time algorithm for finding a sparsek-connected spanning subgraph of an-connected graph,” Algorithmica, vol. 7, no. 1, pp. 583–596, 1992.

[13] A. Aboud and V. V. Williams, “Popular conjectures imply strong lower bounds for dynamic problems,” in 2014 IEEE 55th Annual Symposium on Foundations of Computer Science. IEEE, 2014, pp. 434–443.
[35] 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, vol. 6, no. 7, p. 12, 2007.

[36] 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, 2017, pp. 1130–1143.

[37] J. L. Bentley, “Multidimensional divide-and-conquer,” Communications of the ACM, vol. 23, no. 4, pp. 214–229, 1980.

[38] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan, “Making data structures persistent,” Journal of computer and system sciences, vol. 38, no. 1, pp. 86–124, 1989.

[39] J. L. Bentley and D. Wood, “An optimal worst case algorithm for reporting intersections of rectangles,” IEEE Transactions on Computers, vol. 29, no. 07, pp. 571–577, 1980.

[40] B. Chazelle, “A functional approach to data structures and its use in multidimensional searching,” SIAM Journal on Computing, vol. 17, no. 3, pp. 427–462, 1988.

[41] K. Bringmann and M. Künemann, “Quadratic conditional lower bounds for string problems and dynamic time warping,” in 2015 IEEE 56th Annual Symposium on Foundations of Computer Science. IEEE, 2015, pp. 79–97.

[42] W. Jin and X. Sun, “Fully dynamic st edge connectivity in subpolynomial time,” in 2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS). IEEE, 2022, pp. 861–872.

[43] D. Kang and J. Payor, “Flow rounding,” arXiv preprint arXiv:1507.08439, 2015.

[44] D. D. Sleator and R. E. Tarjan, “A data structure for dynamic trees,” Journal of computer and system sciences, vol. 26, no. 3, pp. 362–391, 1983.

[45] M. A. Bender and M. Farach-Colton, “The lca problem revisited,” in Latin American Symposium on Theoretical Informatics. Springer, 2000, pp. 88–94.

[46] 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.

[47] J. Chuzhoy and T. Saranurak, “Deterministic algorithms for decremental shortest paths via layered core decomposition,” in Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA). SIAM, 2021, pp. 2478–2496.

[48] A. Bernstein, M. P. Gutenberg, and T. Saranurak, “Deterministic decremental sssp and approximate min-cost flow in almost-linear time,” arXiv preprint arXiv:2101.07149, 2021.