Efficient Fault-Tolerant Search by Fast Indexing of Subnetworks

Davide Bilò[1], Keerti Choudhary[2], Sarel Cohen[3], Tobias Friedrich[4], Martin Schirneck[5]

[1]Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Italy
[2]Department of Computer Science and Engineering, Indian Institute of Technology Delhi, India
[3]School of Computer Science, The Academic College of Tel Aviv-Yaffo, Israel
[4]Hasso Plattner Institute, University of Potsdam, Germany
[5]Faculty of Computer Science, University of Vienna, Austria
davide.bilo@univaq.it, keerti@iitd.ac.in, sarelco@mta.ac.il, tobias.friedrich@hpi.de, martin.schirneck@univie.ac.at

Abstract

We design sensitivity oracles for error-prone networks. For a network problem Π, the data structure preprocesses a network $G = (V, E)$ and sensitivity parameter f such that, for any set $F \subseteq V \cup E$ of up to f link or node failures, it report a solution for Π in $G-F$. We study three exemplary problems Π.

- L-Hop Shortest Path: Given $s, t \in V$, is there a shortest s-t-path in $G-F$ with at most L links?
- k-Path: Does $G-F$ contain a simple path with k links?
- k-Clique: Does $G-F$ contain a clique of k nodes?

Our main technical contribution is a new construction of (L, f)-*replacement path coverings* ((L, f)-RPC) in the parameter realm where $f = o(\log L)$. An (L, f)-RPC is a family $\mathcal{G}$ of subnetworks of G which, for every $F \subseteq E$ with $|F| \leqslant f$, contain a subfamily $\mathcal{G}_F \subseteq \mathcal{G}$ such that (i) every subnetwork in $\mathcal{G}_F$ contains no link of F and (ii) for each $s, t \in V$, if $G-F$ contains a shortest s-t path with at most L links, then some subnetworks in $\mathcal{G}_F$ retains at one of such paths. Our (L, f)-RPC has almost the same size as the one by Weimann and Yuster (2013) but it improves the query time to access $\mathcal{G}_F$ from $\widetilde{O}(f^2 L^f)$ to $\widetilde{O}(f^{\frac{5}{2}} L^{o(1)})$. It also improves both the size and query time of the (L, f)-RPC by Karthik and Parter (2021) by nearly a factor of L. We then derive oracles for L-Hop Shortest Path, k-Path, and k-Clique from this. Notably, our solution for k-Path improves the query time of the one by Bilò et al. (2022) for $f = o(\log k)$.

Introduction

Networks are central structures in computer science as they can model different types of relations we encounter in real-world applications. Numerous algorithms and data structures have been developed to solve problems in static networks where nodes and links do not change over time. However, as networks in real life are prone to transient failures, many of these algorithms require recomputations from scratch even when only a few network components are malfunctioning. In many applications one may have an a priori known bound f on the number of simultaneous failures, especially in the context of independent failures where the likelihood of multiple failures decreases exponentially. This is called the *fault-tolerant* setting or *sensitivity* analysis and the resulting data structures are called *sensitivity oracles*. In the past two decades, many sensitivity oracles have been designed for classical network problems, e.g., connectivity (Patrascu and Thorup 2007; Duan and Pettie 2009, 2010, 2017), shortest paths (Cho, Shin, and Oh 2024; Demetrescu et al. 2008; Chechik et al. 2012; Duan and Pettie 2009; Bilò et al. 2023; Duan and Ren 2022; Dey and Gupta 2024; Gu and Ren 2021) and routing (Chechik et al. 2012).

We continue this line of work by design data structures that, given a network problem Π, a network $G = (V, E)$ with n nodes and m links, and a parameter f, preprocesses G into a *oracle with sensitivity* f that, when queried with a set $F \subseteq V \cup E$ of size $|F| \leqslant f$, reports a solution for Π in $G-F$. Specifically, we study the following problems Π.

- L-Hop Shortest Path: Given two nodes $s, t \in V$, is there a shortest s-t path in $G-F$ with at most L links?
- k-Path: Does $G-F$ contain a simple path with k links?
- k-Clique: Does $G-F$ contain a clique of k nodes?

In fact, we can handle all problems with the property that any certificate solution $S \subseteq G$ for Π, like e.g., a k-path, is also a certificate solution of G' for Π, for every $S \subseteq G' \subseteq G$. Although we focus on decision problems, our data structures are also capable of reporting a certificate solution.

A naive solution consists in enumerating all possible subnetworks, one for each $F \subseteq V \cup E$ with $|F| \leqslant f$, and storing a static data structure for each computed subnetwork. This is, however, prohibitively expensive w.r.t. both space and preprocessing time as there are $\binom{|V|+|E|}{\leqslant f} = \Theta(n^f + m^f)$ subnetworks if node and link failures can occur. We need a more efficient way to construct subnetworks that allow us to make the data structure fault tolerant. Moreover, we also need some compact and time-efficient indexing scheme to quickly access the correct subnetworks upon query.

(L, f)-Replacement Path Coverings. Weimann and Yuster (2013) were the first to design a data structure with sensitivity f for L-Hop Shortest Path. They used it as a building block to handle general hop-lengths in a so-called *distance sensitivity oracle* (f-DSO). Their main tool was an (L, f)-*replacement path covering*[1], or (L, f)-RPC. This is a family $\mathcal{G}$ of subnetworks of G which, for every failure set

[1]The name was introduced later by Karthik and Parter (2021).

Total Subnetworks	Query Time	Subnetworks Relevant to F	Randomization	Reference
$\widetilde{O}(fL^f)$	$\widetilde{O}(f^2L^f)$	$\widetilde{O}(fL^{f-\lvert F\rvert})$	randomized	Weimann and Yuster (2013)
$\widetilde{O}(fL)^{f+1}$	$\widetilde{O}(f^2L)$	$\widetilde{O}(fL)$	deterministic	Karthik and Parter (2021)
$\widetilde{O}(fL^{f+o(1)})$	$\widetilde{O}(f^{\frac{5}{2}}L^{o(1)})$	$\widetilde{O}(fL^{o(1)})$	randomized	Theorem 1

Table 1: Comparison of (L,f)-replacement path coverings. The sensitivity f and cut-off parameter L satisfy $f=o(\log L)$. The number of subnetworks relevant for the failure set F refers to the size $|\mathcal{G}_F|$, while the query time is the time to compute $\mathcal{G}_F$.

$F\subseteq E$ with $|F|\leqslant f$, has a subfamily $\mathcal{G}_F\subseteq\mathcal{G}$ such that (i) no subnetwork in $\mathcal{G}_F$ contains any element of F and (ii) for every pair of nodes $s,t\in V$, if $G-F$ contains an L-hop shortest s-t path, then some subnetwork in $\mathcal{G}_F$ retains such a path. That means, if an L-hop shortest s-t-path exists in $G-F$, the minimum s-t-distance over all subnetworks in $\mathcal{G}_F$ is the true s-t distance in $G-F$.

Weimann and Yuster (2013) obtained their (L,f)-RPC using $O(fL^f\log n)$ copies of G and, in each one, removing any link independently with probability $1/L$. It is an easy application of Chernoff bounds that w.h.p.[2] there are $|\mathcal{G}_F|=O(fL^{f-|F|}\log n)$ subnetworks that do not contain any link of F and such that, for every $s,t\in V$, at least one element in $\mathcal{G}_F$ retains an L-hop shortest s-t path in $G-F$, if there is any. Unfortunately, due to the randomness, the most efficient way to find $\mathcal{G}_F$ is to go through all subnetworks in $\mathcal{G}$ individually, taking time $O(f^2L^f\log n)$.

Karthik and Parter (2021) derandomized this construction via error-correcting codes. For $L\geqslant f$, their (L,f)-RPC has size $O(fL\log n)^{f+1}$, which is an $O(f^fL\,(\log n)^f)$-factor larger than the solution by Weimann and Yuster. The size of the relevant subfamily $\mathcal{G}_F$ is $\widetilde{O}(fL)$,[3] independently of $|F|$. The huge advantage of the deterministic construction is the much better query time. They showed how to retrieve $\mathcal{G}_F$ in time $\widetilde{O}(f^2L)$. A side-by-side comparison is given in Table 1.

While the randomized (L,f)-RPC is smaller than the deterministic one, the latter has a much better query time. Thus, a natural question is whether one can design an (L,f)-RPC whose size is as small as the former and whose query time is at least as efficient as the latter. We answer this question affirmatively. The bottleneck of the construction by Weimann and Yuster (2013) is not randomness, but *independence*. This is the reason one has to scan all subnetworks in order to find $\mathcal{G}$. Our construction can be seen as an indexing of the Weimann and Yuster graphs. The crucial difference is that the subnetworks we generate are no longer independent. Instead, the construction process naturally groups them into *sampling trees*. The query algorithm merely traces a root-to-leaf path in the tree by always choosing an arbitrary child node whose subnetwork contains no link of F.

Theorem 1. *Let G be a directed/undirected network with n nodes, possibly weighted, and let $f=o(\log L)$. We can build a randomized (L,f)-replacement path covering $\mathcal{G}$ of size $\widetilde{O}(fL^{f+o(1)})$ such that, given any $F\subseteq V\cup E$ with $|F|\leqslant f$, computes in time $\widetilde{O}(f^{\frac{5}{2}}L^{o(1)})$ a collection $\mathcal{G}_F\subseteq\mathcal{G}$ of subnetworks satisfying the following properties w.h.p.:*

(i) $|\mathcal{G}_F|=\widetilde{O}(fL^{o(1)})$;

(ii) No subnetwork in $\mathcal{G}_F$ contains an element of F;

(iii) For any two nodes $s,t\in V$ that admits an L-hop shortest s-t path in $G-F$, there is at least one subnetwork in $\mathcal{G}_F$ that retains one of such paths.

The number of subnetworks in our construction is only an $L^{o(1)}$-factor away from the one by Weimann and Yuster (2013). The $\widetilde{O}(f^{5/2}L^{o(1)})$-query time is even better then the one by Karthik and Parter (2021). The assumption $f=o(\log L)$ is to ensure that parameters of the (L,f)-RPC scale only logarithmically in the size of the input network G (hidden in the $\widetilde{O}$-notation). Our construction also works for larger sensitivities like, e.g., $f=o(\log(n)/\log\log n)$, a common bound in the literature (Weimann and Yuster 2013), (Chechik, Cohen, Fiat, and Kaplan 2017), (Bilò, Chechik, Choudhary, Cohen, Friedrich, and Schirneck 2024), and even up to $f=o(\log n)$. However, then the total number of subnetworks, the number of relevant subnetworks $|\mathcal{G}_F|$, and the query time all increase by an $n^{o(1)}$-factor.

An important open problem is whether our approach can be derandomized without asymptotically affecting the size of the (L,f)-RPC and the query time. Also, Karthik and Parter (2021) gave a lower bound on the number of subnetworks in any (L,f)-RPC. They showed that for any set of integer parameters n, L, f such that $(L/f)^{f+1}\leqslant n$, there exists a network G with n nodes such that any (L,f)-RPC for G must contain $\Omega((L/f)^f)$ networks. Even the construction by Weimann and Yuster (2013) is off by a factor $\widetilde{O}(f^{f+1})$. Closing this gap is another open problem in the area.

Next, we apply the data structure from Theorem 1 to the L-Hop Shortest Path problem.

Theorem 2. *Let G be a (directed) network with n nodes and real edge weights that does not contain negative cycles. Let $f=o(\log L)$. There exists a randomized L-hop distance oracle with sensitivity f for pairwise L-hop shortest paths that takes space $\widetilde{O}(fL^{f+o(1)}n^2)$ and has query time $\widetilde{O}(f^{5/2}L^{o(1)})$. The oracle can be preprocessed in time $\widetilde{O}(fL^{f+o(1)}\,T_{APSP})$, where T_{APSP} is the time to compute all-pairs shortest paths in G.*

It has often been observed that real-world networks are modeled well by networks with small diameter, see the works of Watts and Strogatz (1998); Albert, Jeong, and

[2] We say an event occurs *with high probability* (w.h.p.) if it has probability at least $1-n^c$ for a constant $c>0$.

[3] The $\widetilde{O}(\cdot)$ notation hides poly-logarithmic factors in the number n of nodes of the network.

Query Time	Space	Preprocessing Time	Reference
$\widetilde{O}\left(\left(\frac{f+k}{f}\right)^f\left(\frac{f+k}{k}\right)^k fk\right)$	$\widetilde{O}\left(\left(\frac{f+k}{f}\right)^f\left(\frac{f+k}{k}\right)^k fk\right)$	$\left(\frac{f+k}{f}\right)^f\left(\frac{f+k}{k}\right)^k f \cdot 1.66^k\,\mathsf{poly}(n)$	Bilò et al. (2022)
$f^2\,2^k\,\mathsf{poly}(k)$	$\widetilde{O}(2^k \cdot n^2)$	$2^k\,\mathsf{poly}(k) \cdot n^\omega$	Alman and Hirsch (2022)
$\widetilde{O}\big(4^f f^{2-o(1)} k^{o(1)}\big)$	$\widetilde{O}\left(\left(\frac{k+4}{f}\right)^{f+o(1)} f^{3/2} k\right)$	$\left(\frac{k+4}{f}\right)^{f+o(1)} f^{3/2} \cdot 1.66^k\,\mathsf{poly}(n)$	Theorem 4

Table 2: Comparison of fixed-parameter sensitivity oracles for the k-PATH problem. All results are randomized. The last row assumes $f = o(\log k)$. The quantity $\omega < 2.371552$ is the matrix multiplication exponent.

Barabási (1999); Adamic (1999), the "small-world" networks by Kleinberg (2000), Chung-Lu networks (Chung and Lu 2002), hyperbolic random networks (Friedrich and Krohmer 2018), and the preferential attachment model (Hofstad 2016). Let D be the diameter of an undirected, unweighted network G. By the result of Afek, Bremler-Barr, Kaplan, Cohen, and Merritt (2002) showing that if $G-F$ is still connected, then its diameter is at most $(f+1)D$, Theorem 2 gives a very efficient f-DSO for *general* hop-lengths in networks with, say, polylogarithmic diameter.

Corollary 3. *Let G be an undirected, unweighted network with n nodes, m links, and diameter $D = \omega(1)$. For any constant f, there exists a randomized distance oracle with sensitivity f that takes space $\widetilde{O}(D^{f+o(1)}n^2)$ and has query time $\widetilde{O}(D^{o(1)})$. The oracle can be preprocessed in time $\widetilde{O}(D^{f+o(1)}mn)$ or $\widetilde{O}(D^{f+o(1)}n^\omega)$, where $\omega < 2.371552$ is the matrix multiplication exponent.*

On k-PATH and k-CLIQUE. The k-PATH problem is NP-complete when k is given as part of the input via an easy reduction from Hamilton Path. If k is treated as a parameter, however, then the problem turns out to be *fixed-parameter tractable* (FPT), meaning that it is solvable in time $g(k) \cdot \mathsf{poly}(n)$ for some function g. The current-best algorithms run in time $1.66^k \cdot \mathsf{poly}(n))$ using randomization (Björklund, Husfeldt, Kaski, and Koivisto 2017), or deterministic time $2.554^k \cdot \mathsf{poly}(n)$ (Tsur 2019).

Bilò, Casel, Choudhary, Cohen, Friedrich, Lagodzinski, Schirneck, and Wietheger (2022) introduced *fixed-parameter sensitivity oracles* for FPT-problems. In these oracles the preprocessing time and space requirement must be of the form $g(f,k) \cdot \mathsf{poly}(n)$, and the query time ought to be "significantly faster" than recomputing a solution from scratch. They designed a fixed-parameter oracle with sensitivity f for k-PATH with a query time and space of $O((\frac{f+k}{f})^f(\frac{f+k}{k})^k fk\log n)$, and a $(\frac{f+k}{f})^f(\frac{f+k}{k})^k f2^k \cdot \mathsf{poly}(n)$ preprocessing time. When $f \geqslant k$, Alman and Hirsch (2022) significantly improved the query time to $f^2 2^k \mathsf{poly}(k)$ randomized or $O(f^2 2^{\omega k})$ deterministic. The space requirement is $O(2^k(\log k)n^2)$ in the randomized case and $O(2^k kn^2 \log n)$ for the deterministic version. The preprocessing time is $2^k\mathsf{poly}(k)n^\omega$ (randomized) or $4^k\mathsf{poly}(k)n^\omega$ (deterministic). See Table 2 for an overview.

We consider the case where k is much larger than f, our goal is to make the dependence on k as small as possible.

Theorem 4. *Let G be a directed, unweighted network with n nodes. Let f and k be two integer parameters with $f = o(\log k)$. There exists a randomized fixed-parameter oracle with sensitivity f for k-PATH that takes space $\widetilde{O}((\frac{k+4}{f})^{f+o(1)} f^{3/2} k)$ and has query time $\widetilde{O}(4^f f^{2-o(1)} k^{o(1)})$ w.h.p. The oracle can be preprocessed in randomized time $(\frac{k+4}{f})^{f+o(1)} f^{3/2} \cdot 1.66^k\,\mathsf{poly}(n)$.*

During the analysis, it becomes apparent that the path structure is not actually needed, only the fact that the solution has k links. This allows us to extend sensitivity oracles even to networks motifs that are believed not to have an FPT-algorithm, like k-CLIQUE. It is widely believed k-CLIQUE is not solvable in time $g(k) \cdot \mathsf{poly}(n)$ as the clique detection problem is W[1]-complete. The current best algorithm by Nešetřil and Poljak (1985) computes a clique of k nodes in time $O(n^{\omega k/2})$. We use it to design the first fixed-parameter sensitivity oracle for k-CLIQUE.

Theorem 5. *Let G be an undirected, unweighted network with n nodes. Let f and k be two integer parameters with $f = o(\log k)$. There exists a randomized oracle with sensitivity f for k-CLIQUE that takes space $\widetilde{O}((\frac{k^2+4}{f})^{f+o(1)} f^{3/2} k^2)$ and has query time $\widetilde{O}(4^f f^{2-o(1)} k^{o(1)})$ w.h.p. The oracle can be preprocessed $\widetilde{O}((\frac{k^2+4}{f})^{f+o(1)} f^{3/2} \cdot n^{\omega k/2})$.*

Outline. We first discuss replacement path coverings in general in the next section, followed by their most common application in L-hop distance sensitivity oracles. Afterwards, we show that similar ideas can also be used for k-paths, k-cliques, and other graph motifs.

The Main Tool: Sampling Trees

We turn the idea of removing links to create an (L,f)-replacement path covering upside down. We build a *sampling tree* having the empty subnetwork as its root and, in each level, any child node takes all the links of its parent and randomly *adds* new links from G with some probability depending on f and L, and on an additional parameter h that controls the height of the trees. The probability is fine-tuned in such a way that, among the leaves, the likelihood for any link to exist is precisely the same as it was in the construction of Weimann and Yuster (2013). This brings the number of subnetworks back down to $O(fL^{f+o(1)}\log n)$.

The crucial difference is that the stored subnetworks are no longer independent. Two nodes of the same tree always

share at least all links that are stored in their lowest common ancestor. This arrangement of the subnetworks allows for a very efficient query algorithm even though the construction is randomized. Given a query $F \subseteq E$ with $|F| \leqslant f$, we navigate the sampling tree starting from the root and, in each level, we move to an *arbitrary* child node whose stored subnetwork does not have any link of F. The flip side of such a simple procedure is that the leaf node we reach may be relevant for F only with a small probability; as it will turn out, exponentially small. In turn, it depends only on the parameter h and not on f, L, or the network size. Optimizing h and repeating the query in sufficiently many independent trees ensures a high success probability.

Detailed Construction. Let f, L be positive integers that may depend on n. The *sensitivity* f is assumed to be much smaller than the cut-off parameter L, namely, $f = o(\log L)$. We now implement the data structure that preserves paths with hop-length most L against up to f failing links or nodes in the network. We focus first on fault tolerance against *link failures* and later extend it to node failures. For now, let $F \subseteq E$ with $|F| \leqslant f$. We construct a collection of K sampling trees whose nodes all hold a subnetwork of G. Each sampling tree has height h and any internal node has exactly α children. K, h, and α are parameters to be optimized later.

A single tree has α^h leaves and $O(\alpha^h)$ nodes in total. We associate with each node x a set $A_x \subseteq E$. Intuitively, A_x are the links that are *missing* in the network stored in x; equivalently, node x holds the network $G_x = G - A_x$. If x is the root of the tree, we set $A_x = E$ so that the corresponding network is the empty network (on the same node set V). Now let y be a child of some node x, its set $A_y \subseteq A_x$ is obtained by selecting each link in A_x independently with probability p. This construction is iterated until the tree has height h. In the same fashion, we build all the sampling trees $T_0, T_1, \ldots, T_{K-1}$ where each random choice along the way is made independently of all others. The total number of stored subnetworks is $O(K\alpha^h)$ and the f-covering of G is given by the family $\mathcal{G}$ of all subnetworks that are stored in the leaves of all the K sampling trees.

We will heavily use the fact that the link distribution in the random sets A_x only depends on the depth r of the node x. Moreover, even though two different nodes are not independent, the links present in a single node indeed are. The following lemma has a simple proof by induction over r.

For a positive integer ℓ, we denote $\{0, \ldots, \ell - 1\}$ by $[\ell]$.

Lemma 6. *Let $i \in [K]$ be an index and x a node of the tree T_i at depth $r \geqslant 0$. For any $e \in E$, we have $\mathrm{P}[e \in A_x] = p^r$. Moreover, for any two different links $e, e' \in E$, the events $e \in A_x$ and $e' \in A_x$ are independent.*

Query Algorithm. For the data structure to be efficient, we need to quickly find $\mathcal{G}_F$ when given a query set $F \subseteq E$ with $|F| \leqslant f$. The procedure is summarized in Algorithm 1. Recall that the set $A_x \subseteq E$ contains those links that are *removed* in the subnetwork G_x of the node x. The trees $T_0, \ldots, T_{K-1}$ are searched individually, starting in the respective roots. In each step, the algorithm always chooses some (any) child y of the current node x with $F \subseteq A_x$, and continues the search there. If no such y exists, the current tree is abandoned. Once a leaf is reached, the subnetwork stored there is added to $\mathcal{G}_F$. Up to K subnetworks relevant for the query set F are collected in total time $O(fK\alpha h)$ as every node has α children and the trees have height h.

Algorithm 1: Query algorithm.

```
1  G_F ← ∅;
2  for i = 0 to K−1 do
3      x ← root of T_i;
4      while x is not a leaf do
5          noChildFound ← TRUE;
6          forall children y of x do
7              if F ⊆ A_y then
8                  x ← y;
9                  noChildFound ← FALSE;
10                 break inner for-loop;
11         if noChildFound then
12             continue outer for-loop;
13     G_F ← G_F ∪ {G_x};
```

Analysis. Before we optimize the parameters K, α, h, and p, we show the correctness of the query algorithm. It is clear that no network in $\mathcal{G}_F$ contains a failing link from F since it is explicitly verified that $F \subseteq A_y$ before recursing to y. It could be that $\mathcal{G}_F$ remains empty. This happens if, for every single tree, the outer for-loop of Algorithm 1 is continued in line 12 since a node is encountered whose children all have $F \nsubseteq A_y$. As part of the correctness proof, we bound the probability of this event. In the following, given two nodes $s, t \in V$ and $F \subseteq E$, we denote by $\pi(s, t, F)$ a shortest path from s to t in $G - F$, a.k.a. a *replacement path*.

Lemma 7. *Let $i \in [K]$, $F \subseteq E$ with $|F| \leqslant f$, and $s, t \in V$ such that $\pi = \pi(s, t, F)$ contains at most L links.*

(i) Algorithm 1 reaches a leaf of the sampling tree T_i with probability at least $((1 - (1 - p^f)^\alpha)^h$.

(ii) If Algorithm 1 reaches a leaf x of T_i, then the probability of π existing in G_x is at least $(1 - p^h)^L$.

Proof. To prove Clause (i), we first establish the following claim. Let x be an inner node of T_i with $F \subseteq A_x$ and $y_0, \ldots, y_{\alpha-1}$ its children. Then, the probability that a child has $F \subseteq A_{y_j}$ is at least $1-(1-p^f)^\alpha$. Any set A_{y_j} is obtained from A_x by sampling each link independently with probability p. Since $F \subseteq A_x$, we have $\mathrm{P}[F \subseteq A_{y_j}] = p^{|F|} \geqslant p^f$ and $\mathrm{P}[\,\exists j \in [\alpha] \colon F \subseteq A_{y_j}] = 1 - \prod_{j=0}^{\alpha-1} \mathrm{P}[F \nsubseteq A_{y_j}]$. So the latter probability is lower bounded by $1 - \left(1 - p^f\right)^\alpha$.

The derivation depends only the condition $F \subseteq A_x$ and not on the path through T_i by which the query algorithm reaches the node x. Since Algorithm 1 maintains this condition, we can iterate that argument for each of the h parent-child transitions, which proves Clause (i).

For Clause (ii), consider the algorithm reaching a leaf x of T_i at depth h. Evidently, we have $F \subseteq A_x$, but for all

other links $e \in E \setminus F$, it holds that $\mathrm{P}[e \in A_x] = p^h$ by Lemma 6. The replacement path π survives in G_x with probability $\mathrm{P}[E(\pi) \cap A_x = \emptyset] = (1-p^h)^{|E(\pi)|} \geqslant (1-p^h)^L$. □

Optimizing the Parameters. The number of subnetworks is $O(K\alpha^h)$ out of which the query algorithm selects at most K in time $O(fKah)$. This incentives us to choose all those parameters as small as possible, especially the height h of the sampling trees. Moreover, the probability to reach a leaf of a tree is exponentially small in h (Lemma 7 (i)). However, once a leaf is actually reached, the probability of the stored network holding a relevant replacement path $\pi(s,t,F)$ *grows* exponentially with h. We need to cover *all* pairs $s,t \in V$ that have a shortest path in $G-F$ with at most L links. Our strategy for setting the parameters is to keep the height small and instead boost the success probability by choosing a larger branching factor α and number of independent trees K, and balance the selection with a suitable sampling probability p. Let $c > 0$ be a sufficiently large constant. We set the parameters as $h = \sqrt{f \ln L}$, $K = c(\frac{e}{e-1})^h f \ln n$, $\alpha = L^{f/h}$, and $p = L^{-1/h}$.

Lemma 6 states that in any leaf x, at depth h, the probability for any link to be removed (i.e., $e \in A_x$) is $p^h = 1/L$. Not coincidentally, this is the same probability used by Weimann and Yuster (2013). We verify next that the query algorithm indeed finds a suitable collection of networks. The lemma also implies that the networks stored in the leaves of the trees form an (L,f)-replacement path covering w.h.p.

Lemma 8. *W.h.p. over all $F \subseteq E$ with $|F| \leqslant f$ and $s,t \in V$ with $|E(\pi(s,t,F))| \leqslant L$, after the termination of Algorithm 1, the path $\pi(s,t,F)$ exists in a network of $\mathcal{G}_F$.*

Proof. The proof heavily relies on the estimates derived in Lemma 7. The probability to reach a leaf x in some tree T_i whose corresponding network G_x contains the replacement path $\pi(s,t,F)$ is at least $((1-(1-p^f)^\alpha)^h \cdot (1-p^h)^L$. We estimate the two factors separately starting with the second. Inserting the parameters gives $(1-p^h)^L \geqslant (1-L^{-1})^L \geqslant \frac{1}{4}$. Further, observe that $1-\left(1-p^f\right)^\alpha = 1-\left(1-\frac{1}{L^{f/h}}\right)^{L^{f/h}} \geqslant 1-\frac{1}{e}$. The total probability is thus at least $\frac{1}{4}(1-\frac{1}{e})^h = \frac{1}{4}(\frac{e-1}{e})^h$. Repeating the query in $K = c(\frac{e}{e-1})^h f \ln n$ independent trees reduces the *failure* probability for any triple (s,t,F) to $\left(1-\frac{1}{4}\left(\frac{e-1}{e}\right)^h\right)^{c(\frac{e}{e-1})^h f \ln n} \leqslant e^{-\frac{c}{4} f \ln n} = n^{-\frac{c}{4}f}$. A union bound over the at most $|V^2 \times \binom{E}{\leqslant f}| = O(n^{2+2f})$ triples shows that the failure probability of the whole algorithm is of order $O(n^{2+2f-\frac{c}{4}f})$. Choosing a sufficiently large constant c thus ensures a high success probability. □

We are left to compute the number of networks and query time. Let C abbreviate $\frac{e}{e-1} \approx 1.582$ and note that $C^h = C^{\sqrt{f \ln L}} = \left(L^{\frac{\ln C}{\ln L}}\right)^{\sqrt{f \ln L}} = L^{\ln C \sqrt{\frac{f}{\ln L}}} = L^{o(1)}$. The last estimate uses $f = o(\log L)$. Similarly, we have $\alpha = L^{f/h} = L^{\sqrt{f/\ln L}} = L^{o(1)}$. Our choice of parameters thus implies that the whole data structure stores $O(K\alpha^h) = O(fC^h L^f \log n) = O(fL^{f+o(1)} \log n)$ networks and computes a subfamily of $K = O(fL^{o(1)} \log n)$ of them that are relevant for the failure set F in time $O(fK\alpha h) = O(f(L^{o(1)})^2 f(\log n)\sqrt{f \log L}) = O(f^{5/2} L^{o(1)} \log n)$. This proves Theorem 1 for the case of link failures.

If the sensitivity is up to $f = o(\log n)$, then we have $h = \sqrt{f \ln L} = o(\log n)$ since L is at most n. This results in a total number of subnetworks of $O(fC^h L^f \log n) = L^f n^{o(1)}$, $|\mathcal{G}_F| = O(fC^h \log n) = n^{o(1)}$ of which are relevant for a given query, and a query time of $O(f^2 C^h L^{f/h} h) = n^{o(1)}$.

Node Failures. The changes needed to accompany node failures are minuscule. Instead of a set of links, we now associate with every tree node x a set $A_x \subseteq V$ of network nodes. We have $A_x = V$ in the roots and in each child y of x, we include any element of A_x independently with probability p. Note that then the network $G_x = G - A_x$ is the subnetwork of G *induced* by the node set $V \setminus A_x$. As the sampling remains the same, we get an analog of Lemma 6.

Lemma 9. *Let $i \in [K]$ be an index and x a node of the tree T_i at depth $r \geqslant 0$. For any $v \in v$, we have $\mathrm{P}[v \in A_x] = p^r$. For any two different network nodes $v, v' \in V$, the events $v \in A_x$ and $v' \in A_x$ are independent.*

The other lemmas follow from this almost verbatim as before. The only differences are that, if Algorithm 1 reaches a leaf, then the probability of the replacement path existing in the subnetwork is $(1-p^h)^{L+1}$ for if the replacement path has up to L links, it can have up to $L+1$ nodes. This changes the correction factor in Lemma 8 to $(1-L^{-1})^{L+1} \geqslant \frac{1}{8}$, which is counter-acted by choosing the constant c in the definition of the number of trees K marginally larger. In contrast to link failures, there are now at most $|V^2 \times \binom{V}{\leqslant f}| = O(n^{2+f})$ relevant queries to cover.

Distance Sensitivity Oracles

Recall that, given $s,t \in V$ and a set F of at most F failures, an L-hop f-DSO reports an overestimate of the length of $\pi(s,t,F)$ that matches the lower bound if $G-F$ contains an L-hop shortest s-t path. It is straightforward to turn our generic tree structure of Theorem 1 into an L-hop f-DSO. After all, that was the original purpose of Weimann and Yuster (2013) when defining (L,F)-RPCs. The only difference we make is to precompute for every leaf x of all sampling trees the pairwise distances of nodes in G_x. The query works exactly as Algorithm 1 only that, upon query (s,t,F), in line 13 the stored *distance* from s to t in G_x is recorded. The value returned by our L-hop f-DSO is given by the minimum of the computed distances.

The construction of the oracle is dominated by preparing the distances in the leaves. The preprocessing time is $\widetilde{O}(fL^{f+o(1)} \cdot T_{\text{APSP}})$, where T_{APSP} is the time needed to compute all-pairs shortest paths (APSP) in an arbitrary subnetwork of G. It depends on the properties of G. If there are no negative cycles (e.g., because all link weights are non-negative), one can use Dijkstra's or Johnson's algorithm running in time $T_{\text{APSP}} = \widetilde{O}(mn)$. If instead one is willing to use fast matrix multiplication, then APSP can be

computed in time $\widetilde{O}(Mn^\omega)$ for undirected networks with non-negative integer link weights in $[M]$, or $O(Mn^{2.5286})$ for directed networks with integer weights in $\{-M, \dots M\}$ (Alon, Galil, and Margalit 1997; Seidel 1995; Shoshan and Zwick 1999; Zwick 2002). Here, $\omega < 2.371552$ is the matrix multiplication exponent (Duan, Wu, and Zhou 2023; Vassilevska Williams, Xu, Xu, and Zhou 2024).

The space of the data structure is $\widetilde{O}(fL^{f+o(1)}n^2)$, again dominated by storing the distances in the leaves. In turn, the query time remains at $\widetilde{O}(f^{5/2}L^{o(1)})$ as the values $d_{G_x}(s,t)$ can be looked up in constant time. This proves Theorem 2.

Sensitivity Oracles for k-PATH

We now turn to the fixed-parameter sensitivity oracle for the NP-complete k-PATH problem. We only treat link failures here, (i.e., $F \subseteq E$) to ease notation. We further assume that access to an algorithm that computes simple paths with k links (k-paths) in subnetworks of G in a way that respects a certain *inheritance property*. Suppose F has at most f links, $G-F$ has a k-path and the algorithm we use produces such a path P. Then for any subnetwork $H \subseteq G-F$ that still contains P, we require that the same path P is also the output on of the algorithm in H. Note that such a tie-breaking scheme is obtained by assign distinct weights to the links in E and always choosing the k-path of minimum weight.

Our data structure is based on the sampling trees above, where we naturally set $L = k$ and thus assume $k = o(\log k)$. The construction of the trees with parameters K, α, h, and p is almost as before. The difference is that any node x of a tree T_i is not only associated with one set of links A_x, but also with a second one S_x. In a parent-child traversal from x to y, A_y is still obtained from A_x by sampling each link of A_x with probability p.

The construction of S_y is a bit more involved. Let r be the depth of the parent, and thus $r+1$ the depth of the child y. Let S_x be the set associated with its parent. To unify the exposition, if y is the root, we set $S_x = E$. To construct S_y from S_x, we first build a family $\mathcal{P}_y$ of paths in $4^f \alpha^{h-(r+1)} \ln h$ independent rounds. In each round, we sample a subset $I \subseteq A_y$ by selecting each link in A_y independently with probability $p^{h-(r+1)}/2$. If there exists a k-path in the network with link set $S_x \backslash I$, we add the one computed with the inheritance property to $\mathcal{P}_y$. After the last round, we set S_y to the union of the paths in $\mathcal{P}_y$. This ensures $S_y \subseteq S_x$. Intuitively, for an (inherited) k-path from S_x to *not* survive in $\mathcal{P}_y$, it must have been destroyed by *all* the random deletions I. If x is a leaf, we not only store S_x but also the path information in $\mathcal{P}_x$.

The query algorithm is very similar to Algorithm 1. The difference is that line 7 now checks for $F \cap S_x \subseteq A_y$ (previously, $F \subseteq A_y$). In line 13, where a leaf x is reached, the query algorithm searches $\mathcal{P}_x$ for a k-path that is disjoint from F. If one exists, it is output and the whole algorithm terminates; if no such path exists, the search continues with the next tree. If none of the trees $T_0, \dots, T_{K-1}$ produce such a path, it is reported that $G-F$ does not have a k-path.

Analysis. We use different parameters for this oracle: $h = \sqrt{f \ln(k/f)}$, $K = c\ 8^h f \ln n$, $\alpha = (k/f)^{f/h}$, and $p = (f/k)^{1/h}$. Note that $\alpha = 1/p^f$ still holds.

Let y be a node in the tree T_i, r its depth, and S_x the edge set in the parent ($S_x = E$ in the root). Assume $G-F$ has a k-path, let P be the one computed with the inheritance property. $E(P)$ is its set of links. We say y is *well behaved* if it satisfies the following properties: (P1) $F \cap S_x \subseteq A_y$, (P2) $|E(P) \cap A_y| \leqslant p^r k$, and (P3) $P \in \mathcal{P}_y$.

The query algorithm enforces (P1) in every step of the way. We are mainly interested in the probability of (P3) holding in the leave that is reached for a query. We bound this using well-behaved children.

Lemma 10. *Let $i \in [K]$ and x a non-leaf node in T_i.*

- *(i) If x satisfies* (P1)*, then there exists a child y of x also satisfying* (P1) *with probability is at least $1 - \frac{1}{e}$.*
- *(ii) If x satisfies* (P2)*, then, for any child y of x, the probability that y also satisfies* (P2) *is at least $\frac{1}{4}$.*
- *(iii) If x is well behaved and has a child y that satisfies both* (P1) *and* (P2)*, then the probability that y is even well behaved (satisfies* (P3)*) is at least $1 - \frac{1}{h}$.*

Moreover, the events are independent of each other.

Proof. Suppose x satisfies (P1), let S_z be the set of its parent (or $S_x = E$) and let $y_0, \dots, y_{\alpha-1}$ be the children of x. We have $F \cap S_x \subseteq F \cap S_z \subseteq A_x$. The first inclusion is due to $S_x \subseteq S_z$ and the second one due to (P1). Since the elements of A_{y_j} are sampled from A_x with probability p, there exists an index $j \in [\alpha]$ such that $F \cap S_x \subseteq A_{y_j}$ with probability $1 - \prod_{j=0}^{\alpha-1} \mathrm{P}[F \cap S_x \nsubseteq A_{y_j}] \geqslant 1 - (1-p^f)^\alpha \geqslant 1 - \frac{1}{e}$.

We turn to Clause (ii). Let r be the depth of x. We now assume that x satisfies (P2) (but not necessarily the other properties). That means, at most a p^r ratio of the k links of the path P are removed in the network G_x (are in the set A_x). For any child y of x, the *expected* size of $E(P) \cap A_y$ is $p \cdot |E(P) \cap A_x| \leqslant p^{r+1}k$. In fact, the random variable $|E(P) \cap A_y|$ is binomially distributed with parameters $|E(P) \cap A_x|$ and p. The Central Limit Theorem states that $\mathrm{P}[|E(P) \cap A_y| > p^{r+1}k] \leqslant \frac{3}{4}$. Since the depth of y is $r+1$, that implies that y satisfies (P2) with probability $\frac{1}{4}$.

The main part of this proof is Clause (iii) as it involves the new sets S_y. Let x be well behaved and its child y satisfy (P1) and (P2). Consider any of the round in the creation of S_y and let $I \subseteq A_y$ be the sampled subset. We want to know whether the specific k-path P from $G-F$ is included in $\mathcal{P}_y$ in this round. Due to the inheritance property, it is sufficient that $E(P) \subseteq (S_x \backslash I)$ and at the same time $F \cap (S_x \backslash I) = \emptyset$. The latter condition formalizes that the network with edge set $S_x \backslash I$ is a subnetwork of $G-F$.

We bound the probability of the two events. Parent x is well behaved, a fortiori it satisfies (P3), thus $E(P) \subseteq S_x$. Each link in I is drawn from A_y with probability $p^{h-(r+1)}/2$. We have $E(P) \subseteq (S_x \backslash I)$ if none of the links in $E(P) \cap A_y$ are drawn. Since y satisfies (P2), this has probability at least $(1 - \frac{p^{h-(r+1)}}{2})^{|E(P) \cap A_y|} \geqslant (1 - \frac{p^{h-(r+1)}}{2})^{p^{r+1}k}$. Inserting the definition $p = (\frac{f}{k})^{1/h}$, we get $p^{h-(r+1)} =$

$\frac{f}{p^{r+1}k}$ and $p^{r+1}k \geqslant p^h k = f$. Above estimate gives $(1 - \frac{p^{h-(r+1)}}{2})^{p^{r+1}k} = (1 - \frac{f}{2 \cdot p^{r+1}k})^{p^{r+1}k} \geqslant (1 - \frac{f}{2 \cdot f})^f = \frac{1}{2^f}$.

Next is the probability that the sets F and $S_x \backslash I$ are disjoint. Since the child y also satisfies (P1), we have $F \cap S_x \subseteq A_y$. The sample I is also a subset of A_y. So for $F \cap S_x \backslash I = \emptyset$ all links in $F \cap S_x$ must be selected for I. This event has probability $(\frac{p^{h-(r+1)}}{2})^{|F \cap S_x|} \geqslant (\frac{p^{h-(r+1)}}{2})^f = \frac{p^{f(h-(r+1))}}{2^f}$. Recall that we chose the parameter such that $p^f = \alpha^{-1}$. The last estimate is thus equal to $2^{-f}\alpha^{-(h-(r+1))}$.

Since the events $E(P) \subseteq (S_x \backslash I)$ and $F \cap (S_x \backslash I) = \emptyset$ are independent, they occur together with probability at least $4^{-f}\alpha^{-(h-(r+1))}$. We give the construction algorithm of our data structure $4^f \alpha^{h-(r+1)} \ln h$ rounds to try for it. The probability to include the path P in $\mathcal{P}_y$ in any of the rounds is thus at least $1 - \left(1 - \frac{1}{4^f \alpha^{h-(r+1)}}\right)^{4^f \alpha^{h-(r+1)} \cdot \ln h} \geqslant 1 - \frac{1}{h}$. □

We now prove the correctness of the query algorithm. If $G - F$ does not have a k-path, then the procedure indeed reports this fact. If it reaches a leaf x at all, it explicitly scans $\mathcal{P}_x$ for a path that is disjoint from F. In the case that $G - F$ has a k-path, we argue over the parent-child traversals. It is convenient to also include the algorithm jumping into the root of a tree as the first step. Recall that for the root y, we set $A_y = E$ and use the convention $S_x = E$. Therefore, y trivially satisfies the properties (P1) and (P2). Lemma 10 thus shows that the root is well behaved with probability $(1 - \frac{1}{e})\frac{1}{4}(1 - \frac{1}{h})$. Furthermore, since the algorithm actively looks for a child satifying (P1), Lemma 10 can also be iterated over the following h traversals. In summary a well-behaved child is reached with probability $((1 - \frac{1}{e})\frac{1}{4}(1 - \frac{1}{h}))^{h+1} \geqslant \frac{1}{8^h}$. Repeating this in all $K = c \cdot 8^h f \ln n$ trees for a sufficiently large constant $c > 0$ gives a high success probability over the $|\binom{E}{\leqslant f}| = O(n^{2f})$ possible query sets F.

Query Time, Space, and Preprocessing Time. Recall that we chose the parameters as $K = c8^h f \ln n$, $\alpha = (k/f)^{f/h}$, and $h = \sqrt{f \ln(k/f)}$. As before, this implies $8^h = 8^{\sqrt{f \ln(k/f)}} = (k/f)^{\frac{\ln 8}{\ln(k/f)}\sqrt{f \ln(k/f)}} = (k/f)^{o(1)}$, where we use $f = o(\log k)$. That means, we have $K = (k/f)^{o(1)} f \ln n$. However, the assumption on f also implies that $\log f = o(\log k)$ (for any positive base of the logarithm). Via $\log(k/f) = \log(k) - \log(f) = (1 - o(1)) \log(k)$, we get the seemingly stronger statement $f = o(\log(k/f))$. We conclude $\alpha = (k/f)^{f/h} = (k/f)^{\sqrt{f/\ln(k/f)}} = (k/f)^{o(1)}$.

The total query time is $O(K\alpha h f + K f 4^f \log h)$. The first term is derived as in the generic construction, assuming that the test $F \cap S_x \subseteq A_y$ can be done in $O(f)$ time. We explain below how to implement that. The second term describes the time needed to search the collection $\mathcal{P}_x$ in all the leaves the query algorithm may reach. Using the parameters gives $O((k/f)^{o(1)} f^2 \sqrt{f \ln(k/f)} \ln n \; + \; (k/f)^{o(1)} f^2 4^f \log h)$. The second term of order $\widetilde{O}(4^f f^{2-o(1)} k^{o(1)})$ is dominating.

For the space, consider a node y in one of the trees that is not a root, let S_x be the second set associated with its parent. We define $B_y = S_x \cap A_y$. Observe that we have $F \cap S_x \subseteq A_y$ if and only if $F \cap S_x \subseteq B_y$. So it is enough to store B_y instead of A_y and we can indeed make that check in time $O(|F \cap S_x|) = O(f)$. It will be advantageous for the analysis that both B_y and S_y are subsets of S_x.

The final data structure stores the trees T_i for all $i \in [K]$; in each root x the set S_x; in each non-root node y the sets B_y and S_y; and in each leaf y the set S_y and paths in $\mathcal{P}_y$.

Let r be the depth of a node x, and let y be a child of x. It is enough to bound the size of the path collection $\mathcal{P}_x$ since it dominates the size of the union S_x, and in turn the sizes of the subsets B_y and S_y. $\mathcal{P}_x$ contains at most $4^f \alpha^{h-r} \ln h$ different k-paths and thus takes space $O(4^f \alpha^{h-r} k \log h)$. Note that this is independent of the input graph G.

There are α^r many nodes in a tree at depth r, giving space $\widetilde{O}(4^f \alpha^h k)$ per level, and $\widetilde{O}(Kh4^f \alpha^h k)$ for all levels and trees together. Due to $\alpha^h = (k/f)^f$, this is $\widetilde{O}(4^f f^{3/2} (k/f)^{f+o(1)} k) = \widetilde{O}(((k+4)/f)^{f+o(1)} f^{3/2} k)$.

For the preprocessing, we use the k-path algorithm by Björklund et al. (2017) running in time $1.66^k \mathsf{poly}(n)$. We spend time $4^f \alpha^{h-r} \cdot 1.66^k \mathsf{poly}(n)$ preprocessing one tree node at depth r, $4^f \alpha^h \cdot 1.66^k \mathsf{poly}(n)$ in each level, and $((k+4)/f)^{f+o(1)} f^{3/2} \cdot 1.66^k \mathsf{poly}(n)$ in total.

Cliques, Stars, and Cycles. Throughout the analysis, we never actually used the path structure of the solution, only that it had k edges. We can thus replace k-paths by any other graph motif we desire. However, be aware that k-cliques have $\binom{k}{2} = O(k^2)$ edges. Also, there are no known algorithms for finding k-cliques in time $g(k) \cdot \mathsf{poly}(n)$.

References

Adamic, L. A. 1999. The Small World Web. 443–452. Springer.

Afek, Y.; Bremler-Barr, A.; Kaplan, H.; Cohen, E.; and Merritt, M. 2002. Restoration by Path Concatenation: Fast Recovery of MPLS Paths. *Distributed Computing*, 15: 273–283.

Albert, R.; Jeong, H.; and Barabási, A.-L. 1999. Diameter of the World-Wide Web. *Nature*, 401: 130–131.

Alman, J.; and Hirsch, D. 2022. Parameterized Sensitivity Oracles and Dynamic Algorithms Using Exterior Algebras. In *Proceedings of the 49th International Colloquium on Automata, Languages, and Programming (ICALP)*, 9:1–9:19.

Alon, N.; Galil, Z.; and Margalit, O. 1997. On the Exponent of the All Pairs Shortest Path Problem. *Journal of Computer and System Sciences*, 54: 255–262.

Bilò, D.; Chechik, S.; Choudhary, K.; Cohen, S.; Friedrich, T.; Krogmann, S.; and Schirneck, M. 2023. Approximate Distance Sensitivity Oracles in Subquadratic Space. In *Proceedings of the 55th Symposium on Theory of Computing (STOC)*, 1396–1409.

Bilò, D.; Chechik, S.; Choudhary, K.; Cohen, S.; Friedrich, T.; and Schirneck, M. 2024. Improved Distance (Sensitivity) Oracles with Subquadratic Space. In *Proceedings of the 65th Symposium on Foundations of Computer Science (FOCS)*, 1550–1558.

Bilò, D.; Casel, K.; Choudhary, K.; Cohen, S.; Friedrich, T.; Lagodzinski, J. G.; Schirneck, M.; and Wietheger, S. 2022. Fixed-Parameter Sensitivity Oracles. In *Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS)*, 23:1–23:18.

Björklund, A.; Husfeldt, T.; Kaski, P.; and Koivisto, M. 2017. Narrow Sieves for Parameterized Paths and Packings. *Journal of Computer and System Sciences*, 87: 119–139.

Chechik, S.; Cohen, S.; Fiat, A.; and Kaplan, H. 2017. $(1+\varepsilon)$-Approximate f-Sensitive Distance Oracles. In *Proceedings of the 28th Symposium on Discrete Algorithms (SODA)*, 1479–1496.

Chechik, S.; Langberg, M.; Peleg, D.; and Roditty, L. 2012. f-Sensitivity Distance Oracles and Routing Schemes. *Algorithmica*, 63: 861–882.

Cho, K.; Shin, J.; and Oh, E. 2024. Approximate Distance Oracle for Fault-Tolerant Geometric Spanners. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI)*, 20087–20095.

Chung, F.; and Lu, L. 2002. The Average Distances in Random Graphs with Given Expected Degrees. *Proceedings of the National Academy of Sciences*, 99: 15879–15882.

Demetrescu, C.; Thorup, M.; Chowdhury, R. A.; and Ramachandran, V. 2008. Oracles for Distances Avoiding a Failed Node or Link. *SIAM Journal on Computing*, 37: 1299–1318.

Dey, D.; and Gupta, M. 2024. Nearly Optimal Fault Tolerant Distance Oracle. In Mohar, B.; Shinkar, I.; and O'Donnell, R., eds., *Proceedings of the 56th Symposium on Theory of Computing (STOC)*, 944–955.

Duan, R.; and Pettie, S. 2009. Dual-Failure Distance and Connectivity Oracles. In *Proceedings of the 20th Symposium on Discrete Algorithms (SODA)*, 506–515.

Duan, R.; and Pettie, S. 2010. Connectivity Oracles for Failure Prone Graphs. In Schulman, L. J., ed., *Proceedings of the 42nd Symposium on Theory of Computing (STOC)*, 465–474.

Duan, R.; and Pettie, S. 2017. Connectivity Oracles for Graphs Subject to Vertex Failures. In *Proceedings of the 28th Symposium on Discrete Algorithms (SODA)*, 490–509.

Duan, R.; and Ren, H. 2022. Maintaining Exact Distances under Multiple Edge Failures. In *Proceedings of the 54th Symposium on Theory of Computing (STOC)*. To appear.

Duan, R.; Wu, H.; and Zhou, R. 2023. Faster Matrix Multiplication via Asymmetric Hashing. In *Proceedings of the 64th Symposium on Foundations of Computer Science (FOCS)*, 2129–2138.

Friedrich, T.; and Krohmer, A. 2018. On the Diameter of Hyperbolic Random Graphs. *SIAM Journal on Discrete Mathematics*, 32: 1314–1334.

Gu, Y.; and Ren, H. 2021. Constructing a Distance Sensitivity Oracle in O(n^2.5794 M) Time. In Bansal, N.; Merelli, E.; and Worrell, J., eds., *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, 76:1–76:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Hofstad, R. v. d. 2016. *Random Graphs and Complex Networks*, volume 1 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge, UK: Cambridge University Press.

Karthik C.S.; and Parter, M. 2021. Deterministic Replacement Path Covering. In *Proceedings of the 32nd Symposium on Discrete Algorithms (SODA)*, 704–723.

Kleinberg, J. M. 2000. Navigation in a Small World. *Nature*, 406: 845–845.

Nešetřil, J.; and Poljak, S. 1985. On the Complexity of the Subgraph Problem. *Commentationes Mathematicae Universitatis Carolinae*, 26: 415–419.

Patrascu, M.; and Thorup, M. 2007. Planning for Fast Connectivity Updates. In *Proceedings of the 48th Symposium on Foundations of Computer Science (FOCS)*, 263–271.

Seidel, R. 1995. On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs. *Journal of Computer and System Sciences*, 51: 400–403.

Shoshan, A.; and Zwick, U. 1999. All Pairs Shortest Paths in Undirected Graphs with Integer Weights. In *Proceedings of the 40th Symposium on Foundations of Computer Science (FOCS)*, 605–615.

Tsur, D. 2019. Faster Deterministic Parameterized Algorithm for k-Path. *Theoretical Computer Science*, 790: 96–104.

Vassilevska Williams, V.; Xu, Y.; Xu, Z.; and Zhou, R. 2024. New Bounds for Matrix Multiplication: from Alpha to Omega. In *Proceedings of the 35th Symposium on Discrete Algorithms (SODA)*, 3792–3835.

Watts, D. J.; and Strogatz, S. H. 1998. Collective dynamics of 'small-world' networks. *Nature*, 393: 440–442.

Weimann, O.; and Yuster, R. 2013. Replacement Paths and Distance Sensitivity Oracles via Fast Matrix Multiplication. *ACM Transactions on Algorithms*, 9: 14:1–14:13.

Zwick, U. 2002. All Pairs Shortest Paths Using Bridging Sets and Rectangular Matrix Multiplication. *Journal of the ACM*, 49: 289–317.