Fully Dynamic Approximate Distance Oracles for Planar Graphs via Forbidden-Set Distance Labels

Ittai Abraham
Microsoft Research
Silicon Valley Center, USA
ittaia@microsoft.com

Shiri Chechik
Dept. of Computer Science
Weizmann Institute, Israel
shiri.chechik@weizmann.ac.il

Cyril Gavoille**
Université de Bordeaux
LaBRI, France
gavoille@labri.fr

ABSTRACT

This paper considers fully dynamic $(1+\varepsilon)$ distance oracles and $(1+\varepsilon)$ forbidden-set labeling schemes for planar graphs. For a given n-vertex planar graph G with edge weights drawn from $[1, M]$ and parameter $\varepsilon > 0$, our forbidden-set labeling scheme uses labels of length $\lambda = O(\varepsilon^{-1} \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$. Given the labels of two vertices s and t and of a set F of faulty vertices/edges, our scheme approximates the distance between s and t in $G \setminus F$ with stretch $(1+\varepsilon)$, in $O(|F|^2\lambda)$ time.

We then present a general method to transform $(1+\varepsilon)$ forbidden-set labeling schemas into a fully dynamic $(1+\varepsilon)$ distance oracle. Our fully dynamic $(1+\varepsilon)$ distance oracle is of size $O(n \log n \cdot (\varepsilon^{-1} + \log n))$ and has $\tilde{O}(n^{1/2})$ query and update time, both the query and the update time are worst case. This improves on the best previously known $(1+\varepsilon)$ dynamic distance oracle for planar graphs, which has worst case query time $\tilde{O}(n^{2/3})$ and amortized update time of $\tilde{O}(n^{2/3})$.

Our $(1+\varepsilon)$ forbidden-set labeling scheme can also be extended into a forbidden-set labeled routing scheme with stretch $(1+\varepsilon)$.

Categories and Subject Descriptors: F.2.2 Analysis of Algorithms and Problem Complexity: Nonnumerical Algorithms and Problems

General Terms: Algorithms, Theory

Keywords: planar graphs, forbidden sets, fault-tolerance, distance labeling, fully dynamic, distance oracles, compact routing

1. INTRODUCTION

A *dynamic distance oracle* is a preprocessed data structure capable of handling an adversarial online sequence of update and query operations. The objective is usually to minimize the query time, the update time and the space of the data structure. Each distance query involves two vertices s and t whose distance needs to be computed. Each update operation involves inserting/deleting a vertex/edge from the graph. A dynamic algorithm is said to be incremental if it handles only insertion operations, decremental if it handles only deleting operations and fully dynamic if it handles both operations.

We say that the stretch of a distance oracle is k (or simply write k-distance oracle) if the returned distance is at least the real distance and at most k times the real distance.

A closely related notion is that of *forbidden-set distance oracles*. A forbidden-set distance oracle for a given graph G is a processed data structure capable of answering distance queries of the form (s, t, F) where F is a set of faulty vertices (or edges) and s and t are vertices whose distance needs to be computed in the graph $G \setminus F$.

A *forbidden-set distance labeling scheme* is a mechanism that assigns each vertex a small *label*. We are then given distance queries where each query involves a set of forbidden vertices or edges F, and two vertices s and t, and the goal is to approximate $\mathbf{dist}(s, t, G \setminus F)$ using only the labels of s, t and the elements of F (the label of an edge in F is given by the pair of labels of its endpoints). Where $\mathbf{dist}(s, t, G \setminus F)$ is the distance between s and t in the graph $G \setminus F$.

In this paper, we consider dynamic distance oracles and forbidden-set distance labeling scheme for weighted planar graphs. We present several results. The main technical result is an efficient forbidden-set distance labeling scheme with stretch $(1+\varepsilon)$ that is capable of handling any number of vertex/edge failures.

THEOREM 1. *Given an n-vertex planar graph G with edge weights in $[1, M]$ and a parameter $\varepsilon > 0$, one can construct a forbidden-set $(1+\varepsilon)$-approximate distance labeling scheme of label length $\lambda = O(\varepsilon^{-1} \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$. All the labels can be computed in $O(n\lambda)$ time, and each query can be answered in $O(|F|^2\lambda)$ time, where F is the set of faulty vertices/edges.*

We then show that our $(1+\varepsilon)$ forbidden-set labeling scheme can be used to give a fully dynamic $(1+\varepsilon)$ distance oracle with $\tilde{O}(\sqrt{n})$ worst case cost:

THEOREM 2. *Given an n-vertex planar graph G with edge weights in $[1, M]$ and a parameter $\varepsilon > 0$, one can construct a fully dynamic $(1+\varepsilon)$-approximate distance oracle of size $O(n \log n \cdot (\varepsilon^{-1} + \log n))$ that can be constructed in $O(\varepsilon^{-1} n \log^2 n)$ time. Each query operation and each up-*

*Supported by the ANR-11-BS02-014 project "DISPLEXITY", the IUF, the equipe-projet INRIA "CEPAGE", and the European project "EULER".

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
STOC'12, May 19–22, 2012, New York, New York, USA.
Copyright 2012 ACM 978-1-4503-1245-5/12/05 ...$10.00.

date operation takes $O(\varepsilon^{-1}n^{1/2}\log^2 n\log(nM)\cdot(\varepsilon^{-1}+\log n))$ *worst case time.*

Our techniques for transforming the labeling scheme into a fully dynamic approximate distance oracle is general and can be used in other settings. Specifically, one can combine our techniques with the forbidden labeling schemes presented in [1] and [12] to get fully dynamic $(1+\varepsilon)$ distance oracle for graphs of bounded doubling dimension and fully dynamic exact distance oracle for graphs of bounded tree-width or clique-width. Both these oracles are of size $\tilde{O}(n)$ and with worst case update and query time $\tilde{O}(n^{1/2})$.

In addition, we show that we can easily transform our forbidden-set labeling scheme to a forbidden-set distance oracle and a forbidden-set compact routing scheme:

THEOREM 3. *Given an n-vertex planar graph G with edge weights in $[1,M]$ and a parameter $\varepsilon>0$, one can construct a centralized $(1+\varepsilon)$-approximate distance oracle of size $O(n\log n\cdot(\varepsilon^{-1}+\log n))$ that can be constructed in $O(\varepsilon^{-1}n\log^2 n)$ time that supports the following operations. Given an update operation that involves adding/removing an edge/node, the data structure can be modified in $O(\varepsilon^{-1}|U|\log^2 n\log(nM)\cdot(\varepsilon^{-1}+\log n))$ time so that distance queries in the modified graph can be answered in $O(\varepsilon^{-1}|U|\log^2 n\log(nM)\cdot(\varepsilon^{-1}+\log n))$ time, where U is the set of updates that occurred so far.*

THEOREM 4. *Given an n-vertex planar graph G with edge weights in $[1,M]$ and a parameter $\varepsilon>0$, one can construct a forbidden-set routing labeling scheme with stretch $(1+\varepsilon)$, and $o(\varepsilon^{-2}\log^4 n\log(nM))$ labels length, and $o(\varepsilon^{-1}\log^3 n)$ tables size.*

All our sizes are measured in the number of words, like in [29]. Where a word is a space unit big enough to contain an ID of a vertex in the graph or a distance.

Related work.

Our result on forbidden-set distance labeling scheme fall into the framework of algorithms that get ready for a number of failures. As opposed to dynamic algorithms where the assumption is that the network tends to change over time, here the assumption is that the network is fixed and some of the edges/verices may occasionally crash. Pătraşcu and Thorup [27] considered the problem of connectivity oracle in this framework. They showed that one can efficiently construct a linear size data structure such that given a set F of f edge failures and two nodes s and t, can determine in $\tilde{O}(f)$ time if s and t are still connected in the remaining graph. In a similar spirit, we show that one can preprocess a planar graph into a labeling scheme that supports a distance query such that given a set of f failures in $\tilde{O}(f^2)$ time.

There is a vast literature on distance labeling schemes and distance oracles in the failure-prone case, below we mention only the most relevant ones.

The problem of fully dynamic approximate distance oracle for planar graphs was considered by Klein and Subramanian in [24]. Klein and Subramanian show how to construct a $(1+\varepsilon)$-distance oracle for planar graphs with worst case query time of $\tilde{O}(n^{2/3})$ and amortized update time of $\tilde{O}(n^{2/3})$. In this paper, we break these long-standing bounds, showing a fully dynamic distance oracle for planar graphs with worst case query and update time of $\tilde{O}(n^{1/2})$. Note that while the update time in [24] is amortized, both our update time and query time are worst case.

Dynamic distance oracles for general graphs have been extensively studied. Many papers considered the inceremental/decremental/fully dynamic distance oracles with exact distances and achieved different tradeoffs (e.g. [2, 19, 23, 3, 30, 13, 31, 14]). A major breakthrough for fully dynamic exact distance oracle was developed by Demetrescu and Italiano [13]. Demetrescu and Italiano devise a fully dynamic exact distance oracle for directed general graphs with non negative edge weights, with amortized update time $\tilde{O}(n^2)$. Thorup [30] later extended the result to negative edge weights and slightly improved the update time. Thorup [31] also presented an algorithm with worst case update time $\tilde{O}(n^{2.75})$. Baswana et al. [3] considered the decremental case for unweighted directed graphs and obtained amortized update time $\tilde{O}(n^3/m)$.

Approximate dynamic algorithms were also explored for general graphs (e.g. [3, 4, 33, 6]). Roditty and Zwick [33] devise efficient distance oracles for the only incremental and for the only decremental cases, both with $(1+\varepsilon)$ stretch, constant query time and amortized update time $\tilde{O}(n)$.

Bernstein [6] considered the fully dynamic case and presented an algorithm for weighted, undirected graphs with $O(\log\log\log n)$ query time and update time close to $\tilde{O}(m)$ and $(2+\varepsilon)$ stretch. Recently, Roditty and Bernstein [8] present an only decremental distance oracle for unweighted undirected graphs with $(2k-1+\varepsilon)$ stretch, $O(k)$ query time and about $\tilde{O}(n^{2+1/k}/m)$ amortized update time.

Distance oracles supporting failures were also studied. A distance oracle for answering exact distance queries with a single edge fault for general directed graphs is presented in [15]. The size of the data structure is $O(n^2\log n)$ and the query time is $O(\log n)$. This was extended to handle single vertex/edge failures [7], and recently to dual-failures [16]. Approximate distance oracle supporting multiple edge failures is presented in [10].

Routing scheme supporting failures for general graphs has been explored in [22] for single vertex failure and in [10] for two edge failures. Recently, a routing scheme for general graphs supporting multiple edge failures has been presented in [9].

A forbidden-set distance labeling scheme for graphs of bounded treewidth or cliquewidth allowing multiple edge/vertex failures is presented by Courcelle and Twigg [11, 12]. Their scheme uses $O(k^2\log^2 n)$-bits size for graphs of treewidth or cliquewidth k. Their scheme is also extended to forbidden-set routing scheme. Courcelle and Twigg [12] raised as a main open problem the question of constructing efficient forbidden-set distance labeling scheme for planar graphs. In this paper we present a $(1+\varepsilon)$ stretch answer for the open problem.

Later, [1] devise a forbidden-set distance labeling scheme for unweighted graphs of bounded doubling dimension. For stretch $(1+\varepsilon)$, the scheme uses $O(1+\varepsilon^{-1})^{2\alpha}\log^2 n$-bit labels, where α is the doubling dimension of the graph. This scheme is also extended to forbidden-set routing scheme.

Distance oracles for planar graphs in the failure free setting have been explored by Thorup [28, 29] and Klein [25]. They show how to construct a $(1+\varepsilon)$-distance oracle with $O(\varepsilon^{-1}n\log n)$ space and $O(\varepsilon^{-1})$ query time for planar undirected graphs. For planar directed graphs, Thorup [29] shows how to construct $(1+\varepsilon)$-distance oracle of

$O(\varepsilon^{-1} n \log n \log(nM))$ space (where the edge weights are drawn from $\{1, \ldots, M\}$). Recently, Kawarabayashi, Klein and Sommer in [21], show how to construct a $(1+\varepsilon)$-distance oracle for undirected planar graphs with linear size and $O(\varepsilon^{-2} \log^2 n)$ query time.

Very recently, Baswana, Lath and Mehta [5] construct an exact distance oracle for directed planar graphs in the presence of one failure. Their scheme is capable of handling only a single edge/vertex failure and answers distance queries where the source is fixed. More precisely, they show how to construct a data structure for a given source s of size $\tilde{O}(n)$ with query time $O(\log n)$ that is capable of answering queries of the form: given two vertices u and v, report the distance from s to v in $G \setminus \{u\}$. Baswana et al. [5] also consider one failure exact distance oracle of size $\tilde{O}(n\sqrt{n})$, the preprocessing time is $\tilde{O}(n\sqrt{n})$ and the query time is $\tilde{O}(\sqrt{n})$. In comparison, we present a distance oracle with significantly smaller size, faster preprocessing and query time at the price of $(1+\varepsilon)$ stretch. For example, for a single fault and fixed ε, our distance oracle is of size $\tilde{O}(n)$, the preprocessing time is $\tilde{O}(n)$ and the query time is $\tilde{O}(1)$. In addition, we can handle any number of faults.

We note that the idea of transforming a decremental algorithm into a fully dynamic algorithm was first suggested by Henzinger and King in their result on dynamic minimum spanning forest [20].

Techniques.

From a very high level the staring point of our forbidden-set label approach is to combine the forbidden-set label ideas of [11, 12] with the failure-free ideas of [29, 25]. Doing this for planar graphs seems to raise several non-trivial difficulties, our solution requires new planar decomposition properties and uses several new ideas.

A key enabler of our result is a new planar decomposition theorem that provides more refined properties (see Properties 1,2). Roughly speaking, given a spanning tree T, it shows one can recursively decompose a planar graph into two 'small' regions using two paths from T. A crucial property of this decomposition is that in each region there are at most $O(\log n)$ special vertices that we call *apices*. Moreover we show that all non-apex vertices are 'well behaved' in the sense that they can store routing information for themselves (apices can belong to too many regions and require special care).

Our forbidden-set label scheme uses this property in a non-trivial manner to store sufficient information to detour around failed vertices and edges. This is done by having the failed vertex label (denoted FL) store a set of $O(\log^2 n)$ vertex-region pairs (w, R) and their labels $L(w, R)$. Roughly speaking, in regions where there are faults, the FL labels of the faults provide the L label of other failure-free vertex-region pairs that allow finding an alternative path between s and t that is $(1+\varepsilon)$ close to the shortest path in the graph with the failures.

An additional complication is when the fault is an apex in a region. In this case the label of the fault cannot store the required information (because a vertex can be an apex in many regions). Our approach is to prepare for this type of fault in advance by always computing the distance in a region R as if all apices are faulty. This means we need special care to deal with the case that some of the apices are not faulty: Since each region has $O(\log n)$ apices, we store labels for when the apices are not-fault and each vetrex that stores information about a region R also stores these labels for all the apices or R.

The main proof is constructed by three nested induction statements. The outer induction statement (see Lemma 7) is an induction on the length of the path and is quite standard. The two remaining nested induction statements (see Lemma 6) are rather subtle statements on certain tuples of four vertexes and two paths which we call *legitimate tuples*. These tuples (see Definition 7) play a crucial role in the proof. The induction statement on legitimate tuples is an induction on the regions induced by the tree of separators and in each region an inner induction is on the length of the legitimate pairs.

Our construction of a fully dynamic $(1+\varepsilon)$ distance oracle from a $(1+\varepsilon)$ forbidden-set labeling scheme with worst case cost is quite general. Roughly speaking we keep two forbidden-set distance oracles in each interval of $\tilde{O}(\sqrt{n})$ rounds: the first copy is used for answering queries, the second copy is gradually constructed each round for the first half of the interval, and then gradually updated two updates each round for the second half of the interval. At the end of the stage, the second copy is ready to become the new first copy and the new second copy is stared from scratch.

This reduction already provides additional new and non-trivial fully dynamic distance oracles for graphs of bounded tree-width (and clique-width) and unweighted graphs with bounded doubling dimension (based on known exact and $(1+\varepsilon)$ forbidden-set labeling schemes for these families). Additionally, this approach may potentially lead to new fully dynamic distance oracles for other families of graphs (or even general graphs). We also note that our techniques and the results concerning planar graphs (Theorems 1–5) can be extended to graphs of bounded Euler-genus, i.e., the graphs that can be embedded on surface (orientable or not) of given genus.

The rest of this paper is organized as follows. In Section 2 we describe the planar decomposition theorem used in our construction, and introduce some preliminaries. In Section 3 we present as a warm-up a simple distance labeling scheme for the failure-free case. Section 4 describes the construction of the forbidden-set labels. Section 5 describes the query algorithm. Finally, in Section 6, we outline the stretch analysis. Our fully dynamic approximate distance oracle is presented in Section 11. In addition, for simplicity, we first present a construction for unweighted graphs, that handles only vertex deletions and uses slightly larger labels length. We later show how to extend these results for weighted graphs, for other updates operations (edge deletions, node insertions, edge insertions and weight changes) and the modifications needed to slightly reduce the labels length.

2. TREE OF SEPARATORS

Consider a planar graph G. Denote by $V(G)$ and $E(G)$ respectively the sets of vertices and edges of G. Let T be a shortest path tree rooted at some vertex root(T), and let G' be the planar graph obtained by triangulating a plane embedding of G. The root path of some vertex v of G, denoted by T_v, is the path between root(T) and v in the tree T. Two nodes of a tree are *relatives* if one is the ancestor of the other one.

We will make the use of the well-known two-path separator for planar graphs [26][Lemma 2, p. 179].

LEMMA 1. *Given a subgraph C of G, one can find in linear time an edge $(u,v) \in E(G') \setminus E(T)$ such that $C \setminus (T_u \cup T_v)$ is divided into two subgraphs A, B of at most $2|C|/3$ vertices such that no edge of G links a vertex of A to a vertex of B. Moreover, A and B lie on different faces of the plane graph $T_u \cup T_v \cup \{(u,v)\}$.*

We recursively apply Lemma 1 and we define a separator hierarchy tree $\mathcal{T}$ as follows. The tree $\mathcal{T}$ is binary, each node μ of $\mathcal{T}$ corresponds to an edge (u,v) of G' induced by Lemma 1 applied on some subgraph C of G. The root of $\mathcal{T}$ is the edge (u,v) of Lemma 1 in the case $C = G$. The two children of (u,v) are then the two edges corresponding to the two path separators when applying Lemma 1 to the subgraphs A and B. The decomposition stops whenever we find (u,v) for C such that $C \subseteq T_u \cup T_v$, that is there is no more subgraphs A and B.

Such a tree $\mathcal{T}$ has depth $O(\log n)$ and can be constructed in $O(n \log n)$ time.

Let us introduce some definitions. Consider a node $\mu = (u,v)$ of $\mathcal{T}$. The *cluster* of a node μ (denoted by $\textsc{Cluster}(\mu)$) is the subgraph C of G on which we invoked Lemma 1 to produce (u,v). The *level* of μ (denoted by $\textsc{Level}(\mu)$) is the number of edges in $\mathcal{T}$ from μ to the root of $\mathcal{T}$.

The *cycle-separator* of node μ is a subgraph of G' defined recursively as follows. If μ is the root of $\mathcal{T}$, then $\textsc{Cycle-Sep}(\mu) = T_u \cup T_v \cup \{(u,v)\}$, otherwise $\textsc{Cycle-Sep}(\mu) = \textsc{Cycle-Sep}(\mu') \cup T_u \cup T_v \cup \{(u,v)\}$, where μ' is the parent of μ. The *separator* of μ (denoted by $\textsc{Sep}(\mu)$) is simply the subgraph of Cycle-Sep induced by the edges of T, i.e., $\textsc{Sep}(\mu) = \textsc{Cycle-Sep}(\mu) \cap E(T)$. Note that $\textsc{Sep}(\mu)$ is a subtree of T containing root(T).

Consider a vertex x of G. The node μ of $\mathcal{T}$ of smallest level such that x belongs to $\textsc{Sep}(\mu)$ is called the *home* of x, and is denoted by $\textsc{Home}(x)$. This definition is justified by the fact that the set of nodes of $\mathcal{T}$ containing in its separator any given vertex of G forms a non-empty subtree of $\mathcal{T}$.

Note that if μ' is the parent of μ then vertices of $\textsc{Cycle-Sep}(\mu')$ separate $\textsc{Cluster}(\mu)$ from the rest of the graph. We would like to have all vertices of $\textsc{Cycle-Sep}(\mu')$ store information about $\textsc{Cluster}(\mu)$. Unfortunately, this would cause some vertices to store too much information. Our solution is to define a subset of $\textsc{Cycle-Sep}(\mu')$ which we call $\textsc{Frame}(\mu)$ and a set of special vertices in $\textsc{Frame}(\mu)$ that we call $\textsc{RelApices}(\mu)$. Roughly speaking we will show that (1) vertices in $\textsc{Frame}(\mu)$ separate $\textsc{Cluster}(\mu)$ from the rest of the graph; (2) the size of $\textsc{RelApices}(\mu)$ is $\tilde{O}(1)$; (3) If vertices in $\textsc{Frame}(\mu) \setminus \textsc{RelApices}(\mu)$ store information about $\textsc{Cluster}(\mu)$ the total information each vertex stores is $\tilde{O}(1)$.

The *apices* of μ (denoted by $\textsc{Apices}(\mu)$) are the vertices of $\textsc{Cycle-Sep}(\mu)$ with degree $\geqslant 3$ in $\textsc{Cycle-Sep}(\mu)$.

Observe the following useful property:

PROPERTY 1. *For every node μ of $\mathcal{T}$, $|\textsc{Apices}(\mu)| \leqslant 2 \cdot \textsc{Level}(\mu) + 1$.*

PROOF. Consider the graph S obtained from $\textsc{Cycle-Sep}(\mu)$ by cutting all edges of $\textsc{Cycle-Sep}(\mu)$ not in T into two degree-1 vertices. In other words, we delete each edge (u,v) of $\textsc{Cycle-Sep}(\mu) \cap E(T)$ not in T and we add a pending degree-1 vertex to u and to v. The graph S is a tree rooted at root(T) whose number of leaves is precisely twice the number of edges we have cut. Moreover, the number of nodes of degree $\geqslant 3$ in S is the same as in $\textsc{Cycle-Sep}(\mu)$.

By induction on $\textsc{Level}(\mu)$, the number of leaves of S is $\ell = 2 \cdot (\textsc{Level}(\mu) + 1)$. Note that the number of nodes of degree $\geqslant 3$ in any tree of ℓ leaves is $\leqslant \ell - 1$. It follows that S contains at most $2 \cdot \textsc{Level}(\mu) + 1$ vertices of degree $\geqslant 3$, and thus $|\textsc{Apices}(\mu)| \leqslant 2 \cdot \textsc{Level}(\mu) + 1$. □

The *tails* of μ, denoted by $\textsc{Tails}(\mu)$, is the subgraph induced by the all the edges of $\textsc{Sep}(\mu)$ not in $\textsc{Sep}(\mu')$, if μ has a parent μ', or simply induced by the edges of $\textsc{Sep}(\mu)$ if μ is the root. In other words, $\textsc{Sep}(\mu) = \textsc{Tails}(\mu) \cup \textsc{Sep}(\mu')$. Each one of the sub-paths $\textsc{Tails}(\mu) \cap T_u$ and $\textsc{Tails}(\mu) \cap T_v$ is called *tail*. Let $\mathcal{P}$ be the set of all tails appearing in the nodes of $\mathcal{T}$.

The next crucial definition emphasizes that only some parts of $\textsc{Sep}(\mu')$, called the *frame* of μ, is needed for separating $\textsc{Cluster}(\mu)$ from the rest of the graph. Informally, one can see tree $\textsc{Sep}(\mu')$ as the union of internally disjoint paths (called *segments*) meeting only in apices. Then, we keep in $\textsc{Frame}(\mu)$ only segments touching $\textsc{Cluster}(\mu)$, i.e., having at least a neighbor in $\textsc{Cluster}(\mu)$. Segments that are in $\textsc{Tails}(\mu)$ are excluded from $\textsc{Frame}(\mu)$ and considered as candidate for μ's child frames.

More precisely, we define it as follows. If μ is the root of $\mathcal{T}$, $\textsc{Frame}(\mu)$ is the empty graph. Otherwise, let μ' be the parent of μ. Then, $\textsc{Frame}(\mu)$ is the subgraph of $\textsc{Sep}(\mu')$ induced by all the vertices x of $\textsc{Sep}(\mu')$ for which there is a vertex y with a neighbor in $\textsc{Cluster}(\mu)$ such that there is a path from x to y in $\textsc{Sep}(\mu') \setminus (\textsc{Apices}(\mu') \setminus \{x\})$. Note that $\textsc{Frame}(\mu)$ is in general a forest, and may contain some component reduced to a single vertex (e.g., an apex x having a neighbor in the cluster). By construction, a path linking a vertex of $\textsc{Cluster}(\mu)$ to a vertex out the cluster has to intersect $\textsc{Frame}(\mu)$.

The *region* of μ (denoted by $\textsc{Reg}(\mu)$) is the subgraph of G induced by all the vertices of $\textsc{Cluster}(\mu) \cup \textsc{Frame}(\mu)$.

The last crucial observation is about vertices that belong to more than one frame. We say that a vertex $v \in \textsc{Frame}(\mu)$ is *singular* in μ if each node $\tilde{\mu}$ with $v \in \textsc{Frame}(\tilde{\mu})$ is a relative of μ, i.e., either is an ancestor or a descendant of μ. The intuition behind this concept is that singular vertices in the frame of μ can only be adjacent to vertices of a single cluster, namely $\textsc{Cluster}(\mu)$ or some of its descendant (all descendant clusters being included in $\textsc{Cluster}(\mu)$). Whereas non-singular vertices can be adjacent to disjoint clusters.

We will extensively use the following property about non-singular vertices:

PROPERTY 2. *If $v \in \textsc{Frame}(\mu) \setminus \textsc{Apices}(\mu)$ is non-singular in μ, then at most one child of μ can contain v in its region.*

PROOF. As preliminaries of the proof, we need to show the following three observations:

The first observation is that the set of nodes of $\mathcal{T}$ containing a given vertex v in its frame induced a subtree of $\mathcal{T}$, hereafter denoted by $F(v)$. For that, consider three nodes μ_1, μ_2, μ_3 of $\mathcal{T}$ such that μ_1 is the parent of μ_2, and μ_2 is some ancestor of μ_3. We show that if a vertex

$v \in \textsc{Frame}(\mu_1) \setminus \textsc{Frame}(\mu_2)$, then $v \notin \textsc{Frame}(\mu_3)$. Because on one side $\textsc{Cluster}(\mu_3) \subset \textsc{Cluster}(\mu_2)$ and on the other side $v \notin \textsc{Frame}(\mu_2)$ implies that v is not part of a segment touching $\textsc{Cluster}(\mu_2)$. So, v cannot be part of a segment touching $\textsc{Cluster}(\mu_3)$ and is not in $\textsc{Frame}(\mu_3)$.

The second observation is that for every non-root node μ, the plane graph $\textsc{Cycle-Sep}(\mu')$ has a face where the subgraph $\textsc{Cluster}(\mu)$ lies and such that its face-walk, hereafter denoted by $W(\mu)$, includes $\textsc{Frame}(\mu)$. Recall that the face-walk of a face f in a plane graph is the subgraph composed of all vertices and edges lying on the border of f. This can be shown as follows by induction on the level of the parent μ' of μ. Let $C = \textsc{Cluster}(\mu')$, and let $A = \textsc{Cluster}(\mu)$.

Let us first assume that $\mu' = (u', v')$ is the root of $\mathcal{T}$. Then, by Lemma 1 applied on C, subgraph A lies on one of the two faces of $T_{u'} \cup T_{v'} \cup \{(u', v')\} = \textsc{Sep}(\mu')$, say f_A. The face-walk of f_A, $W(\mu_A)$, clearly contains $\textsc{Frame}(\mu)$.

Let us assume now that μ' is not the root of $\mathcal{T}$, and denote by μ'' the parent of μ'. By induction hypothesis, C lies on some face of $\textsc{Cycle-Sep}(\mu'')$, say f_C. We remarque that $\textsc{Tails}(\mu') \cup \{(u', v')\}$ must be a part of C, and thus lies on f_C. Otherwise either A would be empty, or one of the root path $T_{u'}$ or $T_{v'}$ would intersect the tree $\textsc{Sep}(\mu')$, creating by this way a cycle in T. In both cases, this is a contradiction. So, $\textsc{Tails}(\mu') \cup \{(u', v')\}$ lies on f_C. It follows that adding $\textsc{Tails}(\mu') \cup \{(u', v')\}$ to $\textsc{Cycle-Sep}(\mu'')$ splits face f_C into two faces, say f_A and f_B, containing respectively the clusters A and B. Since $\textsc{Cycle-Sep}(\mu') = \textsc{Cycle-Sep}(\mu'') \cup \textsc{Tails}(\mu') \cup \{(u', v')\}$, we have proved that f_A and f_B are faces of $\textsc{Cycle-Sep}(\mu')$. By the Jordan's Curve, only the segments included in the face-walk of f_A can touch A. It follows that $\textsc{Frame}(\mu)$ is included in the face-walk $W(\mu_A)$ of f_A, concluding the proof of the second observation.

The last observation is that if v is non-singular in μ, then the subgraph $\textsc{Cycle-Sep}(\mu')$ contains a face-walk $W \neq W(\mu)$ that contains v, where μ' is the parent of μ. Note that the nodes of tree $F(v)$ are singular or non-singular, the root of $F(v)$ being necessarily singular. Define $\tilde{\mu}$ as the node of $F(v)$ with the largest level such that v is singular, and let $\tilde{\mu}_A$ and $\tilde{\mu}_B$ be the children of $\tilde{\mu}$. All descendants of $\tilde{\mu}_A$ and $\tilde{\mu}_B$ must be non-singular. W.l.o.g. assume that μ is a descendant of $\tilde{\mu}_A$. By the second observation, there are two faces f_A and f_B of $\textsc{Cycle-Sep}(\tilde{\mu})$ whose face-walks $W(\tilde{\mu}_A)$ and $W(\tilde{\mu}_B)$ both contains v. In $\textsc{Cycle-Sep}(\mu')$ face f_B exists, whereas $\textsc{Cluster}(\mu)$ lies on face f_A. The face-walk $W(\tilde{\mu}_B)$ of face f_B is precisely the face-walk W we are looking for.

We are now ready to prove Property 2. Let μ_A, μ_B be the two children of μ, and let $C = \textsc{Cluster}(\mu)$, $A = \textsc{Cluster}(\mu_A)$, and let $B = \textsc{Cluster}(\mu_B)$. We have $v \in \textsc{Frame}(\mu)$, so it implies that $v \notin \textsc{Cluster}(\mu)$ and $v \notin \textsc{Cluster}(\mu_A)$ since $\textsc{Cluster}(\mu_A) \subset \textsc{Cluster}(\mu)$. Moreover, if for a child μ_A we have $v \notin \textsc{Frame}(\mu_A)$, then $v \notin \textsc{Reg}(\mu_A)$ and we are done because the vertices of $\textsc{Reg}(\mu_A)$ are, by definition, either in the cluster or in the frame of μ_A.

So let us assume that $v \in \textsc{Frame}(\mu_A) \cap \textsc{Frame}(\mu_B)$. In particular, $v \in W(\mu_A) \cap W(\mu_B)$, the face-walks in $\textsc{Cycle-Sep}(\mu)$ containing the frames of μ_A and μ_B. By the third observation applied to μ_A, there is a face-walk $W \neq W(\mu_A)$ in $\textsc{Cycle-Sep}(\mu)$ containing v. Similarly, there is a face-walk $W' \neq W(\mu_B)$ in $\textsc{Cycle-Sep}(\mu)$ containing v. We have possibly $W = W'$, however $W(\mu_A) \neq W(\mu_B)$. It follows that v belongs to at least three different face-walks of $\textsc{Cycle-Sep}(\mu)$, or equivalently, belongs to the border of at least three different faces of $\textsc{Cycle-Sep}(\mu)$. This contradict the fact $v \notin \textsc{Apices}(\mu)$ and must be of degree $\leqslant 2$ in $\textsc{Cycle-Sep}(\mu)$. Therefore, we have prove that, for at least one of the two children of μ, say μ_A, $v \notin W(\mu_A)$ and thus $v \notin \textsc{Frame}(\mu_A)$ as required. □

Let $\textsc{NewApices}(\mu) = \textsc{Apices}(\mu) \setminus \textsc{Apices}(\mu')$ be the new apices of node μ w.r.t. its parent μ'. The *relevant apices* of μ (denoted by $\textsc{RelApices}(\mu)$) is the set of vertices in $(\textsc{Apices}(\mu) \cap \textsc{Frame}(\mu)) \setminus \textsc{NewApices}(\mu)$. Note that by Property 1, $|\textsc{NewApices}(\mu)| \leqslant 2$.

An important corollary of Property 2 is that as long as a vertex v does not become a relevant apex then the set of regions it belongs to forms two paths in $\mathcal{T}$. More precisely:

Property 3. *For every vertex v, the set of nodes μ of $\mathcal{T}$ such that $v \in \textsc{Reg}(\mu) \setminus \textsc{RelApices}(\mu)$ induces at most two root paths in $\mathcal{T}$.*

Proof. Consider the subgraph $F(v)$ (resp. $C(v)$) induced by all the nodes of $\mathcal{T}$ such that $v \in \textsc{Frame}(\mu)$ (resp. $v \in \textsc{Cluster}(\mu)$). Recall that the vertices of each region $\textsc{Reg}(\mu)$ belongs either to $\textsc{Cluster}(\mu)$ or to $\textsc{Frame}(\mu)$. So, the subset X of nodes of $\mathcal{T}$ we are looking for verifies $X \subseteq C(v) \cup F(u)$.

From the construction of $\mathcal{T}$, $C(v)$ is a root path of $\mathcal{T}$ since sibling clusters are disjoint. And, from the first observation of the proof of Property 2, $F(v)$ is a tree. Let r be the root of $F(v)$. Note that once $v \in \textsc{Frame}(\mu)$, then no descendents of μ can contain v in its cluster. It follows that path $C(v)$ ends at r.

By definition of singularity, all the nodes of $F(v)$ where v is singular form a path in $F(v)$ that starts from r and that ends to some singular node, say μ. Then, from that node, each proper descendent of μ is non-singular. Let μ_1, μ_2 be the children of μ, if they exist in $F(v)$.

If $v \in \textsc{Apices}(\mu_i)$ for some $i \in \{1, 2\}$, then v is also an apex in all descendents of μ_i in $F(v)$. Thus none of μ_i and its descendents can belong to X. So, $v \notin \textsc{Apices}(\mu_i)$. Now, as long as v is not an apex in a non-singular node μ_i one can apply Property 2 and conclude that μ_i has at most one child in X. This forms at most one path for each child of μ. Overall, vertex v is contained in at most two root paths of $\mathcal{T}$. □

The last usefull property is the following.

Property 4. $\sum_{\mu \in \mathcal{T}} |V(\textsc{Reg}(\mu))| = O(n \log n)$.

Proof. Let $h \leqslant \log_{3/2} n + O(1)$ be the maximum level of a node in $\mathcal{T}$. We use a double counting depending whether a given vertex v is a relevant apex of a node μ or not.

From Property 3, each vertex v contributes to at most $2h + 1$ regions of $\mathcal{T}$ as non relevant apices. So, in total, the number of non relevant apices in the regions sums to at most $(2h + 1) \cdot n = O(n \log n)$.

From Property 1, each region of level i contains at most $2i + 1 \leqslant 2h + 1$ relevant apices. The tree $\mathcal{T}$ contains at most $|E(G')| - (n - 1) < 2n$ nodes, since each node of $\mathcal{T}$ corresponds to a distinct non-tree edge of the triangulated plane graph G' with $3n - 6$ edges. Therefore, the number of relevant apices in all the regions sums to $(2h + 1) \cdot 2n$.

All together, the sum of vertices in all the regions of the nodes of $\mathcal{T}$ is $(2h + 1) \cdot 3n = O(n \log n)$. □

The notions of apices, tails and illustration of Property 2 are presented in Figure 1.

We observe that all definitions we have presented in this section and all technical results we prove further in the paper, but those of Lemma 1 and of Property 2, are purely combinatorial. They are independent on the topology of the surface on which G is embedded. In fact, our results extend to graphs of Euler-genus g by slightly enhanced the tree decomposition $\mathcal{T}$. It can be shown that in such graphs there are at most $2g$ root paths of tree T whose removal leaves the graph planar (see [17, 21]). We create therefore a new root node on the top of $\mathcal{T}$ composed of these $2g$ root paths. For bounded g, the analysis remains essentially the same.

Denote by $\textsc{Ancestors}(\mu)$ be the set of proper ancestor nodes of μ in $\mathcal{T}$, and let $\textsc{Ancestors}[\mu] = \textsc{Ancestors}(\mu) \cup \{\mu\}$.

In the remaining, we will mainly focus on the regions of the nodes of $\mathcal{T}$, a notion which is more convenient to use. We extend all the definitions we saw so far about a node μ of $\mathcal{T}$ to its region $R = \textsc{Reg}(\mu)$. This mainly concerns Tails, RelApices, Ancestors, Level. For instance, $\textsc{RelApices}(R)$ is just a short for $\textsc{RelApices}(\mu)$ such that $R = \textsc{Reg}(\mu)$.

3. FAILURE FREE DISTANCE LABELS

As a warm-up we describe a simple distance labeling scheme for the failure-free case. This scheme will later be modified (quite substantially) to handle failures.

Consider an Euler tour of T.

Definition 1 (Euler Tour Labels). *For each vertex v of G, let $\mathrm{ET}(v) = \langle a, b, \mathbf{dist}(v, \mathrm{root}(T), T)\rangle$ where a and b are respectively the entering and leaving time of v in the Euler tour of T.*

Observe that given $\mathrm{ET}(u), \mathrm{ET}(v)$ it is possible to compute if one of them is an ancestor of the other in the tree T. Since T is a shortest path spanning tree of G, in case one is an ancestor of the other, it is possible to compute their distance in G.

For a subgraph H of G and a subset of vertices X of G, we denote by $H + X$ the subgraph composed of H, the vertices of X, and of all the edges of G linking a vertex of H to a vertex of X. For a set S of nodes in $\mathcal{T}$, let $\textsc{Regions}(S)$ be the set of corresponding regions, namely, $\textsc{Regions}(S) = \bigcup_{\mu \in S} \textsc{Reg}(\mu)$. Finally, let $\mathcal{R} = \textsc{Regions}(V(\mathcal{T}))$ be the set of all the regions of $\mathcal{T}$. For a tail $P \in \mathcal{P}$, let $\textsc{Reg}(P) \in \mathcal{R}$ be the region of minimum level such that $P \subset \textsc{Tails}(R)$.

Recall that a ρ-net in a given graph is a ρ-dominating subset of vertices that are pairwise at distance at least ρ. A ρ-dominating set is a subset S of vertices such that every vertex is at distance at most ρ from S. For all path P, vertex $c \in P$, and reals $\rho_1, \rho_2 \geqslant 0$, denote by $\mathcal{N}(P, c, \rho_1, \rho_2)$ any ρ_1-net of the sub-path composed of all the vertices of P at distance at most ρ_2 from c in P. Note that $|\mathcal{N}(P, c, \rho_1, \rho_2)| = O(\rho_2/(\rho_1 + 1))$.

Definition 2 (Failure-Free Geometric Net-Points). *For all region R and vertex $v \in R$, and for each tail P of R, let $c_v \in P$ be the closest vertex to v in R, i.e., any vertex $c_v \in P$ such that $\mathbf{dist}(v, c_v, R) = \min_{x \in P} \mathbf{dist}(v, x, R)$. Let $FFN(v, R) = \{x \in \mathcal{N}(P, c_v, 2^i\varepsilon/8, 2^i) : P \text{ tail of } R, 0 \leqslant i \leqslant \log(nM)\}$.*

We now define the failure-free label of a vertex v of G.

Definition 3 (Failure-Free Label). *The failure-free label $FFL(v)$ of a vertex v is defined as follows:*

$$FFL(v) = \{E_0(v, R) : R \in \textsc{Regions}(\textsc{Ancestors}[\textsc{Home}(v)])\}$$

where $E_0(v, R)$ is:

$$E_0(v, R) = \{I_0(v, x, R) : x \in FFN(v, R)\}$$

where $I_0(v, x, R) = \langle \mathrm{ET}(v), \mathrm{ET}(x), \mathbf{dist}(v, x, R)\rangle$ is a tuple that stores the ET *label of the endpoints v, x and their distance in R.*

This concludes the construction of the labels. Note that the label $FFL(v)$ of each vertex v stores $E_0(v, R)$ for $O(\log n)$ regions. The size of $E_0(v, R)$ is $O(\varepsilon^{-1} \log(nM))$ as there are $O(\varepsilon^{-1} \log(nM))$ net-points in $FFN(v, R)$. We thus get, that the label size for $FFL(v)$ is $O(\varepsilon^{-1} \log n \log(nM))$.

Let us now turn to the query phase.

In the query phase, we get the labels $FFL(s)$ and $FFL(t)$ of two vertices s and t, whose distance needs to be estimated. The query phase is schematically done as follows:

1. Compute a weighted *sketch* graph $H = H(s, t)$.
2. Compute a shortest path distance from s to t in H and return it.

We compute the graph H as follows. The set of vertices in H is the set of vertices that are stored in $FFL(s)$ and $FFL(t)$. For every $I_0(v, x, R)$ that is stored in $FFL(s)$ or $FFL(t)$, add an edge (v, x) with weight $\mathbf{dist}(v, x, R)$. For every two vertices x_1 and x_2 of H that are relatives in T, we also add an edge between them of weight the distance in G between them, using $\mathrm{ET}(x_1)$ and $\mathrm{ET}(x_2)$.

This concludes the construction of the graph H. The next step is to invoke a shortest path algorithm between s and t in H and return that distance.

We now show that the returned distance is $(1+\varepsilon)$ approximated.

Lemma 2. $\mathbf{dist}(s, t, H) \leqslant (1 + \varepsilon) \cdot \mathbf{dist}(s, t, G)$.

Proof. Let $Q = Q(s, t)$ be a shortest path from s to t in G, and let R be the region with minimal $\textsc{Level}(R)$ such that $\textsc{Tails}(R) \cap Q \neq \varnothing$. Since the tails of all proper ancestors of R do not intersect Q, the entire path Q belongs to R. Let $R_s = \textsc{Reg}(\textsc{Home}(s))$. Note also that $R \in \textsc{Regions}(\textsc{Ancestors}[R_s])$, since R_s itself satisfies $\textsc{Tails}(R_s) \cap Q \neq \varnothing$. Therefore, $E_0(s, R)$ appears in the label of s. Similarly, $E_0(t, R)$ appears in the label of t.

Let x be a vertex in $\textsc{Tails}(R) \cap Q$, and let P be a tail of R containing x. Let $d_1 = \mathbf{dist}(s, x, Q)$ and $d_2 = \mathbf{dist}(x, t, Q)$, so $d_1 + d_2 = \mathbf{dist}(s, t, Q)$. Let i be the integer such that $2^i \leqslant d_1 < 2^{i+1}$. Let c_s be the closest vertex to s on P. The distance in R from s to c_s is at most d_1, so the distance on P from c_s to x is at most $2d_1 < 2^{i+2}$. By construction of the label of s, there is in $FFL(s)$ a net-point $x_1 \in P$ at distance at most $2^{i+2} \cdot \varepsilon/8 \leqslant d_1 \cdot \varepsilon/2$ from x. So, the edge (s, x_1) is in H whose length is $\mathbf{dist}(s, x_1, R) \leqslant d_1 + d_1 \cdot \varepsilon/2$. Similarly, in the label $FFL(t)$ there is a net-point $x_2 \in P$ close to x, so that the edge (t, x_2) is in H with length $\mathbf{dist}(t, x_2, R) \leqslant d_2 + d_2 \cdot \varepsilon/2$. And, the distance between x_1 and x_2 on P

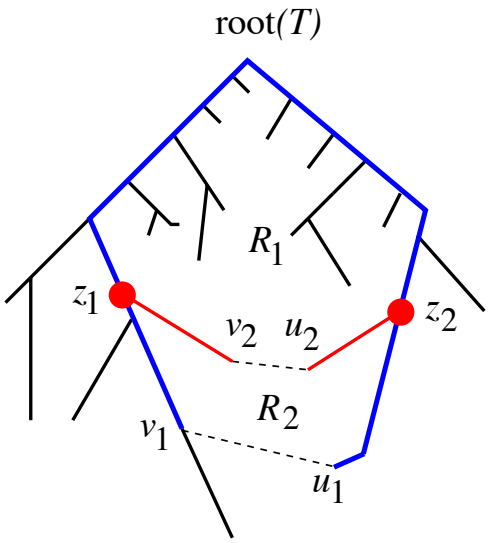

Figure 1: Illustration of the notion of apices. Assume (T_{v_1}, T_{u_1}) is the separator of the root of $\mathcal{T}$. The separator (T_{v_1}, T_{u_1}) separates the plane into two parts, the interior and exterior. Let R be the region of the interior part. Assume (T_{v_2}, T_{u_2}) is the separator of region R. The vertices z_1 and z_2 are the apices of R, and the red paths the tails of R. The frame of R is the subgraph colored blue. Note that all vertices in the first separator (T_{v_1}, T_{u_1}) but the apices belong to only one of R_1 and R_2.

is $\mathbf{dist}(x_1, x_2, P) \leqslant (d_1 + d_2) \cdot \varepsilon/2$. Note that (x_1, x_2) are relatives in T, so the edge (x_1, x_2) is in H.

Therefore, the graph H contains the edges (s, x_1), (x_2, t) whose lengths are with respect to R, and the edge (x_1, x_2) whose length is with respect to P, a shortest path in G. We get that $\mathbf{dist}(s, t, H) \leqslant \mathbf{dist}(s, x_1, R) + \mathbf{dist}(x_1, x_2, P) + \mathbf{dist}(x_2, t, R) \leqslant [d_1 + d_1 \cdot \varepsilon/2] + [(d_1 + d_2) \cdot \varepsilon/2] + [d_2 + d_2 \cdot \varepsilon/2] = (1 + \varepsilon) \cdot (d_1 + d_2) = (1 + \varepsilon) \cdot \mathbf{dist}(s, t, Q)$, as required. □

4. FORBIDDEN-SET DISTANCE LABELS

In this section we describe the labels of the vertices.

Observe that given $\mathrm{ET}(u), \mathrm{ET}(v)$ and $\mathrm{ET}(f)$ it is possible to compute if the faulty vertex f is on the path in T between u and v, in the case u, v are relatives in T.

In the failure-free case each vertex stops storing information once it becomes part of one of the separator paths. For the failure-prone setting, we continue storing information about the vertex even after it becomes part of one of the separator paths. We would like to store for each vertex v information on all the regions it belongs to. However, a vertex may belong to many regions. So the idea is to store information about a vertex as long as it does not become an apex. If a vertex becomes an apex in a region R, then we use the fact that each region has few apices hence any vertex that stores information about region R will also store the fault-labels of all the apices of R and apices in the child regions R_1, R_2 of R.

Definition 4 (Relevant Regions). *For every vertex v, let $\mathrm{RR}(v) = \{R \in \mathcal{R} : v \in R \setminus \textsc{RelApices}(R)\}$ be the set of all regions R where v is not a relevant apex.*

Note that by Property 3, $|\mathrm{RR}(v)| = O(\log n)$.

In the following, we denote by $\textsc{Arn}(R) = \textsc{RelApices}(R) \cup \textsc{NewApices}(R)$ the apices of R that are relevant or new. We now define for every $v \in R$ a set of net-points $N(v, R)$. This set will contain all the relevant and new apices $\textsc{Arn}(R)$ and a sufficiently fine net of points from each tails P of R.

Definition 5 (Geometric Net-Points). *Let R be a region, and let $\tilde{R} = R \setminus \textsc{RelApices}(R)$. For every path P of R, and for every vertex $v \in R$, let $c_v \in P$ be the closest vertex to v in $\tilde{R} + \{v\}$, i.e., any vertex $c_v \in P$ such that $\mathbf{dist}(v, c_v, \tilde{R} + \{v\}) = \min_{x \in P} \mathbf{dist}(v, x, \tilde{R} + \{v\})$.*

Define $N(v, R) = \{x \in N(v, P) : P \text{ tail of } R\} \cup \textsc{Arn}(R)$, where $N(v, P)$ is the set composed of the vertices of $\mathcal{N}(P, c_v, 2^i\varepsilon/384, 2^i)$ for each $0 \leqslant i \leqslant \log(nM)$.

We are now ready to define the forbidden-set distance label of a node v. Intuitively, it is composed of the fault-labels of a node v in all its relevant regions, where the fault-label of v in region R is a set of failure-free labels for all net-points w of tails of R plus the fault-labels of all the apices of R in the child regions R_1, R_2 of R.

Definition 6 (Forbidden-Set Distance Labels). *The fault-label $FL(v)$ of a vertex v is defined as follows:*

$$FL(v) = \{FL(v, R) : R \in \mathrm{RR}(v)\} \cup L(v)$$

where $FL(v, R)$ is defined as:

$$\begin{aligned} FL(v, R) \quad = \quad & \{L(w, R) : w \in N(v, R)\} \\ & \cup \{L(w, R_1), L(w, R_2) : w \in \textsc{Arn}(R)\} \end{aligned}$$

where R_1, R_2 are the child regions of region R (if exists).

The label $L(w, R)$ is defined as:

$$L(w, R) = \{E(w, R)\} \cup L(w)$$

where $L(w)$ is:

$$L(w) = \{E(w, R') : R' \in \mathrm{RR}(w)\}$$

where $E(w,R')$ *is:*

$$E(w,R') = \{I(w,x,R') : x \in N(w,R')\}$$

where $I(w,x,R') = \langle \mathrm{ET}(w), \mathrm{ET}(x), \mathbf{dist}(w,x,\tilde{R}'), \mathbf{1}(w,x)\rangle$, *with* $\tilde{R}' = (R' \setminus \textsc{RelApices}(R')) + \{w,x\}$, *is a tuple that stores the* ET *label of the endpoints* w,x, *their distance in* $\tilde{R}'$, *and a binary indicator* $\mathbf{1}(w,x)$ *that is* 1 *iff* $(w,x) \in E(G)$.

The label $L(w,R)$ is reminiscent of the failure-free labels of the previous section. The main difference is that distances are computed in $\tilde{R}$ (the region R minus the relevant apices). The idea is that a fault f needs to store all the possible detours (up to a $1+\varepsilon$ approximation), this is done by having each fault store a fault-label $FL(v)$. The fault-label contains a set of vertex-region pairs (w,R) and their labels $L(w,R)$. Roughly speaking, in regions where there are faults, the FL labels of the faults provide the L label of other vertex-regions that allows to find an alternative path between s and t that is close to their shortest path in the graph with the failures.

An additional complication is when the fault is an apex in a region. In this case the label of the fault cannot store the required information (because a vertex can be an apex in many regions). Our approach is to prepare of this type of fault in advance by always computing the distance in a region R as if all apices are faulty. This is done by considering only the subgraph $\tilde{R}$ (the region without the relevant apices). Obviously, this means we need to deal with the case that some of the apices may be not faulty - this is done by having $FL(v,R)$ store the L labels in the child regions R_1, R_2 for the new apices of R. This allows f to provide the L labels of the apices that are not faulty if they are needed.

Consider the labels $L(v,R)$ and $L(v,R')$ such that R' is a descendant region of R. If the vertex v is not a relevant apex of R' then observe that $L(v,R') = L(v,R)$.

Lemma 3. *For every vertex* v, *the size of the label* $FL(v)$ *is* $O(\log^2 n \log^2(nM)/\varepsilon^2)$.

Proof. The number of net-points in $N(v,R)$ is $O(\log(nM)/\varepsilon)$, thus the size of $N(v,R)$ is $O(\log(nM)/\varepsilon)$.

The label $L(v,R)$: there are $O(\log n)$ regions in $\mathrm{RR}(v)$, for each such region $O(\log(nM)/\varepsilon)$ data size is stored. Hence, the size of $L(v,R)$ is $O(\log n \log(nM)/\varepsilon)$.

The label $FL(v,R)$: there are $O(\log(nM)/\varepsilon)$ net-points in $N(v,R)$, for each such net-point w the size of $L(w,R)$ is $O(\log n \log(nM)/\varepsilon)$. In addition, there are at most $\log n$ vertices in $\textsc{Arn}(R)$, for each such vertex $O(\log n \log(nM)/\varepsilon)$ data size is stored. Therefore the total size of $FL(v,R)$ is $O(\log n \log^2(nM)/\varepsilon^2)$.

The label $FL(v)$: there are $O(\log n)$ regions in $\mathrm{RR}(v)$, for each such region $FL(v,R)$ is stored, which is of size $O(\log n \log^2(nM)/\varepsilon^2)$. Therefore the total size is $O(\log^2 n \log^2(nM)/\varepsilon^2)$. □

We say that a label L contains a label L', or simply write $L' \in L$, if the label L' is explicitly stored in the label of L.

5. DISTANCE QUERIES

A *distance query* (s,t,F) involves a source s, a target t and a subset F of *faulty* (or *forbidden*) vertices. Its input consists of the labels $\{FL(s), FL(t)\} \cup \{FL(v) : v \in F\}$. Answering a query (s,t,F) based on the given labels, namely, finding an approximate distance from s to t in the graph $G \setminus F$, is done as in the fault-free case, namely, first compute a weighted *sketch* graph $H = H(s,t,F)$ and then compute a shortest path distance from s to t in H and return it.

Next, we show how to construct the weighted graph H. The general idea is that only "safe" edges are added to H, where an edge is safe if the path it represents also occurs in $G \setminus F$. In fact, for our purposes, it suffices to consider only the labels $\{L(s), L(t)\} \cup \{FL(v) : v \in F\}$. Hereafter, when we say that a label or edge-set appear in the query, we mean that it appears in $\{L(s), L(t)\} \cup \{FL(v) : v \in F\}$.

Building the sketch graph H.

We compute the graph H as follows.

1. For every edge-set $E(v,R)$ that appears in the query we do the following: For each tuple $I(v,x,R)$ stored in $E(v,R)$, we add the edge (v,x) to H with weight $\tilde{\omega}(v,x)$ if it is *safe*.
2. In addition, for every two net-points that belong to the same tail $P \in \mathcal{P}$ we also add an edge between them whose weight is the distance between them, if there is no fault between them in P.

   Specifically, for every two net-points $x_1 \in E(w_1,R)$ and $x_2 \in E(w_2,R)$ such that x_1, x_2 are relatives in T, we add an edge between x_1 and x_2 (whose weight is the distance between them) if there is no fault on the path in T between x_1 and x_2. Checking if the edge (x_1,x_2) should be added is simply done by the Euler tour labels $\mathrm{ET}(x_1), \mathrm{ET}(x_2)$ and $\{\mathrm{ET}(f) : f \in F\}$.

We now explain how to determine if the edge (v,x) is *safe*.

Let $\tilde{R} = R \setminus \textsc{RelApices}(R)$ and let Q be a tail of $\textsc{Tails}(R)$ such that $x \in Q$.

For the safety check we need the following observation. By construction a faulty vertex f satisfies $f \in R$ and f is not a relevant apex of R if and only if $E(f,R) \in L(f)$. Thus, having the labels of the faulty vertices, we can easily find all faulty vertices f such that $f \in R$ and f is not a relevant apex of R.

Safety of an edge $(v,x) \in E(v,R)$.

1. If either v or x are faulty then set $\mathrm{safe}(v,x) := \textsc{False}$;
2. Otherwise, if $\mathbf{1}(v,x) = 1$, namely $(v,x) \in E(G)$, then set $\mathrm{safe}(v,x) := \textsc{True}$;
3. Otherwise, for every faulty vertex f such that $f \in R$ and f is not a relevant apex of R do the following: Use $E(f,R)$ to approximate $\mathbf{dist}(f,x,\tilde{R}+\{x\})$ (by considering all net-points x_1 in $N(f,Q)$ and taking the edge that gives the minimal $\tilde{\omega}(f,x_1) + \mathbf{dist}(x_1,x,Q)$). Let $\tilde{\mathbf{dist}}(f,x,\tilde{R}+\{x\})$ be the approximation. Check if $\tilde{\mathbf{dist}}(f,x,\tilde{R}+\{x\}) \leqslant (1+\varepsilon/48)\cdot\tilde{\omega}(v,x)$, if the inequality holds, then set $\mathrm{safe}(v,x) := \textsc{False}$. Otherwise if the inequality does not hold for all $f \in R$ that are not a relevant apex of R then set $\mathrm{safe}(v,x) := \textsc{True}$.

We next show that the sketch graph H distance preserver of $G \setminus F$, namely, the distances in H are greater equal to the distances in $G \setminus F$.

Lemma 4. *For every pair of nodes* x_1, x_2, $\mathbf{dist}(x_1,x_2,H) \geqslant \mathbf{dist}(x_1,x_2,G \setminus F)$.

PROOF. Note that is enough to show the claim for only edges in H. Consider an edge (v,x) that was added to H. It is not hard to verify that edges from the second type that are added to H are indeed safe, that is do not contain a faulty vertex on the path they represent. We now show if safe(v,x) is TRUE for $(v,x) \in E(v,R)$, then there is no fault on the path (v,x) represents.

Recall that the edge (v,x) represents a path in the subgraph $\tilde{R}+\{v,x\}$ and thus does not contain any relevant apex of R, except maybe v or x itself, but we already checked that v and x are not faulty. Hence, we only need to check the safety of the edge (v,x) against faulty vertices in R that are not relevant apices of R. Consider a faulty vertex $f \in \tilde{R}$. Using $E(f,R) \in L(f) \in FL(f)$, we can find $\tilde{\mathbf{dist}}(f,x,\tilde{R}+\{x\})$ such that $\mathbf{dist}(f,x,\tilde{R}+\{x\}) \leqslant \tilde{\mathbf{dist}}(f,x,\tilde{R}+\{x\}) \leqslant (1+\varepsilon/48)\cdot\mathbf{dist}(f,x,\tilde{R}+\{x\})$. This is by straightforward calculations and due to the fact that the net-points $N(f,Q)$ contains a net-point on Q at distance at most $\varepsilon/96\cdot\mathbf{dist}(f,x,\tilde{R}+\{x\})$ from x (again by straight forward calculations). Note that if safe(v,x) is TRUE then $\tilde{\mathbf{dist}}(f,x,\tilde{R}+\{x\}) > (1+\varepsilon/48)\cdot\tilde{\omega}(v,x)$. Thus, $\mathbf{dist}(f,x,\tilde{R}+\{x\}) > \tilde{\omega}(v,x)$. It is not hard to see that f cannot be on the path (v,x) represents. Note also that if safe(v,x) is FALSE then $\mathbf{dist}(f,x,\tilde{R}+\{x\}) \leqslant (1+\varepsilon/48)\cdot\tilde{\omega}(v,x)$. □

We get that only safe edges are added to H, it is left to show that indeed H contains a short path between s and t.

6. ANALYSIS

In this section, we prove the existence of a path from s to t in H of length at most $(1+\varepsilon)\cdot\mathbf{dist}(s,t,G\setminus F)$ by Lemma 6 (the more involved part of the analysis) and by Lemma 7. The very high level of the analysis is as follows. Lemma 7 is derived from Lemma 6 almost trivially. Loosely speaking, Lemma 6 shows the following (with some more technical details to handle apices). Consider a shortest path $Q(y_1,y_2,G\setminus F)$ between two vertices y_1 and y_2 in $G\setminus F$ and assume the query contains the labels $L(x_1)$ and $L(x_2)$ of two net-points x_1 and x_2 that are "close" (with respect to the distance from y_1 to y_2) to y_1 and y_2 respectively and consider the tail P with minimal LEVEL(P) such that P contains some vertex r in the path from x_1 to x_2. Then using similar arguments as in the failure-free case, there exists two net-points r_1 and r_2 on P such that the path obtained by the concatenation of the paths from x_1 to r_1, from r_1 to r_2 and from r_2 to x_2 is of length roughly the distance from y_1 to y_2 in $G\setminus F$. Therefore if these paths are safe, then we have a path from x_1 to x_2 in the graph H that is "close" to the path from y_1 to y_2 in $G\setminus F$. In case one of these paths is not safe then using the labels of the faulty vertices it is possible to find two other net-points z_1 and z_2 that are "close" to the path from y_1 to y_2 in $G\setminus F$, z_1 is close to some vertex w_1 on that path and z_2 is close to some other vertex w_2 on that path and the graph H contains a path from z_1 to z_2 of length close to $\mathbf{dist}(w_1,w_2,G\setminus F)$.

Now, Lemma 7 is derived easily from Lemma 6. Consider the path from s to t in $G\setminus F$, by Lemma 6 there exist four vertices z_1,z_2,w_1,w_2 such that w_1 and w_2 are on the path $Q(s,t,G\setminus F)$ and z_1 and z_2 are "close" to w_1 and w_2. Moreover, there exists a path in H of length roughly $\mathbf{dist}(w_1,w_2,G\setminus F)$. Now using induction on the length of a path, we get that there is also a path from s to z_1 of length roughly $\mathbf{dist}(s,z_1,G\setminus F)$ which is also roughly $\mathbf{dist}(s,w_1,G\setminus F)$. Similarly, there exists a path from z_1 to t of length roughly $\mathbf{dist}(z_2,t,G\setminus F)$ which is also roughly $\mathbf{dist}(w_2,t,G\setminus F)$. Concatenating these three paths we get a path from s to t of length roughly $\mathbf{dist}(s,t,G\setminus F)$.

We now introduce some definitions that would be helpful for the following lemmas.

DEFINITION 7 (LEGITIMATE TUPLE). *Consider some region R, we say that the tuple $(y_1,y_2,x_1,x_2,Q_1,Q_2)$ is legitimate for the region R if the following properties are satisfied.*

(1) For $i=1$ and $i=2$, we have either the vertices x_i and y_i belong to some tail $Q \subset$ TAILS(ANCESTORS$[R]$) or ($x_i=y_i$ and $Q=\varnothing$). In addition, if $x_i \neq y_i$ then all vertices on the subpath $Q(x_i,y_i)\setminus\{x_i\}$ are not relevant apices of R.

(2) $Q(y_1,y_2,G\setminus F)\cup\{x_1,x_2\} \subseteq R$ and $(Q(y_1,y_2,G\setminus F)\setminus\{y_1,y_2\}) \cap$ TAILS(ANCESTORS(R)) $=\varnothing$.

(3) The labels $L(x_1,R)$ and $L(x_2,R)$ appear in the query.

(4) $\mathbf{dist}(x_1,y_1,Q_1\setminus F)+\mathbf{dist}(x_2,y_2,Q_2\setminus F) \leqslant \varepsilon/8\cdot\mathbf{dist}(y_1,y_2,G\setminus F)$.

An illustration of the notion legitimate tuple is presented in Figure 2.

Note that Property (4) implies that the paths from x_1 to y_1 on Q_1 and from x_2 to y_2 on Q_2 are free from faults, since otherwise assume w.l.o.g that the path from x_1 to y_1 on Q_1 contains a fault then $\mathbf{dist}(x_1,y_1,Q_1\setminus F)=\infty$.

The next lemma shows that for every region R and legitimate tuple $(y_1,y_2,x_1,x_2,Q_1,Q_2)$ for R, one of the two cases must hold. Either there is a path from x_1 to x_2 of length "close" to the distance between y_1 and y_2 in $G\setminus F$ or there exists a fault that is "close" to the path $Q(y_1,y_2,G\setminus F)$ and that $FL(f,R')$ appear in the query for every $R' \in$ REGIONS(ANCESTORS$[R]$).

LEMMA 5. *For every region R, a legitimate tuple $(y_1,y_2,x_1,x_2,Q_1,Q_2)$ for R and a vertex r of (TAILS$(R)\cap Q(y_1,y_2,G\setminus F))\setminus\{y_1,y_2\}$, one of the two cases must hold.*

(1) $2\cdot\mathbf{dist}(x_1,y_1,H)+2\cdot\mathbf{dist}(x_2,y_2,H)+\mathbf{dist}(x_1,x_2,H) \leqslant (1+\varepsilon)\cdot\mathbf{dist}(y_1,y_2,G\setminus F)$

(2) There exists a faulty vertex $f \in F$ such that $FL(f,R')$ appear in the query for every $R' \in$ REGIONS(ANCESTORS$[R]$) and:
$\mathbf{dist}(f,r,\tilde{R}) \leqslant 3\cdot\mathbf{dist}(y_1,y_2,G\setminus F)$,
$\mathbf{dist}(f,y_1,\tilde{R}+\{y_1\}) \leqslant 3\cdot\mathbf{dist}(y_1,y_2,G\setminus F)$, and
$\mathbf{dist}(f,y_2,\tilde{R}+\{y_2\}) \leqslant 3\cdot\mathbf{dist}(y_1,y_2,G\setminus F)$, where $\tilde{R}=R\setminus$ RELAPICES(R).

PROOF. Let $S_1 \subset$ TAILS(R) be the tail such that $r \in S_1$.

Observe that $r \in \tilde{R}$. To see this, recall that by definition of legitimate tuple, we have $(Q(y_1,y_2,G\setminus F)\setminus\{y_1,y_2\}) \cap$ TAILS(ANCESTORS(R)) $=\varnothing$. Hence, $r \cap$ TAILS(ANCESTORS(R)) $=\varnothing$, so clearly r is not a relevant apex of R.

Note that for every two vertices $x \in S_1$ and $y \in R$, the set $N(y,R)$ contains a net-point on S_1 at distance at most $\varepsilon/96\cdot\mathbf{dist}(y,x,\tilde{R}+\{y\})$ from x. Specifically, the closest vertex r_1 to r in $N(x_1,R)$ is at distance at most $\varepsilon/96\cdot\mathbf{dist}(x_1,r,\tilde{R}+$

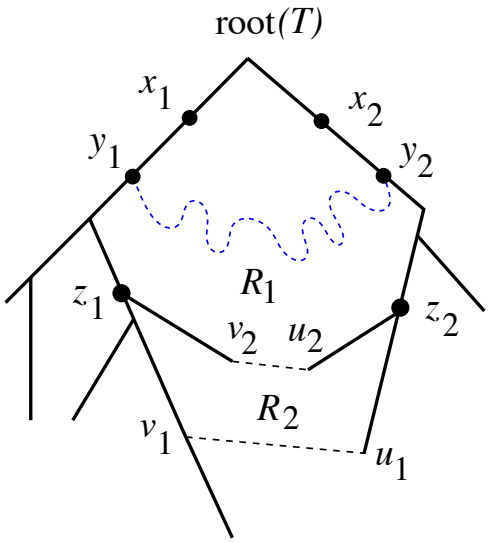

Figure 2: Illustration for legitimate tuple. Assume (T_{v_1}, T_{u_1}) is the separator of the root of $\mathcal{T}$. The separator (T_{v_1}, T_{u_1}) separates the plane into two parts, the interior and exterior. Let R be the region of the interior part. Assume (T_{v_2}, T_{u_2}) is the separator of region R. Let $Q_1 = T_{v_1}$ and $Q_2 = T_{v_2}$ (notice that since (v_1, u_1) is the first separator then the entire separator paths are the tails). Let the blue dashed path be the shortest path from y_1 to y_2 in $G \setminus F$. Notice that the entire dashed path belongs to the cluster of R_1, that is, it does not intersect with ancestors separators of R_1. Assume the label $L(x_1, R_1)$ and $L(x_2, R_1)$ appear in the query. Assume also that x_1 and x_2 are "close" enough to y_1 and y_2 resp. The tuple $(y_1, y_2, x_1, x_2, Q_1, Q_2)$ is legitimate in R_1.

$\{x_1\}$) from r and the closest vertex r_2 to r in $N(x_2, R)$ is at distance at most $\varepsilon/96 \cdot \mathbf{dist}(x_2, r, \tilde{R} + \{x_2\})$ from r. (it could be that $r_1 = r_2$) We now consider two cases. The first case is when all edges (x_1, r_1), (r_1, r_2) and (r_2, x_2) are safe. The second case is when at least one edge among (x_1, r_1), (r_1, r_2) and (r_2, x_2) is not safe.

Consider the first case, namely, all edges (x_1, r_1), (r_1, r_2) and (r_2, x_2) are safe. Consider the path $\tilde{Q}_1 = Q(x_1, y_1, Q_1) \circ Q(y_1, r, G \setminus F) \circ Q(r, r_1, S_1)$ and let $\tilde{Q}_2 = Q(x_2, y_2, Q_1) \circ Q(y_2, r, G \setminus F) \circ Q(r, r_2, S_1)$. Note that $\tilde{Q}_1 \subseteq \tilde{R} + \{x_1\}$ and that $\tilde{Q}_2 \subseteq \tilde{R} + \{x_2\}$. We thus get, $\mathbf{dist}(x_1, r_1, H) \leqslant \tilde{\omega}(x_1, r_1) \leqslant \omega(\tilde{Q}_1)$, $\mathbf{dist}(r_1, r_2, H) \leqslant \mathbf{dist}(r_1, r_2, S_1) = \mathbf{dist}(r_1, r_2, G)$ and $\mathbf{dist}(r_2, x_2, H) \leqslant \tilde{\omega}(x_2, r_2) \leqslant \omega(\tilde{Q}_2)$, where $\omega(Q')$ is the length (the sum of its edge weight) of the path Q'. Using straight forward calculations, we get that $\mathbf{dist}(x_1, x_2, H) + 2 \cdot \mathbf{dist}(x_1, y_1, H) + 2 \cdot \mathbf{dist}(x_2, y_2, H) \leqslant (1 + \varepsilon) \cdot \mathbf{dist}(y_1, y_2, G \setminus F)$ and the lemma follows.

Consider the second case, namely, one of the edges (x_1, r_1), (r_1, r_2) and (r_2, x_2) is not safe, we next show that there is a fault f that is not a relevant apex of R and that is "close" to these edges. Let $d = \mathbf{dist}(y_1, y_2, G \setminus F)$ and let $d_1 = \mathbf{dist}(y_1, r, G \setminus F)$ and $d_2 = \mathbf{dist}(r, y_2, G \setminus F)$.

We assume that one of the edges (x_1, r_1), (r_1, r_2), (r_2, x_2) is not safe.

We consider two subcases. The first subcase is when the edge (r_1, r_2) is not safe. The second subcase is when at least one of the edges (x_1, r_1) and (r_2, x_2) is not safe. Consider the first subcase, namely, the edge (r_1, r_2) is not safe. Notice that in this case both r_1 and r_2 belong to S_1, namely, one of them is an ancestor of the other in the tree T. Moreover, the edge (r_1, r_2) is not safe only in the case where there is a fault f on the shortest path from r_1 to r_2 in the tree T. In addition, note that by construction f is not a relevant apex of R (recall that the tail S_1 does not intersect with ancestor separators).

Since f is not a relevant apex of R and therefore also not a relevant apex of any $R' \in \textsc{Regions}(\textsc{Ancestors}(R))$, by definition $FL(f, R')$ appear in $FL(f)$ and thus appear in the query for every $R' \in \textsc{Regions}(\textsc{Ancestors}[R])$.

Moreover, $\mathbf{dist}(y_1, f, \tilde{R} + \{y_1\}) \leqslant \mathbf{dist}(y_1, r, \tilde{R} + \{y_1\}) + \mathbf{dist}(r, f, \tilde{R}) \leqslant d_1 + \varepsilon/96 \cdot \max\{\mathbf{dist}(x_1, r, S_1), \mathbf{dist}(x_2, r, S_1)\} \leqslant d_1 + \varepsilon d$. The exact same analysis shows that $\mathbf{dist}(y_2, f, \tilde{R} + \{y_2\}) < d_2 + \varepsilon d$. We also have $\mathbf{dist}(r, f, \tilde{R}) \leqslant (\varepsilon/96) \cdot \max\{\mathbf{dist}(x_1, r, S_1), \mathbf{dist}(x_2, r, S_1)\} < \varepsilon \, .d$ The lemma follows.

We left with the second subcase, where one of the edges (x_1, r_1), (r_2, x_2) is not safe. Assume the edge (x_1, r_1) is not safe, the other case where (r_2, x_2) is not safe is symmetric.

Since the edge (x_1, r_1) is not safe, there exists a faulty vertex f such that f is not a relevant apex of R and in addition $\mathbf{dist}(f, r_1, \tilde{R}) \leqslant (1 + \varepsilon/48) \cdot \mathbf{dist}(x_1, r_1, \tilde{R} + \{x_1\})$.

Since f is not a relevant apex of R and therefore also not a relevant apex of any $R' \in \textsc{Regions}(\textsc{Ancestors}(R))$, by definition $FL(f, R')$ appear in $FL(f)$ and thus appear in the query for every $R' \in \textsc{Regions}(\textsc{Ancestors}[R])$.

Next, we show that f is at distance at most $3d$ from all y_1, y_2 and r.

We have, $\mathbf{dist}(y_1, f, \tilde{R} + \{y_1\}) \leqslant \mathbf{dist}(y_1, r, \tilde{R} + \{y_1\}) + \mathbf{dist}(r, r_1, \tilde{R}) + \mathbf{dist}(r_1, f, \tilde{R}) \leqslant d_1 + \varepsilon/96 \cdot d_1 + (1 + \varepsilon/48) \cdot \mathbf{dist}(x_1, r_1, \tilde{R} + \{x_1\}) \leqslant d_1 + \varepsilon/96 \cdot d_1 + (1 + \varepsilon/48) \cdot (\varepsilon/8 \cdot d_1 + d_1 + \varepsilon/96 \cdot d_1) \leqslant 2d_1 + \varepsilon d_1 \leqslant 3d_1 \leqslant 3d$,

$\mathbf{dist}(y_2, f, \tilde{R} + \{y_2\}) \leqslant \mathbf{dist}(y_2, r, \tilde{R} + \{y_2\}) + \mathbf{dist}(r, r_1, \tilde{R}) + \mathbf{dist}(r_1, f, \tilde{R}) \leqslant d_2 + \varepsilon/96 \cdot d_1 + (1 +$

$\varepsilon/48) \cdot \mathbf{dist}(x_1, r_1, \tilde{R} + \{x_1\}) \leqslant d_2 + \varepsilon/96 d_1 + (1 + \varepsilon/48) \cdot (\varepsilon/8 \cdot d_1 + d_1 + \varepsilon/96 \cdot d_1) \leqslant d_1 + d_2 + \varepsilon d_1 \leqslant d + \varepsilon d_1 \leqslant 3d$.

Finally, $\mathbf{dist}(r, f, \tilde{R}) \leqslant \mathbf{dist}(r, r_1, \tilde{R}) + \mathbf{dist}(r_1, f, \tilde{R}) \leqslant \varepsilon/96 \cdot d_1 + (1 + \varepsilon/48) \cdot \mathbf{dist}(x_1, r_1, \tilde{R} + \{x_1\}) \leqslant \varepsilon/96 \cdot d_1 + (1 + \varepsilon/48) \cdot (\varepsilon/8 \cdot d_1 + d_1 + \varepsilon/96 \cdot d_1) \leqslant d_1 + \varepsilon d_1 \leqslant 3d$.

The lemma follows. □

The following is the main technical lemma, the proof has several cases, one of the main cases requires the use of Lemma 5.

LEMMA 6. *For every region R and legitimate tuple $(y_1, y_2, x_1, x_2, Q_1, Q_2)$ for R the following holds. There exist four vertices z_1, z_2, w_1, w_2 such that w_1 and w_2 are vertices on the sub-path of $Q(y_1, y_2, G \setminus F)$, the labels of $L(z_1)$ and $L(z_2)$ appear in the query and $2 \cdot \mathbf{dist}(z_1, w_1, H) + 2 \cdot \mathbf{dist}(z_2, w_2, H) + \mathbf{dist}(z_1, z_2, H) \leqslant (1+\varepsilon) \cdot \mathbf{dist}(w_1, w_2, G \setminus F)$.*

PROOF. The proof is by induction on the level of R and then by the number of edges on the subpath from y_1 to y_2 in $G \setminus F$. For simplicity, we assume uniqueness on the shortest paths in $G \setminus F$.

First, consider the base case where the number of edges in $Q(y_1, y_2, G \setminus F)$ is 1. Assume w.l.o.g. that $\varepsilon < 1$ we thus have $x_1 = y_1$ and $x_2 = y_2$.

Consider the region R' with minimum $\text{LEVEL}(R')$ having a tail $\tilde{P}$ such that $\{x_1, x_2\} \cap \tilde{P} \neq \varnothing$. Assume w.l.o.g that $x_1 \in \tilde{P}$, the label $L(x_2)$ contains $E(x_2, R')$ that contains the edge (x_1, x_2) since $\varepsilon < 1$, and we are done (note that this is the only place in the proof where we use the fact that the graph is unweighted, later, in Section 7, we show how to deal with the weighted case).

We now assume that the lemma holds for every region R' of level ℓ' for $\ell' > \ell$ and legitimate tuple $(y_1, y_2, x_1, x_2, Q_1, Q_2)$ and also for every region R' of level ℓ and legitimate tuple $(y'_1, y'_2, x'_1, x'_2, Q'_1, Q'_2)$ of R' such that the number of edges on the subpath from y'_1 to y'_2 in $G \setminus F$ is less than k.

Next, we prove the lemma for some region R of level ℓ and a legitimate tuple $(y_1, y_2, x_1, x_2, Q_1, Q_2)$ such that the number of edges on the subpath from y_1 to y_2 in $G \setminus F$ is $k > 1$. Note that it could be that R belongs to a leaf in $\mathcal{T}$.

To ease notation, if $x_1 = y_1$ then we set $Q_1 = \{y_1\}$. Similarly, if $x_2 = y_2$ then we set $Q_2 = \{y_2\}$. Consider the path $Q = Q(x_1, y_1, Q_1) \circ Q(y_1, y_2, G \setminus F) \circ Q(x_2, y_2, Q_2)$. Let $d = \mathbf{dist}(y_1, y_2, G \setminus F)$. Let $\tilde{R} = R \setminus \text{RELAPICES}(R)$.

We consider two cases. Case (1) is when $\text{TAILS}(R) \cap (Q(y_1, y_2, G \setminus F) \setminus \{y_1, y_2\}) = \varnothing$. Case (2) is when $\text{TAILS}(R) \cap (Q(y_1, y_2, G \setminus F) \setminus \{y_1, y_2\}) \neq \varnothing$.

Consider case (1), where $\text{TAILS}(R) \cap (Q(y_1, y_2, G \setminus F) \setminus \{y_1, y_2\}) = \varnothing$.

Observe that in this case R cannot be a leaf in $\mathcal{T}$. To see this, note that all internal vertices in $Q(y_1, y_2, G \setminus F)$ do not appear in the current separator and in all ancestor separators. As all vertices belong to some separator of $\mathcal{T}$, it follows that R cannot be a leaf in $\mathcal{T}$. Let R_1 and R_2 be the two child regions of R. Note that the entire path $Q(y_1, y_2, G \setminus F)$ is contained in one of the child regions of R, assume w.l.o.g. that $Q(y_1, y_2, G \setminus F) \subseteq R_1$. We consider two subcases. Subcase (1.1) is when $\tilde{R}$ does not contain a faulty vertex and subcase (1.2) is when $\tilde{R}$ contains a faulty vertex f. Consider subcase (1.1). There are two subcases in subcase (1.1). Subcase (1.1.1) is when there is a vertex $r \in \text{ARN}(R)$ such that $r \in Q(x_1, y_1, Q_1) \cup Q(x_2, y_2, Q_2)$. (Recall that $\text{ARN}(R) = \text{RELAPICES}(R) \cup \text{NEWAPICES}(R)$.) Subcase (1.1.2) is when there is no vertex $r \in \text{ARN}(R)$ such that $r \in Q(x_1, y_1, Q_1) \cup Q(x_2, y_2, Q_2)$.

Consider subcase (1.1.1). Assume w.l.o.g. that $r \in Q(x_1, y_1, Q_1)$. Note that $N(x_2, R) (\subseteq L(x_2, R))$ contains the net-point r. There are no faulty vertices in $\tilde{R}$, thus the edges $(x_1, r), (x_2, r)$ are safe. We get that, $\mathbf{dist}(x_1, x_2, H) \leqslant \mathbf{dist}(x_1, r, Q_1) + \mathbf{dist}(x_2, r, \tilde{R} + \{x_2\})$. It is not hard to verify that $2 \cdot \mathbf{dist}(x_1, y_1, H) + 2 \cdot \mathbf{dist}(x_2, y_2, H) + \mathbf{dist}(y_1, y_2, H) \leqslant (1+\varepsilon) \cdot \mathbf{dist}(y_1, y_2, G \setminus F)$. The lemma follows where $w_1 = y_1$, $w_2 = y_2$, $z_1 = x_1$ and $z_2 = x_2$.

Consider subcase (1.1.2). There is no vertex in $\text{ARN}(R) \cap (Q(x_1, y_1, Q_1) \cup Q(x_2, y_2, Q_2))$. Specifically, x_1 and x_2 are not in $\text{ARN}(R)$. Therefore, $L(x_1, R) = L(x_1, R_1)$ and $L(x_2, R) = L(x_2, R_1)$ and it is not hard to show that the tuple $(y_1, y_2, x_1, x_2, Q_1, Q_2)$ is legitimate for R_1. The claim follows by the induction hypothesis on R_1 and on $(y_1, y_2, x_1, x_2, Q_1, Q_2)$.

Consider subcase (1.2). If $Q(x_1, y_1, Q_1) \cap \text{ARN}(R) \neq \varnothing$, then set x'_1 to be the closest vertex to y_1 in $\text{ARN}(R) \cap Q(x_1, y_1, Q_1)$. Note that $FL(f, R) \subseteq FL(f)$ contains $L(x'_1, R_1)$. If $Q(x_1, y_1, Q_1) \cap \text{ARN}(R) = \varnothing$, then set x'_1 to be x_1. Note that in this case $L(x_1, R) = L(x_1, R_1)$. Similarly, we find a vertex x'_2 on the path $Q(x_2, y_2, Q_2)$ such that $L(x_2, R_1)$ appears in the query and that $(Q(x'_2, y_2, Q_2) \setminus \{x'_2\}) \cap \text{ARN}(R) = \varnothing$. It is not hard to verify that the tuple $(y_1, y_2, x'_1, x'_2, Q_1, Q_2)$ is legitimate for R_1. The claim follows by the induction hypothesis on R_1 and on $(y_1, y_2, x_1, x_2, Q_1, Q_2)$.

For illustrations of subcase (1.2) see Figures 3 and 4.

Consider case (2), where $\text{TAILS}(R) \cap (Q(y_1, y_2, G \setminus F) \setminus \{y_1, y_2\}) \neq \varnothing$.

Pick r to be an arbitrary vertex in $\text{TAILS}(R) \cap (Q(y_1, y_2, G \setminus F)) \setminus \{y_1, y_2\}$. Let $S_1 \subset \text{TAILS}(R)$ be the tail of R such that $r \in S_1$.

By Lemma 5 one of the following two subcases occurs. Either $\mathbf{dist}(x_1, x_2, H) + 2 \cdot \mathbf{dist}(x_1, y_1, H) + 2 \cdot \mathbf{dist}(x_2, y_2, H) \leqslant (1 + \varepsilon) \cdot \mathbf{dist}(y_1, y_2, G \setminus F)$, we call this subcase (2.1). Or otherwise, there exists a fault $f \in R$ such that $FL(f, R')$ for every $R' \in \text{REGIONS}(\text{ANCESTORS}[R])$ exists in the query and f is at distance at most $3d$ from y_1, y_2 and r in R, we call this subcase (2.2).

In the first subcase (2.1) the lemma immediately follows where $w_1 = y_1$, $w_2 = y_2$, $z_1 = x_1$ and $z_2 = x_2$.

Consider subcase (2.2). Using the labels $FL(f, R')$ for $R' \in \text{REGIONS}(\text{ANCESTORS}[R])$ we can find the labels $L(x'_1, R)$ and $L(x'_2, R)$ of two possibly other net-points x'_1 and x'_2 and conclude the claim using the induction hypothesis.

Let $d_1 = \mathbf{dist}(y_1, r, G \setminus F)$ and $d_2 = \mathbf{dist}(r, y_2, G \setminus F)$ and $d = d_1 + d_2$. Assume w.l.o.g. that $d_1 \geqslant d_2$. The faulty vertex f is at distance at most $3d \leqslant 6d_1$ from both r and y_1 and $FL(f, R')$ for every $R' \in \text{REGIONS}(\text{ANCESTORS}[R])$ exists in the query. Thus, $FL(f, R)$ appears in the query. The set $N(f, R)$ contains a vertex $x'_2 \in R$ on S_1 such that $\mathbf{dist}(r, x'_2, S_1) \leqslant \varepsilon/96 \cdot \mathbf{dist}(f, r, \tilde{R}) \leqslant \varepsilon/96 \cdot 6d_1 = \varepsilon/16 \cdot d_1$. By construction, we get that the label $L(x'_2, R) \in FL(f, R)$ and thus appears in the query. If $x_1 = y_1$ then set $x'_1 = x_1$. Otherwise $FL(f, \text{REG}(Q_1))$ also appears in the query as $\text{REG}(Q_1) \in \text{REGIONS}(\text{ANCESTORS}[R])$. The set $N(f, R)$ contains a vertex $x'_1 \in R$ on Q_1 such that $\mathbf{dist}(y_1, x'_1, Q_1) \leqslant \varepsilon/96 \cdot \mathbf{dist}(f, y_1, \tilde{R}) \leqslant \varepsilon/96 \cdot 6d_1 = \varepsilon/16 \cdot d_1$. By construction, we get that the label $L(x'_1, R) \in FL(f, \text{REG}(Q_1)))$ and thus

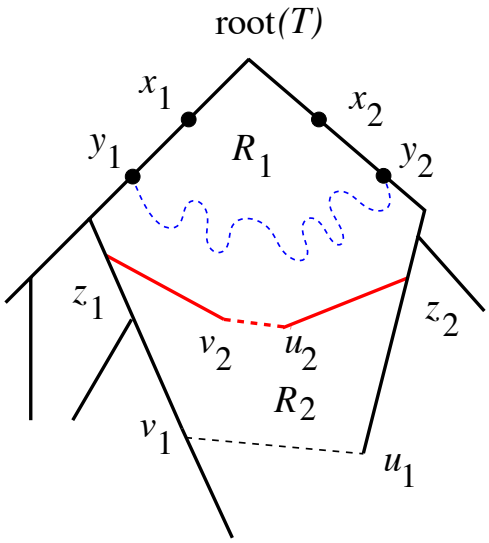

Figure 3: Illustration for case (1.2). Let R be the interior region induced by the separator (T_{v_1}, T_{u_1}). The regions R_1 and R_2 are the child regions of R obtained by the separator (T_{v_2}, T_{u_2}). Note that the entire path Q belongs to R_1. In this case $L(x_1, R) = L(x_1, R_1)$ and $L(x_2, R) = L(x_2, R_1)$ and the claim follows by induction hypothesis on R_1 and on $(y_1, y_2, x_1, x_2, Q_1, Q_2)$.

appears in the query. If x_1 is closer to y_1 than x'_1 then set $x'_1 = x_1$, this is done in order to make sure that $x'_1 \in R$. We get that, $\mathbf{dist}(y_1, x'_1, Q_1) + \mathbf{dist}(r, x'_2, S_1) \leqslant \varepsilon/8 \cdot d_1 = \varepsilon/8 \cdot \mathbf{dist}(y_1, r, G \setminus F)$. Moreover, if the path from x'_2 to r is not free from faults, namely, it contains a fault f, then using similar arguments as claimed above f is not a relevant apex of R. Therefore we may use the label $FL(f, R)$ to get a closer net-point on the path from x'_2 to r and continue this process until finding a net-point x such that the path from x to r is failure-free (or until $x = r$ and clearly $r \notin F$ as r belongs to a path in $G \setminus F$), set x'_2 to that net-point. Note that, the number of edges on the shortest path from y_1 to r in $G \setminus F$ is strictly less than on the path $Q(y_1, y_2, G \setminus F)$. Therefore, the claim follows by induction hypothesis on R and on $(y_1, r, x'_1, x'_2, Q_1, S_1)$.

For illustration of subcase (2.2) see Figure 5. □

LEMMA 7. *For every two vertices u and v such that the labels $L(u)$ and $L(v)$ appear in the query, the graph H contains a path from u to v with stretch at most $1 + \varepsilon$.*

PROOF. The proof is by induction on $\mathbf{dist}(u, v, G \setminus F)$. Assume correctness for all pairs u', v' such that $\mathbf{dist}(u', v', G \setminus F) < d$ and let u and v be a pair of vertices such that $\mathbf{dist}(u, v, G \setminus F) = d$ and that the labels $L(u)$ and $L(v)$ appear in the query. Let Q be the shortest path from u to v in $G \setminus F$.

Note that the tuple $(y_1 = u, y_2 = v, x_1 = u, x_2 = v, \varnothing, \varnothing)$ is legitimate for the region G (the entire graph). Property 1 is satisfied trivially as $x_1 = y_1$ and $x_2 = y_2$. Property (2) is satisfied as all vertices belong to G. Property (3) is satisfied by the assumption the labels $L(u)$ and $L(v)$ appear in the query. Finally, Property (4) is satisfied again trivially as $x_1 = y_1$ and $x_2 = y_2$. By invoking Lemma 6 on the region G (the entire graph) and on $(u, v, u, v, \varnothing, \varnothing)$ we get that there exist four vertices z_1, z_2, w_1, w_2 such that w_1 and w_2 are vertices on the sub-path of Q, the labels of $L(z_1)$ and $L(z_2)$ appear in the query and $2 \cdot \mathbf{dist}(z_1, w_1, G \setminus F) + 2 \cdot \mathbf{dist}(z_2, w_2, G \setminus F) + \mathbf{dist}(z_1, z_2, H) \leqslant (1+\varepsilon) \cdot \omega(Q(w_1, w_2))$. Note that $\mathbf{dist}(u, z_1, G \setminus F) < \mathbf{dist}(u, v, G \setminus F)$ and that $\mathbf{dist}(v, z_2, G \setminus F) < \mathbf{dist}(u, v, G \setminus F)$. By induction hypothesis $\mathbf{dist}(u, z_1, H) \leqslant (1 + \varepsilon) \cdot \mathbf{dist}(u, z_1, G \setminus F) \leqslant (1 + \varepsilon)(\mathbf{dist}(u, w_1, G \setminus F) + \mathbf{dist}(w_1, z_1, G \setminus F)) < (1 + \varepsilon) \cdot \mathbf{dist}(u, w_1, G \setminus F) + 2\mathbf{dist}(w_1, z_1, G \setminus F)$. We also get that $\mathbf{dist}(v, z_2, H) < (1+\varepsilon) \cdot \mathbf{dist}(v, w_2, G \setminus F) + 2\mathbf{dist}(w_2, z_2, G \setminus F)$. We conclude that, $\mathbf{dist}(u, v, H) < (1+\varepsilon) \cdot \mathbf{dist}(x, y, G \setminus F)$ are required. □

By Lemma 7 we conclude the following desired corollary.

COROLLARY 1. *The graph H contains a shortest path from s to t of length at most $(1 + \varepsilon) \cdot \mathbf{dist}(s, t, G \setminus F)$.*

7. WEIGHTED CASE

In this section we describe the modifications needed to handle weighted graphs. In the preprocessing time, before constructing the separators hierarchy tree $\mathcal{T}$ add in the middle of every edge a dummy vertex. The ID of the dummy vertex is set to be the ID of the edge. Now construct the separators hierarchy tree $\mathcal{T}$ on the new graph.

If the net-points $N(v, P)$ for some vertex $v \in V$ and a tail $P \in \mathcal{P}$ contains some dummy vertex that represents an edge (x, y) then add both x and y to $N(v, P)$ and also store the edge (x, y). The rest of the preprocessing phase is the same.

In addition, we slightly modify the definition of legitimate tuple so that the nodes y_1, y_2, x_1 and x_2 are real nodes and not dummy ones.

The query phase is also similar to the unweighted case with a slight modification. If one of the labels contain a dummy vertex that represents an edge (x, y) such that both x and y are non-faulty then add the edge (x, y).

Note that the only place in the proof where we use the fact that the graph is unweighed is in the base case in Lemma 6. Instead of considering the base case, we add another case (3) in the induction step.

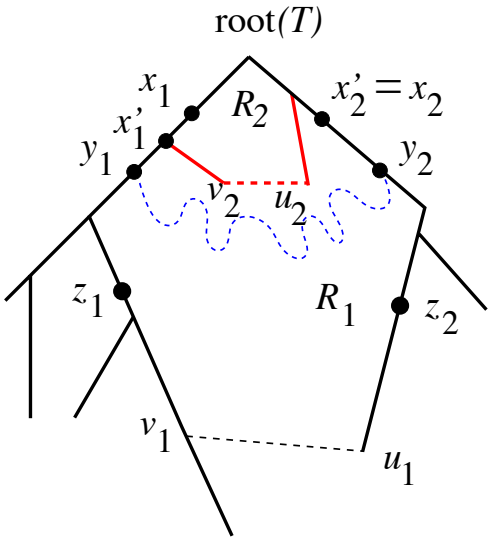

Figure 4: Illustration for case (1.2). Let R be the interior region induced by the separator (v_1, u_1). The regions R_1 and R_2 are the child regions of R obtained by the separator (v_2, u_2). Note that the entire path $Q(y_1, y_2, G \setminus F)$ belongs to R_1. The vertex x'_1 is a new apex of R. Notice that since the (internal) path from x_1 to y_1 does not contain any relevant apex of R, then clearly the (internal) path from x'_1 to y_1 does not contain any relevant apex of R. In addition, the internal path from x'_1 to y_1 does not contain any new apex of R, hence the (internal) path from x'_1 to y_1 does not contain any relevant apex of R_1.

Namely, in the induction step we have the following cases. Case (1) is when $\textsc{Tails}(R) \cap (Q(y_1, y_2, G \setminus F) \setminus \{y_1, y_2\}) = \varnothing$. Case (2) is when $\textsc{Tails}(R) \cap (Q(y_1, y_2, G \setminus F) \setminus \{y_1, y_2\}) \neq \varnothing$ and $k > 1$. Case (3) is when $\textsc{Tails}(R) \cap (Q(y_1, y_2, G \setminus F) \setminus \{y_1, y_2\}) \neq \varnothing$ and $k = 1$.

Cases (1) and (2) are similar to the unweighted case, where in case (2) we pick r to be a real vertex in $(\textsc{Tails}(R) \cap (Q(y_1, y_2, G \setminus F)) \setminus \{y_1, y_2\})$ and not a dummy vertex, the rest of the analysis of this case is the same. Now consider the new case (3), where $\textsc{Tails}(R) \cap (Q(y_1, y_2, G \setminus F) \setminus \{y_1, y_2\}) \neq \varnothing$ and $k = 1$. Note that here unlike the unweighted case, it is not necessarily the case that $x_1 = y_1$ and that $x_2 = y_2$. Notice also that in this case the tails of R contains the dummy vertex $\tilde{y}$ on the edge (y_1, y_2). Let $S_1 \subset \textsc{Tails}(R)$ be the tail of R such that $\tilde{y} \in S_1$. Notice that either the vertices y_1 and x_1 belong to S_1 or the vertices y_2 and x_2. Assume w.l.o.g that $x_1, y_1 \in V(S_1)$. The net-points $N(x_1, S_1)$ contains the dummy vertex $\tilde{y}$ and thus also the vertices y_1, y_2 and the edge (y_1, y_2). We get that the edge (y_1, y_2) is added to H. Moreover, the edges (y_1, x_1) and (y_2, x_2) are safe and thus are also added to H. We get that, $2 \cdot \mathbf{dist}(x_1, y_1, H) + 2 \cdot \mathbf{dist}(x_2, y_2, H) + \mathbf{dist}(y_1, y_2, H) \leqslant 2\varepsilon/8 \cdot \mathbf{dist}(y_1, y_2, G \setminus F) + \mathbf{dist}(y_1, y_2, G \setminus F) < (1+\varepsilon) \cdot \mathbf{dist}(y_1, y_2, G \setminus F)$, and the claim follows where $z_1 = x_1, z_2 = x_2, w_1 = y_1, w_2 = y_2$.

8. OTHER UPDATES

To simplify notations, the construction we describe handles only vertex faults. In this section we explain the slight modifications needed to handle edge failures, edge insertions, vertex insertions and weight changes. Note that a change of edge weight is similar to two consecutive changes: a deletion of the edge and then an insertion of the edge (u, v) with the new weight. We thus focus on edge failures, edge insertions and vertex insertions.

The query now involves two vertices s and t, a set of faulty vertices F_V, a set of faulty edges F_E, a set of new vertices N_V and a set of new edges N_E. The goal is to estimate the distance between s and t in the graph after the updates.

The label of an edge (x, y) is the concatenation of the labels of its endpoints, $(FL(x), FL(y))$.

The safety check of an edge (x, y) with respect to the faulty edge (u, v) in a region R is similar to the check against a faulty vertex and is done as follows. We check the safety of the edge (x, y) against both u and v. We determine that the edge (x, y) is unsafe if it is unsafe against both u and v, otherwise safe. It is not hard to verify that if the edge (x, y) is safe against the faulty edge (u, v) then the path (x, y) represents does not contain the edge (u, v).

To handle an insertion of an edge (u, v), add all safe edges in $L(u)$ and $L(v)$ to H and in addition add the edge (u, v) itself to H. The only change in the analysis is as follows. Let Q be the shortest path from s to t in $G \setminus F$. If the edge (u, v) is not part of the path Q then clearly the insertion of the edge (u, v) does not affect our analysis. Otherwise, if the edge (u, v) is part of the path Q then using induction (on the length of the path) we can claim that the graph H contains a path Q_1 from s to u with stretch $1 + \varepsilon$ and a path Q_2 from v to t with stretch $1 + \varepsilon$. Consider the path $\tilde{Q} = Q_1 \circ (u, v) \circ Q_2$ that is obtained by concatenating the path Q_1, the edge (u, v) and the path Q_2. The path $\tilde{Q}$ is a path from s to t in H with stretch at most $1 + \varepsilon$.

To handle a vertex insertion do the following. Let x be the new vertex. In the query phase, simply add the vertex to the sketch graph H. We then treat each incident edge insertion as a separate operation and simply invoke the method for edge insertion described above.

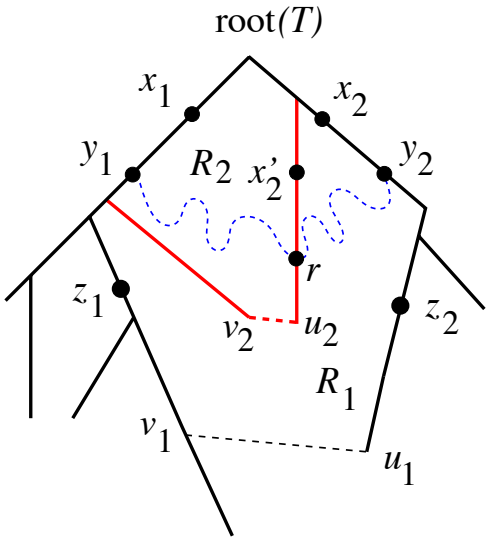

Figure 5: Illustration for case (2). Let R be the interior region induced by the separator (T_{v_1}, T_{u_1}). The regions R_1 and R_2 are the child regions of R obtained by the separator (T_{v_2}, T_{u_2}). The separator of R intersects with $Q(y_1, y_2, G \setminus F) \setminus \{y_1, y_2\}$.

9. REDUCING THE LABEL SIZE

In this section we show how to slightly reduce the labels size to $O(\varepsilon^{-1} \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$.

Recall that the set $N(v, P)$ contains $O(\varepsilon^{-1} \log(nM))$ net-points on the path $P \in \mathcal{P}$. Using these labels, we show in Section 3 a variant of distance labeling scheme that requires labels of length $O(\varepsilon^{-1} \log n \log(nM))$. Thorup [29] shows that it possible to reduce the length of the label to $O(\varepsilon^{-1} \log n)$. More specifically, he shows that it possible to use only $O(\varepsilon^{-1})$ net-points rather than the construction of the $O(\varepsilon^{-1} \log(nM))$ net-points described in Section 3.

Consider a subgraph H and let P be some shortest path in H. For a given $\tilde{\varepsilon} > 0$, the connection (b, v) is said to $\tilde{\varepsilon}$-cover the connection (a, v) in H for vertices $a, b \in P$ and $v \in H$ if $\mathbf{dist}(b, v, H) + \mathbf{dist}(b, a, P) \leqslant (1 + \tilde{\varepsilon}) \cdot \mathbf{dist}(a, v, H)$. A set of connections $C(v, P)$ is an $\tilde{\varepsilon}$-cover set for the vertex v with respect to P and H if for every vertex $a \in P$, there exists a connection $(b, v) \in C(v, P)$ such that (b, v) $\tilde{\varepsilon}$-covers (a, v) in H.

Thorup shows how to construct $\tilde{\varepsilon}$-cover sets $C(v, P)$ of size $O(1/\tilde{\varepsilon})$ for every vertex $v \in H$ in time $O((1/\tilde{\varepsilon})|V(H)| \log n)$ ([29] Lemma 3.18 together with Lemma 3.11).

For the purpose of handling failures we cannot just substitute the net-points stored in $N(v, P)$ with the improved net-points of [29]. To see the difficulty, consider a shortest path Q from s to t in $G \setminus F$ and let P be the first tail that crosses Q, that is $Q \cap P \neq \varnothing$. Let x be a vertex in $Q \cap P$ and let (b, s) be a connection that $\tilde{\varepsilon}$-covers (x, s). Namely, $\mathbf{dist}(b, s, G) + \mathbf{dist}(b, x, G) \leqslant (1 + \tilde{\varepsilon}) \cdot \mathbf{dist}(s, x, G)$. Note that it could be that b is relatively far away from x. For example, assume that there are two disjoint shortest paths from s to x, one is the subpath $Q(s, x)$ of Q between s and x and the second shortest path $\tilde{Q}(s, x)$ go through b and assume that $\mathbf{dist}(b, s, G), \mathbf{dist}(b, x, G) = 1/2 \cdot \mathbf{dist}(s, x, G)$. Note that in this case (b, s) indeed $\tilde{\varepsilon}$-cover (x, s). Now assume that the path from x to b is faulty. If we indeed replace the net-points stored in $N(f, P)$ with improved net-points of [29], then using the label $FL(f, \textsc{Reg}(P))$, we could get the label $L(b, \textsc{Reg}(P))$. However the distance between s and b in $G \setminus F$ could be $1.5 \cdot \mathbf{dist}(s, x, G)$ since it could be that the only path between s and b in $G \setminus F$ goes through x. Thus the label $L(b, \textsc{Reg}(P))$ could not help us to approximate the distance between s and x in $G \setminus F$. In other words, in the failure-prone setting we need that the net-point that covers some vertex $x \in P$ be "close" to x on the path P.

Therefore, simply replacing all our net-points with Thorup's improved net-points seems challenging in our setting.

However, we show that replacing only the net-points of $N(v, P)$ in the label of $L(v, R)$ with Thorup's improved net-points does work and it replace a factor of $\varepsilon^{-1} \log(nM)$ with the factor $\varepsilon^{-1} + \log n$. Recall that there are two places where we use the net-points of $N(v, P)$. First, in the construction of the labels $L(v, R)$ and second in the construction of the labels $FL(v, R)$. The labels $L(v, R)$ are only used to approximate distances between vertices in R (as if there are no faults). In case some of these edges are not safe, we use the FL labels of the faulty vertices to get the L labels of other vertices "close" to the path between s and t in $G \setminus F$. Therefore in the construction of the labels $L(v, R)$ we can use Thorup's improved net-points. Recall that we also add to the net-set $N(v, R)$ all relevant and new apices of R. Thus, using Thorup's improved net-points, we can replace a factor of $\varepsilon^{-1} \log(nM)$ with a factor of $O(\varepsilon^{-1} + \log n)$.

Formally the labels are defined as follows. The definition of the net-points $N(v, P)$ is unchanged. In addition, we also introduce another set of net-points $\tilde{N}(v, P)$ where $\tilde{N}(v, P)$ is the cover set obtained by Thorup's construction on P and R, together with the relevant apices of R. The edge-set $\tilde{E}(v, R)$ is exactly as $E(v, R)$ with using $\tilde{N}(v, P)$ instead of $N(v, P)$.

The label $L(v, R)$ for some region R is defined as before but with using $\tilde{E}(v, R')$ instead of $E(v, R')$.

The labels $FL(v, R)$ and $FL(v)$ are unchanged. It is not hard to verify that the new labels length are of size $O(\varepsilon^{-1} \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$.

10. CONSTRUCTION AND QUERY TIME

Construction Time.

We now show that our data structure can be constructed in $O(\varepsilon^{-1} n \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$ time.

The tree decomposition $\mathcal{T}$ can be constructed in $O(n \log n)$ time. It is not hard to verify that finding the regions and the relevant apices of the regions can be done in $\sum_{R \in \mathcal{R}} |R| = O(n \log n)$ by Property 4.

The edge-sets $\tilde{E}(v, R)$ for every vertex $v \in R$ can be constructed as shown in [29] in $O(\varepsilon^{-1}|R| \log n)$ time. One thing to notice it that in our construction we also add to $\tilde{E}(v, R)$, all the relevant and new apices of R. However, we can first invoke Thorup construction and then for the relevant apices we can simply invoke a shortest path algorithm in R (to find the distances from all vertices in R to the relevant apices). As there are $O(\log n)$ relevant apices, this can be done in $O(|R| \log n)$ time. We thus get that the total construction time for all $\tilde{E}(v, R)$ for every region R and vertex $v \in R$ is $O(\varepsilon^{-1} n \log^2 n)$.

Let $\tilde{R} = R \setminus \textsc{RelApices}(R)$. One thing to notice is that, as opposed to Thorup's construction where the distances are with respect to the region R, our distances for a vertex x are with respect to $\tilde{R} + \{x\}$. This is only a minor technical issue. The only slight change in Thorup's construction is the following. While calculating the distances from some net-point $x \in \textsc{Tails}(R)$ to a set of vertices S, we first find the distances in the induced graph $\tilde{R} \cap S$ and for every vertex u in $(R \setminus \tilde{R}) \cap S$ we find the vertex v in $\tilde{R} \cap S$ such that the length of the path obtained by concatenating the edge (v, u) with the shortest path from u to x is minimal. The analysis and running time are unchanged.

Consider a region R of $\mathcal{T}$ and a tail P of $\textsc{Tails}(R)$. We now show that it possible to find the net-points $N(v, P)$ for every vertex $v \in R$ in time $O(\varepsilon^{-1}|R| \log(nM) \log\log M)$. Finding for every vertex $v \in R$ the closest (with respect to distances in R) vertex c_v on P can be done in $O(|R|)$, by for example contracting the path P into a single vertex, invoking Dijkstra algorithm and then expanding the path P back. Once having the closest vertex c_v on P for a vertex v, the set of net-points $N(v, P)$ can be constructed trivially in $O(\varepsilon^{-1} \log(nM) \log\log M)$ time, by numbering the vertices in P so that we can get the next vertex at distance d' from some vertex x' in P in $O(\log\log M)$ (using predecessor queries).

We get that the sets of net-points $N(v, P)$ for every vertex $v \in R$ can be constructed in time $O(\varepsilon^{-1}|R| \log(nM) \log\log M)$. Thus, all sets of net-points $N(v, P)$ for every vertex $v \in R$ and tail P of R can be constructed in time $O(\varepsilon^{-1} n \log n \log(nM) \log\log M)$.

The label $L(v, R)$ is just the concatenation of some edge-sets $\tilde{E}(v, R')$ and thus can be constructed in time proportional to the size of $L(v, R)$. This is also the case for the labels $FL(v, R)$ and $FL(v)$.

We conclude that the labels can be constructed in $O(\varepsilon^{-1} n \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$ time.

Query time.

Consider a source s, a target t and a subset U of updates.

There are $O(\varepsilon^{-1}|U| \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$ edges and $O(|U|)$ vertices in the query.

In order to keep the number of edges that are added to H, $O(\varepsilon^{-1}|U| \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$, we do the following. For every tail $P \in \mathcal{P}$ we first number the net-points in P that appear in the query, according to their appearance in P. Second, instead of adding the safe edges between all pairs of net-points in P, we add only safe edges between pairs of consecutive net-points. It is not hard to verify that this does not change the analysis and that the number of potential edges stays $O(\varepsilon^{-1}|U| \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$.

For each such edge we need to check the safety of the edge against all faulty vertices and edges.

In order to check if the edge (u, v) in some region R is safe with respect to some faulty vertex f (or an edge (f_1, f_2)), we need to approximate the distance $\mathbf{dist}(f, v, R)$ ($\mathbf{dist}(f_1, v, R)$ and $\mathbf{dist}(f_2, v, R)$) using $\tilde{E}(f, R)$. This can be done in $O(\varepsilon^{-1})$ time. In fact, rather than approximating the distances $\mathbf{dist}(f, v, R)$ with ε-factor, we instead can settle for a constant approximation and save a $1/\varepsilon$ factor from the query time. Namely, we construct the labels for both $\tilde{\varepsilon} = \varepsilon$ and $\tilde{\varepsilon} = 1$ and store both labels. This requires adjusting the net-points in $N(f, P)$ by a constant factor, in order to get the same analysis. Thus the total time for constructing the graph H is $O(\varepsilon^{-1}|U|^2 \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$. There are $O(\varepsilon^{-1}|U| \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$ edges and $O(|U|)$ vertices in H, thus invoking Dijkstra on H takes $O(\varepsilon^{-1}|U| \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$ time.

We thus conclude that the query time is $O(\varepsilon^{-1}|U|^2 \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$.

11. FULLY DYNAMIC APPROXIMATE DISTANCE ORACLE FOR PLANAR GRAPHS

In this section we prove Theorem 2, namely, we show that our labeling scheme can be tweaked to give fully dynamic approximate distance oracle of size $O(n \log^2 n \cdot (\varepsilon^{-1} + \log n))$ with worst case query and update time of $\tilde{O}(n^{1/2})$. This improves on the previously known fully dynamic approximate distance oracle, which has worst case query time of $\tilde{O}(n^{2/3})$ and amortized update time of $\tilde{O}(n^{2/3})$.

Our data structure supports vertex removal, edge removal, vertex insertion, edge insertion and edge weight change. Where the assumption is that during all these updates the graph stays planar.

Centralized Forbidden-Sequence Approximate Distance Oracles

In this subsection we prove Theorem 3. We show how to transform our forbidden-set labeling scheme to a *centralized dynamic approximate distance oracle* (`FADO`) of size $O(n \log^2 n \cdot (\varepsilon^{-1} + \log n))$ with worst case update time and query time $\tilde{O}(|U|)$, where U is the sequence of updates that occurred since the construction of this data structure. Denote by $\mathcal{D}_{\texttt{FADO}}(\tilde{G})$ the `FADO` data structure that is constructed on the graph $\tilde{G}$.

We later see how to use this data structure to get a fully dynamic approximate distance oracle of size $O(n \log^2 n \cdot (\varepsilon^{-1} + \log n))$ with worst case query time and update time of $\tilde{O}(n^{1/2})$.

The Data Structure.

Our data structure is basically a data structure capable of answering distance queries in the failure-free setting with some slight additions. Most of the data stored in the labels of our labeling scheme can be constructed (using the stored failure-free distance oracle) in time proportional to the size

of the data, and thus there is no reason to store it in our centralized data structure.

Recall that as opposed to the failure-free setting, in the failure-prone setting it is essential that the length of an edge $(v,u) \in \tilde{E}(v,R)$ reflects the distance between v and u in $\tilde{R} + \{v,u\}$ rather than its distance in G. This is essential due to the following scenario. It could be that the distance between u and v is very short in G, relative to their distance in $\tilde{R} + \{v,u\}$, but the short path goes through some faulty vertex $f \in F$. Also assume that f is not in R and thus cannot provides us additional information that can indicates an alternative path between u and v in R.

We now formally described our data structure. Construct the tree decomposition $\mathcal{T}$. Next, for every region $R \in \mathcal{R}$ construct the edge-sets $\tilde{E}(v,R)$ for every $v \in R$.

For every vertex $v \in R$ and every tail P of R, store the closest vertex to v on P with respect to the distances in the subgraph of $R \setminus \textsc{RelApices}(R)$.

In addition, we store the set U of the updates that occurred from the beginning of this data structure. Finally, since it is too "costly" to construct the sketch graph H in the query phase, we rather gradually construct the graph H during the updates and store it as part of our data structure. For every edge $(u,v) \in \tilde{E}(v,R)$ that is added to H, we store an indication that the source of the edge (u,v) is $\tilde{E}(v,R)$.

The Update Phase.

We now explain how to modify the sketch graph H given an additional update.

The update phase for a vertex deletion is as follows. Let f be the deleted vertex. The first stage is to check the safety of all edges in H with respect to the new fault f.

In the second stage, construct $FL(f)$ and add to H all safe edges stored in $FL(f)$.

The update phase for adding an edge (u,v) is done as follows. Add all safe edges in $L(u)$ and $L(v)$ and in addition add the edge (u,v) to the graph H.

Removing an edge (x,y) is similar to a vertex deletion and is done as follows. Notice that if one of x and y is already faulty then there is no need to do anything, so assume both x and y are not faulty. The first stage is to check the safety of all edges in H with respect to the new faulty edge (x,y).

In the second step, construct $FL(u)$ and $FL(v)$ and add to H all safe edges stored in $FL(u)$ and $FL(v)$.

The update phase for adding a new vertex is done simply by adding the vertex to the graph H.

Changing the weight of an edge (u,v) is done as follows. We first remove the edge (u,v) and then add the edge (u,v) back to H (with the new weight).

The Query Phase.

In the query phase we are given two non-faulty vertices s and t whose distance needs to be approximated in G', where G' is the graph $\tilde{G}$ after the sequence of updates U.

The query phase is done as follows. Construct the labels $L(s)$ and $L(t)$. Add all safe edges from $L(s)$ and $L(t)$ to H, compute the shortest path from s to t in H and return $\mathbf{dist}(s,t,H)$. In the end of the query phase, delete the edges added to H from $L(s)$ and $L(t)$.

Correctness.

We need to show that the approximate distance returned in the query phase is indeed at least the real distance and at most $1+\varepsilon$ the real distance. The correctness of the algorithm is similar to the one presented in Section 5. It is not hard to verify that constructed graph H is exactly the same graph constructed by the labeling-scheme for the query (s,t,U) as described in Section 5. Thus the correctness is derived from the correctness of Section 5.

Analysis.

We now analyze the construction time, update time and query time.

The construction time of the data structure: Constructing the tree decomposition can be done in $O(n \log n)$ time. Consider a region $R \in \mathcal{R}$.

The construction time for all $\tilde{E}(v,R)$ for every region $R \in \mathcal{R}$ and vertex $v \in R$ is $O(\varepsilon^{-1} n \log^2 n)$ as shown in Section 10. In addition, for every tail P of R, the closest vertex c_v to v on P with respect to the distances in the subgraph R can be done in time $O(\sum_{R \in \mathcal{R}} |R|) = O(n \log n)$ from Property 4. We conclude that the total construction time is $O(\varepsilon^{-1} n \log^2 n)$.

We now analyze the update phase for a vertex deletion.

In the first step, we check the safety of edges in H with respect to the new fault f. In order to check if the edge (u,v) is safe with respect to some faulty vertex f, we need to approximate the distance $\mathbf{dist}(f,v,R)$. This can be done in $O(1)$ time (as mentioned earlier it is enough to approximate $\mathbf{dist}(f,v,R)$ with a constant factor by adjusting the net-points in $N(f,R)$ by some constant factor).

There are $O(\varepsilon^{-1}|U| \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$ edges in H. For each such edge the safety check against the new fault f takes $O(1)$ time. Thus the total time for this part is $O(\varepsilon^{-1}|U| \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$.

In addition, we add all safe edges in the label $FL(f)$. It is not hard to verify that constructing $FL(f)$ can be done in time proportional to the size of $FL(f)$. There are $O(\varepsilon^{-1} \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$ edges in $FL(f)$, for each such edge we need to check its safety against all faulty vertices and edges. This can be done in $O(\varepsilon^{-1}|U| \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$ time.

The analysis for edge deletion/insertion are similar for vertex deletion. We thus conclude the total time for the update phase is $O(\varepsilon^{-1}|U| \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$.

We turn turn to analyze the query phase. We are given two vertices s and t and add all safe edges in $L(s)$ and $L(t)$. The number of edges in $L(s)$ and $L(t)$ is $O(\log n \cdot (\varepsilon^{-1} + \log n))$. The safety of each such edge can be done in $O(|U|)$ time. Thus the total time for this part is $O(|U|(\varepsilon^{-1} + \log n))$. In addition, we invoke a shortest path algorithm from s to t that takes $O(|E(H)|) = O(\varepsilon^{-1}|U| \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$ time.

We conclude that the query phase takes $O(\varepsilon^{-1}|U| \log^2 n \log(nM) \cdot (\varepsilon^{-1} + \log n))$ time.

We now turn to bound the size of the data structure. We have, $\sum_{R \in \mathcal{R}} |R| = O(n \log n)$ by Property 4. For each region R and node $v \in R$, we store $\tilde{E}(v,R)$, which is of size $O(\varepsilon^{-1} + \log n)$. We thus conclude that the size of the data structure is $O(n \log n \cdot (\varepsilon^{-1} + \log n))$.

We thus conclude Theorem 3.

Fully Dynamic Approximate Distance Oracles for Planar Graphs

We now show how to transform the `FADO` data structure to a fully dynamic approximate distance oracle of size $O(n \log^2 n \cdot (\varepsilon^{-1} + \log n))$ with worst case query and update time of $\tilde{O}(n^{1/2})$.

Notice that using the FADO data structure, we can easily get a fully dynamic approximate distance oracle with worst case query time $\tilde{O}(n^{1/2})$ and amortized update time of $\tilde{O}(n^{1/2})$. We simply construct the FADO data structure $\mathcal{D}_{\texttt{FADO}}(G)$, where G is the current graph and after every $n^{1/2}$ updates we simply reconstruct the FADO data structure for the updated graph.

Note that in order to get worst case update time of $\tilde{O}(n^{1/2})$ it is not enough to simply reconstruct the new FADO data structure during the next $n^{1/2}$ updates and use it once we finish to construct it, since during the construction of the data structure, some new updates accumulated and should appear in the sketch graph H.

We prove the following general theorem.

THEOREM 5. *Consider an oracle for n-vertex m-edge graphs that has size s, construction time c, update time $u(|U|)$, and query time $q(|U|)$, where U is the sequence of at most $O(\sqrt{n+m})$ updates that occurred since the construction of the oracle. One can transform the oracle into dynamic oracle of size $O(s+n+m)$, with worst case update time $O(u(r)+(c+n+m)/r)$ and worst case query time $q(r)$ for every integer r.*

Consider an oracle of size s, construction time c, update time $u(|U|)$ and query time $q(|U|)$ and let $r \leqslant |U|$ be some non-negative integer. Denote by $\mathcal{O}(\tilde{G})$ the oracle that is constructed on the graph $\tilde{G}$. And by $\mathcal{O}(\tilde{G}, U)$ the oracle $\mathcal{O}(\tilde{G})$ after handling the sequence of updates U.

We maintain two different such oracles. For the sake of analysis partition the sequence of $|U|$ updates into intervals of $r' = r/2$ each. Let $I_1, I_2, \ldots$ be the set of intervals. Let G_i be the graph at the beginning of the interval I_i.

During the interval I_i we maintain two oracles, for $i > 1$. The first oracle D_1 is constructed on the graph G_{i-1} and contains all updates of I_{i-1} and the updates of I_i occurring so far. The second oracle D_2 is constructed on the graph G_i during the first half of the interval I_i. More precisely, in the beginning of the interval I_i, we have the graph G_{i-1} and the list of r' updates of the interval I_{i-1}. During the first half of the interval I_i we construct the graph G_i and afterwards the oracle $D_2 = \mathcal{O}(G_i)$. During the second half of the interval I_i we add to D_2 all updates occurring in I_i (by adding to D_2 two updates each time).

In addition we store all updates occurring in the current interval I_i. Finally, we keep the graph $\tilde{G} = G_i$.

Given an n-vertex m-edge graph, and given a set of r' updates, clearly one can construct the updated graph in $O(n + m + r')$ time. Namely, one can construct the updated graph together with the new oracle on the updated graph in time at most $c_0 \cdot (c+n+m+r')$ for some constant $c_0 > 0$. Let $t = c_0 \cdot (c + n + m + r')/r'$.

The Update Phase.

Formally, during the j'th update in the interval I_i we do the following.

(1) If $i > 1$ and $j \leqslant r'/2$: make the next $2t$ steps for constructing the updated graph $\tilde{G}$ $(= G_i)$ and the data structure D_2 $(= \mathcal{O}(G_i))$ according to the graph $\tilde{G}$.

(2) If $i > 1$ and $j > r'/2$: add to D_2 $(= \mathcal{O}(G_i))$ the $(2(j - r') - 1)$-th and $2(j - r')$-th updates of the interval I_i.

(3) Add the j'th update to D_1 $(= \mathcal{O}(G_{i-1}))$.

In the end of interval i for any $i > 1$: we set $D_1 = D_2$, $D_2 = \perp$.

We say that an oracle $\mathcal{O}$ is ready if it contains all updates that occurred since the construction of it.

We now state some important observations of the update phase. In the end of the interval I_i, the oracle D_2 satisfies $D_2 = \mathcal{O}(G_i, I_i)$, namely, the oracle D_2 is ready.

Let I_i^j be the first j updates in the interval I_i. In the end of the j'th update of the interval I_i we have, $D_1 = \mathcal{O}(G_{i-1}, I_{i-1} \cup I_i^j)$, namely, the oracle D_1 is ready.

The Query Phase.

In the query phase we simply invoke the query algorithm of D_1 and return the value.

Correctness.

As stated above D_1 is ready, namely, the data structure D_1 contains all updates that occurred since its construction. The correctness is thus obvious from the correctness of the oracle.

Analysis.

In is not hard to verify that each of the steps make in the updates phase takes $O(u(2r') + t)$ time. Thus, the update phase takes $O(u(r) + (c + n + m)/r)$ time. The size of the dynamic oracle is $O(s+n+m)$ due to the storage of the two oracles and the graph $\tilde{G}$.

In the query phase we simply invoke the query algorithm of D_1. Note that there are at most $2r' = r$ updates in D_1. Therefore the query phase takes at most $q(r)$ time.

Combining Theorems 3 and 5 (with $r = \sqrt{n+m}$ and $m = O(n)$) we conclude Theorem 2.

We note that one can combine our techniques presented in this section with the forbidden labeling schemes presented in [1] and [12] to get the following theorems (we omit the details).

THEOREM 6. *Given an n-vertex of tree-width or clique-width $\tilde{O}(1)$, one can construct a fully dynamic exact distance oracle of size $\tilde{O}(n)$. Each query operation and each update operation takes $\tilde{O}(n^{1/2})$ worst case time.*

THEOREM 7. *For all fixed α and $\varepsilon > 0$, and given an n-vertex unweighted graph of doubling dimension α, one can construct a fully dynamic $(1+\varepsilon)$ approximate distance oracle of size $\tilde{O}((1 + \varepsilon^{-1})^{2\alpha} n)$. Each query operation and each update operation takes $\tilde{O}(n^{1/2})$ worst case time.*

12. ROUTING SCHEME

We can easily transform our forbidden-set labeling scheme to a forbidden-set compact routing scheme resulting with Theorem 4.

We now describe our routing scheme. In this setting, we assume that all vertices are familiar with the labels of the faulty nodes/edges. In addition, we attach the labels $L(s)$ and $L(t)$ of the source s and destination t to the header of the message. Thus, any vertex that receives the message during the routing process is familiar with the labels $\{L(s), L(t)\} \cup \{FL(v) : v \in F\}$. We later, consider the somewhat more natural setting, where the vertices are not familiar with the labels of the faulty vertices.

For the sake of routing, we need an additional property from the $\tilde{\varepsilon}$-cover set described earlier. Namely, for every connection (v, a) in $\tilde{E}(v, R)$, consider the shortest path Q in

$\tilde{R} + \{v, a\}$ between v and a, where $\tilde{R} = R \setminus \text{RELAPICES}(R)$. Then, for every vertex x of Q the connection (x, a) belongs to $\tilde{E}(x, R)$. We call this property the *continuous* property. Thorup [29] shows that it possible to construct set covers of size $O(\varepsilon^{-1} \log n)$ with this additional property. When constructing the edge-sets $\tilde{E}(u, R)$ we thus use these set covers with the continuous property. This blows-up the labels length by a factor of $\log n$.

We now describe the modifications in the preprocessing phase.

For every vertex v of any tail P of a region R, we construct a shortest path tree $T_{\text{route}}(v, R)$ rooted at v spanning the set $B(v, R) = \{x \in R : v \in \tilde{N}(x, P)\}$ of vertices x of R such that v is a net-point in $\tilde{N}(x, P)$. Note that due to the continuous property, it is possible to form a shortest path tree from the set $B(v, R)$. During the routing process we may need to route a message from some vertex in $B(v, R)$ to v or from v to some vertex in $B(v, R)$. For that purpose we use the labeled based tree routing scheme [32, 18]. That scheme uses $o(\log n)$ labels and routing tables size. Let $\lambda(x, T_{\text{route}}(v, R))$ and $RT(x, T_{\text{route}}(v, R))$ be the label and routing table (resp.) given to the vertex $x \in B(v, R)$ by this tree routing scheme based on $T_{\text{route}}(v, R)$. Now, for every edge $(u, v) \in \tilde{E}(u, R)$, we add to $I(u, v, R)$ the labels of both $\lambda(u, T_{\text{route}}(v, R))$ and $\lambda(v, T_{\text{route}}(v, R))$, allowing routing to both directions, that is from u to v and from v to u in $T_{\text{route}}(v, R)$. This blows-up the labels length by a factor of $o(\log n)$. In addition, if u is a relevant apex of R, we also add to $I(u, v, R)$ the routing table $RT(x, T_{\text{route}}(v, R))$. In addition, for every vertex x of $T_{\text{route}}(v, R)$ that is not a relevant apex of R we store at x the routing table $RT(x, T_{\text{route}}(v, R))$.

During the routing process on the tree $T_{\text{route}}(v, R)$ either from v to some vertex x or from some vertex x to v, the label of the destination is attached to the header of the message. Note that for every edge $(u, v) \in \tilde{E}(u, R)$, all vertices along the shortest path the edge (u, v) represents, are not relevant apices of R, except maybe u itself. Therefore, every vertex x on that path contains $RT(x, T_{\text{route}}(v, R))$ and can thus forward the message to the right port, for the vertex u itself we store its routing table $RT(u, T_{\text{route}}(v, R))$ in $I(u, v, R)$ hence u can use it to forward the message.

Consider a vertex x. There are $O(\log n)$ regions in $\text{RR}(x)$, for each such region R, there are $O(\log n/\varepsilon)$ net-points in $\tilde{N}(x, R)$, for each such net-point v, the routing table $RT(x, T_{\text{route}}(v, R))$ is of $o(\log n)$ size. All in all, we get that the routing table at a vertex x is of size $o(\log^3 n/\varepsilon)$.

It is not hard to verify that given that routing information and the labels of s, t, F, for any edge (u, v) in H (constructed according to the query (s, t, F)) stored in some region R, one can route on the shortest path from u to v in the subgraph R. This implies that the stretch obtained by this routing scheme is exactly the same stretch of our distance labeling scheme. We thus have Theorem 4.

Our routing scheme can be tweaked to handle setting where the vertices do not know in advance which vertices are faulty and only discover the faulty vertices during the routing process at the price of an additional factor of $O(|F|)$ in the stretch. This requires an additional modification – every vertex needs to be familiar with the labels of its neighbors. We attach to the header of the message the set of the discovered faulty vertices along with a copy of the current graph H. In each step an attempt is made to route to the destination according to graph H, if no additional fault is encountered then the message will be delivered to the destination (assuming the source and the destination are still connected). Otherwise, if a new faulty vertex is encountered, we attach the label of the faulty vertex and update H accordingly. Notice that the latter can happen at most $|F|$ times. This gives the following .

THEOREM 8. *Given an n-vertex planar graph G with edge weights in $[1, M]$, one can efficiently construct a routing scheme that given a source vertex s and a target vertex t, in the presence of a set of failures F (unknown to s), can route a message from s to t in a distributed manner over a path of length at most $O(|F| \cdot \mathbf{dist}(s, t, G \setminus F))$. The scheme assigns each vertex v a label of length $o(\log^4 n \log(nM))$ and routing table of $o(\deg(v) \cdot \log^3 n \log(nM))$ size, where $\deg(v)$ is the degree of v in G. The message header passed during the routing process is of size $o(|F| \cdot \log^3 n \log(nM))$.*

We note that the factor of $|F|$ appearing in the stretch is in fact necessary. To see this, consider a source vertex s and a target t and assume the graph G is composed of $|F| + 1$ disjoint shortest paths from s to t. Now, pick $|F|$ of these shortest path and assume the last vertex (the vertex adjacent to t) in each of these shortest paths fail. Since the source do not know in advance which vertices fail it might try to route the message through the faulty paths first. This yields a stretch of $\approx 2|F|$.

Acknowledgement. We are grateful to David Peleg for very helpful discussions.

13. REFERENCES

[1] I. Abraham, S. Chechik, C. Gavoille and D. Peleg. Forbidden-set distance labels for graphs of bounded doubling dimension. *29th ACM Symp. on Principles of Distributed Computing (PODC)*, 192–200, 2010.

[2] G. Ausiello, G. F. Italiano, A. Marchetti-Spaccamela, and U. Nanni. Incremental algorithms for minimal length paths. *J. Algorithms, 12(4)*, 615–638, 1991.

[3] S. Baswana, R. Hariharan, and S. Sen. Improved decremental algorithms for transitive closure and all-pairs shortest paths. In *34th ACM Symp. on Theory of Computing (STOC)*, 117–123, 2002.

[4] S. Baswana, R. Hariharan, and S. Sen. Maintaining all-pairs approximate shortest paths under deletion of edges. In *14th ACM Symp. on Discrete Algorithms (SODA)*, 394–403, 2003.

[5] S. Baswana, U. Lath, and A. S. Mehta. Single source distance oracle for planar digraphs avoiding a failed node or link. In *23rd ACM Symp. on Discrete Algorithms (SODA)*, 223-232, 2012.

[6] A. Bernstein. Fully dynamic approximate all-pairs shortest paths with query and close to linear update time. In *50th IEEE Symp. on Foundations of Computer Science (FOCS)*, 50–60, 2009.

[7] A. Bernstein and D. Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *41st ACM Symp. on Theory of Computing (STOC)*, 101–110, 2009.

[8] A. Bernstein, L. Roditty. Improved dynamic algorithms for maintaining approximate shortest paths under deletions. In *22nd ACM Symp. on Discrete Algorithms (SODA)*, 1355–1365, 2011.

[9] S. Chechik. Fault-tolerant compact routing schemes for general graphs. In *38th Int'l Coll. on Automata, Languages and Programming (ICALP)*, 101–112, 2011.

[10] S. Chechik, M. Langberg, D. Peleg, and L. Roditty. f-sensitivity distance oracles and routing schemes. In *18th European Symposia on Algorithms (ESA)*, 84–96, 2010.

[11] B. Courcelle and D. A. Twigg. Compact forbidden-set routing. In *24th Symp. on Theoretical Aspects of Computer Science (STACS)*, 37–48, 2007.

[12] B. Courcelle and D. A. Twigg. Constrained-path labellings on graphs of bounded clique-width. *Theory of Computing Systems, 47*, 531–567, 2010.

[13] C. Demetrescu and G. F. Italiano. A new approach to dynamic all pairs shortest paths. *J. of the ACM, 51*, 2004.

[14] C. Demetrescu and G. F. Italiano. Fully dynamic all pairs shortest paths with real edge weights. *J. of Computer and System Sciences, 72(5)*, 813–837, 2006. Special issue of *FOCS* '01.

[15] C. Demetrescu and M. Thorup. Oracles for distances avoiding a link-failure. In *13th ACM Symp. on Discrete Algorithms (SODA)*, 838–843, 2002.

[16] R. Duan and S. Pettie. Dual-failure distance and connectivity oracles. In *20nd ACM Symp. on Discrete Algorithms (SODA)*, 506–515, 2009.

[17] D. Eppstein. Dynamic generators of topologically embedded graphs. In *14th ACM Symp. on Discrete Algorithms (SODA)*, 599–608, 2003.

[18] P. Fraigniaud and C. Gavoille. Routing in Trees. In *28th Int'l Coll. on Automata, Languages and Programming (ICALP)*, 757–772, 2001.

[19] M. R. Henzinger and V. King. Fully dynamic biconnectivity and transitive closure. In *36th IEEE Symp. on Foundations of Computer Science (FOCS)*, 664–72, 1995.

[20] M. R. Henzinger and V. King. Maintaining Minimum Spanning Forests in Dynamic Graphs. *SIAM J. Computing, 31(2)*, 364–374, 2001.

[21] K. Kawarabayashi, P. N. Klein, and C. Sommer. Linear-space approximate distance oracles for planar, bounded-genus, and minor-free graphs. In *38th Int'l Coll. on Automata, Languages and Programming (ICALP)*, 135–146, 2011.

[22] N. Khanna and S. Baswana. Approximate shortest path oracle under vertex failure. In *26th Symp. on Theoretical Aspects of Computer Science (STACS)*, 513–524, 2010.

[23] V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *40th IEEE Symp. on Foundations of Computer Science (FOCS)*, 81–91, 1999.

[24] P. Klein, and S. Subramanian. A fully dynamic approximation scheme for shortest path problems in planar graphs. *Algorithmica 23*, 235–249, 1998.

[25] P. Klein. Preprocessing an undirected planar network to enable fast approximate distance queries. In *13th ACM Symp. on Discrete Algorithms (SODA)*, 820–827, 2002.

[26] R. Lipton, and R. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math. 36*, 177–189, 1979.

[27] M. Pătraşcu and M. Thorup. Planning for fast connectivity updates. In *48th IEEE Symp. on Foundations of Computer Science (FOCS)*, 263–271, 2007.

[28] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. In *42nd IEEE Symp. on Foundations of Computer Science (FOCS)*, 242–251, 2001.

[29] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. of the ACM 51(6)*, 993–1024, 2004.

[30] M. Thorup. Fully-dynamic all-pairs shortest paths: faster and allowing negative cycles. In *9th Scandinavian Workshop on Algorithm Theory (SWAT)*, 384–396, 2004.

[31] M. Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *37th ACM Symp. on Theory of Computing (STOC)*, 112–119, 2005.

[32] M. Thorup and U. Zwick. Compact routing schemes. In *13th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, 1–10, 2001.

[33] L. Roditty and U. Zwick. Dynamic approximate all-pairs shortest paths in undirected graphs. In *45th IEEE Symp. on Foundations of Computer Science (FOCS)*, 499–508, 2004.