An Optimal Dual Fault Tolerant Reachability Oracle [*]

Keerti Choudhary

Department of Computer Science and Engineering
IIT Kanpur, Kanpur - 208016, India.
keerti@cse.iitk.ac.in

Abstract

Let $G=(V,E)$ be an n-vertices m-edges directed graph. Let $s\in V$ be any designated source vertex. We address the problem of reporting the reachability information from s under two vertex failures. We show that it is possible to compute in polynomial time an $O(n)$ size data structure that for any query vertex v, and any pair of failed vertices f_1, f_2, answers in $O(1)$ time whether or not there exists a path from s to v in $G\setminus\{f_1,f_2\}$.

For the simpler case of single vertex failure such a data structure can be obtained using the dominator-tree from the celebrated work of Lengauer and Tarjan [TOPLAS 1979, Vol. 1]. However, no efficient data structure was known in the past for handling more than one failures. We, in addition, also present a labeling scheme with $O(\log^3 n)$-bit size labels such that for any $f_1, f_2, v \in V$, it is possible to determine in poly-logarithmic time if v is reachable from s in $G\setminus\{f_1,f_2\}$ using only the labels of f_1, f_2 and v.

Our data structure can also be seen as an efficient mechanism for verifying double-dominators. For any given $x, y, v \in V$ we can determine in $O(1)$ time if the pair (x,y) is a double-dominator of v. Earlier the best known method for this problem was using dominator chain from which verification of double-dominators of only a single vertex was possible.

1998 ACM Subject Classification G.2.2 Graph Algorithms

Keywords and phrases Fault tolerant, Directed graph, Reachability oracle, Labeling scheme

Digital Object Identifier 10.4230/LIPIcs.ICALP.2016.XXX

1 Introduction

Networks in most real life applications are prone to failures. These failures, though unpredictable, are transient due to some simultaneous repair process that is undertaken in the application. This motivates the research on designing fault tolerant structures for various graph problems. In the past few years, a lot of work has been done in designing fault tolerant structures for various fundamental graph problems, see [7, 9, 14].

We address the problem of building a compact data structure for answering reachability queries from a designated source on vertex failures. The only previous known result for this problem was for single failure. The single fault tolerant reachability is closely related to the notion of *dominators* as follows. Given a directed graph G and a source vertex s, we say that a vertex x dominates a vertex v if every path from s to v contains x. Lengauer and Tarjan [12] introduced a data structure called *dominator tree* which is a tree rooted at s such that for any v in G, the set of ancestors of v in the tree is precisely the set of dominators of v. Thus, for any two vertices x and v in G, v becomes unreachable from s on failure of x if and

[*] This work was partially supported by Google India PhD Fellowship Award.

only if x is ancestor of v in the dominator tree. A lot of work has been done on computing dominators in optimal and near-optimal time, see [3, 10, 11, 12].

In this work, we focus on building efficient data structures for answering reachability queries upon two vertex failures. One simple solution to this problem is an $O(n)$ space and $O(n)$ query time data structure using the work of Baswana et. al [1] on fault tolerant reachability subgraphs (FTRS). They show that for every $k \geqslant 1$, we can compute in polynomial time a subgraph H of G with at most $2^k n$ edges that preserves the reachability information from s even after k edge/vertex failures. Such a subgraph is referred as a k-FTRS for G. Thus in this case a 2-FTRS serves as our data structure for answering reachability queries. To answer a reachability query after two vertex failures, we can run any standard graph traversal algorithm (BFS/DFS) from s in the 2-FTRS avoiding the failed vertices.

Another extreme solution to this problem is an $O(n^2)$ space data structure with $O(1)$ query time. This is possible by building n dominator trees, one for each graph $G \setminus \{y\}$, where $y \in V$. Now on failure of vertices x, y, a vertex v will be reachable from s if and only if x is not ancestor of v in the dominator tree of $G \setminus \{y\}$. To the best of our knowledge, these are the only known solutions for the dual failure case.

This brings us to the following central question of our work - Is it possible to achieve the best of the above two results? In other words, can we have an oracle of $O(n)$ space that still answers reachability queries in $O(1)$ time? We give an affirmative answer to this question. We also present a labeling scheme for this problem. Our results are summarized as follows:

Oracle: There exists a data structure of $O(n)$ size that given any two failing vertices f_1, f_2 and a query vertex v, takes $O(1)$ time to determine if v is reachable from s in $G \setminus \{f_1, f_2\}$.

Labeling Scheme: There exists a compact labeling scheme for answering reachability queries under two failures. Each vertex stores a label of $O(\log^3 n)$ bits such that for any two failing vertices f_1, f_2 and any destination vertex v, it is possible to determine whether v is reachable from s in $G \setminus \{f_1, f_2\}$ by processing the labels associated with f_1, f_2 and v only.

Our result also implies a data structure for the closely related problem of double dominator verification. A pair of vertices (x, y) is said to be *double-dominator* of a vertex v if each path from s to v contains either x or y, but none of x and y are dominators of v. Using our data structure together with the dominator-tree of Lengauer and Tarjan [12], one can obtain an $O(n)$ space data structure that for any given triplet $x, y, v \in V$ verifies in $O(1)$ time if (x, y) is double-dominator of v. The best previously known result for this could verify double-dominators only for a fixed s, v pair in $O(1)$ time using an $O(n)$ space data structure called dominator chain [16].

1.1 Related work

Demetrescu et al. [7] showed that any oracle for reporting distances from a single source, rather than just the reachability information upon vertex failures in a directed weighted graph must require $\Omega(m)$ space. Nonetheless, for the problem of reporting distances between an arbitrary pair of vertices, they give a construction of $O(n^2 \log n)$ size data structure that for any $u, v, x \in V$ reports the length of the shortest path from u to v avoiding x in constant time. Duan and Pettie [9] extended this result to dual failures by designing a data structure of $O(n^2 \log^3 n)$ space which could answer distance queries after any two vertex failures in an $O(\log n)$ time.

Parter and Peleg [14] addressed the problem of computing a sparse subgraph that preserves the distances from source s on failure of a single vertex. In particular, they show that for any given unweighted graph G we can compute a subgraph H with $O(n^{3/2})$ edges such that for any two vertices v, x, the distance of v from s in the graph $H \setminus \{x\}$ is the same as that in $G \setminus \{x\}$. They also show that this bound is tight. Parter [13] extended this result to dual failure in undirected graphs by showing an upper bound of $O(n^{5/3})$, and also showed that this bound is tight. Baswana and Khanna [2] showed that for the undirected and single failure case if one is willing to settle for an approximation then there is a subgraph with $O(n \log n)$ edges that preserves the distances up to a multiplicative error of 3. Parter and Peleg [15] improved this result and obtained a subgraph with at most $3n$ edges.

Another closely related problem is the replacement paths problem. In this problem we are given a source s and a target t and for each edge e on the shortest path from s to t the algorithm computes the shortest path from s to t avoiding e. Many variants of this problem were studied along the years. For a recent overview see [17] and references therein.

The questions of finding graph spanners, approximate distance oracles and compact routing schemes that are resilient to f vertex or edge failures in undirected graphs were studied in [8, 5, 4].

1.2 Our Techniques

Consider a reachability tree T rooted at source s. Let v be any vertex in T, and P be the path from s to v in T. Let us first consider the simple case when only a single vertex, say x, fails in G. If x lies on P and v is still reachable from s, then we can find an alternate path consisting of - a prefix of P, followed by a "detour", say D, avoiding P (and x), followed by a suffix of P. This simple decomposition can be used to easily handle reachability queries for one vertex failure. However, in the case of the dual failure, non trivialities arise when the second failing vertex lies on detour D. A natural direction to proceed from here is to define secondary detours which are disjoint from both P as well as D, but handling secondary detours is quite complicated.

So we take an alternative approach in which instead of initially starting with a single tree we begin with two independent trees - T_1 and T_2, defined by Georgiadis and Tarjan [11]. They satisfy the condition that for any v, the path from s to v in two trees intersect only at the dominators of v. This allows us to reduce the problem to the case when exactly one failure is an ancestor of v in T_1, and the other failure is ancestor of v in T_2. Now let P_1, P_2 be the paths from s to v in T_1, T_2, and let f_1, f_2 be the failed vertices lying respectively on P_1, P_2. Then, the structure of an alternate path to v gets simplified as follows: It consists of a prefix of either P_1 or P_2 up to a vertex a (lying respectively before f_1 or f_2) followed by a detour which is disjoint from P_1, P_2 (and f_1, f_2), and followed again by a suffix of P_1 or P_2. Note that the starting and terminating vertices of detour need not lie on the same tree-path.

So, we get a very clean and simple representation of detours. The main challenge lies in how to efficiently search for an appropriate detour avoiding f_1, f_2. The problems arising and how they are tackled is discussed in detail in Section 4.

1.3 Organization of the paper

We describe notations and terminologies in Section 2. In Section 3, we briefly sketch the solution for the single failure case using detours. The overview of the paper is presented in Section 4. In Section 5, we present the reachability oracle for a simpler class of graphs that are 2-vertex strongly connected. In Section 6, we present the oracle for general graphs. In Appendix, we provide the details of the labeling scheme.

2 Preliminaries

Given a directed graph $G = (V, E)$ on $n = |V|$ vertices and $m = |E|$ edges, and a source vertex $s \in V$, the following notations will be used throughout the paper.

- f_1, f_2: A given pair of failed vertices.
- $par_T(x)$: The parent of vertex x in tree T.
- $depth_T(x)$: The depth of vertex x in tree T.
- $\textsc{Path}_T(x, y)$: The path from vertex x to vertex y in tree T.
- $\textsc{Path}_T(\bar{x}, y)$: $\textsc{Path}_T(x, y) \setminus \{x\}$
- $\textsc{Path}_T(x, \bar{y})$: $\textsc{Path}_T(x, y) \setminus \{y\}$
- $\textsc{Path}_T(\bar{x}, \bar{y})$: $\textsc{Path}_T(x, y) \setminus \{x, y\}$
- $idom(x)$: The immediate dominator of vertex x. (See Definition 1).
- T_1, T_2: A pair of independent trees for G rooted at s. (See Definition 2).
- $P[x, y]$: The subpath of path P lying between vertices x, y, assuming x precedes y on P.
- $P{::}Q$: The path formed by concatenating paths P and Q in G. Here it is assumed that the last vertex of P is the same as the first vertex of Q.

We now define immediate dominators and independent spanning trees which are crucial to our fault tolerant reachability oracle.

▶ **Definition 1** ([12]). A vertex w is said to be the *immediate dominator* of v if (i) w is a dominator of v, and (ii) every dominator of v (other than v itself) is also a dominator of w.

▶ **Definition 2** (Georgiadis and Tarjan [11]). Given a directed graph G and a designated source s, a pair of trees T_1, T_2 rooted at s are said to be *independent spanning trees* if for each $v \neq s$ the paths from s to v in T_1 and T_2 intersect only at the dominators of v. (It was shown by Georgiadis and Tarjan [11] that such a pair of trees can be computed in an $O(m)$ time).

Below we state a few basic properties of dominators in a directed graph.

▶ **Property 1.** *Let T be a reachability tree rooted at s, and y_0, y_1 be vertices such that $y_0 = idom(y_1)$. Then for any $z \in \textsc{Path}_T(\bar{y}_0, y_1)$, $idom(z)$ belongs to $\textsc{Path}_T(y_0, y_1)$.*

▶ **Property 2.** *Let T be a reachability tree rooted at s, and y_1, y_2 be vertices such that y_1 is ancestor of y_2, and $idom(y_1) = idom(y_2)$. Then for any $z \in \textsc{Path}_T(y_1, y_2)$ either y_1 is a dominator of z or $idom(z) = idom(y_1)$.*

For efficient implementation of our oracle, we use the following optimal result by Demaine et. al [6] for answering the range minima queries on tree paths.

▶ **Theorem 3** (Demaine et al. [6]). *A tree T on n vertices and edge weights in the range $[0, n^3]$ can be preprocessed in $O(n \log n)$ time to build a data structure of $O(n)$ size so that given any $u, v \in T$, the edge of smallest weight on the tree path from u to v can be reported in $O(1)$ time.*

The following corollary is immediate from Theorem 3.

▶ **Corollary 4.** *Given a tree, say T, on n vertices, with each vertex assigned an integral weight in range $[0, n^3]$, we can obtain in polynomial time an $O(n)$ size data structure that for any two vertices x, y, outputs in $O(1)$ time the vertex with minimum weight on $\textsc{Path}_T(\bar{x}, y)$.*

3 Review of reachability oracle for single failure

In order to understand the dual fault tolerant reachability oracle we first briefly discuss the case of single failure. We here describe an alternative reachability oracle using detours instead of dominator tree. Let T be any reachability tree of G rooted at s, and f, v be respectively the failed and the query vertex. Also assume f is an ancestor of v in T. Notice that if v is reachable from s in $G \setminus \{f\}$, then there must exist a path starting from $\text{PATH}_T(s, \bar{f})$ and terminating at $\text{PATH}_T(\bar{f}, v)$ which, except for its endpoints, does not pass through any ancestor of v in T. (See Figure 1(i)). So for each $w \in V$, we can define a detour $D(w)$ to be a path starting from the highest possible ancestor of w in T and terminating at w such that none of the internal vertices of the path pass through an ancestor of w. Now on failure of f it suffices to search whether there exists a vertex lying in $\text{PATH}_T(\bar{f}, v)$ whose detour starts from an ancestor of f. This can be achieved by assigning to each vertex w a weight equal to the depth of the first vertex on $D(w)$. By doing this the problem of reachability under one vertex failure reduces to the problem of solving range minima on weighted trees, for which already an optimal solution exists. (See Corollary 4).

4 Overview

Let us consider the failure of a pair of vertices f_1, f_2 in G, and let v be the query vertex. Note that if any of the tree paths - $\text{PATH}_{T_1}(s, v)$ or $\text{PATH}_{T_2}(s, v)$ is intact, then v will be reachable from s. Also, if both $\text{PATH}_{T_1}(s, v)$ or $\text{PATH}_{T_2}(s, v)$ contains a common failed vertex, say f_1, then v will not be reachable from s. This is because then f_1 would be a dominator of v. Thus the non-trivial case is when $\text{PATH}_{T_1}(s, v)$ contains only f_1 and $\text{PATH}_{T_2}(s, v)$ contains only f_2, or the vice-versa. So whenever a query vertex v is given to us, we may assume that the following condition is satisfied.

$$\boxed{\mathcal{C} : f_1 \text{ lies on } \text{PATH}_{T_1}(s, v) \setminus \text{PATH}_{T_2}(s, v), \text{ and } f_2 \text{ lies on } \text{PATH}_{T_2}(s, v) \setminus \text{PATH}_{T_1}(s, v).}$$

Now consider the sets $S_A(v)$ and $S_B(v)$ as defined below. (For a better understanding of these sets see Figure 1(ii)).

- $S_A(v)$: Set of vertices lying either above f_1 on $\text{PATH}_{T_1}(s, v)$ or above f_2 on $\text{PATH}_{T_2}(s, v)$.
- $S_B(v)$: Set of vertices lying either below f_1 on $\text{PATH}_{T_1}(s, v)$ or below f_2 on $\text{PATH}_{T_2}(s, v)$.

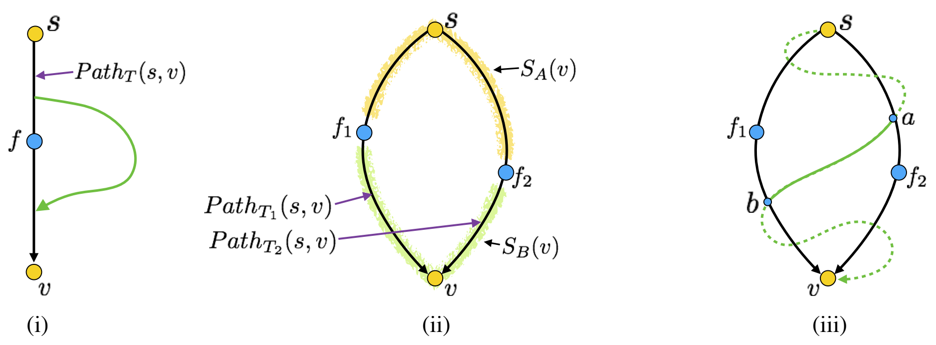

Figure 1 (i) Representation of a path bypassing f in the single vertex failure case; (ii) Representation of sets $S_A(v)$ and $S_B(v)$ when condition $\mathcal{C}$ is satisfied for vertex v; (ii) A path from $a \in S_A(v)$ to $b \in S_B(v)$ when v is reachable from s in $G \setminus \{f_1, f_2\}$.

It turns out that if v is reachable from s in $G \setminus \{f_1, f_2\}$, then there must exists a path from set $S_A(v)$ to $S_B(v)$ avoiding the vertices of both $\textsc{Path}_{T_1}(s, v)$ and $\textsc{Path}_{T_2}(s, v)$. This fact is formally stated in the following lemma.

▶ **Lemma 5.** *Given a pair of failed vertices f_1, f_2, a vertex v is reachable from s if and only if G contains a path P satisfying the following conditions. (See Figure 1(iii)).*

C1. The first and last vertices of P lies respectively in sets $S_A(v)$ and $S_B(v)$.

C2. None of the internal vertices of P lies on $\textsc{Path}_{T_1}(s, v)$ or $\textsc{Path}_{T_2}(s, v)$.

Proof. We first consider the case that v is reachable from s after the failure of f_1, f_2. Let Q be any path from s to v in $G \setminus \{f_1, f_2\}$. Let a be the last vertex on Q lying in the set $S_A(v)$, and b be the first vertex on $Q[a, v]$ that lies in the set $S_B(v)$. The vertices a, b are well defined since $s \in S_A(v)$ and $v \in S_B(v)$. It is easy to see that none of the intermediate vertices of path $Q[a, b]$ lies on $\textsc{Path}_{T_1}(s, v)$ and $\textsc{Path}_{T_2}(s, v)$. Hence $Q[a, b]$ is the required path which starts from a vertex in $S_A(v)$ and terminates to a vertex in $S_B(v)$.

Now to prove the converse let us assume that there exists a path, say P, from $a \in S_A(v)$ to $b \in S_B(v)$ such that none of the intermediate vertices of P lies on $\textsc{Path}_{T_1}(s, v)$ and $\textsc{Path}_{T_2}(s, v)$. Let $i, j \in [1, 2]$ be such that $a \in \textsc{Path}_{T_i}(s, f_i)$ and $b \in \textsc{Path}_{T_j}(f_j, v)$. Note that the paths $\textsc{Path}_{T_i}(s, a)$, $\textsc{Path}_{T_j}(b, v)$, and P cannot contain the failed vertices f_1 or f_2. Hence $\textsc{Path}_{T_i}(s, a)::P::\textsc{Path}_{T_j}(b, v)$ is a path from s to v in $G \setminus \{f_1, f_2\}$. ◀

For simplicity we refer to a path satisfying the conditions *C1* and *C2* stated in the above lemma as an $S_{A,B}(v)$ path. In order to efficiently compute such a path we define a pair of detours $D^1(w)$ and $D^2(w)$ for each vertex $w \in V$ as follows.

- $\mathrm{D}^i(w)$: a path starting from the highest possible ancestor of w in T_i and terminating at w such that none of the internal vertices of the path are ancestor of w in T_1 or T_2.

Note that the detours $D^1(w)$ and $D^2(w)$ can be seen as a simple generalization of the detour $D(w)$ which was defined for the single failure case in Section 3. However, we show that this simple generalization is not sufficient to answer the reachability queries in dual failure. To understand this subtle point consider an $S_{A,B}(v)$ path with a, b as its endpoints. If the endpoint b is equal to v, then P could be simply either $D^1(v)$ or $D^2(v)$. The problem arises when $b \neq v$. This is because if b is an ancestor of v in T_1, then P might contain vertices from $\textsc{Path}_{T_2}(s, b)$. (Recall that the internal vertices of P are disjoint from $\textsc{Path}_{T_2}(s, v)$, but not necessarily disjoint from $\textsc{Path}_{T_2}(s, b)$). So in this case P can neither be $D^1(b)$ nor be $D^2(b)$. For a more clear insight into this consider the graph and its two independent spanning trees in Figure 2. Since in-degree of each vertex in the graph is at most two, $\mathrm{D}^i(w)$ for each $w \in V$ is simply the incoming edge from $par_i(w)$ to w. Thus the path P connecting $S_A(v)$ to $S_B(v)$ is a concatenation of as many detours as there are number of edges in P. Determining whether a concatenation of all these single-edge detours can give us an $S_{A,B}(v)$ path is difficult to achieve in $O(1)$ time.

This shows that a simple generalization of detours from a single tree to two trees is not sufficient. To tackle the problem we extend the notion of detours to 'Parent Detours' and 'Ancestor Detours'. These detours unlike the normal detour terminates at an appropriate ancestor of w in T_1 or T_2. We formally define the parent-detours and ancestor-detours in the following sections, and show how they can be used to solve the problem of dual fault tolerant reachability.

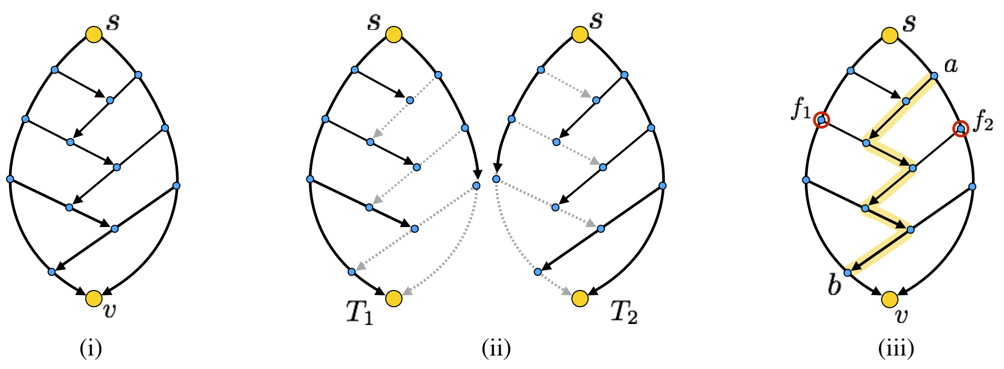

Figure 2 (i) A graph G with in-degree of each vertex bounded by two; (ii) A pair of independent spanning trees T_1 and T_2 for G; (iii) A path P from $a \in S_A(v)$ to $b \in S_B(v)$ in $G \setminus \{f_1, f_2\}$.

5 Reachability oracle for 2-vertex strongly connected graphs

In this section we describe an $O(n)$ space and $O(1)$ time reachability oracle for 2-vertex strongly connected graphs. By 2-vertex strongly connectedness we have that on removal of any vertex f ($f \neq s$), all the vertices in $G \setminus \{f\}$ are still reachable from s. Thus each vertex is dominated only by source s and by itself. This implies that for any vertex w, $\textsc{Path}_{T_1}(s, w)$ and $\textsc{Path}_{T_2}(s, w)$ intersects only at the endpoints s, w.

Consider a query vertex v which is reachable from s in $G \setminus \{f_1, f_2\}$. Let us assume that condition $\mathcal{C}$ is satisfied for v. Let P be any $S_{A,B}(v)$ path, and a, b be respectively the first and last vertices on P. Without loss of generality we can assume that b lies on $\textsc{Path}_{T_1}(s, v)$. See Figure 3(i). We make the following additional assumptions.

1. None of the $S_{A,B}(v)$ paths terminates at v (i.e. b cannot be v).
2. b is the lowest vertex on $\textsc{Path}_{T_1}(s, v)$ at which an $S_{A,B}(v)$ path terminates.

▶ Remark. The assumption 1 is justified since if $b = v$, then v will be reachable from s using the detours $D^1(v)$ or $D^2(v)$.

We now state a lemma which provides the motivation for defining the parent detours.

▶ **Lemma 6.** *Let a, b, P be as described above, and c be the child of b on $\textsc{Path}_{T_1}(s, v)$. Then,*
(i) Vertex f_2 is an ancestor of c in T_2.
(ii) None of the internal vertices of P lie on $\textsc{Path}_{T_1}(s, c)$ or $\textsc{Path}_{T_2}(s, c)$.

Proof. Let z denote the LCA of vertices c and v in tree T_2. (See Figure 3(ii)). Consider the path $Q = \textsc{Path}_{T_2}(z, c)$. It is easy to see that none of the internal vertices of Q lies on $\textsc{Path}_{T_2}(s, v)$. Also, the internal vertices of Q appearing on $\textsc{Path}_{T_1}(s, v)$ must lie below c on $\textsc{Path}_{T_1}(s, v)$. This is because, by definition of independent spanning trees, $\textsc{Path}_{T_1}(s, c)$ and Q can intersect only at the vertices s and c.

We now prove claim 1. Let d be the first point of intersection of Q with $\textsc{Path}_{T_1}(c, v)$. (See Figure 3(ii)). Then the internal vertices of $Q[z, d]$ are disjoint from both $\textsc{Path}_{T_1}(s, v)$ and $\textsc{Path}_{T_2}(s, v)$. Now if z is an ancestor of f_2 in T_2, then $Q[z, d]$ forms an $S_{A,B}(v)$ path, terminating at descendant of b in T_1, thereby violating assumption 2. Hence f_2 must be either same as z or an ancestor of z. This shows that f_2 is an ancestor of c in T_2.

In order to prove claim 2, we first show that P is disjoint from Q. Let us suppose on the contrary, that there exists a vertex, say z', belonging to $P \cap Q$. Also let d' be the first vertex of $Q[z', c]$ lying on $\textsc{Path}_{T_1}(c, v)$. (See Figure 3(iii)). Then $P[a, z']::Q[z', d']$ forms an

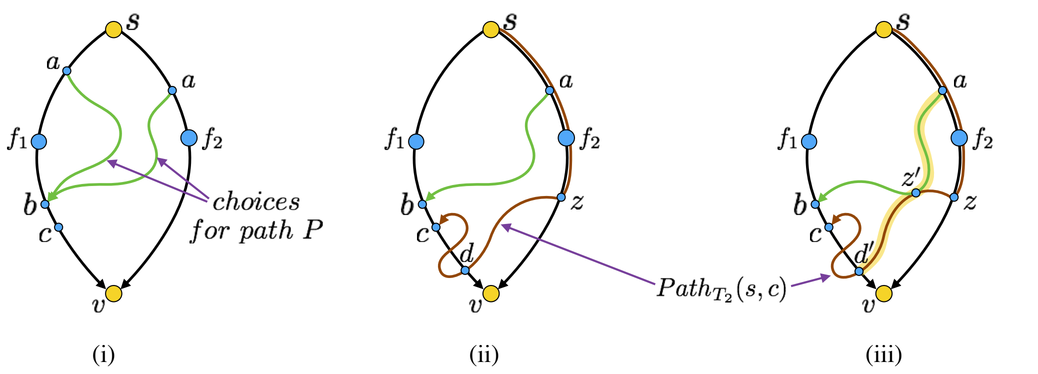

Figure 3 (i) Possibilities for path P when $b \in \textsc{Path}_{T_1}(\bar{f}_1, \bar{v})$; (ii) Representation of $\textsc{Path}_{T_2}(s,c)$ and $z = LCA(c,v)$ in T_2; (iii) Violation of assumption 2 if $P \cap \textsc{Path}_{T_2}(z,c)$ is non-empty.

$S_{A,B}(v)$ path terminating at a descendant of b in T_1. This again violates assumption 2. Thus $P \cap Q = \emptyset$. Now since the internal vertices of P are disjoint from $\textsc{Path}_{T_1}(s,v), \textsc{Path}_{T_2}(s,v)$, they must be disjoint from $\textsc{Path}_{T_1}(s,c)$ and $\textsc{Path}_{T_2}(s,z)::Q = \textsc{Path}_{T_2}(s,c)$, as well. ◀

The above lemma implies that f_1 is an ancestor of c in T_1, and f_2 is an ancestor of c in T_2. Thus $S_A(v) = S_A(c)$. Hence we have the following corollary.

▶ **Corollary 7.** *P is an $S_{A,B}(c)$ path terminating at $par_{T_1}(c)$.*

In order to capture the above fact we define parent-detours for each $w \in V$ which instead of terminating at w terminates at either $par_{T_1}(w)$ or $par_{T_2}(w)$.

- $\mathrm{PD}^i_j(w)$: a path starting from the highest possible ancestor of w in T_i and terminating at $par_{T_j}(w)$ s.t. none of the internal vertices of the path lie on $\textsc{Path}_{T_1}(s,w)$ or $\textsc{Path}_{T_2}(s,w)$.

By above definition of parent-detour it follows that P can be replaced by either $\mathrm{PD}^1_1(c)$ or $\mathrm{PD}^2_1(c)$ depending upon whether it starts from an ancestor of c in T_1 or T_2. Now let x_1 denote the child of f_1 in T_1 lying on $\textsc{Path}_{T_1}(s,v)$, and x_2 denote the child of f_2 in T_2 lying on $\textsc{Path}_{T_2}(s,v)$. Then the parent-detours of vertices x_1, x_2 may not be of any help, since they would terminate at f_1 and f_2. However, the parent-detours of vertices in $S_B(v) \setminus \{x_1, x_2\}$ will suffice to determine whether v is reachable from s or not.

Notice that in above discussion, we observed that P is an $S_{A,B}(c)$ path terminating at $b = par_{T_1}(c)$. This shows that for vertices lying on $\textsc{Path}_{T_1}(\bar{x}_1, v)$, we only need to worry about parent-detours terminating at $par_{T_1}(\cdot)$, i.e. $\mathrm{PD}^1_1(\cdot)$ and $\mathrm{PD}^2_1(\cdot)$. Whereas, for vertices on $\textsc{Path}_{T_2}(\bar{x}_2, v)$, we need to worry about parent-detours terminating at $par_{T_2}(\cdot)$, i.e. $\mathrm{PD}^1_2(\cdot)$ and $\mathrm{PD}^2_2(\cdot)$. We thus have the following lemma.

▶ **Lemma 8.** *Let x_1 and x_2 be as defined above. A vertex v is reachable from s in $G \setminus \{f_1, f_2\}$ if and only if at least one of the following vertices lie in $S_A(v)$.*

(i) The first vertex of $\mathrm{D}^1(v)$ or $\mathrm{D}^2(v)$.
(ii) The first vertex of either $\mathrm{PD}^1_1(w)$ or $\mathrm{PD}^2_1(w)$ for some $w \in \textsc{Path}_{T_1}(\bar{x}_1, v)$.
(iii) The first vertex of either $\mathrm{PD}^1_2(w)$ or $\mathrm{PD}^2_2(w)$ for some $w \in \textsc{Path}_{T_2}(\bar{x}_2, v)$.

5.1 Implementation of the oracle

We first introduce the following notations for detours and parent detours.

- $\beta^i(v)$: $depth_{T_i}$(first vertex on $\mathrm{D}^i(v)$).
- $\gamma^i_j(v)$: $depth_{T_i}$(first vertex on $\mathrm{PD}^i_j(v)$).

Now let f_1, f_2 be a given pair of failed vertices and v be a given query vertex. Our first step is to check if condition $\mathcal{C}$ is satisfied. Recall that this requires only verifying the ancestor-descendant relationship in trees T_1 and T_2. One simple method to achieve this for any given tree T is to perform the pre-order and the post-order traversal of T, and store the vertices in the order they are visited. Now x will be ancestor of y in T if and only if x appears before y in the pre-order traversal, and after y in the post-order traversal.

Algorithm 1 presents the pseudo-code for answering reachability query for a vertex v assuming condition $\mathcal{C}$ is satisfied. This can be explained in words as follows. For $i = 1, 2$, we first check if $D^i(v)$ starts from an ancestor of f_i in T_i or not. This is done by comparing the value of $\beta^i(v)$ with the depth of f_i in T_i. Next we compute the vertices x_1, x_2. Finally for $i, j \in \{1, 2\}$, we compute a vertex $w \in \textsc{Path}_{T_j}(\bar{x}_j, v)$ for which $\gamma^i_j(\cdot)$ is minimum. If $\gamma^i_j(w)$ is less than the depth of f_i in T_i, then it implies that $\mathrm{PD}^i_j(w)$ starts from an ancestor of f_i in T_i, so we return True. If we reach to the end of code, that means we have not been able to find any path for v, so we return False.

Algorithm 1: Oracle for reachability to v in 2-vertex strongly connected graphs.

```
1 if $\beta^1(v) < depth_{T_1}(f_1)$ or $\beta^2(v) < depth_{T_2}(f_2)$ then Return True;
2 $x_1 \leftarrow$ the vertex with minimum depth on $\textsc{Path}_{T_1}(\bar{f}_1, v)$;
3 $x_2 \leftarrow$ the vertex with minimum depth on $\textsc{Path}_{T_2}(\bar{f}_2, v)$;
4 foreach $i, j \in \{1, 2\}$ do
5     $w \leftarrow$ a vertex on $\textsc{Path}_{T_j}(\bar{x}_j, v)$ for which $\gamma^i_j(\cdot)$ is minimum;
6     if $\gamma^i_j(w) < depth_{T_i}(f_i)$ then Return True;
7 end
8 Return False;
```

The above oracle can be easily implemented in $O(1)$ time, by having a total of six weight functions - one each for storing the depth of a vertex in trees T_1, T_2, and the other four for storing the values $\gamma^i_j(\cdot)$, for $i, j \in \{1, 2\}$. By doing this the vertices x_1, x_2 can be computed in constant time since they are respectively the vertices with minimum depth on the paths $\textsc{Path}_{T_1}(\bar{f}_1, v)$ and $\textsc{Path}_{T_1}(\bar{f}_1, v)$. Also Step 4 can be carried out in an $O(1)$ time. We can thus state the following theorem.

▶ **Theorem 9.** *A 2-vertex strongly connected graph on n vertices can be preprocessed in polynomial time for a given source vertex s to build a data structure of $O(n)$ size such that for any query vertex v, and pair of failures f_1, f_2, it takes $O(1)$ time to determine if there exists any path from s to v in $G \setminus \{f_1, f_2\}$.*

6 Reachability oracle for general graphs

In this section we explain the reachability oracle for general graphs. Consider a query vertex u in G. Let $u_0, u_1, ..., u_k$ be the dominators of u with $u_0 = s$ and $u_k = u$. Thus $\textsc{Path}_{T_1}(s, u)$ and $\textsc{Path}_{T_2}(s, u)$ intersect only at u_i's. (See Figure 4(i)). As in Section 5, we assume that condition $\mathcal{C}$ holds for u, so none of the u_i's can be equal to f_1 or f_2. Now let $i, j \in [1, k]$ be such that $f_1 \in \textsc{Path}_{T_1}(\bar{u}_{i-1}, \bar{u}_i)$ and $f_2 \in \textsc{Path}_{T_2}(\bar{u}_{j-1}, \bar{u}_j)$. It is easy to see that if $i \neq j$, then u is reachable from s by the path $\textsc{Path}_{T_1}(s, u_{i-1})::\textsc{Path}_{T_2}(u_{i-1}, u_i)::\textsc{Path}_{T_1}(u_i, u)$.

(See Figure 4(ii)). Thus we consider the case when $i = j$ [1]. For simplicity, we use symbols, v and $idom(v)$ to respectively denote the vertices u_i and u_{i-1}. Notice that in order to check reachability of u from s, it suffices to check if v is reachable from s in $G \setminus \{f_1, f_2\}$.

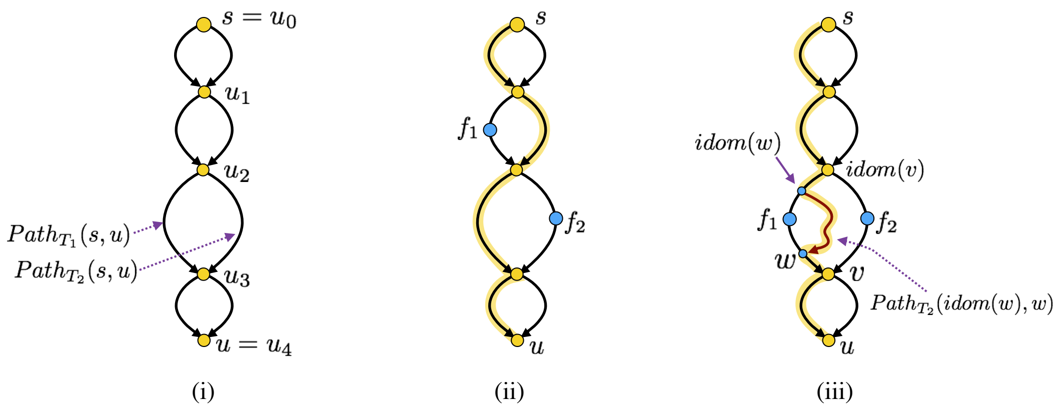

Figure 4 (i) Representation of dominators of u; (ii) A path from s to u (highlighted in yellow) when f_1 lies on $\text{PATH}_{T_1}(\bar{u}_1, \bar{u}_2)$ and f_2 lies on $\text{PATH}_{T_2}(\bar{u}_2, \bar{u}_3)$; (iii) A path from s to u (highlighted in yellow) when there exists a vertex $w \in S_B(v)$ for which $idom(w)$ lies in the set $S_A(v) \setminus \{idom(v)\}$.

We now divide our analysis into various different cases as follows:

Case 1. *There exists an $S_{A,B}(v)$ path terminating at vertex v.*
In this case v will be reachable from s using either of the detours $D^1(v)$ or $D^2(v)$.

Case 2. *There exists a vertex $w \in S_B(v)$ for which $idom(w) \in S_A(v) \setminus \{idom(v)\}$.*
In this case also we can show that v is reachable from s by the following argument. Without loss of generality let us assume that w is an ancestor of v in T_1. Since $idom(w)$ is an ancestor of w in T_1, it must lie on $\text{PATH}_{T_1}(\overline{idom(v)}, \bar{f}_1)$. (See Figure 4(iii)). Consider the path $Q = \text{PATH}_{T_2}(idom(w), w)$. Note that f_2 cannot lie on Q. This is because otherwise $\text{PATH}_{T_1}(idom(v), w)$ and $\text{PATH}_{T_2}(idom(v), f_2)::Q[f_2, w]$ will form two vertex disjoint paths from $idom(v)$ to w, which would violate the fact that $idom(w) \neq idom(v)$. Also f_1 cannot lie on Q, as Q is disjoint from $\text{PATH}_{T_1}(\overline{idom(w)}, \bar{w})$. Thus v is reachable from s by the path $\text{PATH}_{T_1}(s, idom(w))::Q::\text{PATH}_{T_1}(w, v)$.

Case 3. *None of the $S_{A,B}(v)$ path terminates at v, and there does not exist a vertex in $S_B(v)$ whose immediate dominator lies in $S_A(v) \setminus \{idom(v)\}$.*
This is the most non-trivial case of dual fault tolerant reachability oracle. We now provide analysis for this case.

Let us suppose v is reachable from s in $G \setminus \{f_1, f_2\}$. Then without loss of generality we can assume that there exists an $S_{A,B}(v)$ path (say P) terminating at an ancestor of v in T_1. In case there are multiple $S_{A,B}(v)$ paths, then we take P to be that path which terminates at lowest vertex on $\text{PATH}_{T_1}(\bar{f}_1, \bar{v})$. Let a, b be respectively the first and last vertices on P. By Property 1, we know that $idom(b)$ cannot be an ancestor of $idom(v)$ in T_1. Therefore, $idom(b)$ must be equal to $idom(v)$. This is because $idom(b)$ cannot lie in $S_A(v) \setminus \{idom(v)\}$,

[1] One can verify in $O(1)$ time whether $u_{i-1} = u_{j-1}$ (i.e. if $i = j$) since vertex $u_{i-1} = LCA(f_1, v)$ and vertex $u_{j-1} = LCA(f_2, v)$ in the dominator tree of G.

and if $idom(b)$ lies in $S_B(v)$ then $P \cap S_B(v)$ will contain both b and $idom(b)$, which would violate the definition of an $S_{A,B}(v)$ path.

Now consider the vertex c which is child of b on $\textsc{Path}_{T_1}(s,v)$. It turns out that in a general graph the parent-detours of c may not be of any help. This is because the analysis for 2-vertex strongly connected graphs crucially exploited the fact that $idom(b) = idom(c) = s$. But in general graphs, if $idom(b)$ is not equal to $idom(c)$, then it can be shown that Lemma 6 no longer holds. To be more precise, we can show that the internal vertices of P might not be disjoint from $\textsc{Path}_{T_2}(s,c)$.

However, the problem can be resolved if we take c to be the first descendant of b on $\textsc{Path}_{T_1}(s,v)$ whose immediate dominator is the same as that of b. This motivates us to define the notion of pseudo-child (and pseudo-parent) as follows.

▶ **Definition 10.** Given a reachability tree T rooted at s, a vertex x is said to be pseudo-parent of y in T (and y is said to be pseudo-child of x) if x is the nearest ancestor of y in T whose immediate dominator is the same as that of y.

Note that in a 2-vertex strongly connected graph, the definition of pseudo-parent and pseudo-child degenerates to normal notion of parent and child. This is because, immediate dominator of all the vertices (other than s) in such graph is equal to s.

We now state a lemma which is an analogue of Lemma 6 for general graphs.

▶ **Lemma 11.** *Let a, b, P be as described above, and c be the $\underline{\textit{pseudo-child}}$ of b on $\textsc{Path}_{T_1}(s,v)$. Then,*
(i) Vertex f_2 is an ancestor of c in T_2.
(ii) None of the internal vertices of P lie on $\textsc{Path}_{T_1}(s,c)$ or $\textsc{Path}_{T_2}(s,c)$.

As a corollary of the above lemma we get that P is an $S_{A,B}(c)$ path terminating at pseudo-parent of c in T_1. We thus define ancestor-detours which are a generalization of parent-detours as follows.

- $\mathrm{AD}^i_j(w)$: a path starting from the highest possible ancestor of w in T_i and terminating at pseudo-parent of w in tree T_j such that none of the internal vertices of the path lie on $\textsc{Path}_{T_1}(s,w)$ or $\textsc{Path}_{T_2}(s,w)$.

Now let x_1 be the first descendant of f_1 on $\textsc{Path}_{T_1}(s,v)$ whose immediate dominator is equal to $idom(v)$. Similarly, let x_2 be the first descendant of f_2 on $\textsc{Path}_{T_1}(s,v)$ whose immediate dominator is equal to $idom(v)$. Then the ancestor-detours of x_1, x_2 will not be of any help as they would terminate at either f_1, f_2 or their ancestors. Now as in Section 5, we can argue that ancestor-detours of vertices on $\textsc{Path}_{T_1}(\bar{x}_1, v) \cup \textsc{Path}_{T_2}(\bar{x}_2, v)$ suffice to answer the reachability query for vertex v. This completes the analysis of the third case.

We thus have the following lemma.

▶ **Lemma 12.** *Let v be a vertex satisfying condition $\mathcal{C}$ such that $f_1 \in \textsc{Path}_{T_1}(\overline{idom(v)}, v)$ and $f_2 \in \textsc{Path}_{T_1}(\overline{idom(v)}, v)$. Also let x_1 and x_2 be as defined above. Then v is reachable from s in $G \setminus \{f_1, f_2\}$ if and only if either of the following statements holds true.*

(i) [Case 1] The first vertex of $\mathrm{D}^1(v)$ or $\mathrm{D}^2(v)$ lies in $S_A(v)$.
(ii) [Case 2] There exists a vertex $w \in S_B(v)$ for which $idom(w) \in S_A(v) \setminus \{idom(v)\}$.
(iii) [Case 3] There exists a vertex $w \in \textsc{Path}_{T_1}(\bar{x}_1, v)$ such that $idom(w) = idom(v)$ and the first vertex of either $\mathrm{AD}^1_1(w)$ or $\mathrm{AD}^2_1(w)$ lies in $S_A(v)$.
(iv) [Case 3] There exists a vertex $w \in \textsc{Path}_{T_2}(\bar{x}_2, v)$ such that $idom(w) = idom(v)$ and the first vertex of either $\mathrm{AD}^1_2(w)$ or $\mathrm{AD}^2_2(w)$ lies in $S_A(v)$.

6.1 Implementation of the oracle

We now explain the implementation of reachability oracle for general graphs. As in Section 5, we define the following notations.

- $\alpha^i(v)$: $depth_{T_i}(idom(v))$.
- $\beta^i(v)$: $depth_{T_i}(\text{first vertex on } \mathrm{D}^i(v))$.
- $\gamma^i_j(v)$: $depth_{T_i}(\text{first vertex on } \mathrm{AD}^i_j(v))$.

Let f_1, f_2 be a given pair of failed vertices and v be a given query step. We assume that condition $\mathcal{C}$ is satisfied, and failures f_1, f_2 lies respectively on $\textsc{Path}_{T_1}(idom(v), v)$ and $\textsc{Path}_{T_2}(idom(v), v)$. We first check for $i = 1, 2$, if $D^i(v)$ starts from an ancestor of f_i in T_i or not. This is done by comparing the value of $\beta^i(v)$ with the depth of f_i in T_i.

Next we compute the vertices x_1, x_2 as follows. Recall that x_1 is the highest ancestor of v in $\textsc{Path}_{T_1}(\bar{f}_1, v)$ whose immediate dominator is equal to $idom(v)$. So to obtain x_1, we call the range minima query for vertices on $\textsc{Path}_{T_1}(\bar{f}_1, v)$ with $\langle \alpha^1(\cdot), depth_{T_1}(\cdot) \rangle$ as the weight function. By comparing the value of $\alpha^1(\cdot)$, it is able to filter out those vertices in $\textsc{Path}_{T_1}(\bar{f}_1, v)$ whose immediate dominator is at minimum depth in T_1, i.e. it is equal to $idom(v)$. After this it assigns x_1 to be that vertex which has minimum depth in T_1. Vertex x_2 is computed in a similar manner.

Now notice that to find whether there exists a vertex in $S_B(v)$ whose immediate dominator lies in $S_A(v) \setminus \{idom(v)\}$, we only need to restrict ourself to paths $\textsc{Path}_{T_1}(\bar{f}_1, \bar{x}_1)$ and $\textsc{Path}_{T_1}(\bar{f}_1, \bar{x}_1)$. This is because Property 2 implies that immediate dominator of vertices in $\textsc{Path}_{T_1}(x_1, v)$ is either equal to $idom(v)$ or lies in $\textsc{Path}_{T_1}(x_1, v)$ itself. Similarly, for $\textsc{Path}_{T_2}(x_2, v)$. So for $i = 1, 2$, we perform the range minima query to find a vertex, say w, on $\textsc{Path}_{T_i}(\bar{f}_i, \bar{x}_i)$ for which $\alpha^i(w)$ is minimum. If $\alpha^i(w)$ is less than the depth of f_i in T_i, then we report that v is reachable from s.

Finally for $i, j \in \{1, 2\}$, we compute a vertex $w \in \textsc{Path}_{T_j}(\bar{x}_j, v)$ for which $\langle \alpha^i(\cdot), \gamma^i_j(\cdot) \rangle$ is minimum. The term $\alpha^i(\cdot)$ is added in front so that we are able to filter out those vertices whose immediate dominator is equal to $idom(v)$. Now if $\gamma^i_j(w)$ is less than the depth of f_i in T_i, then it implies that $\mathrm{AD}^i_j(w)$ starts from an ancestor of f_i in T_i, so we return True.

If we reach to the end of code, that means we have not been able to find any path for v, so we return False.

Algorithm 2: Oracle for reachability to v in general graphs.

```
1  if $\beta^1(v) < depth_{T_1}(f_1)$ or $\beta^2(v) < depth_{T_2}(f_2)$ then Return True;
2  $x_1 \leftarrow$ a vertex on $\textsc{Path}_{T_1}(\bar{f}_1, v)$ for which $\langle \alpha^1(\cdot), depth_{T_1}(\cdot) \rangle$ is minimum;
3  $x_2 \leftarrow$ a vertex on $\textsc{Path}_{T_2}(\bar{f}_2, v)$ for which $\langle \alpha^2(\cdot), depth_{T_2}(\cdot) \rangle$ is minimum;
4  foreach $i \in \{1, 2\}$ do
5      $w \leftarrow$ a vertex on $\textsc{Path}_{T_i}(\bar{f}_i, \bar{x}_i)$ for which $\alpha^i(\cdot)$ is minimum;
6      if $\alpha^i(w) < depth_{T_i}(f_i)$ then Return True;
7  end
8  foreach $i, j \in \{1, 2\}$ do
9      $w \leftarrow$ a vertex on $\textsc{Path}_{T_j}(\bar{x}_j, v)$ for which $\langle \alpha^i(\cdot), \gamma^i_j(\cdot) \rangle$ is minimum;
10     if $\gamma^i_j(w) < depth_{T_i}(f_i)$ then Return True;
11 end
12 Return False;
```

As in Algorithm 1, we can argue that the Steps 2, 3, 5 and 9, can be implemented in $O(1)$ time. Thus Algorithm 2 takes constant time to answer reachability queries. We thus conclude with the following theorem.

▶ **Theorem 13.** *A directed graph $G = (V, E)$ on n vertices can be preprocessed in polynomial time for a given source vertex $s \in V$ to build a data structure of $O(n)$ size such that for any $f_1, f_2, v \in V$, it takes $O(1)$ time to determine if there exists a path from s to v in $G \setminus \{f_1, f_2\}$.*

Acknowledgements. I am very grateful to my advisor, Prof. Surender Baswana, for many helpful discussions and for reviewing this paper.

References

1. Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault tolerant subgraph for single source reachability: Generic and optimal. In *Proceedings of the Forty-Eigth Annual ACM Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 19-21, 2016 (to appear)*.
2. Surender Baswana and Neelesh Khanna. Approximate shortest paths avoiding a failed vertex: Near optimal data structures for undirected unweighted graphs. *Algorithmica*, 66(1):18–50, 2013.
3. Adam L. Buchsbaum, Loukas Georgiadis, Haim Kaplan, Anne Rogers, Robert Endre Tarjan, and Jeffery Westbrook. Linear-time algorithms for dominators and other path-evaluation problems. *SIAM J. Comput.*, 38(4):1533–1573, 2008.
4. Shiri Chechik. Fault-tolerant compact routing schemes for general graphs. *Inf. Comput.*, 222:36–44, 2013.
5. Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. f-sensitivity distance oracles and routing schemes. *Algorithmica*, 63(4):861–882, 2012.
6. Erik D. Demaine, Gad M. Landau, and Oren Weimann. On cartesian trees and range minimum queries. *Algorithmica*, 68(3):610–625, 2014.
7. Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008.
8. Michael Dinitz and Robert Krauthgamer. Fault-tolerant spanners: better and simpler. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, pages 169–178, 2011.
9. Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In *SODA'09: Proceedings of 19th Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 506–515, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.
10. Wojciech Fraczak, Loukas Georgiadis, Andrew Miller, and Robert Endre Tarjan. Finding dominators via disjoint set union. *J. Discrete Algorithms*, 23:2–20, 2013.
11. Loukas Georgiadis and Robert Endre Tarjan. Dominators, directed bipolar orders, and independent spanning trees. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, pages 375–386, 2012.
12. Thomas Lengauer and Robert Endre Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Program. Lang. Syst.*, 1(1):121–141, 1979.
13. Merav Parter. Dual failure resilient BFS structure. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 481–490, 2015.

14 Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 779–790, 2013.

15 Merav Parter and David Peleg. Fault tolerant approximate BFS structures. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1073–1092, 2014.

16 Maxim Teslenko and Elena Dubrova. An efficient algorithm for finding double-vertex dominators in circuit graphs. In *2005 Design, Automation and Test in Europe Conference and Exposition (DATE 2005), 7-11 March 2005, Munich, Germany*, pages 406–411, 2005.

17 Virginia Vassilevska Williams. Faster replacement paths. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1337–1346, 2011.

A Appendix

In this section we describe the labeling scheme for answering reachability queries from s to any vertex v on two vertex failures. In the first subsection we develop a labeling scheme for reporting minima and second minima on tree paths. In the next subsection, we see how these labels can be used to obtain a labeling scheme for answering reachability queries.

A.1 Labeling scheme for minima and second minima on tree paths

We assume that a reachability tree T rooted at s is given to us, and each vertex x is assigned a positive integral weight $wt(x)$ in the range $[0, n^2]$. Note that second-minima may have two interpretations depending upon whether or not the second minimum value is allowed to be the same as the first one. In this paper, by second-minimum we always mean the smallest value obtained after removing all occurrence of the first-minimum value. The first minimum and second minimum operations are respectively denoted by $\min_1(\cdot)$ and $\min_2(\cdot)$. Now given any two vertices x, y, where x is ancestor of y in T, we define the following notations.

- $\textsc{fm}(x,y) := \min_1\{wt(z) \mid z \in \textsc{Path}_T(\bar{x}, y)\}$.
- $\textsc{sm}(x,y) := \min_2\{wt(z) \mid z \in \textsc{Path}_T(\bar{x}, y)\}$.

Our aim is to develop compact labeling scheme for reporting $\textsc{fm}(x,y)$ and $\textsc{sm}(x,y)$ described above. In order to achieve this we first do a heavy-light path decomposition of T. Let $\mathcal{P}$ be the collection of paths obtained by this decomposition. For each vertex x, we use $P(x)$ to denote the path in $\mathcal{P}$ containing x. Now for each vertex x and each integer $i \in [1, \log n]$, we define the following notations.

- $\textsc{fm}_a(x,i) := \min_1\{wt(y) \mid y \text{ lies } \underline{\text{a}}\text{bove } x \text{ in } P(x) \text{ and } |depth_T(x) - depth_T(y)| \le 2^i\}$.
- $\textsc{sm}_a(x,i) := \min_2\{wt(y) \mid y \text{ lies } \underline{\text{a}}\text{bove } x \text{ in } P(x) \text{ and } |depth_T(x) - depth_T(y)| \le 2^i\}$.

Similarly, we define the notations $\textsc{fm}_b(x,i)$ and $\textsc{sm}_b(x,i)$. For each vertex x, let $\delta(x)$ denote the string of $O(\log^2 n)$ bits obtained by concatenation of: (i) $depth_T(x)$, (ii) $wt(x)$, and (iii) $\textsc{fm}_a(x,i), \textsc{sm}_a(x,i), \textsc{fm}_b(x,i), \textsc{sm}_b(x,i)$ for each $i \in [1, \log n]$.

- $\delta(x) := \langle depth_T(x), wt(x), \textsc{fm}_a(x,i), \textsc{sm}_a(x,i), \textsc{fm}_b(x,i), \textsc{sm}_b(x,i)\ \forall\ i \in [1, \log n]\rangle$.

Note that if x is ancestor of y, and $P(x) = P(y)$, then $\textsc{fm}(x,y)$ and $\textsc{sm}(x,y)$ can be computed by just looking at the strings $\delta(x)$ and $\delta(y)$ as follows. We first extract out

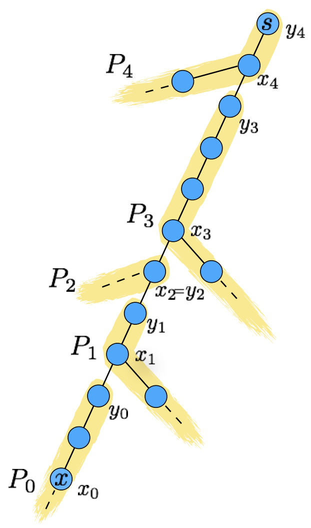

Figure 5 Heavy-Light Path Decomposition: Representation of paths $P_0, P_1, .., P_t$ where $P_0 = P(x)$ and $P_1, .., P_t$ are paths in $\mathcal{P}$ starting from an ancestor of x. Note that t will be bounded by $\log n$.

$depth_T(x)$, $depth_T(y)$ from $\delta(x), \delta(y)$ respectively. Now let k be the greatest integer such that $2^k \leq depth_T(y) - depth_T(x) - 1$. Then,

$$\textsc{fm}(x, y) = \min\nolimits_1\{wt(y), \textsc{fm}_b(x, k), \textsc{fm}_a(y, k)\}, \text{ and} \tag{1}$$

$$\textsc{sm}(x, y) = \min\nolimits_2\{wt(y), \textsc{fm}_b(x, k), \textsc{fm}_a(y, k), \textsc{sm}_b(x, k), \textsc{sm}_a(y, k)\} \tag{2}$$

For the case when x, y belongs to different paths in $\mathcal{P}$, observe that if x is ancestor of y, then $\textsc{Path}_T(x, y)$ can be obtained by joining together at most $\log n$ different paths from $\mathcal{P}$.

Consider any vertex $x \in T$. Let $P_0 = P(x)$, and $P_1, .., P_t$ be the paths in $\mathcal{P}$ that starts from an ancestor of x. Also for $i = 1$ to t, let x_i, y_i be the vertices in $\textsc{Path}_T(s, x) \cap P_i$ of respectively the maximum and the minimum depth value. (See Figure 5). By definition of heavy-light path decomposition we have that t must be bounded by $\log n$. We define label of x, denoted by $\ell(x)$, as the string of $O(\log^3 n)$ bits obtained by concatenating together all $\delta(x_i)$'s and $\delta(y_i)$'s, where i ranges from 0 to t.

- $\ell(x) := \langle \delta(x_i), \delta(y_i) \ \forall\ i \in [0, t] \rangle$, where x_i, y_i and t are as defined above.

Now, let z, x be a pair of query vertices such that z is ancestor of x. We give below the steps for computing $FM(z, x)$ and $SM(z, x)$ by using labels $\ell(x)$ and $\ell(z)$.

1. Let $W = \emptyset$.
2. Compute t = number of paths in $\mathcal{P}$ by scanning the string $\ell(x)$. Let $P_0, P_1, .., P_t$ denote these paths, and x_i, y_i's be the corresponding vertices.
3. Repeat for $i = 0$ to t.
   – Add $wt(x_i)$ to set W.
   – If $depth_T(x_i) \leq depth_T(z) \leq depth_T(y_i)$, i.e. $z \in \textsc{Path}_T(y_i, x_i)$, then compute $\textsc{fm}(z, x_i)$ and $\textsc{sm}(z, x_i)$ using Eqns. 1, 2 and add them to W. Also break the loop.
   – If $z \notin \textsc{Path}_T(y_i, x_i)$, then compute $\textsc{fm}(y_i, x_i)$ and $\textsc{sm}(y_i, x_i)$ using Eqns. 1, 2 and add them to W.
4. Return $\textsc{fm}(z, x) = min_1(W)$ and $\textsc{sm}(z, x) = min_2(W)$.

We conclude with the following theorem.

▶ **Theorem 14.** *Given a rooted tree T on n vertices and vertex weights in range $[0, n^2]$ we can design a labeling scheme of $O(\log^3 n)$ bits such that for any two vertices x, y, $\textsc{fm}(x, y)$ and $\textsc{sm}(x, y)$ can be computed in poly-logarithmic time by just looking at the labels of x, y.*

▶ Remark. If in addition to weight $wt(x)$, each vertex x also has a tag field of $O(\log n)$ bits, say $\textsc{tag}(x)$, then while reporting $\min_1(\cdot)$ we can also report the corresponding vertex's tag. This can be easily done by defining a new weight function $wt'(x)$ which is concatenation of both $wt(x)$ and $\textsc{tag}(x)$. Unless explicitly stated we assume $\textsc{tag}(x)$ to be empty. The tags of two different vertices may be either same or distinct.

A.2 Labeling scheme for answering reachability queries

In this subsection we describe labeling scheme for answering reachability queries from s after two vertex failures. The label of a vertex x, denoted by $\sigma(x)$, will be a concatenating of many small strings $\sigma_i(x)$'s. We first define labels to check ancestor-descendant relations in trees T_1 and T_2. Consider a DFS traversal of one of the trees, say T_1. A vertex x will be an ancestor of vertex y in T_1, if and only if the start-time of x is less than the start-time of y, and the finish-time of x is greater than the finish-time of y. We thus define strings $\sigma_1(\cdot)$ to $\sigma_4(\cdot)$ as follows.

- $\sigma_1(x) :=$ start-time of x in T_1.
- $\sigma_2(x) :=$ finish-time of x in T_1.
- $\sigma_3(x) :=$ start-time of x in T_2.
- $\sigma_4(x) :=$ finish-time of x in T_2.

We also define the labels $\sigma_5(\cdot)$ to $\sigma_6(\cdot)$ to store depth fields.

- $\sigma_5(x) :=$ depth of x in T_1.
- $\sigma_6(x) :=$ depth of x in T_2.

Let u be a query vertex, and f_1, f_2 be a pair of failures. As in Section 6, let $u_0, u_1, ..., u_k$ be the dominators of u with $u_0 = s$ and $u_k = u$. We now consider following steps of our dual fault tolerant reachability oracle.

Step 1. *Check if condition $\mathcal{C}$ is satisfied or not.*
This can be done by just comparing the labels $\sigma_1(\cdot)$ to $\sigma_4(\cdot)$ of vertices f_1, f_2 and u.

Let us suppose condition $\mathcal{C}$ holds true. Let $i, j \in [1, k]$ be such that $f_1 \in \textsc{Path}_{T_1}(\bar{u}_{i-1}, \bar{u}_i)$ and $f_2 \in \textsc{Path}_{T_2}(\bar{u}_{j-1}, \bar{u}_j)$.

Step 2. *Check whether $i = j$, or in other words $u_{i-1} = u_{j-1}$.*
Recall that u_{i-1} is the dominator of u lying above f_1 in T_1, and u_{j-1} is the dominator of u lying above f_2 in T_2. These can also be defined as below.

$$u_{i-1} = \text{ arg } \min_y \{depth_{T_1}(y) \mid y = idom(x) \text{ for some } x \in \textsc{Path}_{T_1}(\bar{f_1}, u)\}$$

$$u_{j-1} = \text{ arg } \min_y \{depth_{T_2}(y) \mid y = idom(x) \text{ for some } x \in \textsc{Path}_{T_2}(\bar{f_1}, u)\}$$

So $\sigma_7(\cdot)$ and $\sigma_8(\cdot)$ are defined as follows.

- $\sigma_7(x) :=$ label for reporting $\langle \min_1(\cdot), \textsc{tag}\rangle$ on tree-paths of T_1 when $\textsc{tag}(x) = idom(x)$ and $wt(x) = depth_{T_1}(idom(x))$.
- $\sigma_8(x) :=$ label for reporting $\langle \min_1(\cdot), \textsc{tag}\rangle$ on tree-paths of T_2 when $\textsc{tag}(x) = idom(x)$ and $wt(x) = depth_{T_2}(idom(x))$.

Now if $i \neq j$, then u will be reachable from s, thus lets assume $i = j$. Let u_i be represented by v, and so $u_{i-1} = idom(v)$. Also let x_1 be the first descendant of f_1 on $\textsc{Path}_{T_1}(s, v)$ whose immediate dominator is equal to $idom(v)$. Similarly, let x_2 be the first descendant of f_2 on $\textsc{Path}_{T_1}(s, v)$ whose immediate dominator is equal to $idom(v)$. Note that we cannot have access to labels of vertices x_1, x_2 and v. Thus, the construction will be not be a straight forward implementation of Algorithm 2.

Step 3. *Check if there exists a vertex $w \in S_B(v)$ for which $idom(w) \in S_A(v) \setminus \{idom(v)\}$.*
Since v is dominator of all vertices in $S_B(u) \setminus S_B(v)$, we can even search for w in the whole set $S_B(u)$. If w lies on $\textsc{Path}_{T_1}(s, u)$, then its immediate-dominator will have second minimum depth among immediate-dominators of all vertices in $\textsc{Path}_{T_1}(\bar{f}_1, u)$. So we just need to compute $\min_2\{depth_{T_1}(idom(x)) \mid x \in \textsc{Path}_{T_1}(\bar{f}_1, u)\}$ and see if it is less than $depth_{T_1}(f_1)$. Similar procedure is required for tree T_2. We thus define the labels $\sigma_9(\cdot)$ and $\sigma_{10}(\cdot)$ as follows.

- $\sigma_9(x) :=$ label for reporting $\min_2(\cdot)$ on tree-paths of T_1 when $wt(x) = depth_{T_1}(idom(x))$.
- $\sigma_{10}(x) :=$ label for reporting $\min_2(\cdot)$ on tree-paths of T_2 when $wt(x) = depth_{T_2}(idom(x))$.

Step 4. *Check if either $D^1(v)$ starts from an ancestor of f_1 in T_1, or $D^2(v)$ starts from an ancestor of f_2 in T_2.*
Note that immediate dominator of each vertex $x \in \textsc{Path}_{T_1}(\bar{f}_1, u)$ is either equal to $idom(v)$, or a descendant of $idom(v)$ in T_1. Moreover, among all vertices in $\textsc{Path}_{T_1}(\bar{f}_1, u)$ whose immediate dominator is same as $idom(v)$, vertex v has the maximum depth. Thus any $\textsc{tag}$ of v can filtered out by setting $wt(x)$ as the ordered pair $\langle depth_{T_1}(idom(x)), n - depth_{T_1}(x)\rangle$. Here for $i = 1, 2$, we need to compute $\beta^i(v) = depth_{T_i}(\text{first vertex of } D^i(v))$ and compare it with $depth_{T_i}(f_i)$. We thus define $\sigma_{11}(x)$ and $\sigma_{11}(x)$ respectively as follows with $\textsc{tag}$ value as $\beta^1(x)$ and $\beta^2(x)$.

- $\sigma_{11}(x) :=$ label for reporting $\langle \min_1(\cdot), \textsc{tag}\rangle$ on tree-paths of T_1 when $\textsc{tag}(x) = \beta^1(x)$ and $wt(x) = \langle depth_{T_1}(idom(x)), n - depth_{T_1}(x)\rangle$.
- $\sigma_{12}(x) :=$ label for reporting $\langle \min_1(\cdot), \textsc{tag}\rangle$ on tree-paths of T_1 when $\textsc{tag}(x) = \beta^2(x)$ and $wt(x) = \langle depth_{T_1}(idom(x)), n - depth_{T_1}(x)\rangle$.

Step 5. *Check if there exists a vertex $w \in \textsc{Path}_{T_1}(\bar{f}_1, u) \setminus \{x_1\}$ such that $idom(w) = idom(v)$ and $\mathrm{AD}_1^i(w)$ starts from an ancestor of f_i in T_i, for some $i \in \{1, 2\}$.*
Lets first consider the case $i = 1$. Then we to search for a vertex $x \in \textsc{Path}_{T_1}(\bar{f}_1, u) \setminus \{x_1\}$ for which $wt(x) = \langle depth_{T_1}(idom(x)), \gamma_1^1(x)\rangle$ is minimum, and check if $\gamma_1^1(x)$ is less than $depth_{T_1}(f_1)$. We add $depth_{T_1}(idom(x))$ in the beginning of weight function so as to filter out those vertices whose immediate dominator is same as $idom(v)$.

The problem here is that we can perform minima queries on tree path $\textsc{Path}_{T_1}(\bar{f}_1, u)$, but do not have any method to exclude out vertex x_1. So we append the field $depth_{T_1}$(pseudo-parent of x in T_1) at the end of weight function, and query for both $\min_1$ and $\min_2$. By doing this we will get two minimums, and by comparing the depth of the pseudo-parents, we can easily filter out which one is from x_1. (Recall that pseudo-parent of x_1 lies in $\textsc{Path}_{T_1}(s, f_1)$). So we define the labels $\sigma_{13}(\cdot)$ and $\sigma_{14}(\cdot)$ as follows.

- $\sigma_{13}(x)$:= label for reporting $\min_1(\cdot)$ and $\min_2(\cdot)$ on tree-paths of T_1 when $wt(x) = \langle depth_{T_1}(idom(x)),\ \gamma_1^1(x),\ depth_{T_1}(\text{pseudo-parent of } x \text{ in } T_1)\rangle$.
- $\sigma_{14}(x)$:= label for reporting $\min_1(\cdot)$ and $\min_2(\cdot)$ on tree-paths of T_1 when $wt(x) = \langle depth_{T_1}(idom(x)),\ \gamma_1^2(x),\ depth_{T_1}(\text{pseudo-parent of } x \text{ in } T_1)\rangle$.

Step 6. *Check if there exists a vertex $w \in \text{PATH}_{T_2}(\bar{f}_2, u) \setminus \{x_2\}$ such that $idom(w) = idom(v)$ and $\text{AD}_2^i(w)$ starts from an ancestor of f_i in T_i, for some $i \in \{1, 2\}$.*
This can be taken care in exactly similar manner as in Step 5. The labels $\sigma_{15}(\cdot)$ and $\sigma_{16}(\cdot)$ are defined as follows.

- $\sigma_{15}(x)$:= label for reporting $\min_1(\cdot)$ and $\min_2(\cdot)$ on tree-paths of T_2 when $wt(x) = \langle depth_{T_2}(idom(x)),\ \gamma_2^1(x),\ depth_{T_2}(\text{pseudo-parent of } x \text{ in } T_2)\rangle$.
- $\sigma_{16}(x)$:= label for reporting $\min_1(\cdot)$ and $\min_2(\cdot)$ on tree-paths of T_2 when $wt(x) = \langle depth_{T_2}(idom(x)),\ \gamma_2^2(x),\ depth_{T_2}(\text{pseudo-parent of } x \text{ in } T_2)\rangle$.

The final label of a vertex x, denoted by $\sigma(x)$ is the concatenation of all the smaller strings $\sigma_{11}(x)$ to $\sigma_{16}(x)$ described above. We conclude with the following theorem.

▶ **Theorem 15.** *A directed graph $G = (V, E)$ on n vertices can be preprocessed for a source vertex s to compute labels of $O(\log^3 n)$ bits such that for any two failing vertices f_1, f_2 and a destination vertex v, whether v is reachable from s in $G \setminus \{f_1, f_2\}$ can be determined in poly-logarithmic time by only processing the labels associated with f_1, f_2 and v.*