Fault Tolerant Reachability for Directed Graphs

Surender Baswana, Keerti Choudhary, Liam Roditty

Fault Tolerant Reachability for Directed Graphs

Surender Baswana[1], Keerti Choudhary[1], and Liam Roditty[2] *

[1] Department of Computer Science and Engineering
IIT Kanpur, Kanpur - 208016, India.
{sbaswana,keerti}@cse.iitk.ac.in
[2] Department of Computer Science
Bar Ilan University, Ramat Gan 52900, Israel.
liam.roditty@biu.ac.il

Abstract. Let $G = (V, E)$ be an n-vertices m-edges directed graph. Let $s \in V$ be any designated source vertex, and let T be an arbitrary reachability tree rooted at s. We address the problem of finding a set of edges $\mathcal{E} \subseteq E \backslash T$ of minimum size such that on a failure of any vertex $w \in V$, the set of vertices reachable from s in $T \cup \mathcal{E} \backslash \{w\}$ is the same as the set of vertices reachable from s in $G \backslash \{w\}$.
We obtain the following results:

- The optimal set $\mathcal{E}$ for any arbitrary reachability tree T has at most $n - 1$ edges.
- There exists an $O(m \log n)$-time algorithm that computes the optimal set $\mathcal{E}$ for any given reachability tree T.

For the restricted case when the reachability tree T is a Depth-First-Search (DFS) tree it is straightforward to bound the size of the optimal set $\mathcal{E}$ by $n - 1$ using semidominators with respect to DFS trees from the celebrated work of Lengauer and Tarjan [13]. Such a set $\mathcal{E}$ can be computed in $O(m)$ time using the algorithm of Buchsbaum et. al [4].
To bound the size of the optimal set in the general case we define semidominators with respect to arbitrary trees. We also present a simple $O(m \log n)$ time algorithm for computing such semidominators. As a byproduct, we get an alternative algorithm for computing dominators in $O(m \log n)$ time.

1 Introduction

Networks in most real life applications are prone to failures. 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.

We distinguish between two models for fault tolerance. In the *Pre-Design* fault-tolerant model the network is designed from scratch such that it will fulfill

* This research was partially supported by Israel Science Foundation (ISF) and University Grants Commission (UGC) of India. The research of the second author was partially supported by Google India under the Google India PhD Fellowship Award.

certain robustness requirements that are known at the design phase. In the *Post-Design* fault-tolerant model the network already exists and it has to fulfill new robustness requirements. This model reflects the scenario in which the network is a physical network such as a road network or an electricity network. Such networks are rarely redesigned from scratch but new robustness requirements are introduced through the years.

In this paper we consider the fault tolerant reachability problem in the post-design fault-tolerant model. From theoretical perspective this model is more challenging as it is stronger than the pre-design model.

We now define the problem formally. Given a directed graph $G = (V, E)$ and a source vertex s, a subgraph H is said to be a Fault Tolerant Reachability Subgraph (FTRS) if for any pair of vertices $v, w \in V$, v is reachable from s in $G\backslash\{w\}$ if and only if v is reachable from s in $H\backslash\{w\}$. We consider the following problem. We are given an arbitrary reachability tree T of s and we are required to find a smallest set $\mathcal{E} \subseteq E$ of edges which on addition to T gives an FTRS. We show that for each tree T, the optimal set $\mathcal{E}$ is of size at most $n-1$. We also present an algorithm that computes this optimal set in $O(m \log n)$ time.

Parter and Peleg [15] considered the question of finding a sparse subgraph of G that preserves the distances from s under a single vertex failure. They showed that if G is an undirected unweighted graph then it has a subgraph H with $O(n^{3/2})$ edges such that for every $v, x \in V$, it holds that the distance from s to v in $H \setminus \{x\}$ is the same as in $G \setminus \{x\}$. They also showed a lower bound of $\Omega(n^{3/2})$. Recently Parter [14] showed a bound of $O(n^{5/3})$ for the case of dual failure. She also showed that this result is tight. For the case of weighted graphs, Demetrescu et al. [8] established a lower bound of $\Omega(m)$ for single source fault tolerant shortest paths structure. Therefore, in light of these lower bounds, it makes sense to relax the problem requirements in order to obtain a graph H of linear (or almost linear) size, as required by many real world applications.

Baswana and Khanna [1] showed that if one is willing to consider only undirected graphs and 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 [16] improved this result and obtained a subgraph with at most $3n$ edges.

Focusing on the reachability in directed graphs instead of approximation of shortest paths in undirected graphs, can be viewed as a different type of relaxation.

The fault tolerant reachability is closely related to the notion of *dominators*. Given a directed graph G and a designated vertex s we say that vertex v dominates vertex w if every path from s to w contains v. A vertex v is said to be the immediate dominator of w if (i) v is a dominator of w, and (ii) every dominator of w (other than v itself) is also a dominator of v.

In a celebrated work [13], Lengauer and Tarjan show that dominators can be computed in $O(m\alpha(m, n))$ time, where $\alpha(m, n)$ is the inverse Ackermann function. Buchsbaum et. al [4] showed how to implement the algorithm of Lengauer and Tarjan in $O(m)$ time. For more details on the work of dominators, see [4, 11,

12]. To the best of our knowledge, all non trivial previous results are based on an initialization phase in which a Depth-First-Search (DFS) tree is computed.

Given a DFS tree T rooted at s it is straightforward using the ideas of Lengauer and Tarjan [13] to find an FTRS. This is implicit in [13] and is based on the notion of semidominators for DFS trees. This solves the FTRS problem in the *weaker* model of pre-design fault tolerant in which the tree can be chosen at the network design phase and hence can be chosen to be a DFS tree.

Our main result is that given any arbitrary tree T, we can efficiently compute an optimal set $\mathcal{E}$ whose addition to T gives an FTRS. This solves the FTRS problem in the more general model of post-design. In order to achieve this we define semidominators with respect to arbitrary reachability trees and not just DFS trees. We expect that this definition could be of independent interest. As a byproduct of our new definition of semidominators we obtain the first non-trivial algorithm for computing dominators that does not rely on a DFS tree computation. The algorithm, however, has a running time of $O(m \log n)$ which is slower than the state of the art for this problem by a logarithmic factor.

1.1 Related work

Most of the previous work on fault tolerant structures is on all-pair shortest paths (APSP). Demetrescu, Thorup, Chowdhury and Ramachandran [8] designed an $O(n^2 \log n)$ size data structure that can report the distance from u to v avoiding x for any $u, v, x \in V$ in $O(1)$ time. Bernstein and Karger [3] improved the preprocessing time of [8] to $O(mn \text{ polylog } n)$. Duan and Pettie [10] improved the result of [8] by designing a data structure of $O(n^2 \log n)$ size that can handle two vertex faults.

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 in the graph without e. Many variants of this problem were studied along the years. For a recent overview see [17] and reference therein.

The questions of finding graph spanners, approximate distance oracles and compact routing schemes that are resilient to f vertex or edge failures were studied in [5, 6, 9].

1.2 Organization of the paper

We describe notations and terminologies in Section 2. For sake of completeness we provide an overview of the FTRS for a DFS tree in Section 3. Here we also highlight the difficulty in extending the existing algorithm for DFS tree to arbitrary tree. In Section 4, we generalize the concept of semidominators to arbitrary tree. In Section 5, we show that for any arbitrary tree T, there exists an optimal set $\mathcal{E}$ of at most $n-1$ edges whose addition to T will make it an FTRS. Furthermore, if we are given the semidominators for all vertices in T, the set $\mathcal{E}$ of edges can be computed in $O(n)$ time. In Section 6, we provide a simple

$O(m \log n)$ time algorithm to compute semidominators. In Section 7, we show the computation of dominators from semidominators.

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.

- T: Any arbitrary tree of G rooted at s.
- $T(v)$: The subtree of T rooted at a vertex v.
- $par(v)$: The parent of v in T.
- $\text{LCA}(u, v)$: The Lowest Common Ancestor of u and v in tree T.
- $\text{PATH}(a, b)$: The tree path from a to b in T. Here a is assumed to be an ancestor of b.
- $\text{OUT}(S)$: The set of all those vertices in $V \backslash S$ which have an incoming edge from some vertex in set S.
- $deg(S)$: The sum of the degrees of all vertices in the set S.
- $\text{SDOM}(v)$: Semidominator of v w.r.t. tree T.
- (σ, w): The sequence obtained by appending w at the end of a sequence σ.
- $\langle P :: Q \rangle$: 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.

Let $\mathcal{P} : \langle v_1, v_2, \ldots, v_n \rangle$ be a sequence of the vertices V defined by any preorder traversal of tree T. For notational convenience henceforth, a vertex will also be denoted by its preorder numbering. Thus, $u \leq v$ would imply that the preorder number of u is less than that of v.

Definition 1. *A simple path $P = (u_0, u_1, \ldots, u_t = v)$ in G is said to be a detour with respect to a given tree T if u_0 is an ancestor of v in T, and for $0 < i < t$, none of the u_i's is an ancestor of v in T.*

A detour from u to v can be seen as an alternate path to v when some intermediate vertex on $\text{PATH}(u, v)$ fails. It follows from Definition 1 that an edge $(u, v) \in T$ is also a detour for vertex v. However, such a detour cannot be used to handle any failures. The next definition is important to characterize those detours that are important for fault tolerant.

Definition 2. *A detour to v with respect to T that emanates from the ancestor of v with minimal preorder number is called a* highest detour *of v.*

We denote the highest detour of v with $\text{HD}(v)$. In case that there is more than one highest detour we pick one arbitrarily.

3 DFS tree versus arbitrary tree

Lengauer and Tarjan [13] presented an algorithm for computing immediate dominators. As part of their work they defined semidominators over a DFS tree T. Their definition of semidominators can be reformulated as follows:

Definition 3 (Semidominators in DFS Tree). *Let T be a DFS tree. For any vertex v ($v \neq s$), the semidominator* $\textsc{sdom}(v)$ *is defined to be the highest ancestor of v from which there is a detour to v. In other words,* $\textsc{sdom}(v)$ *is equal to the first vertex on* $\mathrm{HD}(v)$.

It can be shown that a subgraph H of G that is composed of a reachability tree T and the highest detours of all the vertices with respect to T is an FTRS of G. For the case when T is a DFS tree, Lengauer and Tarjan [13] gave an $O(m\alpha(m,n))$ time algorithm for computing such an FTRS with at most $2n-1$ edges. In order to prove the $2n-1$ bound they used a crucial property of DFS tree which in simple words can be re-stated as follows.

Property 1. If $(\textsc{sdom}(v), v)$ is not an edge in G then we can always find a highest detour $\mathrm{HD}(v)$ for v which can be represented as $\langle \mathrm{HD}(w) :: \textsc{path}(w,y) :: (y,v)\rangle$, where y is an in-neighbor of v and w is either equal to y or an ancestor of y.

For the case when T is an arbitrary tree, and not a DFS tree, Property 1 no longer holds. A simple example that illustrates the situation for general trees is shown in Figure 1. Thus it is not immediately clear whether we can obtain an FTRS by adding at most $n-1$ edges to an arbitrary tree given by an adversary. In order to achieve our goal, we define semidominators for arbitrary trees in the next section.

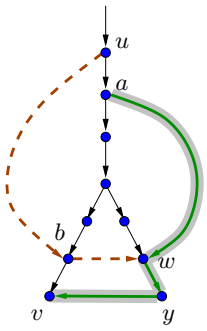

Fig. 1. The highlighted and dashed paths in the figure represent respectively the highest detours $\mathrm{HD}(v)$ and $\mathrm{HD}(w)$ for vertices v and w. Detour $\mathrm{HD}(v)$ cannot be expressed as $\langle \mathrm{HD}(w) :: \textsc{path}(w,y) :: (y,v)\rangle$ because $\mathrm{HD}(w)$ passes through an ancestor of v.

4 Semidominators with respect to arbitrary trees

Given an arbitrary tree T, let D be a detour from a vertex u to a vertex v with minimum number of non-tree edges. Let (u_1, v_1) be the first edge in D and let $(u_2, v_2), (u_3, v_3), \ldots, (u_k, v_k)$ be the sequence of non-tree edges in the order they appear in $D \setminus (u_1, v_1)$. Here $u_1 = u$ and $v_k = v$. Consider the edge (u_i, v_i), where $1 < i \leq k$. Since the segment of D from v_{i-1} to u_i is a path in T it follows that $u_i \in T(v_{i-1})$. Moreover, $v_i \notin T(v_{i-1})$ as if $v_i \in T(v_{i-1})$ we can replace the segment of D from v_{i-1} to v_i by $\text{PATH}(v_{i-1}, v_i)$, thereby reducing the number of non-tree edges.

Consider now only the vertices $(u, v_1, v_2, \ldots, v_k)$. From the above discussion it follows that these vertices satisfy the relation that $v_1 \in \text{OUT}(u)$, and for $1 < i \leq k$, $v_i \in \text{OUT}(T(v_{i-1}))$. This motivates us to define the notion of a valid sequence as follows.

Definition 4 (Valid sequence). *A sequence of vertices $(u, v_1, v_2, \ldots, v_k = v)$ is said to be a valid sequence with respect to tree T if the following two conditions hold:*

(i) (u, v_1) is an edge in G.
(ii) for $1 < i \leq k$, $v_i \in \text{OUT}(T(v_{i-1}))$.

Let u and v be any two vertices such that u is an ancestor of v in T. It follows from Definition 4 that if there exists a detour from u to v in T, then there exists a valid sequence from u to v. However, the other direction is not always true, that is, a valid sequence from u to v in T may not correspond to a detour in T. For example, consider the sequence $\sigma = (u, b, w, v)$ in Figure 1. This is a valid sequence but there is no detour from u to v. The one to one correspondence between detours and valid sequences holds only when T is a DFS tree.

We will now define semidominator with respect to arbitrary trees using valid sequence. In arbitrary trees, it will turn out that if there is a valid sequence from $\text{SDOM}(v)$ to v and $\text{SDOM}(v) \neq par(v)$, then there are two vertex disjoint paths from $\text{SDOM}(v)$ to v. Our FTRS for any tree given tree T will store these vertex disjoint paths in a compact manner.

Definition 5. *A vertex u is semidominator of v if (i) u is an ancestor of v, (ii) there is a valid sequence from u to v, and (iii) there is no other vertex on $\text{PATH}(s, u)$ which has a valid sequence to v.*

Remark 1. There is a detour from an ancestor u of v to v in a DFS tree if and only if there is a valid sequence from u to v. Hence in the case of DFS tree, Definition 5 degenerates to Definition 3.

The following lemma provides an alternative definition to semidominators. We shall use these two definitions interchangeably henceforth.

Lemma 1. *Let u be a vertex with minimum preorder number such that there exists a valid sequence from u to v. Then u is the semidominator of v.*

Proof To prove the lemma it suffices to show that u must be an ancestor of v. Let us assume, towards a contradiction, that u is not an ancestor of v. Thus, there is a vertex w such that $w = \text{LCA}(u, v)$, $w \neq v$ and $w \neq u$. Let $(u = v_0, v_1, v_2, \ldots, v_k = v)$ be a valid sequence from u to v. Let w_1 and w_2 be the children of w such that $v_0 = u \in T(w_1)$ and $v_k = v \in T(w_2)$. Let v_j be the first vertex of the sequence that doest not lie in $T(w_1)$. If $j = 1$, then the in-neighbor v_0 of v_1 lies in $T(w_1)$. If $j > 1$, then v_j has an in-neighbor say u_j that lies in $T(v_{j-1}) \subseteq T(w_1)$. Thus, $v_j \in \text{OUT}(T(w_1))$. The sequence $\sigma = (w, w_1, v_j, v_{j+1}, \ldots, v_k)$ is also a valid sequence that reaches v, and since $w < u$, we get a contradiction. □

5 FTRS for any arbitrary tree

In this section we provide the construction of an optimal size FTRS containing any given arbitrary tree. Our starting point is the following lemma that will be used to show that in order to have two vertex disjoint paths from $\text{SDOM}(v)$ to v, for each v, we need to keep only one extra incoming edge per vertex.

Lemma 2. *Let $u, v, w \in V$ be three vertices such that $v \in \text{OUT}(T(w))$, $v \notin \text{OUT}(u)$, and u is some common ancestor of v and w. Let H be a subgraph of G containing tree T and an edge (y, v), where $y \in T(w)$. If H contains two vertex disjoint paths from u to w, then H also contains two vertex disjoint paths from u to v.*

Proof Let us assume towards a contradiction that H does not contain two vertex disjoint paths from u to v. Then it follows from Menger's Theorem that there exists a vertex x (other than u, v) such that every path from u to v in H passes through x, therefore vertex x is on $\text{PATH}(u, v)$. Let P and Q be two vertex disjoint paths from u to w in H (see Figure 2). Since $w \neq x$, at least one out of these two paths, say Q, does not pass through x. Now $\langle Q :: \text{PATH}(w, y) :: (y, v)\rangle$ gives a path from u to v not passing though x. (Though this concatenated path may contain loops, but we can remove all these loops). Thus we get a contradiction. So H must contain two vertex disjoint paths from u to v. □

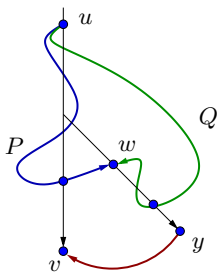

Fig. 2. Two vertex disjoint paths P and Q from u to w.

We now show that a subgraph of G containing a tree T is an FTRS if it contains 2 vertex disjoint paths from $\text{SDOM}(v)$ to v, for each possible vertex v.

Lemma 3. *Any subgraph H of G satisfying the following conditions is an FTRS: (i) H contains tree T, (ii) If v is any vertex such that $\text{SDOM}(v) \neq par(v)$, then there exists two vertex disjoint paths from $\text{SDOM}(v)$ to v in H.*

Proof Consider the failure of an ancestor x of a vertex w in T. Suppose w is reachable from s in $G \setminus \{x\}$. Then there must exist a detour D from u to v, where u and v are respectively vertices lying above and below x in $\text{PATH}(s, w)$. Now a detour from u to v implies that there exists a valid sequence from u to v. So the semidominator of v is either equal to u or an ancestor of u. Since here $\text{SDOM}(v) \neq par(v)$, H contains two vertex disjoint paths, say P and Q from $\text{SDOM}(v)$ to v. Without loss of generality we can assume that x does not lie in P. Thus $\langle \text{PATH}(s, \text{SDOM}(v)) :: P :: \text{PATH}(v, w) \rangle$ is a replacement path to w (avoiding x) contained in subgraph H. □

For the rest of this section, let $\sigma(v)$ denote a valid sequence from $\text{SDOM}(v)$ to v of minimum possible length. The following lemma will be used in the construction of an optimal size FTRS containing any arbitrary tree T.

Lemma 4. *Consider a vertex v and its minimum length valid sequence $\sigma(v) = (u = v_0, \ldots, v_{k-1}, v_k = v)$. Let $w = v_{k-1}$ be the second last vertex in $\sigma(v)$. If $|\sigma(v)| > 2$, then $\text{SDOM}(w) = \text{SDOM}(v)$ and $|\sigma(w)| < |\sigma(v)|$.*

Proof To prove this lemma we use the alternative definition of semidominators as given in Lemma 1. If $\text{SDOM}(w) < \text{SDOM}(v)$, then $(\sigma(w), v)$ will be a valid sequence for v starting from a vertex whose preorder number is less than that of $\text{SDOM}(v)$. This is because $v \in \text{OUT}(T(w))$. Similarly, if $\text{SDOM}(v) < \text{SDOM}(w)$, then $\sigma(v) \setminus \{v\}$ will be a valid sequence for w starting from a vertex whose preorder number is less than that of $\text{SDOM}(w)$. Thus $\text{SDOM}(w)$ must be equal to $\text{SDOM}(v)$. Now suppose $|\sigma(v)| \leq |\sigma(w)|$. Then $\sigma(v) \setminus \{v\}$ would be a valid sequence from $\text{SDOM}(w)$ to w of length strictly less than $|\sigma(w)|$. Hence $|\sigma(w)|$ must be less than $|\sigma(v)|$. □

Let L be the list of vertices in G arranged in the non-decreasing order of length of $\sigma(v)$. The following theorem presents an algorithm for computing an FTRS of optimal size assuming that the minimum length valid sequences for all vertices is known.

Theorem 1. *Given any arbitrary reachability tree T rooted at s, Algorithm 1 computes an optimal set $\mathcal{E}$ such that $T \cup \mathcal{E}$ is an FTRS. Moreover, the size of $\mathcal{E}$ is always bounded by $n - 1$.*

Proof Note that H already contains the tree T. So it follows from Lemma 3 that it is sufficient to prove that for any vertex v if $\text{SDOM}(v) \neq par(v)$, then H contains two vertex disjoint paths from $\text{SDOM}(v)$ to v. The proof is by induction on list L. Consider a vertex v such that $\text{SDOM}(v) \neq par(v)$. If $(\text{SDOM}(v), v)$ is a forward edge, then the tree path from $\text{SDOM}(v)$ to v, and the edge $(\text{SDOM}(v), v)$ are two vertex disjoint paths from $\text{SDOM}(v)$ to v in H. Thus let us consider the

Algorithm 1: Computing an FTRS H containing an arbitrary tree T.

```
1  H ← T;
2  foreach v ∈ L do
3  |  if (SDOM(v), v) is a forward edge then
4  |  |  H ← H ∪ (SDOM(v), v);
5  |  else if |σ(v)| > 2 then
6  |  |  w ← Second last vertex in σ(v);
7  |  |  y ← In-neighbor of v lying in T(w);
8  |  |  H ← H ∪ {(y, v)};
9  |  end
10 end
```

case when $|\sigma(v)| > 2$. Let w be the second last vertex of $\sigma(v)$ and y be an in-neighbor of v lying in $T(w)$. So Lemma 4 implies that $\text{SDOM}(w) = \text{SDOM}(v)$, and w precedes v in list L.

We first consider the case when $\text{SDOM}(w) \neq par(w)$. Since w will appear before v in L, by induction hypothesis H contains two vertex disjoint paths from $\text{SDOM}(w)$ to w. Now the edge (y, v) lies in H, so Lemma 2 implies that H contains two vertex disjoint paths from $\text{SDOM}(v)$ to v also. Next let us consider the case when $\text{SDOM}(w) = par(w)$. Since v lies outside $T(w)$ but in the subtree rooted at $par(w) = \text{SDOM}(v)$, we have that $\text{SDOM}(v) = \text{LCA}(w, v) = \text{LCA}(y, v)$. Thus in this case $\text{PATH}(\text{SDOM}(v), v)$ and $\text{PATH}(\text{SDOM}(v), y)::(y, v)$ form two vertex disjoint paths from $\text{SDOM}(v)$ to v. Hence, it can be seen that Algorithm 1 computes an FTRS containing tree T.

Note that we add an extra incoming edge to v only if $\text{SDOM}(v) \neq par(v)$. In this case, there exist two vertex disjoint paths from $\text{SDOM}(v)$ to v in G. So v will be reachable from s even after failure of $par(v)$. Thus any FTRS must keep an additional incoming edge to v in this case. Hence the subgraph H is indeed a minimum size FTRS containing tree T. □

Remark 2. Given $\text{SDOM}(v)$ and a minimum length $\sigma(v)$ for each v, Algorithm 1 takes $O(n)$ time to compute optimal FTRS containing any given tree T.

In the following section, we present an $O(m \log n)$ time algorithm for computing $\text{SDOM}(v)$ and a minimum length $\sigma(v)$ for all v. This implies that given any tree T, we can compute an optimal FTRS containing T in $O(m \log n)$ time.

6 Algorithm for computing semidominators and valid sequences

Our algorithm for computing semidominators is an iterative algorithm. It processes the vertices in the increasing order of the preorder numbering $\mathcal{P}$ of T. Let v_i denote the vertex at the i^{th} place in $\mathcal{P}$. During i^{th} iteration, the algorithm computes the set W_i consisting of all those vertices whose semidominator is v_i.

Consider a vertex v_i. Let B denote the set of all those vertices w for which there exists a valid sequence starting from v_i and ending at w. The set B can be computed as follows. We initialize B as the out-neighbors of v_i. Next we add $\text{OUT}(T(w))$ to B for each w in B, and proceed recursively. By the alternative definition of semidominators as in Lemma 1 we have that $W_i = B \setminus (\cup_{j<i} W_j)$.

In order to design an efficient implementation of the algorithm outlined above, there are two requirements. The first requirement is that while computing valid sequences from v_i we should not process those vertices whose semidominator have already been computed. For this purpose, we keep a flag variable *active/inactive* corresponding to each vertex w in G. At any instant of time the active vertices are those vertices whose semidominator has not yet been computed. The second requirement is that given any vertex w we should be able to compute the set of active nodes in $\text{OUT}(T(w))$ efficiently. In order to fulfill these requirements, we use a data structure $\mathcal{D}$ that supports the following two operations efficiently.

1. ActiveOutNghbrs($\mathcal{D}, T(w)$): return the set of active nodes in $\text{OUT}(T(w))$.
2. MarkInActive($\mathcal{D}, S$): mark the vertices in set S as inactive. This is done by simply deleting from $\mathcal{D}$ the incoming edges to all vertices present in set S.

The data structure $\mathcal{D}$ is a suitably augmented segment tree formed on an Euler tour of the tree T. The data structure takes $O(deg(A) \log n))$ time to perform ActiveOutNghbrs($\mathcal{D}, T(w)$) operation, where A is the set of vertices reported. It takes $O(deg(S) \log n)$ time to perform MarkInActive($\mathcal{D}, S$) operation. We provide the complete details of the data structure in Subsection 6.1.

Algorithm 2 gives the pseudo code for computing semidominators. It maintains a queue $\mathcal{Q}$ throughout the run of algorithm. The semidominator of the vertices is computed in the order they are enqueued. Initially all the vertices in G except root are marked as active. A vertex is marked inactive as soon as it is enqueued in $\mathcal{Q}$. In the i^{th} iteration the algorithm computes the set of all those vertices whose semidominator is v_i as follows. First it computes the set S of all the active out-neighbors of v_i. This set is enqueued and for each $w \in S$, $\sigma(w)$ is set as (v_i, w). Next while $\mathcal{Q}$ is non empty, it removes the first vertex say x from $\mathcal{Q}$. For each active node w in $\text{OUT}(T(x))$, $\sigma(w)$ is assigned as $(\sigma(x), w)$ and w is enqueued in $\mathcal{Q}$. This process is repeated until $\mathcal{Q}$ becomes empty. Vertex v_i is assigned as semidominator of all the vertices enqueued in the i^{th} iteration.

Figure 3 illustrates the execution of our algorithm. Figure 3(a) depicts the first iteration which is supposed to compute W_1. The vertices that are enqueued before the while loop are $\langle 2, 14 \rangle$. The execution of the while loop will place vertices 15 and 16 into the queue in this order. It can be visually inspected that these vertices constitute W_1. Similarly Figure 3(b) depicts the second iteration that is supposed to compute W_2. The vertices that are enqueued before entering the while loop are $\langle 3, 7, 12 \rangle$. The execution of the while loop will place vertices 5,10,4,8,11 into the queue in this order. It can be visually inspected that these vertices constitute W_2.

Algorithm 2: Computing semidominator and the corresponding valid sequence

```
1  Q ← ∅;
2  for i = 1 to n do
3  |  S ← Set of active vertices lying in OUT(v_i);
4  |  ENQUEUE(Q, S);
5  |  MARKINACTIVE(D, S);
6  |  foreach w ∈ S do σ(w) = (v_i, w);
7  |  while (Q ≠ ∅) do
8  |  |  x ← DEQUEUE(Q);
9  |  |  SDOM(x) ← v_i;
10 |  |  S ← ACTIVEOUTNGHBRS(D, T(x));
11 |  |  ENQUEUE(Q, S);
12 |  |  MARKINACTIVE(D, S);
13 |  |  foreach w ∈ S do σ(w) = (σ(x), w);
14 |  end
15 end
```

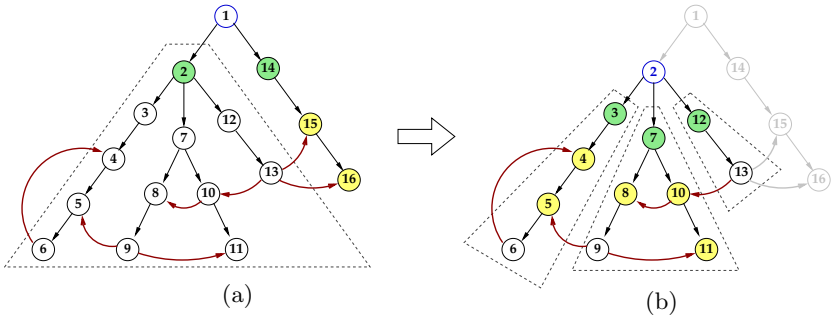

Fig. 3. The filled vertices in Figure (a) and (b) respectively constitute the sets W_1 and W_2. Figure (b) shows that all the vertices in W_1 are marked inactive in round 2.

For each vertex u, let $\sigma(u)$ denote a valid sequence from SDOM(u) to u of minimum possible length. Let L be the list of vertices in G arranged in the non-decreasing order of $|\sigma(u)|$. Then it can be proved by induction on L that Algorithm 2 correctly computes (i) semidominator of u, and (ii) a minimum length valid sequence from SDOM(u) to u, for each vertex u in G.

We now analyze the time complexity of Algorithm 2. The total time taken by Step 3 in the algorithm is $O(m)$. The time taken by steps 5, 10, and 12 is $O(\log n)$ times the sum of degrees of vertices enqueued in $\mathcal{Q}$. Since each vertex is enqueued at most once, the running time of the algorithm is $O(m \log n)$.

Theorem 2. *There exists an $O(m \log n)$ time algorithm that for any given graph with n vertices and m edges, and any given reachability tree T, computes the semidominator and a minimum length valid sequence for each vertex in G.*

From Theorems 1 and 2, we directly get the following result.

Theorem 3. *There exists an $O(m \log n)$ time algorithm that for any given graph G with n vertices and m edges, and any given reachability tree T rooted at source, computes an optimal set $\mathcal{E}$ such that $T \cup \mathcal{E}$ is an FTRS for G.*

6.1 Data structure

Let $\mathcal{T}$ be a segment tree [2] whose leaf nodes from left to right correspond to the sequence $\langle v_1, ..., v_n \rangle$ (see Figure 4). Our data structure will be $\mathcal{T}$ whose nodes are suitably augmented as follows. Let (u, v) be an edge in G. We store a copy of the edge as the ordered pair (u, v) at all ancestors of u (including itself) in tree $\mathcal{T}$. Thus each edge in G is stored at $O(\log n)$ levels in $\mathcal{T}$. Let $\mathcal{E}(b)$ be the collection of edges stored at any node b in $\mathcal{T}$. We keep the set $\mathcal{E}(b)$ sorted by the second endpoint of the edges in a doubly link list. For each edge $(u, v) \in E$, we also store pointer to all $\log n$ copies of it in $\mathcal{T}$. The size of the data structure is $O(m \log n)$ in the beginning.

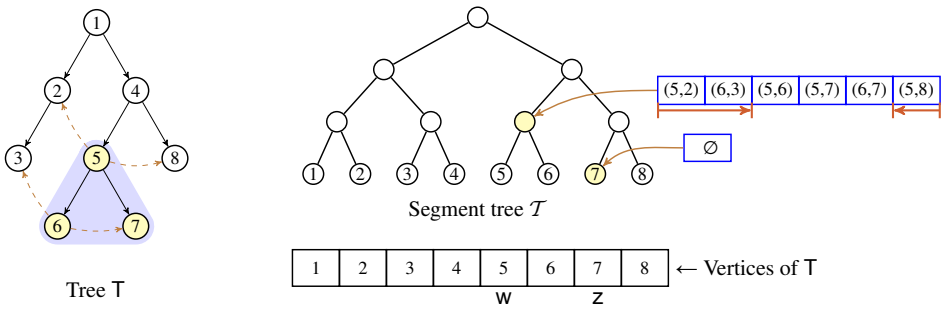

Fig. 4. Data Structure.

The operation MarkInActive($\mathcal{D}, S$) involves deletion of incoming edges to all the vertices in set S. Since we store pointers to all $\log n$ copies of an edge, a single edge can be deleted from the data structure in $O(\log n)$ time. So the time taken by this operation is $O(deg(S) \log n)$.

We now show that $\mathcal{D}$ can perform the operation ActiveOutNghbrs($\mathcal{D}, T(w)$) quite efficiently. Let S_0 be the set of active nodes in out($T(w)$). Note that the preorder numbering of the vertices in $T(w)$ will be a contiguous subsequence of $[1, .., n]$, and w would be the vertex of minimum preorder number in $T(w)$. Let z be the vertex with maximum preorder number in subtree $T(w)$ (This information can be precomputed in total $O(n)$ time for all vertices in the beginning). So $[w, .., z]$ denotes the set of vertices in $T(w)$.

Notice that any contiguous subsequence of $[1, .., n]$ can be expressed as disjoint union of at most $\log n$ subrees in $\mathcal{T}$. Let $\tau_1, ..., \tau_\ell$ denote these subrees for the subsequence $[w, .., z]$. For $i = 1$ to ℓ, let E_i denote the set of all those edges

(x, y) such that x is a leaf node of τ_i and y lies outside the set $[w, .., z]$. It can be observed that the desired set S_0 corresponds to the set of second-endpoints of all edges in the set $\cup_{i=1}^{\ell} E_i$. Let $b_1, ..., b_\ell$ respectively denote the roots of subtrees $\tau_1, ..., \tau_\ell$ in $\mathcal{T}$. Then set E_i can be computed by scanning the list $\mathcal{E}(b_i)$ from beginning (and respectively end) till we encounter an edge (u, v) with v lying in range $[w, .., z]$. (See Figure 4). Thus the time taken by the operation ACTIVE-OUTNGHBRS$(\mathcal{D}, T(w))$ is bounded by $O(deg(S_0) + \log n)$, where S_0 is the set of vertices reported.

This data structure can be preprocessed in $O(m \log n)$ time as follows. First we compute set $\mathcal{E}(b)$ for each leaf node b of $\mathcal{T}$. This takes $O(m)$ time. Now $\mathcal{E}(b)$ for an internal node b can be computed by simply merging the lists $\mathcal{E}(b_1)$, $\mathcal{E}(b_2)$ where b_1 and b_2 are children of b. The space complexity of $\mathcal{D}$ is also $O(m \log n)$.

Theorem 4. *Given a graph G, it can be preprocessed in $O(m \log n)$ time to build a data structure of size $O(m \log n)$ to perform the following operations.*

1. ACTIVEOUTNGHBRS*$(\mathcal{D}, T(w))$: return the set of active nodes in* OUT$(T(w))$.
2. MARKINACTIVE*$(\mathcal{D}, S)$: mark the vertices in set S as inactive.*

The time taken by both of the above operations is $O(deg(S) \log n)$ where S is the set of vertices reported in the first case, and S is the set of vertices marked inactive in the second case.

7 Computation of dominators from semidominators

A vertex $u \neq v$ is said to be a dominator of v if every path from s to v passes through u. Thus u is a dominator of v if and only if either $u = s$, or v becomes unreachable from s on removal of u from G. For each vertex v, we use $D(v)$ to denote the set of dominators of v. In order to compute the set $D(v)$ it suffices to compute $idom(v)$ defined as follows.

Definition 6 ([13]). *Vertex u is said to be immediate dominator of v, denoted by $u = idom(v)$, if u is a dominator of v and every other dominator of v (other than vertex u itself) is also a dominator of u.*

The algorithm for computing immediate dominators from semidominators is almost the same as for the restricted case when T is a DFS tree. For the sake of completeness, we now provide this algorithm. The starting point is the concept of relative dominators defined as follows.

Definition 7 ([4]). *A vertex w is said to be a relative dominator of v if w is a descendant of* SDOM(v) *on* PATH$($SDOM$(v), v)$ *for which* SDOM(w) *has the minimum preorder numbering.*

The following relationship between relative dominators, immediate dominators, and semidominators was shown by Buchbaum et al. [4] for DFS tree. We show that this relation holds even for any arbitrary tree as well.

Lemma 5 ([4]). *For any vertex v, if* $\text{SDOM}(rdom(v)) = \text{SDOM}(v)$*, then* $idom(v) = \text{SDOM}(v)$*, otherwise,* $idom(v) = idom(rdom(v))$.

Proof Let w be a relative dominator of v, and $u = idom(w)$. For the case when $w = v$, it is easy to see that $idom(v) = \text{SDOM}(v)$. Thus we consider the case $w \neq v$. In this case, in order to prove that $idom(v) = u$, we need to show that u is a dominator of v and there does not exists any other dominator of v on $\text{PATH}(u, v)$.

We first show that v is unreachable from s in $G \setminus \{u\}$. Since $u = idom(w)$, we have that w is unreachable from s in $G\backslash\{u\}$. Assume towards a contradiction that there is a path from s to v in $G\backslash\{u\}$. Then there must exist a detour D from a to b where a is an ancestor of u and b is a descendant of w belonging to $\text{PATH}(w, v)$. Since detour D implies existence of a valid sequence from a to b, it follows that $\text{SDOM}(b) \in \text{PATH}(s, a)$. This contradicts that w is a relative dominator of v. Hence v is unreachable from s in $G\backslash\{u\}$. Thus u is a dominator of v.

In order to show that u is the immediate dominator of v, it suffices to show that there does not exist any vertex on $\text{PATH}(u, v)\backslash\{u\}$ whose removal disconnects v from s. Assume towards a contradiction that there exists such a vertex x. Since there are two vertex disjoint paths from $\text{SDOM}(v)$ to v, so x can not lie on $\text{PATH}(\text{SDOM}(v), v)\backslash\{\text{SDOM}(v), v\}$. Also note that x can not lie on $\text{PATH}(u, w)\backslash\{u, w\}$ as there are two vertex disjoint paths from $idom(w)$ to w. Since $\text{SDOM}(v)$ is an ancestor of w, this contradicts the existence of x. □

Lemma 5 suggests that once we have computed relative dominators, the immediate dominators can be computed in $O(n)$ time by processing the vertices of T in a top down manner. The task of computing relative dominators can be formulated as a data structure problem on a rooted tree as follows.

Each tree edge (u, y) is assigned a weight equal to $\text{SDOM}(y)$. It can be seen that if (a, w) is minimum weight edge on $\text{PATH}(\text{SDOM}(v), v)$, then w is a relative dominator of v. So in order to compute relative dominators, all we need is to compute the least weight edge on any given path of tree T. This problem turns out to be an instance of *Bottleneck Edge Query* (BEQ) problem on trees with integral weights. Demaine et al. [7] recently presented the following optimal solution for this problem.

Theorem 5 (Demaine et al. [7]). *A tree on n vertices and edge weights in the range $[1, n]$ can be preprocessed in $O(n)$ time to build a data structure of $O(n)$ size so that given any $u, v \in V$, the edge of smallest weight on $\text{PATH}(u, v)$ can be reported in $O(1)$ time.*

We process tree T in a top down order to compute $idom(v)$ as follows. We first compute $rdom(v)$ in $O(1)$ time by performing BEQ query between v and $\text{SDOM}(v)$. Using the data structure stated in Theorem 5, it takes $O(1)$ time. Let $w = rdom(v)$. If $w = v$, then we set $idom(v) \leftarrow \text{SDOM}(v)$. Otherwise, we set $idom(v) \leftarrow idom(w)$. Since we process the vertices in a top down fashion, $idom(w)$ has already been computed. Hence it takes $O(1)$ time to compute $idom(v)$. So it can be concluded that we can compute immediate dominators of all vertices in $O(n)$ time only if we know semidominators of all vertices.

References

1. 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.
2. J. L. Bentley. Solutions to Klee's rectangle problems. *Unpublished manuscript, Dept of Comp Sci, Carnegie-Mellon University, Pittsburgh PA*, 1977.
3. Aaron Bernstein and David Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *STOC'09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 101–110, New York, NY, USA, 2009. ACM.
4. 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.
5. Shiri Chechik. Fault-tolerant compact routing schemes for general graphs. *Inf. Comput.*, 222:36–44, 2013.
6. Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. f-sensitivity distance oracles and routing schemes. *Algorithmica*, 63(4):861–882, 2012.
7. Erik D. Demaine, Gad M. Landau, and Oren Weimann. On cartesian trees and range minimum queries. *Algorithmica*, 68(3):610–625, 2014.
8. 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.
9. Michael Dinitz and Robert Krauthgamer. Fault-tolerant spanners: better and simpler. In Cyril Gavoille and Pierre Fraigniaud, editors, *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. ACM, 2011.
10. 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.
11. Wojciech Fraczak, Loukas Georgiadis, Andrew Miller, and Robert Endre Tarjan. Finding dominators via disjoint set union. *J. Discrete Algorithms*, 23:2–20, 2013.
12. 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.
13. 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.
14. Merav Parter. Dual failure resilient BFS structure. *arXiv:1505.00692*, 2015.
15. 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.
16. Merav Parter and David Peleg. Fault tolerant approximate BFS structures. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1073–1092. SIAM, 2014.
17. Virginia Vassilevska Williams. Faster replacement paths. In Dana Randall, editor, *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. SIAM, 2011.