Single source distance oracle for planar digraphs avoiding a failed node or link

Surender Baswana* Utkarsh Lath* Anuradha S. Mehta*

Abstract

Let $G = (V, E)$ be a directed planar graph on $n = |V|$ vertices, and let $s \in V$ be any fixed source vertex. We show that G can be preprocessed in $O(n \operatorname{polylog} n)$ time to build a data structure of $O(n \operatorname{polylog} n)$ size which can answer the following query in $O(\log n)$ time for any $u, v \in V$:

report distance from s to v in the graph $G \backslash \{u\}$

We also address the all-pairs version of this problem and present a data structure with $O(n\sqrt{n} \operatorname{polylog} n)$ preprocessing time and space which guarantees $O(\sqrt{n} \operatorname{polylog} n)$ query time.

1 Introduction

The problem of finding shortest paths in graphs is one of the most fundamental algorithmic problems of computer science. In the last few decades, researchers have explored the following variant of the shortest paths problem, called the *replacement paths* problem. Let $G = (V, E)$ be a directed weighted graph on $n = |V|$ vertices and $m = |E|$ edges. For any vertices $u, v, x \in V$, let $P(u, v, x)$ denote the shortest path from u to v in the graph $G \backslash \{x\}$. The aim is to have a compact data structure so that the path $P(u, v, x)$ and its length can be reported efficiently. In addition to being a problem of independent interest, the replacement paths problem appears to be a natural extension of the shortest paths problem. Efficient solution of this problem and its variants have been useful in dynamic routing as well [19].

The all-pairs version of the replacement paths problem (where u, v and x can be chosen arbitrarily) has been solved quite efficiently. Bernstein and Karger [3], improving the work of Demetrescu et al. [4], designed a data structure which can report length of $P(u, v, x)$ for any $u, v, x \in V$ in $O(1)$ time and, it takes $O(n^2 \operatorname{polylog} n)$ space and $O(mn \operatorname{polylog} n)$ preprocessing time. These bounds match, up to poly-logarithmic factors, the best bounds known for the classical all-pairs shortest paths problem. Duan and Pettie [6] addressed an even harder variant of the all-pairs replacement paths problem to handle failure of any two vertices.

For the single pair version of the replacement paths problem (where u and v are two fixed vertices and x can be any vertex from V), no significant breakthrough has yet been made. For this problem, the trivial upper bound $O(mn)$ is still the best known upper bound, and there is a lower bound of $\Omega(m\sqrt{n})$ established by Hershberger et al. [11]. However, for various weaker versions of this problem, efficient results have been obtained in the past [1, 2, 15, 16, 17].

We address the replacement paths problem in planar graphs. Planar graphs have occupied the center stage of the classical shortest paths problem and many interesting and novel results have been achieved [5, 12, 13, 20]. Emek et al. [7] designed an $O(n \log^3 n)$ time solution for the single pair version of the replacement paths problem in planar graphs. Through a series of refinements [13, 21], this bound has been improved to $O(n \log n)$. The result has also been extended to bounded genus graphs by Erickson and Nayyeri [8].

We consider the single source version of the replacement paths problem : Given a directed weighted planar graph and a source vertex s, design a compact data structure to report $P(s, v, x)$ for any $v, x \in V$. As the central contribution of this paper, we present an extremely compact data structure for this problem. This data structure takes $O(n \operatorname{polylog} n)$ space and can be built in $O(n \operatorname{polylog} n)$ time. Interestingly, it takes just $O(\log n)$ time for reporting the length of $P(s, v, x)$ for any $v, x \in V$, hence the name "oracle". Moreover, the entire path $P(s, v, x)$ can also be reported by spending $O(\operatorname{loglog} n)$ time per edge on the path. This data structure also leads to a near optimal algorithm for the single source second shortest paths problem in planar graphs.

We also obtain the first subquadratic space and sublinear query time solution for the all-pairs version of the replacement paths problem in planar graphs. Let $S \in [n^{3/2}, n^2]$. We present a data structure which occupies $O(S)$ space and can report $P(u, v, x)$ for any $u, v, x \in V$ in $O(n^2/S \operatorname{polylog} n)$ time. This data structure is obtained, with effortless ease, by combining our data structure for the single source replacement paths problem with the data structure for all-pairs shortest paths devised by Djidjev [5]. We just provide its sketch in Appendix.

*Department of CSE, I.I.T. Kanpur, India. Email: {sbaswana,utkarsh.lath,anuradha.s.mehta}@gmail.com.
Research supported by the Indo-German Max Planck Center for Computer Science (IMPECS).

1.1 Overview of the result and new ideas

Let T be the shortest paths tree rooted at source s in G. Sleator and Tarjan [18] gave a technique for breaking down a tree into vertex disjoint paths. Using this technique in a straight forward manner, it suffices if we can solve the single source replacement paths problem where the failing vertex belongs to a given path Q in tree T. So we focus on the latter problem and design a near optimal data structure for it. At the heart of our data structure lies an efficient solution to a related problem, we call, *Shortest Paths Avoiding Suffix Subpaths* (SPASS). To build our data structure, we recursively partition Q into equal halves, and build a balanced binary tree $\mathcal{H}(Q)$ with $O(n)$ nodes. Each node ν of this tree corresponds to a subpath of Q. We augment ν with a data structure to solve the SPASS problem for its subpath. Interestingly, the replacement path $P(s, v, x)$, for any $v \in V, x \in Q$, can be computed by querying SPASS data structure associated with a single node in $\mathcal{H}(Q)$.

It turns out that SPASS can be reduced to the Multiple Source Shortest Paths (MSSP) problem. Klein [12] designed an $O(n \log n)$ space and time algorithm to solve MSSP in planar graphs. This by itself would complete the solution to our problem, but solving $O(n)$ instances of the SPASS problem on the given graph would take $O(n^2)$ space and time. To circumvent this problem, we build a relatively small sized graph for each node of $\mathcal{H}(Q)$ and show that it suffices to make SPASS queries on these graphs only. To build these small size graphs, we proceed as follows.

1. We make a couple of insightful observations about SPASS data structures associated with various nodes of $\mathcal{H}(Q)$. In particular, SPASS data structure at a node bears some similarity with SPASS data structure of some of its ancestor. This similarity allows us to obtain a small sized graph for a node by *compressing* the subgraph associated with some of its ancestor. This compression involves a careful pruning and shortcutting paths by edges.

2. We discover an interesting property about replacement paths in planar graph (Theorem 3.1). This property, which might be of independent interest, proves to be helpful for our problem in the following manner. In order to reduce the size of graph associated with a node in $\mathcal{H}(Q)$ without destroying planarity, we need to add some edges which are *short-cuts* of certain paths. Note that, addition of edges, in general, may lead to creation of erroneous replacement paths in a graph. The property (Theorem 3.1) ensures that no such erroneous path gets created if these edges are assigned weights *suitably*.

Organization of the paper After introducing various notations in section 2, we describe an important property of replacement paths in planar graphs in section 3. Section 4 and 5 present efficient solution of a restricted version of the single source replacement paths problem wherein the failing vertex belongs to a given path. This forms the core of our main result. We use this in conjunction with a divide and conquer strategy to solve the single source replacement paths problem in section 6.

2 Preliminaries

Let $G = (V, E)$ be the given planar directed graph and $\omega : E \longrightarrow R^+$ be a weight function on its edges. Let $s \in V$ be a source vertex and T be the shortest path tree in G rooted at vertex s. Without loss of generality, we assume uniqueness of shortest paths and replacement paths. Throughout this paper, we will be working with that planar embedding of G which has s lying on the boundary of the infinite face. Let Q be any path in G. We define the following notations.

- $T(u)$: the subtree of T rooted at u.
- $Q(u, v)$: the subpath of Q from vertex u to vertex v. It is defined only if $u, v \in Q$, and u precedes v.
- $P(u, v)$: the shortest path from u to v in G.
- $\delta(u, v)$: the length of $P(u, v)$.
- $P(u, v, Q)$: the shortest path from u to v in G when all the vertices of path Q fail.
- $\delta(u, v, Q)$: the length of $P(u, v, Q)$.
- FIRST(Q) : the first vertex of Q.
- LAST(Q) : the last vertex of Q.

DEFINITION 2.1. *(**family**): Given a path Q in T, let R be a subpath of Q. Let x be the vertex immediately after* LAST(R) *on Q. We define* FAMILY(R, Q) *(see Figure 1) as the set of vertices present in* $T($FIRST$(R))\backslash T(x)$.

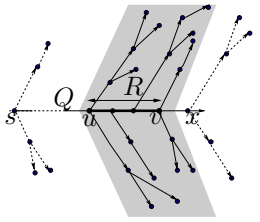

Figure 1: The vertices in the shaded region constitute FAMILY(R, Q) where $R = Q(u, v)$.

DEFINITION 2.2. *(**G**(x)): The subgraph of G induced by vertices of $T(x)$ and augmented by vertex s and edges from s as follows. For each $v \in T(x)$ with neighbors outside $T(x)$, add an edge (s, v) with weight* $= \min_{(u,v)\in E, u\notin T(x)}(\delta(s, u) + \omega(u, v))$.

It can be observed that we have to just work on $G(x)$ instead of G if we want to handle failure of any vertex

belonging to $T(x)$. Notice that a replacement path possesses optimal substructure property. This leads to the following useful observation.

OBSERVATION 2.1. *Let v be a vertex and let vertex $x \in P(s,v)$. If $P(s,v,x)$ passes through a vertex $w \in P(x,v)$, then $P(s,v,x) = P(s,w,x) + P(w,v)$.*

3 Bounded difference property of replacement paths

Let $v, w \in V$ be any two vertices and x be a failing vertex. It is easy to observe that the difference between the lengths of the two replacement paths $P(s,v,x)$ and $P(s,w,x)$ in a general graph may be quite arbitrary. However, for planar graphs, this difference is bounded if v and w satisfy certain *topological* conditions.

Let (u,v) be an edge of G that is not present in tree T. Let a be the lowest common ancestor of u and v in tree T. Notice that $P(a,u)$, edge (u,v), and $P(a,v)$ together form a separator in G. See Figure 2(i). In fact, ignoring the directedness of the graph, this separator is actually the fundamental cycle defined by (u,v) and tree T. Let w be any vertex such that s and w belong to different regions of the plane defined by this cyclic separator. The *bounded-difference-property* of replacement paths in planar graphs claims the following : No matter which vertex fails, the replacement path to v can not be *arbitrarily* longer than the replacement path to w. This claim holds even when an entire subpath of T (not just a vertex) fails. The following theorem states this property in a formal manner.

THEOREM 3.1. *Consider an embedding of the fundamental cycle defined by any non-tree edge (u,v) and tree T. Let w be any vertex such that s and w lie on different sides of this cycle. Then for any path Z in the tree T with $s, v, w \notin Z$,*

$$\delta(s,v,Z) - \delta(s,w,Z) \leq \delta(s,u) + \omega(u,v) - \delta(s,w) \quad (3.1)$$

Proof. Let a be the lowest common ancestor of u and v in T (see Figure 2(i)). If $P(s,a)$ remains intact after failure of Z then Equation 3.1 holds obviously. So let us assume that the failing path Z includes a subpath of $P(s,a)$ (see Figure 2(ii)). Notice that the fundamental cycle defined by (u,v) is a separator with s and w lie on two different sides of it. So the path $P(s,w,Z)$ must intersect this separator at some vertex, say y. There are only two possibilities of this path as shown in Figure 2(ii). Firstly observe that

$$\delta(s,y) + \delta(y,v,Z) \leq \delta(s,u) + \omega(u,v) \quad (3.2)$$

Now using triangle inequality, we can infer that $\delta(s,w) \leq \delta(s,y) + \delta(y,w,Z)$ and $\delta(s,v,Z) \leq \delta(s,y,Z) + \delta(y,v,Z)$. Combining together these inequalities and using $\delta(s,w,Z) = \delta(s,y,Z) + \delta(y,w,Z)$ we get

$$\delta(s,w) + \delta(s,v,Z) \leq \delta(s,w,Z) + [\delta(s,y) + \delta(y,v,Z)]$$

Plugging the inequality from Equation 3.2, RHS is bounded by $\delta(s,w,Z) + \delta(s,u) + \omega(u,v)$. Rearranging the terms, we get the desired Equation 3.1.

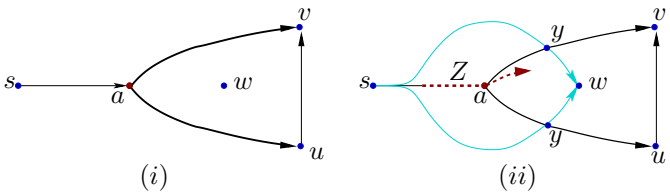

Figure 2: (i) s and w lie on different sides of the separator defined by (u,v), (ii) The replacement path $P(s,w,Z)$ intersects the separator at y.

4 Data structure for failure of vertices on a path $Q \subseteq T$

Let Q be any path in T originating from source s. In this section we describe a data structure to report the replacement path from source s to any vertex in G when the failing vertex lies on Q. This data structure will be based on a problem called *Shortest Paths Avoiding Suffix Subpaths*, or SPASS in short.

4.1 spass problem A subpath R of a path Q is called a *suffix subpath* of Q if $\text{LAST}(R) = \text{LAST}(Q)$.

DEFINITION 4.1. *(The SPASS problem) Given a shortest path Q in T, design a compact data structure for efficiently reporting shortest path from s to any vertex $v \in V$ avoiding (i.e. not passing through any vertex of) any given suffix subpath of Q.*

There is no known efficient solution of SPASS for general graphs. However, for planar graphs, there exists a solution of SPASS with $O(|G(\text{FIRST}(Q))| \log n)$ space and $O(\log n)$ query time. This solution is obtained by a simple reduction to the *Multiple Source Shortest Paths* (MSSP) problem, for which Klein [12] gave an efficient algorithm. We describe the reduction of SPASS to MSSP in Section 5. Note that Emek et al. [7] and Wulff-Nilsen [21] also used reduction to MSSP for solving similar subproblems in their algorithms for single pair replacement paths problem.

4.2 Handling failure of a vertex from Q We begin with showing a nice relationship between our replacement paths problem and SPASS problem. Let $Q(u,v)$ be a subpath of Q, and consider the failure of any vertex $x \in Q(u,v)$. Let w be any descendant of v in T. The replacement path $P(s,w,x)$ would either skip

the entire suffix of $Q(u, v)$ starting from x or it definitely passes through v (see Figure 3).

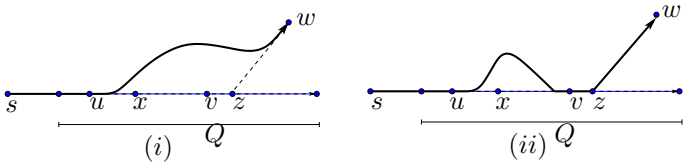

Figure 3: Two cases of $P(s, w, x)$ (shown thick) from perspective of subpath $Q(u, v)$.

In the first case, the replacement path to w can be computed by a single SPASS query on $Q(u, v)$. In the second case, using Observation 2.1, $P(s, w, x)$ can be seen as the concatenation of the replacement path $P(s, v, x)$ and the shortest path $P(v, w)$ already available in T. Based on these ideas, we now present a data structure $\mathcal{H}(Q)$ to solve replacement paths problem when the failing vertex lies on Q.

$\mathcal{H}(Q)$ is a balanced binary tree. The root node of $\mathcal{H}(Q)$ is associated with path Q itself. If Q has only 1 vertex, then $\mathcal{H}(Q)$ is a singleton tree. If Q has more than 1 vertex, we divide Q into 2 subpaths Q_1 and Q_2 with (nearly) equal number of vertices. So, $Q = Q_1 + Q_2$. In this case, $\mathcal{H}(Q)$ is a binary tree with $\mathcal{H}(Q_1)$ and $\mathcal{H}(Q_2)$ as its left child and right child respectively.

Let ν be any node in tree $\mathcal{H}(Q)$. Throughout the paper, we shall use ν synonymously with the subpath of Q associated with ν. We will denote the left child of ν in $\mathcal{H}$ by $\nu.L$, and its right child by $\nu.R$. See Figure 4 for a better understanding of $\mathcal{H}(Q)$ and these concepts.

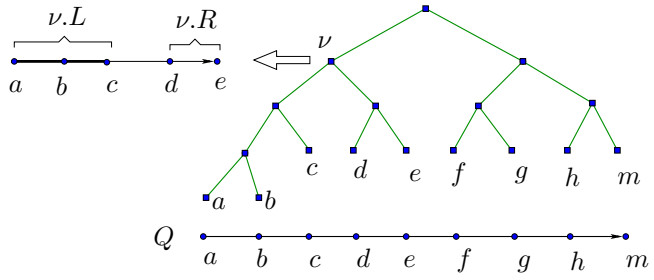

Figure 4: Binary tree $\mathcal{H}(Q)$ constructed on path Q.

Each node ν of $\mathcal{H}(Q)$ is augmented with the following information.

(a) A data structure ν.SPASS which solves SPASS problem on $\nu.L$. In particular, a query ν.SPASS(w, x), for any $x \in \nu.L$ and any $w \in$ FAMILY(ν, Q), will report the length of the shortest path from s to w avoiding the entire suffix of $\nu.L$ starting from x.

(b) For each $x \in \nu.L$, store $\delta(s, \text{LAST}(\nu.L), x)$, i.e., the distance from s to the last vertex of $\nu.L$ avoiding x. (Note that it will be undefined for $x = \text{LAST}(\nu.L)$).

We now describe an algorithm to report $\delta(s, w, x)$ for any failing vertex $x \in Q$ and destination vertex w. Without loss of generality assume that w is descendant of x in T (otherwise distance to w remains intact even after the failure of x). We introduce a notation here. Let PATH-PARENT(w, Q) denote the vertex of Q at which $P(s, w)$ leaves Q; for example, it is the vertex z in Figure 3. We first compute the *Lowest Common Ancestor* (LCA) of PATH-PARENT(w, Q) and x in $\mathcal{H}(Q)$. Let it be ν. The distance $\delta(s, w, x)$ can be retrieved from the data structure associated with ν as described by Algorithm 1. This algorithm is based on the ideas discussed above. In particular, the **else** part is captured precisely in Figure 3 with $\nu.L$ being the subpath $Q(u, v)$.

Let us analyse the running time of Algorithm 1. Notice that PATH-PARENT(w, Q) is the lowest common ancestor of w and LAST(Q) in tree T. Algorithm 1 makes two LCA queries - one on T and one on $\mathcal{H}(Q)$, which take $O(1)$ time (see [10]). In addition, it makes a single SPASS query which takes $O(\log n)$ time. Hence Algorithm 1 takes $O(\log n)$ time.

Algorithm 1: Reporting $\delta(s, w, x)$ where x is a failing vertex on path Q

$\nu \leftarrow$ LCA$(x,$ PATH-PARENT$(w, Q))$;
if $(x =$ LAST$(\nu.L))$ **then**
  return ν.SPASS(w, x)
else
  return $\min \begin{cases} \nu.\text{SPASS}(w, x) \\ \delta(s, \text{LAST}(\nu.L), x) + \delta(\text{LAST}(\nu.L), w) \end{cases}$

Let us discuss preprocessing of the data structure $\mathcal{H}(Q)$. Assuming that for each node of $\mathcal{H}(Q)$, part (a) has already been computed, we can compute part (b) for all nodes efficiently by pursuing a bottom-up approach as follows. Consider any node $\mu \in \mathcal{H}(Q)$. As part (b) for μ, we need to compute $\delta(s, \text{LAST}(\mu.L), x)$ for all $x \in \mu.L$. If we invoke Algorithm 1 with $w = \text{LAST}(\mu.L)$, it will query node $\nu = \text{LCA}(x, \text{LAST}(\mu.L))$. Notice that ν is descendant of μ in $\mathcal{H}(Q)$ and we have been pursuing a bottom-up approach. Therefore, the node ν is already equipped with part (a) and (b) of its information. Hence it will take $O(\log n)$ time to compute $\delta(s, \text{LAST}(\mu.L), x)$ for any $x \in \mu.L$. Thus all the nodes of $\mathcal{H}(Q)$ can be augmented with part (b) in $O(n \log^2 n)$ time. Leaving this overhead of $O(n \log^2 n)$, the main task in preprocessing of $\mathcal{H}(Q)$ is the computation of part (a), that is, ν.SPASS for each $\nu \in \mathcal{H}(Q)$. If we naively use the MSSP reduction to compute and store ν.SPASS for each $\nu \in \mathcal{H}$, then

ν.SPASS will require $O(|G(\text{FIRST}(\nu))| \log n)$ space and preprocessing time. This will lead to $\Theta(n^2)$ space and preprocessing time for the entire $\mathcal{H}(Q)$. In the following section, we use novel insights to show that SPASS data structures at deeper nodes in $\mathcal{H}(Q)$ can be built on smaller *proxy* graphs instead of the original input graph G. This leads to $O(n \log^3 n)$ bound on the space and the preprocessing time of $\mathcal{H}(Q)$.

5 Efficiently building the data structure $\mathcal{H}(Q)$

We begin with an overview of the MSSP algorithm of Klein [12] followed by the reduction of SPASS to MSSP.

5.1 Overview of mssp algorithm of Klein [12]

Given a directed planar graph G, and a face f in G, MSSP problem aims at preprocessing G to build a data structure that can efficiently answer queries of the type: report $\delta(s,t)$ for any $s \in f, t \in V$. Klein [12] designed an $O(n \log n)$ time algorithm to build an $O(n \log n)$ size data structure for the MSSP problem. The query time achieved is $O(\log n)$. Before giving an overview of this algorithm, we begin with some well known terminologies related to shortest path problem.

Let $\mathcal{T}$ be a tree rooted at a vertex s in G, and let $d_{\mathcal{T}}(v)$ denote the distance from s to vertex v in this tree. An edge $e = (x,y) \in E \backslash \mathcal{T}$ is called tense if $d_{\mathcal{T}}(y) > d_{\mathcal{T}}(x) + \omega(e)$. Tree $\mathcal{T}$ will be the shortest paths tree iff no (non-tree) edge is tense. Relaxation of a tense edge e in a tree $\mathcal{T}$ involves removing the edge incident to y in $\mathcal{T}$ and adding e to $\mathcal{T}$. Hence, the task of transforming any tree $\mathcal{T}$ into the shortest path tree can be achieved through a sequence of *relaxations* of the tense edges.

We now present a high-level description of MSSP algorithm of Klein [12]. Enumerate all vertices on f as s_1, s_2,...,$s_{|f|}$ in anti-clockwise direction. The algorithm starts with the shortest path tree T_1 rooted at s_1 and it constructs the rest of the trees T_i, $i > 1$ incrementally and implicitly. Given a shortest path tree T_{i-1}, the tree T_i is built in the following manner: Consider the source vertex making transition from s_{i-1} to s_i in T_{i-1}. As a result, there will be some non-tree edges in the graph which become tense. Carrying out relaxation until there is no tense edge left will provide T_i. The MSSP algorithm of Klein [12] computes the tense edges and carries out their relaxation in a *specific* order such that exactly those edges get relaxed which belong to $T_i \backslash T_{i-1}$. In this manner T_{i-1} gets transformed to T_i (see Figure 5).

Klein [12] exploited planarity crucially to show that during the entire *movement* of source from s_1 to $s_{|f|}$, every edge in the graph is relaxed at most once. All the trees T_i's can thus be expressed by the shortest path tree T_1 and a sequence of $O(n)$ edge relaxations as the source moves from s_1 to $s_{|f|}$. If an edge e gets relaxed

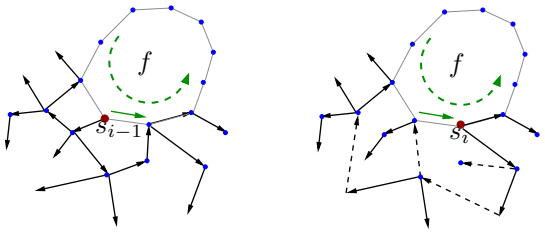

Figure 5: The dashed edges are the edges that get relaxed as the source moves from s_{i-1} to s_i.

while computing shortest path T_i from T_{i-1}, we record the relaxation of e as a pair (e,i) and call i as relaxation index of e. The sequence of edge relaxations is processed to build an $O(n \log n)$ space persistent data structure which implicitly stores all trees T_i. This data structure takes $O(\log n)$ time to report the distance of any vertex $v \in G$ from any vertex $s_i \in f$.

5.1.1 Reduction of spass to mssp

We now give a reduction from SPASS to MSSP. Let $Q = (s = x_0, x_1, \ldots, x_\tau = t)$ be a given shortest path in T. Given an edge $e = (x_i, y)$ with $e \notin Q$, we say that e lies to the left (similarly right) of Q if $i \in \{0, \tau\}$ or e occurs strictly between (x_i, x_{i+1}) and (x_{i-1}, x_i) in the counterclockwise (clockwise) order. Let R be the replacement path from s to a vertex v when some suffix subpath $Q(x_j, t)$ of Q fails. We say that R leaves Q from left (similarly from right) if the first edge on R which is not shared by Q lies to the left (right) of Q. We build data structure SPASS$_l$, to retrieve the paths R that leave Q from the left, by a reduction to MSSP problem as given below. A similar data structure SPASS$_r$ can be built to retrieve paths R that leave Q from the right.

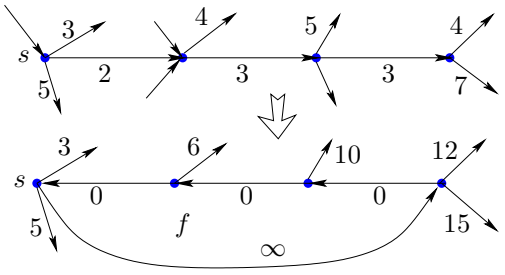

Figure 6: Reduction of SPASS$_l$ to MSSP

We build a graph G' by modifying G as follows (see Figure 6):

- Reverse the direction of all edges on the path Q and assign zero weight to each of them.
- For each vertex $x_i \in Q$, increase the weight of all its outgoing edges which lie to the left of Q by $\delta(s, x_i)$.
- Remove all other edges incident to vertices of Q.
- Add edge (s,t) of weight ∞. This creates a face f

which has all the vertices of Q on it.

Lemma 5.1 states the relationship between SPASS$_l$ on path Q in G and MSSP on face f in G'.

LEMMA 5.1. *Length of the shortest path from s to v avoiding $\langle x_{i+1}, \ldots, x_\tau \rangle$ in G which leaves Q from left is the same as the length of the shortest path from x_i to v in G'.*

Because of this direct relation between SPASS and MSSP we use them interchangeably in the later sections.

5.2 Ideas leading to linear space and preprocessing time for $\mathcal{H}(Q)$ For the sake of compactness, in the following discussion, we use $\mathcal{H}$ to denote $\mathcal{H}(Q)$, FAMILY(ν) to denote FAMILY(ν, Q), and $G(\nu)$ to denote $G($FIRST$(\nu))$.

We introduce a notation here. For a node $\nu \in \mathcal{H}$, define L-PARENT(ν) as the nearest ancestor μ of ν in $\mathcal{H}$ such that ν appears in the left subtree of μ. Conversely, ν is said to be a L-CHILD of μ. See Figure 7.

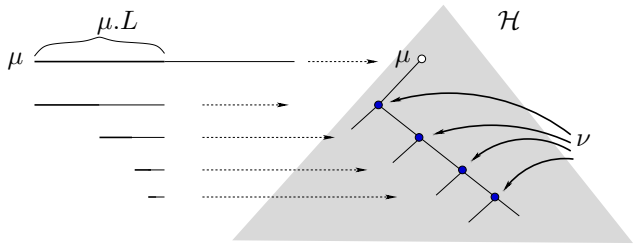

Figure 7: A node μ and its L-CHILDREN in $\mathcal{H}$.

Notice that ν is a suffix subpath of $\mu.L$ and FAMILY(ν) is a subset of FAMILY(μ). As a result, SPASS(μ) and SPASS(ν) have certain common features as follows. Consider any vertex $w \in$ FAMILY(ν) and $x \in \nu.L$ and let us analyse the structure of the path ν.SPASS(w, x). Note that μ.SPASS(w, x) avoids entire suffix of $\mu.L$ starting from x. However, ν.SPASS(w, x) can potentially use some portion of $\mu.L$; in particular, it can use $\nu.R$. This slight distinction between ν and μ leads us to the following observations. If the path ν.SPASS(w, x) does not use any vertex of $\nu.R$, then it is the same as μ.SPASS(w, x) (see Figure 8 (i)). If not, let v be the first vertex of the path ν.SPASS(w, x) which lies on $\nu.R$. One possibility is that the subpath from v to w uses only vertices of FAMILY(ν). This is shown in Figure 8 (ii). Otherwise, let a be the last vertex of the path that does not belong to FAMILY(ν). At first glance, it may appear that the portion of ν.SPASS(w, x) between v and a may be quite arbitrary. However, notice that the path $P(v, a)$, which is already in T, is intact even when any suffix subpath of $\nu.L$ fails. Therefore, using Observation 2.1 the subpath of ν.SPASS(w, x) from v to a is just $P(v, a)$. Figure 8 (iii) depicts this case.

Considering all these cases, observe that any ν.SPASS query can be answered by building SPASS data structure on a graph which, in addition to having all vertices and edges of FAMILY(ν), satisfies the following properties.

1. For all vertices $w \in$ FAMILY$(\nu) \backslash \nu.R$, it has the path μ.SPASS(w, x) for all $x \in \nu.L$. This will take care of ν.SPASS(w, x) shown in Figure 8 (i).
2. For vertices $w \in \nu.R$, it has the path μ.SPASS(y, x) for all y such that (y, w) is an edge in G. Note that y may not necessarily be present in FAMILY(ν). This property, along with property 1, will take care of ν.SPASS(w, x) shown in Figure 8 (ii).
3. For all edges (a, b) such that $a \notin$ FAMILY(ν) and $b \in$ FAMILY(ν), the graph has a path of length $\delta($LAST$(\nu), a) + \omega(a, b)$ from LAST(ν) to b. This property, along with property 1 and 2, will take care of ν.SPASS(w, x) shown in Figure 8 (iii).

For each ν, we construct a small sized graph $G^{reduced}(\nu)$, or $G^r(\nu)$ in short, which satisfies the above three properties. Before providing the complete details of construction of $G^r(\nu)$, we provide a sketch here. We start with $G^r(\mu)$ and sparsify it by keeping only those edges which are either incident to some vertex of the set FAMILY(ν) or belong to some path μ.SPASS(w, x), $x \in \nu.L$. This graph will satisfy the first two properties mentioned above. We then reduce its size by removing some vertices and shortcutting some paths by edges. We call this graph $G^*(\nu)$. Thereafter, we modify $G^*(\nu)$ and construct $G^r(\nu)$ so that it also satisfies the third property. A trivial way to construct $G^r(\nu)$ from $G^*(\nu)$ would be the following. For every edge (a, b) such that $a \notin$ FAMILY(ν) and $b \in$ FAMILY(ν), add an edge from LAST(ν) to b of required length. However, if we do so, the resulting graph may no longer be planar. To overcome this problem, we employ the bounded difference property of replacement paths in planar graphs (Theorem 3.1).

5.3 Construction of $G^r(\nu)$ We construct the graphs $G^r(\nu)$ in a top-down manner on the tree $\mathcal{H}$. The nodes of $\mathcal{H}$ which have no L-PARENT are the ones that represent some suffix subpath of Q. If ν is one such node, then we set $G^r(\nu) = G(\nu)$. For example, for $\nu = Q$, $G^r(\nu) = G$. For other nodes ν we assume that the three properties mentioned in the previous section hold for $G^r(\mu)$, where $\mu =$ L-PARENT(ν). We then construct $G^r(\nu)$ using $G^r(\mu)$, ensuring that the properties hold for $G^r(\nu)$. Thus, by induction, we ensure that for all $\nu \in \mathcal{H}$, $G^r(\nu)$ satisfies the required properties.

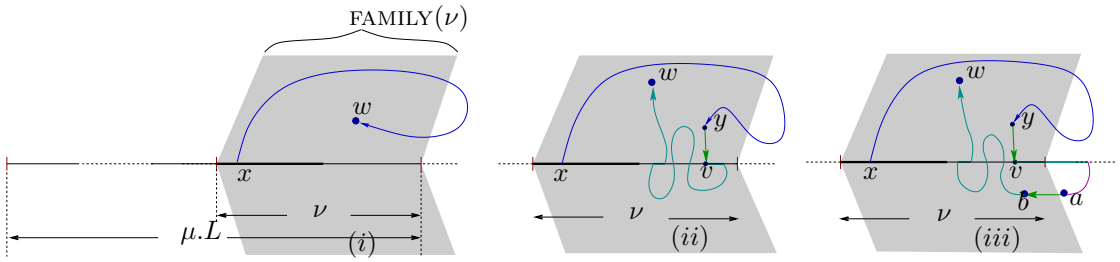

Figure 8: Possible structures of the path ν.SPASS(w, x).

5.3.1 Construction of $G^*(\nu)$ We require $G^*(\nu)$ to satisfy the first two properties mentioned above. Though $G^r(\mu)$ can serve as $G^*(\nu)$, we need a smaller subgraph for our purpose. Let $T^\mu(x)$ denote the shortest path tree rooted at s in the graph $G^r(\mu)\backslash R$, where R is the suffix subpath of $\mu.L$ starting at x. It is sufficient to have the edges of the set $\cup_{x\in\nu.L}T^\mu(x)$ only. These edges can also be viewed as: the edges of $T^\mu(\text{FIRST}(\nu))$ and the edges that get relaxed as the source moves from FIRST(ν) to LAST$(\nu.L)$ while executing MSSP on $G^r(\mu)$.

Let us assign red or blue color to the edges of $\cup_{x\in\nu.L}T^\mu(x)$. Assign red color to all those edges which have at least one end-point in FAMILY(ν). Also assign red color to all those edges that are relaxed when the source moves from FIRST(ν) to LAST$(\nu.L)$ during MSSP on $G^r(\mu)$. The remaining edges are colored blue. We assign colors to the vertices as follows. The end-points of all red edges are colored red and the remaining vertices are colored blue.

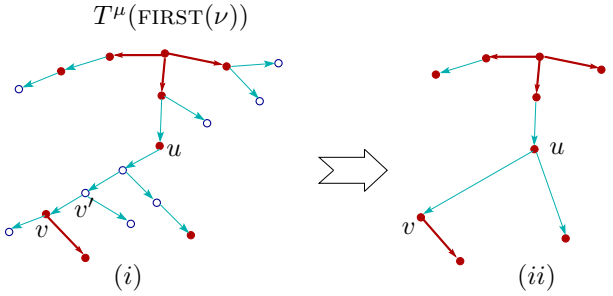

Figure 9: Transformation of $T^\mu(\text{FIRST}(\nu))$ to remove all blue vertices (shown hollow). As shown in (ii), only red vertices (shown solid) remain. The thick edges are red edges, the thin edges are blue edges.

It can be seen that the vertices that belong to or are neighboring to FAMILY(ν) will be colored red. So in order to compute $G^*(\nu)$ of small size, we modify $\cup_{x\in\nu.L}T^\mu(x)$ such that distance between any two red vertices is preserved, but all blue vertices are removed. Notice that each blue edge is an edge of the tree $T^\mu(\text{FIRST}(\nu))$. So we first remove all blue edges and then connect every red vertex with its nearest red ancestor in the tree $T^\mu(\text{FIRST}(\nu))$ (if not already connected) by a new blue edge whose length is equal to the distance between them in the tree. After this procedure, the blue vertices are isolated and can be removed. See Figure 9.

5.3.2 Transforming $G^*(\nu)$ into $G^r(\nu)$ We first introduce some notations. An edge $e = (a, b)$ is a crossing edge if $a \notin$ FAMILY(ν) and $b \in$ FAMILY(ν), and a reverse crossing edge if $a \in$ FAMILY(ν) and $b \notin$ FAMILY(ν). Consider an imaginary line $\mathcal{L}$ passing through LAST(ν) which splits every crossing and reverse crossing edge. $\mathcal{L}$ will act as a separator for $G^*(\nu)$, separating the vertices of FAMILY(ν) from those that lie beyond family (see Figure 10). Starting from vertex $v_0 =$ LAST(ν), we traverse along $\mathcal{L}$ in any one direction (the same will be done in the other direction as well). Suppose we encounter edges $e_1 = (a_1, b_1), e_2 = (a_2, b_2), \ldots$, in that order, during the traversal. For each edge e_l, do the following:

- At the point of intersection of e_l with $\mathcal{L}$, create a new vertex v_l and add an edge from v_{l-1} to v_l. Remove the edge (a_l, b_l) and add edges (a_l, v_l) and (v_l, b_l).
- Assign weights to the new edges as follows.
  (i) $\omega(v_{l-1}, v_l) = \delta(s, a_l) - \delta(s, a_{l-1})$.
  (ii) $\omega(a_l, v_l) = 0$.
  (iii) $\omega(v_l, b_l) = \omega(e_l)$.

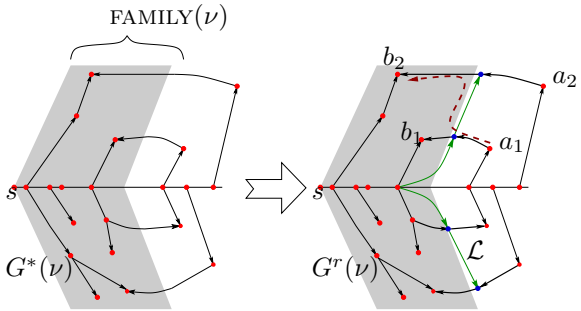

Figure 10: Transforming $G^*(\nu)$ into $G^r(\nu)$ by adding new vertices along (reverse) crossing edges. The dashed edge shows the false path added between a_1 and b_2.

The transformation described above ensures the following. If e_l is a crossing edge, then the length of the

new path from $\text{LAST}(\nu)$ to b_l will be $\delta(\text{LAST}(\nu), a_l) + \omega(e_l)$. Hence $G^r(\nu)$ satisfies the third property mentioned earlier. Furthermore, if e_l is a reverse crossing edge, the new path from $\text{LAST}(\nu)$ to b_l will have length no less than $\delta(s, b_l) - \delta(s, \text{LAST}(\nu)) = \delta(\text{LAST}(\nu), b_l)$.

However, a problem with this transformation is that it introduces some "false paths" in the resulting graph: For every pair of edges e_{l_1} and e_{l_2}, $l_1 < l_2$, now there is a new path from a_{l_1} to b_{l_2} which was not present in the original graph (see Figure 10). We have to ensure that this "false path" does not affect the shortest path avoiding suffix path from s to the vertex b_{l_2}. The bounded difference property of replacement paths (Theorem 3.1) proves to be crucial here as follows.

Notice that in $G^r(\nu)$, e_{l_2} is not present in the shortest path tree rooted at s, and $P(s, a_{l_2})$, e_{l_2}, and $P(s, b_{l_2})$ form a separator in $G^r(\nu)$ that separates s and a_{l_1}. Thus, by bounded difference property of replacement paths (Theorem 3.1), for any path Z in T not containing s and b_{l_2},

$$\delta(s, b_{l_2}, Z) - \delta(s, a_{l_1}, Z) \leq \delta(s, a_{l_2}) + \omega(e_{l_2}) - \delta(s, a_{l_1})$$

It can be verified that the RHS of the above inequality is the same as the length of the "false path" introduced between a_{l_1} and b_{l_2}. Hence, the presence of the "false path" in $G^r(\nu)$ will not alter $\delta(s, b_{l_2}, Z)$ for any Z.

5.4 Analysis We first provide a bound on $\sum_{\nu \in \mathcal{H}} \sum_{v \in \text{FAMILY}(\nu)} deg(v)$. Any vertex v will belong to family of at most one node at any level of $\mathcal{H}$ because whenever a node $\nu \in \mathcal{H}$ is partitioned into two halves L and R, every vertex $v \in \text{FAMILY}(\nu)$ goes either into the family of $\nu.L$ or into family of $\nu.R$, but not both. There are $O(\log n)$ levels in $\mathcal{H}$, hence v belongs to family of at most $\log n$ nodes. Therefore,

$$\sum_{\nu \in \mathcal{H}} \sum_{v \in \text{FAMILY}(\nu)} deg(v) \leq \log n \sum_{v \in G} deg(v) = O(n \log n) \tag{5.3}$$

Now, we provide a bound on the sum of sizes of $G^r(\nu)$ for all $\nu \in \mathcal{H}$. Consider any edge of $G^r(\nu)$. It is of one of the 4 types:

1. A red edge that was incident to some vertex $v \in \text{FAMILY}(\nu)$. The number of such edges is at most the sum of degrees of all the vertices of $\text{FAMILY}(\nu)$. Such edges will be called red edges of type 1. The total number of such edges summed over all $G^r(\nu)$ is $O(n \log n)$ due to Equation 5.3.
2. A red edge that was not incident to any vertex of $\text{FAMILY}(\nu)$. This edge must be one of those edges that were relaxed when the source moved from $\text{FIRST}(\nu)$ to $\text{LAST}(\nu.L)$ during MSSP on $G^r(\mu)$ where $\mu = \text{L-PARENT}(\nu)$. We call it red edge of type 2.
3. A blue edge. Note that at most one blue edge will be incident to any vertex. Moreover, any blue edge is between two red vertices, therefore, we can safely say that number of blue edges is no more than number of red edges. We will ignore counting blue edges henceforth.
4. An edge that was added when $G^r(\nu)$ was constructed from $G^*(\nu)$. Only 3 new edges are added for every edge whose one endpoint lies in $\text{FAMILY}(\nu)$. The number of such edges can be bounded by $O(n \lg n)$ due to equation 5.3. These edges will also be called as red edges of type 1.

We only need to bound the total number of red edges of type 2. Any red edge of type 2 in $G^r(\nu)$ must be present in $G^r(\mu)$. If it was a red edge of type 2 in $G^r(\mu)$, then it must have been *passed* from μ's L-PARENT. Otherwise, it was either a blue edge or a red edge of type 1 in $G^r(\mu)$. We can ensure that no blue edge of $G^r(\mu)$ gets passed down from μ to ν as a red edge of type 2 in $G^r(\nu)$ (see Appendix for details). In that case, every red edge of type 2 in $G^r(\nu)$ must have been a red edge of type 1 in a graph $G^r(\lambda)$ for some ancestor λ of ν; and it must have been "passed on" from there to $G^r(\nu)$ through a series of "L-ANCESTORS" of ν such that it was a red edge of type 2 at all those ancestors. This suggests that for each red edge of type 1 at a node in $\mathcal{H}$, we need to count the number of its descendants in $\mathcal{H}$ where it appears as a red edge of type 2.

Consider a node μ of $\mathcal{H}$ and let $\nu_1, \nu_2, \ldots, \nu_k$ be its L-CHILDREN (see Figure 7). Consider any red edge e in $G^r(\mu)$. Recall that while executing MSSP on $G^r(\mu)$, an edge gets relaxed at most once. Therefore, if $x \in \mu.L$ be such that e got relaxed while computing $T^\mu(x)$ and $x \in \nu_i.L$, then e will be present as a red edge of type 2 only in $G^r(\nu_i)$. Hence, any red edge of type 1 present in $G^r(\mu)$ will be passed on to at most one of its L-CHILDREN as a red edge of type 2. Also, this L-CHILD will in turn pass it on to at most one of its L-CHILDREN and this sequence will go on till we reach a leaf of the tree $\mathcal{H}$. So any red edge of type 1 present at μ will be present as a red edge of type 2 at no more than $\log n$ nodes of $\mathcal{H}$. Hence, the total number of red edges of type 2, summed over all nodes of $\mathcal{H}$ will be at most $\log n$ times the total number of red edges of type 1 summed over all nodes of $\mathcal{H}$. Hence, $\sum_{\nu \in \mathcal{H}} |G^r(\nu)| = O(n \log^2 n)$. The SPASS data structure on a n vertex graph occupies $O(n \log n)$ space, so the total space requirement of data structure associated with $\mathcal{H}(Q)$ is $\sum_{\nu \in \mathcal{H}} O(|G^r(\nu)| \log n) = O(n \log^3 n)$. Building SPASS on n vertex graph takes $O(n \log n)$ time. $G^r(\nu)$ can be built from $G^r(\mu)$ in $O(|G^r(\mu)|)$ time. However, a node μ is L-PARENT of at most $\log n$ nodes of $\mathcal{H}$, so the

preprocessing time of $\mathcal{H}(Q)$ is:

$$\sum_{\nu \in \mathcal{H}, \mu = \text{L-PARENT}(\nu)} O(|G^r(\nu)| \log n + |G^r(\mu)|) = O(n \log^3 n)$$

We can thus state the following theorem.

THEOREM 5.1. *For any path Q present in the shortest path tree T rooted at vertex s in G, a data structure of $O(n \log^3 n)$ size can be built in $O(n \log^3 n)$ time that can report $\delta(s, v, x)$ in $O(\log n)$ time for any $x \in Q, v \in V$.*

6 Data structure for the single source replacement paths problem

In order to solve the single source replacement paths problem, we use Theorem 5.1 and pursue a divide and conquer approach. We compute a path Q in tree T using heavy path decomposition technique given by Sleator and Tarjan [18] as follows. Starting from s, we traverse down the tree T and keep extending Q along that edge from which the largest (in terms of no. of nodes) subtree hangs. We stop when we reach a leaf node. The path Q created in this manner has the following nice property.

Let $v_1, ..., v_k$ be the roots of the subtrees of T connected to the path Q with an edge. Each of $T(v_i)$'s are disjoint and each $T(v_i)$ has fewer than $n/2$ vertices.

First we build a data structure for handling failure of any vertex from Q according to Theorem 5.1. Then, for each $1 \leq i \leq k$, we recursively solve the single source replacement paths problem for $G(v_i)$ (see Definition 2.2). It follows from the property mentioned above that $G(v_i)$ has at most $n/2$ vertices. So the depth of the recursion will be $O(\log n)$ only. We can thus state the following theorem.

THEOREM 6.1. *Given a directed planar graph $G = (V, E)$ on n vertices and a source vertex s, a data structure of $O(n \log^4 n)$ size can be built in $O(n \log^4 n)$ time which can report $\delta(s, v, x)$ in $O(\log n)$ time for any $v, x \in V$.*

7 Conclusion

We presented a single source distance oracle for planar digraphs avoiding any single failed vertex. Its size and construction time are optimal up to poly-logarithmic factors. It can be adapted to handle edge failure as well by introducing a vertex at the center of each edge. We would like to conclude with an open problem: Is it possible to design a compact distance oracle for a planar digraph which can handle multiple failures ?

References

[1] S. Baswana and N. Khanna. Approximate shortest paths avoiding a failed vertex: Optimal size data structures for unweighted graphs. In *STACS*, pages 513–524, 2010.

[2] A. Bernstein. A nearly optimal algorithm for approximating replacement paths and k shortest simple paths in general graphs. In *SODA*, pages 742–755, 2010.

[3] A. Bernstein and D. Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *STOC*, pages 101–110, 2009.

[4] C. Demetrescu, M. Thorup, R. A. Chowdhury, and V. Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008.

[5] H. N. Djidjev. Efficient algorithms for shortest path queries in planar digraphs. In *WG*, pages 151–165, 1996.

[6] R. Duan and S. Pettie. Dual-failure distance and connectivity oracles. In *SODA*, pages 506–515, 2009.

[7] Y. Emek, D. Peleg, and L. Roditty. A near-linear time algorithm for computing replacement paths in planar directed graphs. In *SODA*, pages 428–435, 2008.

[8] J. Erickson and A. Nayyeri. Computing replacement paths in surface embedded graphs. In *SODA*, pages 1347–1354, 2011.

[9] G. N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987.

[10] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

[11] J. Hershberger, S. Suri, and A. Bhosle. On the difficulty of some shortest path problems. *ACM Transaction on Algorithms*, 3:123–139, 2007.

[12] P. N. Klein. Multiple-source shortest paths in planar graphs. In *SODA*, pages 146–155, 2005.

[13] P. N. Klein, S. Mozes, and O. Weimann. Shortest paths in directed planar graphs with negative lengths: a linear-space O(n log^2 n)-time algorithm. *ACM Transactions on Algorithms*, 6(2), 2010.

[14] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. App. Math.*, 36(2):177–189, 1979.

[15] K. Malik, A. K. Mittal, and S. K. Gupta. The k most vital arcs in the shortest path problem. *Operation Research Letters*, 4:223–227, 1989.

[16] E. Nardelli, G. Proietti, and P. Widmayer. Finding the most vital node of a shortest path. *Theor. Comput. Sci.*, 296(1):167–177, 2003.

[17] L. Roditty and U. Zwick. Replacement paths and k-simple shortest paths in unweighted directed graphs. In *ICALP*, pages 249–260, 2005.

[18] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *JCSS*, 26:362–391, 1983.

[19] M. Thorup. Fortifying OSPF/IS-IS against link-failure. manuscript, 2001.

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

[21] C. Wulff-Nilsen. Solving the replacement paths problem for planar directed graphs in $O(n \log n)$ time. In *SODA*, pages 756–765, 2010.

Appendix

Avoiding the passing of blue edges as red edge of type 2

First we describe the situation under which a blue edge in $G^r(\nu)$ could be passed as a red edge of type 2 to some L-CHILD of ν. Consider an edge (u, v) in $G^r(\nu)$ which is either a red edge of type 2 or a blue edge. Notice that both u and v lie outside FAMILY(ν). The construction of $G^*(\nu)$ from $G^r(\mu)$ implies that there exists at least one vertex, say x, in $\nu.L$ such that (u, v) belongs to $T^\mu(x)$; let w be the first such vertex from $\nu.L$. If $w =$ FIRST(ν), then (u, v) is a blue edge, otherwise it will be a red edge of type 2. In the latter case, it means that (u, v) gets relaxed in the shortest path tree when the source vertex moves to w while executing MSSP in graph $G^*(\nu)$. The graph $G^*(\nu)$ is later transformed into $G^r(\nu)$. Notice that difference between $G^r(\nu)$ and $G^r(\mu)$ is that the vertices of $\nu.R$ can be used by any suffix avoiding shortest path. This can *prepone* or *postpone* the relaxation of edge (u, v) while executing MSSP on $G^r(\nu)$ as follows : when the source reaches vertex w, the shortest path to v could pass through some vertices of $\nu.R$ (see Figure 11). However, as the source further moves down $\nu.L$, on reaching some vertex w', a shorter path to v is available which uses the edge (u, v). It will be at this point that the edge (u, v) would get relaxed. So the relaxation of edge (u, v) gets postponed (in a similar fashion, it can be preponed). If (u, v) was a blue edge in $G^r(\nu)$, then this postponing of relaxation of (u, v) will enforce it to be passed on as a red edge of type 2 to some of the L-CHILDREN of ν. However, this can be avoided altogether as discussed below.

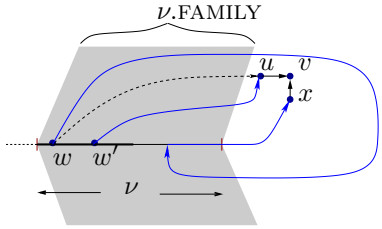

Figure 11: Relaxation of edge (u, v) gets postponed due to presence of $\nu.R$ in $G^r(\nu)$.

Though MSSP on $G^r(\nu)$ solves SPASS for destination vertices lying in $G^r(\nu)$, actually we need to solve SPASS for destination vertices from FAMILY(ν) only. This key observation motivates us to take the following simple-minded approach. For each blue edge or red edge of type 2 whose relaxation got preponed or postponed in $G^r(\nu)$, we restore its relaxation index back to the index as specified by $G^r(\mu)$ (see Subsection 5.1). As a result, a blue edge at ν will be passed only as a blue edge to its L-CHILDREN in $\mathcal{H}$. Interestingly this restoration of the relaxation indices does not affect ν.SPASS(w, x) for any $w \in$ FAMILY(ν) due to the following reason. If ν.SPASS(w, x) does not pass through any vertex of $\nu.R$, then ν.SPASS(w, x) is the same as μ.SPASS(w, x). In this case, the restoration of relaxation index of any edge does not affect ν.SPASS(w, x). If ν.SPASS(w, x) indeed passes through some vertex of $\nu.R$, then ν.SPASS(w, x) will not even pass through any blue or red edge of type 2 (see Figure 8).

All-pairs replacement paths in planar digraphs

Lipton and Tarjan [14] showed that every planar graph G on n vertices has a set of $O(\sqrt{n})$ vertices whose removal splits the graph into connected components each of maximum size $2n/3$. Such a set S is called a *balanced separator* of G. This result can be exploited to construct a tree data structure $T(G)$ as follows. First compute a balanced separator S of G. Let $G_1, ..., G_k$ be the connected components (ignoring direction of the edges) of $G \backslash S$. The root node of $T(G)$ stores G and S, and its children are $T(G_i)$'s built recursively for each $1 \leq i \leq k$. For any node α in $T(G)$, let G_α be S_α respectively the subgraph and the separator stored in it. It follows from the construction of $T(G)$ that for each vertex $v \in V$, there is a unique node $\alpha \in T(G)$ such that $v \in S_\alpha$. Also observe that $T(G)$ has $O(\log n)$ height. The data structure for all-pairs replacement paths problem is obtained by a suitable augmentation of $T(G)$ as follows.

For each vertex $s \in S_\alpha$, we build a data structure $D(s)$ for the single source replacement paths in G_α, and also a similar data structure, $D^{\mathrm{R}}(s)$, in the graph obtained by reversing the edge directions in G_α. Notice the following key observation about this data structure. For any $u, v, x \in V$, there exists a unique node α in $T(G)$ such that $P(u, v, x)$ is present in G_α and it passes through some vertex $s \in S_\alpha$. Hence the data structures $D(s)$ and $D^{\mathrm{R}}(s)$ together store $P(u, v, x)$. In particular, $\delta(s, v, x) + \delta^{\mathrm{R}}(s, u, x) = \delta(u, v, x)$. We can use this observation to answer the query of $\delta(u, v, x)$ as follows.

For all nodes β from LCA(u, v) to the root in $T(G)$, and for each $s \in S_\beta$, we query $D(s)$ and $D^{\mathrm{R}}(s)$ to compute $\delta(s, v, x) + \delta^{\mathrm{R}}(s, u, x)$, and report the minimum. The total number of such queries will be $O(\sqrt{n})$, and each query takes $O(\log n)$ time. Thus it takes $O(\sqrt{n} \log n)$ time to compute $\delta(u, v, x)$.

The space occupied by the data structure described above is $O(n^{3/2} \log^4 n)$. This can be easily generalized to achieve $O(S)$ space and $O(n^2/S \operatorname{polylog} n)$ query time for any $S \in [n^{3/2}, n^2]$ using ϵ-division of planar graphs designed by Frederickson [9].