Improving Robustness of Next-Hop Routing*

Glencora Borradaile, W. Sean Kennedy, Gordon Wilfong, Lisa Zhang

July 13, 2021

Abstract

A weakness of next-hop routing is that following a link or router failure there may be no routes between some source-destination pairs, or packets may get stuck in a routing loop as the protocol operates to establish new routes. In this article, we address these weaknesses by describing mechanisms to choose alternate next hops.

Our first contribution is to model the scenario as the following TREE AUGMENTATION problem. Consider a mixed graph where some edges are directed and some undirected. The directed edges form a spanning tree pointing towards the common destination node. Each directed edge represents the unique next hop in the routing protocol. Our goal is to direct the undirected edges so that the resulting graph remains acyclic and the number of nodes with outdegree two or more is maximized. These nodes represent those with alternative next hops in their routing paths.

We show that TREE AUGMENTATION is NP-hard in general and present a simple $\frac{1}{2}$-approximation algorithm. We also study 3 special cases. We give exact polynomial-time algorithms for when the input spanning tree consists of exactly 2 directed paths or when the input graph has bounded treewidth. For planar graphs, we present a polynomial-time approximation scheme when the input tree is a breadth-first search tree. To the best of our knowledge, TREE AUGMENTATION has not been previously studied.

1 Introduction

In an Internet Protocol (IP) network the header of each packet contains the intended destination for that packet. Each router in the network has a *forwarding table* in which, for each destination, there is a corresponding entry stating which port through which to send out packets. This type of routing is called *next-hop routing*. Since the routings for different destinations are independent, without loss of generality, we will herein talk about routing to one particular destination d. A number of different routing protocols are available for populating routing tables with destination-port pairs. These protocols are called *interior gateway protocols* (IGPs); see References [8, 9, 10] for detailed operations of various of these IGPs.

For our purposes, an IP network is modeled as a graph, each node representing a router and each edge representing a link between routers. The entry in the routing table for destination d at router v is a pair (d, vw) where vw is a directed edge to a neighboring router w of v. The goal of an IGP is to design routing tables so that the union of these edges is a tree directed towards d; we call such a table *valid*. The unique directed path from a router v to destination d in this tree implies a *route* for a message from v to d.

A weakness of next-hop routing is that following a link or router failure there may be no route from some source router to d. While a routing protocol operates to establish new routes (by way of new router table entries), packets may get stuck in a routing loop. The *speed of convergence*, i.e. the time required to recover from such a failure in the network topology, is of fundamental importance in modern IGPs. Indeed, real-time applications such as voice-over IP require quick failure recovery.

There have been many different methods proposed to recover from failures; see References [7] and [11] for surveys. We address a requirement for one such method, *Permutation Routing* [12]. In permutation routing,

*This material is based upon work supported by the National Science Foundation, under Grant No. CCF-0964037. Sean Kennedy is partially supported by a postdoctoral fellowship from the Natrual Sciences and Engineering Research Council of Canada.

there may be multiple entries for destination d at router v. The route used by a message will use the first entry that corresponds to a link or neighbor that is not currently in failure. To avoid routing loops, the union of the edges given by these entries should be a directed acyclic graph with d as the only sink. So long as all the neighbors or edges listed in a routers table do not *all* fail, the tables will imply at least one route to d. In order to implement permutation routing, we start with a network of communication connections between routers and single-entry routing tables corresponding to a tree, often a shortest-path or breadth-first-search (BFS) tree of the graph. We use the edges of the network to add entries to the router tables, thus adding the resiliency required for permutation routing. As we show in this paper, it is the choice of how to add these entries that proves to be challenging.

In this paper, we study a graph theoretic problem which exactly models this issue. Our contribution is two-fold. From the networking perspective, we are the first to offer a rigorous treatment of this approach for improving next-hop routing robustness. From the graph theoretic perspective, we introduce a crisply-stated graph problem that appears to be novel.

1.1 The TREE AUGMENTATION problem

We formally define the problem in graph terms. The input is a mixed graph $G = (V, E \cup \overrightarrow{T})$, where V is the set of nodes, E is a set of undirected edges and $\overrightarrow{T}$ is a set of directed edges that form a directed spanning tree pointing towards the destination node d. (Such a rooted, directed spanning tree is in fact an arborescence; we use the term tree for convenience.) The edges of $\overrightarrow{T}$ give the unique next-hop for the initial routing paths. Our goal is to find an orientation of the undirected edges E so that the resulting directed graph is acyclic and *maximizes* the number of nodes with outdegree at least two. Those nodes that, in the resulting orientation, have outdegree at least two represent those routers with an alternate next hop when a failure occurs. We refer to this problem as TREE AUGMENTATION.

Herein, an *edge* uv refers to an undirected edge between nodes u and v and $\overrightarrow{uv}$ and $\overleftarrow{uv}$ refer to directed edges, or arcs, from u to v and v to u, respectively. For $E' \subseteq E$ we use $\overrightarrow{E'}$ to denote the set of arcs corresponding to some orientation of E'; the details of that orientation will be given by context. We say a node $u \in V$ is *covered* by an orientation if there is an arc $\overrightarrow{uv} \in \overrightarrow{E'}$. Our objective is to maximize *the number of covered nodes* over all possible orientations of subsets E' of E such that $(V, \overrightarrow{E'} \cup \overrightarrow{T})$ is an *acyclic* graph. Note that the problem statement does not require every edge in E be directed in the solution. However, we will see that it never hurts to direct every edge in E.

We also consider a weighted version of TREE AUGMENTATION, for which some nodes may be viewed as more important to be covered. Here, a positive weight $w(v)$ is given for each node v and the objective becomes maximizing the total weight of covered nodes.

1.2 Organization

In Section 2 we present two observations and give simple a $\frac{1}{2}$-approximation for TREE AUGMENTATION. In Section 3 we prove the NP-hardness of TREE AUGMENTATION via a reduction from the SET COVER problem. The remainder of the paper gives positive results for some interesting special cases. In Section 4, we describe a polynomial-time algorithm for graphs of bounded treewidth. In Section 5, we describe a polynomial-time approximation scheme for planar graphs when T is a BFS tree. In Section 6, we describe a polynomial-time algorithm for the special case in which the spanning tree consists of exactly two directed paths. We point out that all our algorithmic results generalize to the weighted case.

2 Observations and a simple approximation

We first note that, algorithmically, the acyclicity constraint is what makes this problem challenging. Consider the connected components of the graph $G = (V, E)$. In each component we want to orient the edges so that each node is covered. If a connected component is a tree, root this tree at an arbitrary node and orient every

edge towards the chosen root. In this way, every node except the root is covered. Further, it is not possible for every node in a tree to be covered. If a connected component is not a tree, we begin with an arbitrary spanning tree S. There must exist one node incident to an edge not in S. We root the spanning tree S at this node and orient every edge in S towards the root. Since the root is incident to an edge outside S, we orient this edge away from the root. All other edges outside S can be oriented arbitrarily. In this way, every node of the connected component is covered. Since all these operations are no more difficult that depth-first search, we get:

Observation 1. *Finding an orientation of E to maximize the number of covered vertices can be done in linear time.*

However, our guiding application requires that we find an acyclic orientation. Suppose we have oriented $E' \subseteq E$ such that $(V, \overrightarrow{T} \cup \overrightarrow{E'})$ is acyclic. Consider a topological ordering of the vertices of this acyclic graph and consider any edge $uv \in E \setminus E'$ in which, without loss of generality, u is before v in the topological ordering. Then $(V, \overrightarrow{T} \cup \overrightarrow{E'} \cup \overrightarrow{uv})$ is also acyclic. Augmenting $(V, \overrightarrow{T} \cup \overrightarrow{E'})$ in this way ensures that u is covered while not affecting the coverage of any other vertex. Repeating this for every edge of $E \setminus E'$ gives:

Observation 2. *Given an orientation of a subset of E' of the non-tree edges E, one can always orient the remaining edges $E \setminus E'$ while maintaining acyclicity and without decreasing the objective.*

Consider the bipartition of E into the set of *back edges* B (edges $uv \in E$ such that u is a descendent of v in $\overrightarrow{T}$) and the *cross edges* C (edges uv in E such that u and v have no ancestor/descendent relationship). Each edge of B can only be oriented in one way, from descendent to ancestor, without introducing a cycle. We let $\overrightarrow{B}$ denote this orientation. Consider two orientations of C: let $\overrightarrow{C}$ be the orientation such that each edge $uv \in C$ is oriented from low to high pre-order (the order given by the first time a vertex is visited by a depth-first search) and let $\overleftarrow{C}$ be the reverse of this orientation (it is, in fact, the order such that each edge $uv \in C$ is oriented from low to high *post*-order).

Lemma 3. *$(V, \overrightarrow{T} \cup \overrightarrow{B} \cup \overrightarrow{C})$ and $(V, \overrightarrow{T} \cup \overrightarrow{B} \cup \overleftarrow{C})$ are acyclic.*

Proof. Trivially, $(V, \overrightarrow{T} \cup \overrightarrow{B})$ is acyclic.

For a contradiction, suppose there is a directed cycle $\overrightarrow{D}$ in $(V, \overrightarrow{T} \cup \overrightarrow{B} \cup \overrightarrow{C})$. Let $\overrightarrow{D}'$ be the cycle obtained from $\overrightarrow{D}$ by contracting the edges in $\overrightarrow{D}$ that are not in $\overrightarrow{C}$ (i.e. $\overrightarrow{D}' \subseteq \overrightarrow{C}$. Let $u_1, u_2, \ldots, u_\ell$ be the vertices of $\overrightarrow{D}'$ in order, starting from an arbitrary vertex u_1 of $\overrightarrow{D}'$; we have that $\ell \geq 2$ for otherwise the edges of $\overrightarrow{D}$ could not have a consistent direction.

By the definition of the orientation of $\overrightarrow{u_i u_{i+1}}$, u_i must appear before u_{i+1} in the pre-order used for each $i = 1, \ldots, \ell - 1$. Likewise, by the definition of the orientation of $\overrightarrow{u_\ell u_1}$, u_ℓ must appear before u_1 in the pre-order used, hence a contradiction.

Likewise, reversing the direction of the edges in $\overrightarrow{C}$ results in an acyclic graph $(V, \overrightarrow{T} \cup \overrightarrow{B} \cup \overleftarrow{C})$. □ □

We can view these orientations in the following way: embed $\overrightarrow{T}$ in a non-crossing way with the root at the top and all edges directed upward; consider a DFS traversal that explores the branches from left to right; $\overrightarrow{C}$ is the orientation in which all the edges of C are oriented from left to right in this embedding and $\overleftarrow{C}$ is the orientation in which all the edges of C are oriented from right to left in this embedding. These orientations are illustrated in Figure 1(c) and (d), respectively. We use this observation to design a simple $\frac{1}{2}$-approximation to the TREE AUGMENTATION problem; that is, the algorithm is guaranteed to return an orientation that covers at least a half of the number of vertices that can be covered in an optimal orientation.

Theorem 4. *The better of the two orientations $\overrightarrow{B} \cup \overrightarrow{C}$ and $\overrightarrow{B} \cup \overleftarrow{C}$ gives a $\frac{1}{2}$-approximation to the* TREE AUGMENTATION *problem.*

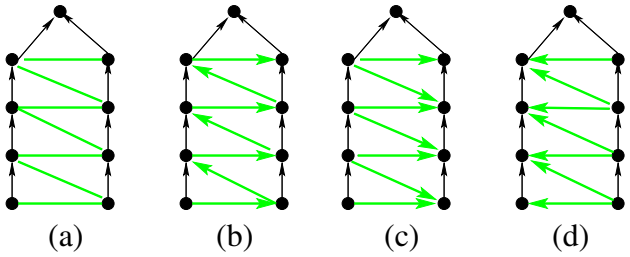


Figure 1: The optimal solution (b) to the input problem (a) covers $n-2$ vertices whereas the left-to-right (c) and right-to-left (d) orientations cover $\frac{n-1}{2}$ vertices each.

Proof. Let V_B be the subset of vertices that are covered by $\overrightarrow{B}$. Let V_X be the subset of vertices that cannot be covered by any orientation (namely, those vertices that are not endpoints of edges in E). The remaining vertices $V_C = V \setminus (V_B \cup V_X)$ are the endpoints of the edges in C. Each vertex in V_C is covered either by $\overrightarrow{C}$ or $\overleftarrow{C}$ (or both). Let $x(\overrightarrow{C})$ and $x(\overleftarrow{C})$, be the number of vertices covered by $\overrightarrow{C}$ and $\overleftarrow{C}$, respectively. We have $x(\overrightarrow{C}) + x(\overleftarrow{C}) \geq |V_C|$. It follows that the number of vertices covered by the better of these two orientations has value:

$$\begin{aligned} \max\left\{|V_B| + x(\overrightarrow{C}), |V_B| + x(\overleftarrow{C})\right\} &\geq \frac{1}{2}\left(2|V_B| + x(\overrightarrow{C}) + x(\overleftarrow{C})\right) \\ &\geq |V_B| + \frac{1}{2}|V_C| \geq \frac{1}{2}|V \setminus V_X| \end{aligned}$$

Since the maximum number of vertices that any orientation can cover is $|V \setminus V_X|$, the better of the two orientations $\overrightarrow{B} \cup \overrightarrow{C}$ and $\overrightarrow{B} \cup \overleftarrow{C}$ is a $\frac{1}{2}$-approximation to the TREE AUGMENTATION problem. The example in Figure 1 (b) illustrates that this analysis is asymptotically tight. □ □

The $\frac{1}{2}$-approximation algorithm generalizes to the weighted case, for which each node v carries a weight $w(v)$. In the above proof we can simply replace the size of a node set with the total weight from the set.

Corollary 5. WEIGHTED TREE AUGMENTATION *admits a* $\frac{1}{2}$*-approximation.*

3 TREE AUGMENTATION is NP-hard

In this section, we prove the following.

Theorem 6. TREE AUGMENTATION *is NP hard.*

Our reduction is from the well-known NP-hard SET COVER problem [6]. An instance of SET COVER consists of a set of elements $X = \{x_1, x_2, \ldots, x_n\}$, a collection of subsets of X, $\mathcal{C} = \{S_1, S_2, \ldots, S_m\}$ and an integer $0 < k \leq m$. The SET COVER problem asks: Is there a subcollection $\mathcal{C}' \subseteq \mathcal{C}$ such that $|\mathcal{C}'| \leq k$ and $\cup_{S_i \in \mathcal{C}'} S_i = X$?

We prove Theorem 6 in three steps. First, we start by describing the gadget which models SET COVER as an instance of TREE AUGMENTATION. Second, we describe the intuition behind our result. Finally, we give the formal details of correctness.

The Reduction

We build an instance $G = (V, E \cup \overrightarrow{T})$ of TREE AUGMENTATION corresponding to an instance of SET COVER as follows. The construction is illustrated in Figure 2.

Vertices V: The root vertex is u. For each set S_j $(j = 1, \ldots, m)$ we define 5 vertices $r_j, s_j, t_j, \ell_j, p_j$. We call the vertices $\{s_1, \ldots, s_m\}$ the *set vertices*. For each element x_i $(i = 1, \ldots, n)$ we define a set of $k+1$ *element vertices* $X_i = \{x_i^1, \ldots, x_i^{k+1}\}$.

Tree arcs $\overrightarrow{T}$: The tree $\overrightarrow{T}$ consists of the following arcs.

1. There is a directed path in $\overrightarrow{T}$ consisting of the arcs $\overrightarrow{p_i p_{i+1}}$ for $1 \leq i < m$, followed by the arc $\overrightarrow{p_m u}$. We call this path the *collection path*.
2. For each set S_j, there is a directed path in $\overrightarrow{T}$ through the vertices r_j, s_j, t_j, u in order. We call these the *set paths*.
3. For each element x_i, there is a directed path in $\overrightarrow{T}$ consisting of the arcs $\overrightarrow{x_i^h x_i^{h+1}}$ for $1 \leq i < k+1$. We call these the *element paths*.
4. The arc $\overrightarrow{x_n^{k+1} p_1}$ is in $\overrightarrow{T}$.
5. For each $j = 1, ..., m$, $\overrightarrow{T}$ contains the arc $\overrightarrow{\ell_j x_1^1}$.

Non-tree edges E: For each set S_j, we connect every element vertex (corresponding to the elements in S_j) to the corresponding set vertex with non-tree edges $\{s_j x_i^1, \ldots, s_j x_i^{k+1} \ : \ x_i \in S_j\}$. For each set S_j there are 4 additional non-tree edges: $\{\ell_j u, r_j u, \ell_j t_j, p_j r_j\}$.

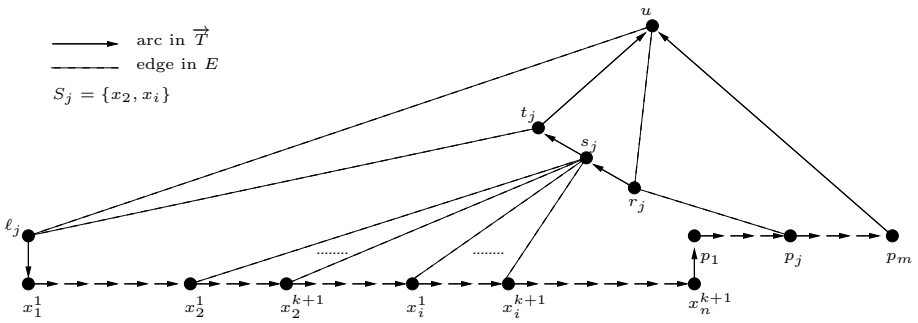


Figure 2: NP-hardness construction. The vertices $\ell_j, p_j, r_j, s_j, t_j$ (and adjacent edges) are only shown for one set S_j.

The Idea

We start by noting that the orientation of $2m$ of the non-tree edges are forced in any acyclic orientation.

Observation 7. *In any feasible solution, i.e., any acyclic orientation, each edge $r_j u$ must be oriented as $\overrightarrow{r_j u}$ since otherwise $u r_j s_j t_j$ would form a directed cycle. Similarly, in any feasible solution, each edge $\ell_j u$ must be oriented as $\overrightarrow{\ell_j u}$ since otherwise $u \ell_j t_j$ would form a directed cycle.*

Hence, in any feasible solution the $2m$ vertices $r_1, ..., r_m$ and $\ell_1, ..., \ell_m$ are covered. So, the only remaining uncovered vertices are t_j, s_j, p_j for each j and the element vertices, $x_i^1, ..., x_i^{k+1}$ for each i.

Notice that if $t_j \ell_j$ is oriented as $\overrightarrow{t_j \ell_j}$ then no vertex x_i^h can be covered for any $x_i \in S_j$, $1 \leq h \leq k+1$ without forming a cycle. However if $\ell_j t_j$ is oriented as $\overrightarrow{\ell_j t_j}$ then each of those element vertices x_i^h can be covered by $\overrightarrow{x_i^h s_j}$ and so we "equate" this case with choosing the set S_j as part of a solution to the set cover problem. Thus minimizing the number of subsets S_j whose union is X will be seen to be equivalent to maximizing the number of t_j's that are not covered. This is the basic idea of our proof.

The Proof

We show that there is a solution to the constructed instance of TREE AUGMENTATION that covers at least $(k+1)n+4m-k$ vertices if and only if there is a solution to the instance of SET COVER.

Solutions to SET COVER imply solutions to TREE AUGMENTATION: Suppose there is a solution $\mathcal{C}' \subseteq \mathcal{C}$ to the SET COVER instance, that is, $|\mathcal{C}'| \leq k$ and $\cup_{S_j \in \mathcal{C}'} S_j = X$. Note that if $|\mathcal{C}'| < k$ we can always add subsets to the solution and it will still cover all the elements. So, without loss of generality, we will assume that $|\mathcal{C}'| = k$. We orient a subset $E' \subset E$ so that it covers $(k+1)n+4m-k$ vertices. Observation 2 then says that the orientation of E' can be extended to an orientation of all of E that covers at least $(k+1)n+4m-k$ vertices and hence is a solution to the TREE AUGMENTATION instance.

E'_1**: Forced edges** As discussed earlier, the edges $\ell_j u$ and $r_j u$ are back edges and so must be oriented toward the root: $\overrightarrow{\ell_j u}, \overrightarrow{r_j u}$. These arcs then cover the $2m$ vertices ℓ_j and r_j for $1 \leq j \leq m$.

E'_2**:** $S_j \in \mathcal{C}'$ For each $S_j \in \mathcal{C}'$, orient $\ell_j t_j$ and $p_j r_j$ as $\overrightarrow{\ell_j t_j}$, $\overrightarrow{p_j r_j}$ respectively. These arcs then cover the k p_j's where $S_j \in \mathcal{C}$. Also for each $x_i \in S_j$, orient the edges $\{s_j x_i^1, \ldots, s_j x_i^{k+1}\}$ toward s_j. Since $\mathcal{C}'$ covers the elements, each x_i^h will have at least one non-tree edge oriented away from it, and hence this covers all of the $(k+1)n$, x_i^j vertices. Therefore the orientation of the edges of E'_2 cover $(k+1)n+k$ additional vertices. Note that adding the edges of E'_2 to $\overrightarrow{T} \cup E'_1$ does not introduce any cycles because it only directs edges *from* the element paths *to* the set paths.

E'_3**:** $S_j \notin \mathcal{C}'$ For each $S_j \notin \mathcal{C}'$, we orient the edge $t_j \ell_j$ as $\overrightarrow{t_j \ell_j}$ and orient all the edges $\{s_j x_i^1, \ldots, s_j x_i^{k+1}\}$ away from s_j. The edges of E'_2 cover the $2(m-k)$ additional vertices t_j and s_j where $S_j \notin \mathcal{C}$. Notice the solution is still acyclic, since for each $S_j \notin \mathcal{C}'$, there is no arc directed into any vertex in the set $\{r_j, s_j, t_j\}$.

It follows $E' = E'_1 \cup E'_2 \cup E'_3$ is a feasible solution of size $(k+1)n+4m-k$ as desired.

No solution to SET COVER implies no solution to TREE AUGMENTATION: Suppose there is no subcollection of $\mathcal{C}$ of size at most k that covers all the elements of X. We will show that any feasible orientation of the non-tree edges $\overrightarrow{E}$ will cover less than $(k+1)n+4m-k$ vertices.

We have two facts resulting from the existence of the set paths to the root in $\overrightarrow{T}$ and directed paths from ℓ_j through all the element vertices and through the collection path to the root. Since we assume all the non-tree edges are oriented (w.l.o.g. by Observation 2), the following facts must hold or else there would be a directed cycle.

Fact 1 If s_j is covered, then p_j is not covered.

Fact 2 If $\overrightarrow{x_i^h s_j} \in \overrightarrow{E}$ for some h $(1 \leq h \leq k+1)$, then $\overrightarrow{\ell_j t_j} \in \overrightarrow{E}$ and t_j cannot be covered.

We have two cases: either some representative vertex x_i^h of each element x_i is covered or there is some element x_i all of whose representatives $x_i^1, x_i^2, \ldots, x_i^{k+1}$ are not covered.

We start with the former case. Suppose that for each $i \in [1, \ldots, n]$ there is some $h(i) \in [1, \ldots, k+1]$ such that $x_i^{h(i)}$ is covered. Let J be the subset of indices $[1, \ldots, m]$ such that the $x_i^{h(i)}$'s are covered by edges oriented to s_j for $j \in J$. As we assume that the best set cover requires more than k sets, $|J| > k$. By Fact 2, t_j is not covered for any $j \in J$. Then, by Fact 1, it follows that for $j \in J$ at most 3 of $\{\ell_j, r_j, s_j, t_j, p_j\}$ are covered. Also note by Facts 1 and 2, at most 4 of $\{\ell_j, r_j, s_j, t_j, p_j\}$ can be covered for $j \notin J$. Therefore, $\overrightarrow{E}$ can cover at most $4m - |J| + (k+1)n < 4m - k + (k+1)n$ vertices.

We now prove the latter case. Suppose that for some $i \in [1, \ldots, n]$, x_i^h is not covered for any $h \in [1, \ldots, k+1]$. By Fact 1, it follows that at most 4 of $\{\ell_j, r_j, s_j, t_j, p_j\}$ are covered. Therefore $\overrightarrow{E}$ can cover at most $4m + (k+1)(n-1) = 4m + (k+1)n - k - 1$ vertices.

4 Bounded treewidth TREE AUGMENTATION

We show in this section that for G of bounded treewidth, TREE AUGMENTATION can be solved optimally via dynamic programming. *Treewidth* is a measure of how far a graph is from being a tree. It is known that many NP-hard graph problems become tractable in graphs of bounded treewidth [5]. Formally, $G = (V, E)$ has treewidth w if there is a tree decomposition (Υ, Γ) of G where each node $\nu \in \Upsilon$ corresponds to a subset S_ν of V, and Γ is a tree on Υ, satisfying the following four properties.

1. No more than $w+1$ vertices of V are mapped to any one node of Γ, i.e. $|S_\nu| \leq w+1$ for $\nu \in \Upsilon$.
2. The union of S_ν is V, i.e. every vertex of V is mapped to some node of Γ.
3. For every edge uv in E, u and v are mapped to some common node of Γ.
4. If, for $\nu_1 \in \Upsilon$ and $\nu_2 \in \Upsilon$, S_{ν_1} and S_{ν_2} contain a common vertex of V, then for all nodes ν of in the (unique) path between ν_1 and ν_2 in Γ, S_ν contains v as well.

We may assume without loss of generality that Γ is a rooted binary tree with $O(|V|)$ nodes [3]. Note that given a graph G and an integer k it is NP-hard to determine if the treewidth of G is at most k [1]. However, for any fixed constant k, Bodlaender gives a linear time algorithm which determines if a graph G has treewidth at most k, and if so, finds a tree-decomposition of G with treewidth at most k in linear time [4].

In this section, we prove:

Theorem 8. *For any constant k, and any graph G of treewidth at most k,* TREE AUGMENTATION *can be solved in linear time using dynamic programming.*

Proof. Let (Υ, Γ) be a tree decomposition of G of treewidth at most k. Since Γ is rooted, for each node $\nu \in \Gamma$, we can define Γ_ν as the subtree of Γ rooted at ν. For a tree node ν of Γ, let G_ν be the subgraph of G whose vertex set is $\{v \in G \mid v \in S_\nu \ s.t. \ \nu \in \Gamma_\nu\}$ and edge set is $\{uv \in E(G) \mid u, v \in S_\nu \ s.t \ \nu \in \Gamma_\nu\}$.

For each tree node ν of Γ, each permutation P of the vertices of S_ν and each subset C of S_ν, we determine two tables, $\overrightarrow{\text{TAB}}_\nu[P, C]$ and $\text{TAB}_\nu[P, C]$:

(i) $\overrightarrow{\text{TAB}}_\nu[P, C]$ is an optimal orientation of the edges G_ν that is consistent with permutation P and covers exactly the vertices C, such that the number of vertices covered by $\overrightarrow{\text{TAB}}_\nu[P, C]$ is maximized.

(ii) $\text{TAB}_\nu[P, C]$ is the number of vertices covered by $\overrightarrow{\text{TAB}}_\nu[P, C]$

If a permutation P contradicts the partial order on the vertices enforced by $\overrightarrow{T}$, then we set $\text{TAB}_\nu[P, C] = -\infty$ and $\overrightarrow{\text{TAB}}_\nu[P, C] = \emptyset$ for all $C \subseteq S_\nu$. Likewise, if no orientation of G_ν can cover C, we set $\text{TAB}_\nu[P, C] = -\infty$ and $\overrightarrow{\text{TAB}}_\nu[P, C] = \emptyset$ for all permutations P of S_ν. For simplicity of presentation, we assume that all entries of TAB are initialized to $-\infty$ and all entries of $\overrightarrow{\text{TAB}}$ are initialized to $\emptyset$.

For the root r of Γ, it follows that the maximum of $\text{TAB}_r[P, C]$ taken over all permutations P and subsets C of S_r is the value of an optimal solution and the corresponding $\overrightarrow{\text{TAB}}_\nu[P, C]$ is an optimal solution. Hence, to complete the proof of Theorem 8, it is enough to show how to determine the entries of our dynamic programming table. We do so in two steps. First, we determine the entries of TAB_ν for each leaf $\nu \in \Gamma$. Second, we determine the entries of TAB_ν for each non-leaf node $\nu \in \Gamma$ given that the entries of TAB_{ν_1} and TAB_{ν_2} for its associated child nodes, ν_1 and ν_2, have already been determined.

Assume $\nu \in \Gamma$ is a leaf node. For each permutation P that is consistent with $\overrightarrow{T}$, $\overrightarrow{E_\nu}$ be the orientation of the edges of G_ν implied by P and let $C^\star$ be the subset of S_ν covered by $\overrightarrow{E_\nu}$. By construction, TAB_ν contains the desired entries.

Assume ν is an internal node such that the TAB entries associated with the children of ν, namely ν_1 and ν_2, have been computed. We consider all permutations P_1 of S_{ν_1} and P_2 of S_{ν_2} together with all subsets $C_1 \subseteq S_{\nu_1}$ and $C_2 \subseteq S_{\nu_2}$. We use the entries of $\text{TAB}_{\nu_1}[P_1, C_1]$ and $\text{TAB}_{\nu_2}[P_2, C_2]$ to construct $\text{TAB}_\nu[P, C]$. Now, not all choices of P_1, P_2, C_1 and C_2 lead to valid solutions. Indeed, the partial order given by the

permutation P_1 must be consistent with P; analogously P_2 must be consistent with P. Additionally, if v is in C_1 and $v \in S_\nu$ then v must also be C; the analogous condition holds for C_2. For $i \in \{1,2\}$, we call P_i and C_i *good* if they satisfy these conditions. We now show that

$$\begin{aligned} \textsc{Tab}_\nu[P,C] = |C| \quad &+ \quad \max_{\text{good } P_1,C_1} \{\textsc{Tab}_{\nu_1}[P_1,C_1] - |C_1 \cap C|\} \\ &+ \quad \max_{\text{good } P_2,C_2} \{\textsc{Tab}_{\nu_2}[P_2,C_2] - |C_2 \cap C|\}, \end{aligned} \tag{1}$$

and $\overrightarrow{\textsc{Tab}}_\nu[P,C]$ can be determined from the entries of $\textsc{Tab}_{\nu_1}$ and $\textsc{Tab}_{\nu_2}$.

Fix a permutation P and subset C of S_ν. Let $\overrightarrow{O}$ be any orientation of G_ν that is valid with respect to P and C. Let A_1 be the vertices covered in G_{ν_1} and A_2 be the vertices covered in G_{ν_2}. By the fourth property of tree decompositions, the number of vertices covered by $\overrightarrow{O}$ is exactly $|C| + |A_1 \setminus C| + |A_2 \setminus C|$. Letting $\overrightarrow{O}_1$ be the restriction of $\overrightarrow{O}$ to G_{ν_1}, for any good P_1 and C_1 we have $|A_1| \leq \textsc{Tab}_{\nu_1}[P_1,C_1]$. Hence, $|A_1| \leq \max_{\text{good } P_1,C_1} \textsc{Tab}_{\nu_1}[P_1,C_1]$, and so, $|A_1 \setminus C| \leq \max_{\text{good } P_1,C_1}(\textsc{Tab}_{\nu_1}[P_1,C_1] - |C_1 \cap C|)$. Similarly, $|A_2 \setminus C| \leq \max_{\text{good } P_2,C_2}(\textsc{Tab}_{\nu_2}[P_2,C_2] - |C_2 \cap C|)$. Hence, $\textsc{Tab}_\nu[P,C]$ is at most the righthand side of Equation 2.

To complete the proof it is enough to show there exists good P_1, C_1 and good P_2, C_2 such that the corresponding entries of $\overrightarrow{\textsc{Tab}}_{\nu_1}[P_1,C_1]$ and $\overrightarrow{\textsc{Tab}}_{\nu_2}[P_2,C_2]$ can be used to give an acyclic orientation of value equal to the righthand side of Equation 2. Let good P_1, C_1 be chosen to maximize $\textsc{Tab}_{\nu_1}[P_1,C_1] - |C_1 \cap C|$, and let good P_2, C_2 be chosen to maximize $\textsc{Tab}_{\nu_2}[P_2,C_2] - |C_2 \cap C|$. We first orient the edges S_ν by the permutation P. By the fourth property of tree decompositions, every remaining edge is completely contained in either G_{ν_1} or completed contained in G_{ν_2}. For these edges we use the orientations of $\overrightarrow{\textsc{Tab}}_{\nu_1}[P_1,C_1]$ and $\overrightarrow{\textsc{Tab}}_{\nu_2}[P_2,C_2]$, respectively. Clearly, this orientation covers the desired number of nodes; it only remains to show it is acyclic.

For the purpose of contradiction, let $u, v \in S_\nu$ be such that the permutation P places u before v but $\textsc{Path}_{vu}$, a directed path from v to u, already exists. If there are multiple such paths, we choose the shortest path. If $\textsc{Path}_{vu}$ contains an internal node $x \in S_\nu$, then $\textsc{Path}_{vx}$ and $\textsc{Path}_{xu}$ are also directed paths. Since $\textsc{Path}_{vu}$ is the shortest, the permutation P must ensure that v is before x and x is before u. However, P also enforces u is before v, which is a contraction. Therefore, none of the internal nodes in $\textsc{Path}_{vu}$ can be in S_ν. By properties 3 and 4 of the tree decomposition, every internal node of $\textsc{Path}_{vu}$ must be contained entirely in one of the subtrees, say the one rooted at ν_1. In addition, u and v must be contained in S_{ν_1}. If a permutation P_1 ensures v is before u then P_1 and P_2 would be inconsistent; if P_1 ensures u is before v then ν_1 would be a earlier node in which a cycle appears. It now follows that $\overrightarrow{\textsc{Tab}}_\nu[P,C]$ is the desired orientation. Moreover, since we can compute it by considering the $O((k!)^4)$ possible choices for P_1, P_2, C_1 and C_2, it follows that it follows that constructing $\textsc{Tab}$ can be done in linear time. □ □

The above dynamic programming argument generalizes to the weighted case, if we keep track of the node weights rather than the number of nodes.

Corollary 9. *For any constant k, and any graph G of treewidth at most k,* Weighted Tree augmentation *can be solved using dynamic programming.*

5 A PTAS for BFS TREE AUGMENTATION in planar graphs

In this section we consider a special case in which the input graph $G = (V, \overrightarrow{T} \cup E)$ is planar and $\overrightarrow{T}$ is the breath-first-search (BFS) tree of G. (Specifically, the T is a BFS tree of the undirected version of $(V, T \cup E)$.) We show that Baker's technique, described in a moment, can be used to design a polynomial-time approximation scheme (PTAS) for this special case. A PTAS is, for a fixed integer d, a polynomial (in $|G|$) time algorithm that finds a solution of value achieving at least a $1 - \frac{1}{d}$ fraction of the optimal solution's value.

Baker's technique is a shifting technique for designing PTASes for planar graph instances of problems [2]. Baker introduced this technique to solve NP-hard problems such as INDEPENDENT SET and VERTEX COVER; the constraints for these problems are defined locally, for neighborhoods of vertices. Usually, Baker's technique cannot be used to solve problems with a global constraints, such as our acyclicity constraint. However, the technique involves separating the graph at BFS layers; when $\overrightarrow{T}$ is a BFS tree, we can guarantee acyclicity across the separators. We show how to use Baker's technique here to prove:

Theorem 10. *There is a PTAS for* TREE AUGMENTATION *when the input is a planar graph* $G = (V, E \cup \overrightarrow{T})$ *such that* $\overrightarrow{T}$ *is a BFS tree of* G.

Proof. Label each vertex with its BFS level from the root of $\overrightarrow{T}$. Let F_i be the subset of arcs and edges $(\overrightarrow{T} \cup E)$ that have one endpoint in level i and one endpoint in level $i+1$. Let $F^k = \cup_{i=k \bmod d} F_i$ for $k = 1, \ldots d$.

Let $\mathcal{S}_k$ be the connected components of $(V, (T \cup E) \setminus F_k)$. Consider a component S of $\mathcal{S}_k$ and attach all the nodes at the smallest level in S to a newly created root node r_S by arcs to create S'; these new arcs plus the arcs of $\overrightarrow{T} \cap S$ forms a BFS spanning tree $\overrightarrow{T}_S$ directed toward r_S. Since $\overrightarrow{T}_S$ has depth at most d, S' has treewidth at most $3d$ by way of:

Theorem 11 (Baker [2])**.** *Given a planar graph* G *with rooted spanning tree of depth* d *a tree decomposition of width at most* $3d$ *can be found in* $O(d|G|)$ *time.*

We are now able to describe the PTAS of Theorem 10. By the algorithm of Section 4, we can optimally solve the TREE AUGMENTATION problem in each of the components of $\mathcal{S}_k$ in polynomial time. For each $k = 1, ..., d$, compute the optimal solutions corresponding to each component of $\mathcal{S}_k$. Let $\overrightarrow{E}_k$ be the orientation given by the union of these solutions with the edges of $F^k \cap E$ oriented from high-to-low BFS level. Return the best solutions of $\{\overrightarrow{E}_1, \ldots, \overrightarrow{E}_d\}$. To complete the proof Theorem 10, it remains only to prove acyclicity and near-optimality:

Acyclicity First notice that these orientations are acyclic. Any directed cycle in $\overrightarrow{E}_k \cap \overrightarrow{T}$ would have to include arcs in multiple components of $\mathcal{S}_k$ and, in particular, would travel from a low BFS-level to a high BFS-level and back, crossing F_i in each direction for some $i = k \bmod d$. However, since T is a BFS tree, each arc in $\overrightarrow{T} \cap F_i$ is oriented from level i to level $i+1$. By design the edges of $F_i \cap E$ are also oriented from level i to level $i+1$. Therefore $\overrightarrow{E}_k \cap \overrightarrow{T}$ is acyclic.

Near-Optimality Let $E^\star$ be a minimal subset of E such that an optimal solution $\overrightarrow{E}^\star$ of $G = (V, \overrightarrow{T} \cup E^\star)$ covers as many vertices as an optimal solution for $G = (V, \overrightarrow{T} \cup E)$. That is, every vertex is the starting point for at most one arc of $\overrightarrow{E}^\star$ and so the maximum number of vertices that can be covered is $|E^\star|$.

F^k is a partition of a subset of $\overrightarrow{T} \cup E$. Therefore

$$\min_k |E^\star \cap F^k| \leq \frac{1}{d} \sum_k |E^\star \cap F^k| \leq \frac{1}{d}|E^\star| \tag{2}$$

Consider the index $k^\star$ that is the argument the above minimum. For each component S of $\mathcal{S}_k$, $\overrightarrow{E}_{k^\star} \cap S$ covers at least as many vertices as $\overrightarrow{E}^\star$, since $\overrightarrow{E}_{k^\star}$ is optimal for S. Therefore $\overrightarrow{E}_{k^\star}$ covers at least $|E^\star| - |E^\star \cap F^{k^\star}|$ vertices. By Inequality 2, $\overrightarrow{E}_{k^\star}$ covers at least $(1 - \frac{1}{d})|E^\star|$ vertices. This completes the proof. □ □

Corollary 12. *There is a PTAS for* WEIGHTED TREE AUGMENTATION *when the input is a planar graph* $G = (V, E \cup \overrightarrow{T})$ *such that* $\overrightarrow{T}$ *is a BFS tree of* G.

6 Two-Arm TREE AUGMENTATION

We consider a special case in which the tree $\overrightarrow{T}$ consists of exactly two directed paths and give a polynomial-time dynamic program for finding an optimal solution to TREE AUGMENTATION. $\overrightarrow{T}$ has root $\ell_0 = r_0$ and two directed paths to the root: the *left arm* with vertices in order from leaf to root $\ell_{n_\ell}, \ell_{n_\ell - 1}, \ldots, \ell_1, \ell_0$ and the *right arm* with vertices in order from leaf to root $r_{n_r}, r_{n_r - 1}, \ldots, r_1, r_0$. We use inequalities to compare the indices of these vertices (i.e., $r_i < r_j$ if $i < j$).

Any edge in E with both endpoints in a single arm is a back edge and must be oriented toward the root; we denote this orientation by $\overrightarrow{B}$ as in Section 2. Each cross edge $e \in C \subseteq E$ has a left endpoint $\ell(e)$ and a right endpoint $r(e)$. We let $\overrightarrow{e} = \overrightarrow{\ell(e)r(e)}$ denote the left-to-right orientation and $\overleftarrow{e} = \overleftarrow{\ell(e)r(e)}$ denote the right-to-left orientation of e.

We sort the cross edges first by left, then by right endpoint. Namely $C = \{e_1, e_2, \ldots, e_m\}$ such that $i < j$ only if either $\ell(e_i) < \ell(e_j)$ or $\ell(e_i) = \ell(e_j)$ and $r(e_i) < r(e_j)$. (Note that we may assume that there are no parallel edges as all parallel edges would need to be oriented consistently to maintain acyclicity.) For each $k = 0, 1, \ldots, m$ and each $j = 1, \ldots, n_r + 1$ we determine

$$\text{the orientation } \overrightarrow{C_{j,k}} \text{ of } C_{j,k} = \{e_i \ : \ i \le k, r(e_i) < r_j\} \text{ that maximizes}$$
$$\text{the number } c_{j,k} \text{ of endpoints of } C_{j,k} \text{ covered by } \overrightarrow{C_{j,k}} \cup \overrightarrow{B}.$$

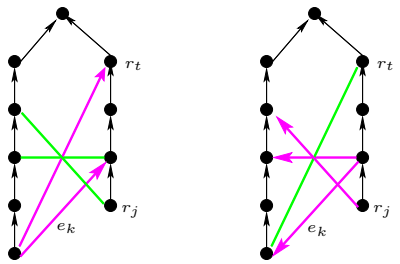
(The notation $\overrightarrow{C_{j,k}}$ does not indicate orienting all the edges in $C_{j,k}$ from left to right.) Clearly, the solution to TREE AUGMENTATION is $\overrightarrow{C_{n_r,m}} \cup \overrightarrow{B}$.

Note that the sets $C_{j,0}$ are empty and so the values $c_{j,0}$ denote the number of vertices covered by $\overrightarrow{B}$. We denote the coverage by $\overrightarrow{B}$ by:

$$\delta(v) = \begin{cases} 1 & \text{if } v \text{ is not covered by } \overrightarrow{B} \\ 0 & \text{otherwise} \end{cases}$$

For any j and for $k > 0$, we determine $\overrightarrow{C_{j,k}}$ from $\overrightarrow{C_{j,t}}$ for $t < k$ and $\overrightarrow{C_{t,k}}$ for $t < k$. This allows us to compute $\overrightarrow{C_{n_r,m}}$ via dynamic programming.

If $r_j \le r(e_k)$, then $e_k \notin C_{j,k}$ and so $C_{j,k} = C_{j,k-1}$; therefore $\overrightarrow{C_{j,k}} = \overrightarrow{C_{j,k-1}}$. If $r_j > r(e_k)$, then $e_k \in C_{j,k}$ and we take the better of two options: $\overrightarrow{e_k}$ or $\overleftarrow{e_k}$. Refer to Figure 3.



(a) (b)

Figure 3: The case in which e_k is oriented (a) $\overrightarrow{e_k}$, from left to right and (b) $\overleftarrow{e_k}$, from right to left.

In the $\overrightarrow{e_k}$ option, to ensure acyclicity, any edge in $C_{j,k}$ that shares e_k's endpoint must also be oriented from left to right. We define t such that e_t is the last edge of $C_{j,k}$ with $\ell(e_t) < \ell(e_k)$. Any acyclic orientation of $C_{j,t}$ combined with this will also be acyclic as any introduced cycle would go through $\ell(e_k)$ which we have prevented. The only additionally covered vertex is $\ell(e_k)$ if it is not already covered by $\overrightarrow{B}$. Formally, letting e_t be the last edge of $C_{j,k}$ with $\ell(e_t) < \ell(e_k)$:

$$\text{Option } \overrightarrow{e_k}: \quad \begin{cases} \overrightarrow{C_{j,k}} & = & \overrightarrow{C_{j,t}} \cup \{\overrightarrow{e} \ : \ \ell(e) = \ell(e_k), \ e \in C_{j,k}\} \\ c_{j,k} & = & c_{j,t} + \delta(\ell(e_k)) \end{cases}$$

In the $\overleftarrow{e_k}$ option, to ensure acyclicity, any edge in $C_{j,k}$ with right endpoint after $r(e_k)$ must also be oriented from right to left. We define t such that $r_t = r(e_k)$. Note that $t < j$ since we are considering the case $r_j > r(e_k)$. Any acyclic orientation of $C_{t,k}$ combined with this will also be acyclic as any introduced cycle would have to go through $r(e_k)$, which we have prevented. The right endpoints of these newly oriented edges may become covered if they were not already covered by $\overrightarrow{B}$. Formally, letting $r_t = r(e_k)$:

$$\text{Option } \overleftarrow{e_k}: \quad \begin{cases} \overrightarrow{C_{j,k}} & = & \overrightarrow{C_{t,k}} \cup \{\overleftarrow{e} \ : \ r(e) \geq r_t, \ e \in C_{j,k}\} \\ c_{j,k} & = & c_{t,k} + \sum_{\overleftarrow{e}: r(e) \geq r_t, e \in C_{j,k}} \delta(r(e)) \end{cases}$$

It is not difficult to see that by storing these two options for each value of j and k, one can give a polynomial-time implementation of the dynamic program.

Theorem 13. Tree augmentation *in the special case of 2 arms can be solved optimally using dynamic programming in polynomial time.*

It is easy to see the following generalization to the weighted case. In the dynamic programming, instead of having binary $\delta(v)$ we have $\delta(v)$ reflect the weight of node v.

Corollary 14. Weighted Tree augmentation *in the special case of 2 arms can be solved optimally using dynamic programming in polynomial time.*

Conclusion

In this paper we study improving robustness in next-hop routing by modeling it as a graph theoretic problem tree augmentation. This work leads to a number of open problems. For example, can the dynamic programming approach be applied to more special cases? We note that the special case of multiple arms is not immediately amenable to dynamic programming. More generally, what is the complexity when the problem has a bounded number of leaves? Does the problem admit a better-than $\frac{1}{2}$-approximation in the general case?

References

[1] S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of Finding Embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8, 1987.

[2] B. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of The ACM*, 41:153–180, 1994.

[3] H. Bodlaender. A Tourist Guide through Treewidth. *Acta Cybernetica*, 11:1–23, 1993.

[4] H. Bodlaender. A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[5] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. In Gunther Schmidt and Rudolf Berghammer, editors, *Graph-Theoretic Concepts in Computer Science*, volume 570 of *Lecture Notes in Computer Science*, pages 13–24. Springer Berlin / Heidelberg, 1992.

[6] M. Garey and D. Johnson. *Computers and intractability: A guide to the theory of NP-completeness.* W. H. Freeman and Co., San Francisco, Calif., 1979.

[7] M. Goyal, M. Soperi, E. Baccelli, G. Choudhury, A. Shaikh, H. Hosseini, and K. Trivedi. Improving Convergence Speed and Scalability in OSPF: A Survey. *IEEE Communications Surveys and Tutorials*, 14(2), 2012.

[8] C. Hedrick. Routing Information Protocol. RFC 1058, June 1988.

[9] J. Moy. OSPF Version 2. RFC 2178, April 1998.

[10] D. Oran. OSI IS-IS Intra-domain Routing Protocol. RFC 1142, February 1990.

[11] A. Raj and O. Ibe. A survey of IP and multiprotocol label switching fast reroute schemes. *Computer Networks*, 51(8):1882–1907, 2007.

[12] H. Q. Vo, O. Lysne, and A. Kvalbein. Permutation Routing for Increased Robustness in IP Networks. In *Proceedings of the 11th international IFIP TC 6 conference on Networking - Volume I*, pages 217–231, 2012.