

Perfect Routing Arborescences for Fast Reroute

Péter Babarczi* and János Tapolcai*†

*Department of Telecommunications and Artificial Intelligence, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, Műegyetem rkp. 3., H-1111 Budapest, Hungary

†HUN-REN-BME Information Systems Research Group, Műegyetem rkp. 3., H-1111 Budapest, Hungary
E-mail: {babarczi, tapolcai}@tmit.bme.hu

Abstract—Owing to its quick reaction to link failures, fast reroute is among the most popular survivable routing approaches in carrier-grade backbone networks. By using pre-configured routing tables, routers can select a failover path for packets based solely on locally available information. Although arc-disjoint spanning arborescences are frequently used to configure forwarding tables in fast reroute, they only provide survivability up to the global connectivity of the network, thus, they fail to maximize survivability in topologies with dense subgraphs. In this paper, we investigate routing arborescences, a generalization of spanning arborescences for fast reroute in which the arborescences are not required to span the entire network. We prove a surprising graph-theoretical result: for any chosen root node in real-world topologies with bidirectional communication links perfect routing arborescences always exist, i.e., each node with local connectivity k to the root is included in exactly k arc-disjoint arborescences. Our constructive proof gives a fast heuristic algorithm to build perfect routing arborescences, offering four optimization options to minimize path length as its primary objective. Extensive simulations are conducted on real-world topologies, demonstrating that our method significantly improves path stretch in the routing arborescences compared to previous methods.

Index Terms—fast reroute, perfect routing arborescences, edge split-off, survivable routing

I. INTRODUCTION

Modern communication networks supporting latency-critical services – such as virtual reality, cloud gaming and financial systems – require rapid recovery from link or node failures to maintain service continuity. As many of these failures are transient in nature, e.g., due to flapping interfaces or short-lived physical disruptions [1], *Fast Reroute* (FRR) plays a critical role in ensuring high availability for these services by providing immediate local failover paths using pre-computed static routing tables [2]. FRR allows packets to be rerouted in a matter of milliseconds based solely on local information available at the router, such as the packet’s destination, its incoming port, and the failure status of adjacent links, often long before any dynamic global routing re-convergence is triggered [3].

However, as FRR does not rely on any control plane messages for failure recovery, only the endpoints of a link are aware of its failure. Therefore, routers not adjacent to the failed link(s) continue forwarding packets as usual, making routing table design a challenging task, as it must ensure loop-free forwarding despite the limited failure visibility. In order to mitigate state space explosion of the routing tables, a common approach is to compute arc-disjoint spanning

arborescences [4] towards each destination and populate the forwarding tables accordingly [2]. Fast reroute can rely on a cyclic order of these arborescences [5] where the forwarding logic follows a given arborescence until the packet encounters a failed link, at which point it switches to the next arborescence with operational out-edge in the order. When a cyclic order does not exist, other routing designs can rely on more involved forwarding functions [2] or rewriting bits in the packet header to avoid infinite forwarding loops [6].

Although a celebrated result by Edmonds [7] states that in a k -edge-connected graph it is always possible to construct k arc-disjoint spanning arborescences rooted at a given node, in practice the network topologies are only 2-edge-connected [8], limiting the achievable resilience to single link failures even in densely connected subgraphs. In order to maximize failure resilience in both sparse and dense subgraphs, more flexible structures are required. In [9] spanning arborescences were generalized as directed acyclic graphs (DAGs), while *routing arborescences* were introduced in [10] for FRR which not necessarily span all network nodes. It was shown that routing arborescences can provide arc-disjoint paths for each node towards the root above global connectivity, improving the resilience in dense subgraphs [11]. When the maximum number of arc-disjoint paths is provided for every node v towards the root t along the arborescences (i.e., equals the local connectivity between v and t), we refer to them as *perfect routing arborescences*.

The algorithm in [11] consistently produced perfect routing arborescences across a wide range of practical scenarios, suggesting that their existence might be a fundamental structural property of undirected graphs. However, it did not provide any theoretical guarantees to support the conjecture. In this paper we make a twofold contribution to bridge this gap between graph-theory and practical FRR routing:

- We provide a constructive proof that in real-world network topologies with only bidirectional communication links, *perfect k -resilient routing arborescences* always exist (Theorem 2). That is, for every node v , there are exactly $k + 1$ arc-disjoint paths along the arborescences towards the root t , where $k + 1$ equals the local connectivity between v and t (see Figure 1).
- Since Theorem 2 guarantees the existence of perfect routing arborescences, Algorithm 1 extended with four optimization options is the first algorithm which focuses on path length as its primary objective, achieving sig-

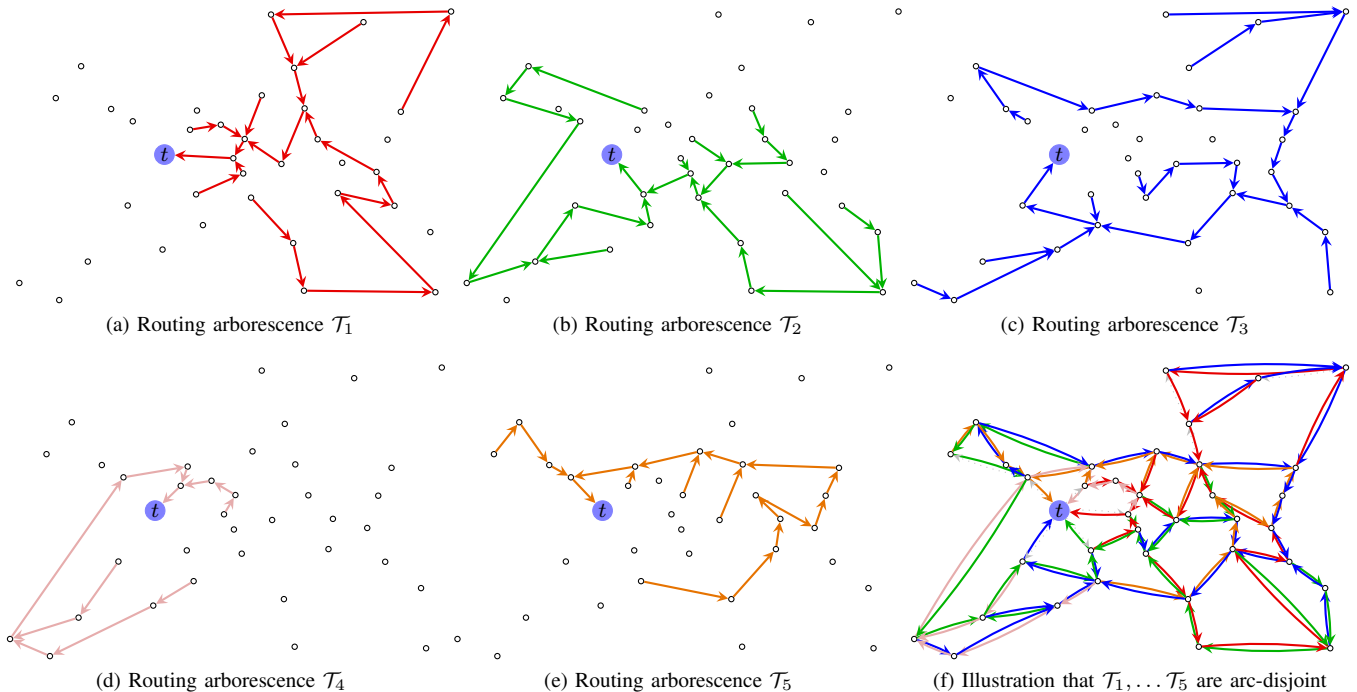


Fig. 1. Perfect routing arborescences covering all in-arcs of root t with $\delta^+(t) = 5$ in the $r(G) = 2$, i.e., 2-edge-connected COST266 network [8] providing arc-disjoint paths equal to the local connectivity. Arborescences $\mathcal{T}_1, \dots, \mathcal{T}_5$ are denoted with different colors, dotted arcs are not part of any arborescence.

nificantly shorter failover paths than previous routing arborescence construction algorithms [9], [11].

The rest of the paper is organized as follows. We summarize the related work on perfectly resilient fast reroute approaches, including spanning and routing arborescences in Section II. We formally define perfect routing arborescences and provide the theoretical background of our study on arc split-off in Section III. As our main contribution, in Section IV we present Theorem 2 with a constructive proof on perfect routing arborescence design in networks with only bidirectional communication links, and introduce an arborescence construction algorithm with four optimization options to minimize path length. In Section V we conduct thorough simulations in real-world topologies to demonstrate the superior performance of our perfect arborescences in terms of path stretch, and discuss their utilization in FRR. Finally, Section VI concludes our work.

II. RELATED WORK

In this section we discuss different levels of resilience in fast reroute. The limitations of perfect resilience is discussed in Section II-A, while practical approaches for perfect k -resilience with spanning arborescences are discussed in Section II-B, and with routing arborescences in Section II-C.

A. Perfect Resilience

Although dynamically adapting the routing tables to the failed links might achieve high resilience, such solutions either take substantial time to converge [3] or not supported in the current infrastructure [2]. Fast reroute provides an

alternative *static routing* approach, where the packet header is matched to the rules in pre-defined routing tables, and the out-edge is selected based only on locally available information. In FRR *perfect resilience* is defined as the ability to route traffic between two nodes with deterministic local failover rules until they remain connected in the underlying topology upon multiple link failures [5].

On one hand, resilient static routing tables for FRR against multiple link failures can always be constructed if both source and destination can be matched, for the price of quadratic routing table size [2]. On the other hand, from a practical point of view a scalable routing is required, leading to different routing approaches for selecting the next out-edge upon link failures, including deterministic approaches with packet header rewrite [6] and randomized methods [12]. It was shown that perfect resilience is always possible against single link failures [13], in special topologies (e.g., outerplanar graphs) or for close source-destination pairs [5], but in general it is not achievable against arbitrary multiple link failures [2] with pre-defined static rules [14], which is the price we have to pay for the scalability and locality of FRR [15].

B. Perfect k -Resilience up to Global Connectivity

Although perfect resilience is not possible for arbitrary multiple link failures, there are several methods which can provide resilient static routing for the more practical (worst case) scenario when the number of link failures is limited to $\leq k$, referred to as *perfect k -resilience*. Arc-disjoint directed spanning trees (i.e., arborescences) are one of the most widespread approaches to guarantee k -resilience depending

on the *global connectivity* $r(G)$ of the network (i.e., $k = r(G) - 1$), while they can be computed in polynomial time [4]. Existing algorithms decompose the graph $G = (V, E)$ into $r(G)$ spanning arborescences rooted at the same destination node t such that none of the arborescences share a link in the same direction [6], [16]. In order to address the main weakness of routing along the spanning arborescences in FRR, the path length was further investigated in [17], where lower bounds were given on the length increase compared to the shortest path – i.e., *path stretch* – together with construction algorithms to minimize it.

In [2] *basic routing* was defined on spanning arborescences, where each node decides on the next out-edge (i.e., arborescence \mathcal{T}_i) based on the destination address, the in-port and the failed out-edges [14]. In [5] a circular order of the $\mathcal{T}_1, \dots, \mathcal{T}_{r(G)}$ spanning arborescences was defined, where $r(G) - 1$ link failures can be survived by forwarding the packet on the next arborescence with an operational out-edge. The correctness of a circular order in arbitrary arborescences can be guaranteed only for $r(G) = \{2, 3\}$, while special arborescences can be constructed for $r(G) = 4$ [2]. For $r(G) = 5$ besides circular routing bouncing back the packet on the arborescence on the reverse direction of the incoming link if the out-edge is failed is necessary as well, while the existence of static routing for $r(G) \geq 6$ with basic routing is unknown [2]. However, from a practical point of view SyPer [18] was proposed as a general framework to synthesize routing tables for a circular order whenever it is possible. The generated prioritized list of out-edges for each in-port provides perfect k -resilience, where the packet is forwarded along the first non-failed edge in the list.

C. Perfect k -Resilience up to Local Connectivity

While the k -resilient spanning arborescences provide high resilience in regular or nearly homogeneous graphs, in heterogeneous networks they fail to exploit the additional redundancy provided by dense subgraphs. In [9] spanning arborescences were grafted into DAGs to achieve resilience above the global connectivity of the network. Similarly, in [14] different traversals of all network links were defined, which were utilized in a greedy forwarding towards the destination until a failed out-edge was reached, where the packet was detoured based on the pre-defined in-port dependent routing tables. In [10] a resilient routing table computation approach for FRR was proposed, which increased the resilience of spanning arborescences beyond the global connectivity by introducing *routing arborescences*, which provide perfect k -resilience based on the local connectivity $r(G, s, t)$ between a source node s and root t , i.e., $k = r(G, s, t) - 1 \geq r(G) - 1$.

In [11] routing arborescences were constructed with a two-step approach in undirected graphs: 1) generate a degree and local connectivity preserving graph-sequence with an iterative graph decomposition revolving around edge split-off [7], and 2) use this graph sequence in the reverse order to extend existing routing arborescences locally to the newly added node(s) by using Integer Linear Programs (ILPs). Note that,

the edge split-off based graph decomposition preserves local connectivity of the nodes, i.e., the degree of the nodes both in G_i and G_{i-1} and their connectivity does not decrease. Although perfect k -resilience can be reached in most practical cases, the algorithm does not provide any theoretical guarantees. Furthermore, odd and even degree nodes had to be handled differently, restricting the order in which the nodes could be split-off and leading to complicated scenarios for odd-degree nodes in the ILPs.

III. PERFECT ROUTING ARBORESCENCES

In this section we first formally define perfect routing arborescences in Section III-A, and introduce node split-off and connectivity-preserving digraph sequences in Section III-B, which will be leveraged in our arborescence construction algorithm.

A. Problem Formulation

We assume that the network topology is given in the form of an undirected graph $G = (V, E)$ as the input of our routing problem, which we transform into a directed graph¹ $D = (V, A)$ by replacing each undirected edge $e \in E$ with *two directed arcs*, one in each direction. In the digraph $D = (V, A)$ let $\delta^{\leftarrow}(v)$ denote the number of arcs entering node v (i.e., in-degree), and $\delta^{\rightarrow}(v)$ the number of arcs exiting node v (i.e., out-degree). A digraph is called *Eulerian* if $\forall v \in V : \delta^{\leftarrow}(v) = \delta^{\rightarrow}(v)$. The local arc-connectivity between $v, w \in V$, i.e., the maximum number of arc-disjoint directed paths connecting v and w is denoted as $r(D, v, w)$. The digraph is strongly connected if $\forall v, w \in V : r(D, v, w) \geq 1$, and its global arc-connectivity is $r(D) = \min_{v, w \in V} r(D, v, w)$.

As in destination based routing the routing tables for each root node t can be calculated independently, we assume t is given as part of the input as well. Arc-disjoint spanning arborescences for t – frequently used for fast reroute – can be generalized by not requiring them to span all the nodes.

Definition 1: We refer to a set of arc-disjoint arborescences $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_l$ directed towards root node t as *routing arborescences* if the unique paths $\mathcal{P}_{c_1}^v, \dots, \mathcal{P}_{c_l}^v$ in the arborescences from v to t are *pairwise arc-disjoint* $\forall v \neq t$, where $c_i \in \{1, \dots, l\}$.

The output of the routing problem is $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_l$ routing arborescences. In traditional solutions with spanning arborescences [6], [16] l equals to the global connectivity $r(D)$. However, with our generalized definition $l = \delta^{\leftarrow}(t) \geq r(D)$ routing arborescences can provide resilience depending on the local connectivity $r(D, v, t) \geq r(D)$ for every node v .

Definition 2: We say that a set of routing arborescences $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_{\delta^{\leftarrow}(t)}$ is *perfect* if there are exactly $r(D, v, t)$ unique directed paths $\mathcal{P}_{c_1}^v, \dots, \mathcal{P}_{c_{r(D, v, t)}}^v$ in the arborescences from each node v to root t , where $c_i \in \{1, \dots, \delta^{\leftarrow}(t)\}$.

An example of *perfect routing arborescences* for root t provided by our novel construction algorithm in Section IV is shown in Figure 1. The notations are summarized in Table I.

¹Shortly we will refer to directed graphs as digraphs throughout the paper.

TABLE I
NOTATION LIST

Notation	Description
$G = (V, E)$	Undirected graph with node set V , edge set E .
$D = (V, A)$	Eulerian digraph constructed from G by replacing $\forall e \in E$ with a directed arc in both directions.
t	Root node $t \in V$ of the routing arborescences.
$\delta^{\leftarrow}(v)$	In-degree of node $v \in V$.
$\delta^{\rightarrow}(v)$	Out-degree of node $v \in V$.
Δ	Maximum (in- and) out-degree in $D = (V, A)$.
$r(D, v, t)$	Local arc-connectivity between node v and root t in D .
$r(D)$	Global arc-connectivity of digraph D .
$\mathcal{T}_1, \dots, \mathcal{T}_{\delta^{\leftarrow}(t)}$	Routing arborescences for root node t .
$\mathcal{T}_{c_1}, \dots, \mathcal{T}_{c_r(D, v, t)}$	Routing arborescence available at node v , where $c_i \in \{1, \dots, \delta^{\leftarrow}(t)\}$
$\mathcal{P}_{c_i}^v$	The unique path in the i^{th} routing arborescence from node v to root t .
$v_i \rightarrow v_j$	Directed arc between node i and node j .
$v_i \rightsquigarrow v_j$	Directed path from node i to node j .
$D_i = (V_i, A_i)$	The i^{th} graph in the Eulerian digraph sequence with $ V_i = i$ nodes.
$h(v)$	Hop distance of node v from root t .

B. Connectivity Preserving Eulerian Digraph Sequence

In order to present our algorithm, we introduce some definitions and observations we will rely on in the design.

Definition 3: Given a digraph $D = (V, A)$, we say that we *split-off* an arc-pair $x \rightarrow y, y \rightarrow z$ from node y if arcs $x \rightarrow y$ and $y \rightarrow z$ are removed and arc $x \rightarrow z$ is added to D , resulting in $D' = (V, A')$. An arc-pair $x \rightarrow y, y \rightarrow z$ is *splittable* if it preserves the local connectivity, i.e., $\forall v, w \in V \setminus \{y\} : r(D, v, w) \leq r(D', v, w)$.

The split-off operation is often used in k -connected graph characterization [7], [19] both to preserve global [20] and local [21] connectivity in undirected graphs, and was already used for Edmonds' arborescence construction [16], [22] as well, but finding splittable edge pairs for odd-degree nodes in undirected graphs is not always possible [23]. However, in an Eulerian digraph every arc is part of a splittable pair:

Theorem 1 (Thm. 2.2 [24]): Let $D = (V, A)$ be an Eulerian digraph. For every arc $u \rightarrow x$ there is an arc $x \rightarrow v$ so that $u \rightarrow x, x \rightarrow v$ is splittable.

Note that, an Eulerian digraph D remains Eulerian after arc split-off. Hence, we can *split-off node* x by splitting off all of its arc-pairs one after the other, which based on Theorem 1 is always possible regardless of the degree of the node. By splitting-off a node at a time from D , we can obtain a digraph sequence with the following characteristics.

Definition 4: A local connectivity-preserving digraph sequence (called *CP-sequence*) of Eulerian digraph $D = (V, A)$ for root node $t \in V$ is $D_{|V|}, D_{|V|-1}, \dots, D_1$, where D_i is an Eulerian digraph with i nodes with the following properties:

- 1) $D_{|V|}$ is the original digraph $D_{|V|} = D$,
- 2) in $D_{i-1} = (V_{i-1} = V_i \setminus \{x \neq t\}, A_{i-1})$ and $D_i = (V_i, A_i)$ all arcs not adjacent to node x are the same; all

nodes have the same in- and out-degree in both graphs; and for any node pair $v, w \in V_{i-1}$ local connectivity is preserved, i.e., $r(D_{i-1}, v, w) \geq r(D_i, v, w)$,

- 3) D_1 has a single node t with $\delta^{\leftarrow}(t)$ loop arcs.

The main idea of Algorithm 1 is to construct a CP-sequence by splitting-off the nodes of D in an appropriate order, which CP-sequence in the reverse direction can be efficiently leveraged in routing arborescence construction.

IV. ROUTING ARBORESCENCE CONSTRUCTION

In Section IV-A we demonstrate that perfect routing arborescences always exist in Eulerian digraphs, and give a construction algorithm in Section IV-B. In order to improve network performance, we suggest four optimization options in Section IV-C to minimize the path length in the perfect arborescences of Algorithm 1, and demonstrate that they do not increase its computational complexity in Section IV-D.

A. Perfect Routing Arborescences in Eulerian Digraphs

Previous proofs and algorithms were mainly focusing on spanning arborescences [16], [22], [25] to provide arc-disjoint forwarding paths which can be efficiently used for fast reroute in destination based routing [2], [5]. As the main contribution of our paper, we generalize these results for routing arborescences, which can be leveraged in the practical design of resilient routing tables for fast reroute algorithms in heterogeneous networks as well, where spanning arborescences fail to utilize the extra redundancy of dense subgraphs. The main theorem of this paper is presented as follows²:

Theorem 2: Let $D_{|V|} = (V, A)$ be an Eulerian digraph with a distinguished root node t . Perfect routing arborescences $\mathcal{T}_1, \dots, \mathcal{T}_{\delta^{\leftarrow}(t)}$ exist in $D_{|V|}$, i.e., $\forall v \in V \setminus \{t\}$ the theoretical maximum number of $r(D_{|V|}, v, t)$ pairwise arc-disjoint paths $\mathcal{P}_{c_1}^v, \dots, \mathcal{P}_{c_{r(D_{|V|}, v, t)}}^v$ are available in the arborescences.

Proof: As a consequence of Theorem 1 $D_{|V|} = (V, A)$ has a CP-sequence, because we can split-off its nodes one after the other. We consider a split-off sequence \mathcal{S} where the nodes were split-off in the order of increasing local connectivity $r(D, v, t)$. Starting from D_1 we will construct the routing arborescences in the reverse order of the CP-sequence by maintaining disjoint arc sets $\mathcal{T}_1, \dots, \mathcal{T}_{\delta^{\leftarrow}(t)}$ throughout the sequence. As the sets are arc-disjoint, we will refer to their arcs by colors.

First, we assign different colors to the $\delta^{\leftarrow}(t)$ loop edges of t in D_1 to obtain $\mathcal{T}_1, \dots, \mathcal{T}_{\delta^{\leftarrow}(t)}$, which provides $r(D_1, t, t) = \delta^{\leftarrow}(t)$ arc-disjoint paths for all nodes in D_1 . We assume that in D_{i-1} there are $\forall v \in V_{i-1} : \mathcal{P}_{c_1}^v, \dots, \mathcal{P}_{c_{r(D_{i-1}, v, t)}}^v$ arc-disjoint (and cycle-free $\forall v \in V_{i-1} \setminus \{t\}$) paths in the $\mathcal{T}_1, \dots, \mathcal{T}_{\delta^{\leftarrow}(t)}$ colored arc sets. Next, by induction we will show how to extend the arc sets from D_{i-1} to D_i .

Due to the reverse order of the CP-sequence, the reverse graph transformation of node split-off between D_{i-1} to D_i involves pinching $\delta^{\leftarrow}(v_i)$ arcs and adding the node v_i on them, shown in Figure 2, where arcs $v_a \rightarrow v_b$ and $v_c \rightarrow v_d$

²Our proof is inspired by Edmonds' disjoint arborescences [25].

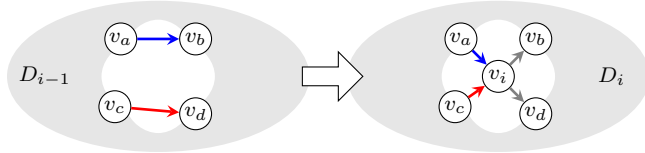


Fig. 2. Arborecence construction: pinching two arcs. In-arcs of v_i always inherit the color of the original arc, while for the out-arcs (denoted with gray) three cases should be considered to color them.

were pinched into arcs $v_a \rightarrow v_i$, $v_i \rightarrow v_b$ and $v_c \rightarrow v_i$, $v_i \rightarrow v_d$, respectively. Let set \mathcal{C} contain the different \mathcal{T}_j colors on the pinched arcs. For the in-arcs of v_i we assign the color of the pinched arc (i.e., $v_a \rightarrow v_i$ has the same color as $v_a \rightarrow v_b$, and $v_c \rightarrow v_i$ has the same as $v_c \rightarrow v_d$). However, when selecting the color of the out-arcs of v_i we need to ensure two properties: 1) all \mathcal{C} arc sets on the pinched arcs should remain directly connected to t , and 2) $\forall v \in V_{i-1} \setminus \{t\}$: $\mathcal{P}_{c_1}^v, \dots, \mathcal{P}_{c_{r(D_i, v, t)}}^v$ paths should remain cycle-free in D_i .

- **Case 1A:** If the colors of the pinched arcs were pairwise disjoint, then we can assign the color of $v_a \rightarrow v_b$ to $v_i \rightarrow v_b$ (and $v_c \rightarrow v_d$ to $v_i \rightarrow v_d$), because from every node the arc-disjoint path to the root remains the same (except in D_i some of them have node v_i in common).
- **Case 1B:** Otherwise, assume that $v_a \rightarrow v_b$ and $v_c \rightarrow v_d$ had the same color \mathcal{T}_j before pinching (i.e., in D_{i-1}), thus, $v_i \rightarrow v_b$ and $v_i \rightarrow v_d$ are candidates for being the single out-arc of v_i in \mathcal{T}_j . Without loss of generality, if v_c was on the unique path from v_a to the root in arc-set \mathcal{T}_j in D_{i-1} , i.e., $v_a \rightarrow v_b \rightsquigarrow v_c \rightarrow v_d \rightsquigarrow t$, then adding $v_i \rightarrow v_b$ to \mathcal{T}_j would introduce a cycle for v_i in D_i and disrupts connectivity for nodes in $v_a \rightarrow v_b \rightsquigarrow v_c$. Therefore, $v_i \rightarrow v_b$ is excluded from having color \mathcal{T}_j . After excluding all candidates which disrupt connectivity, any remaining valid candidate can be added to \mathcal{T}_j . Note that, as $\forall v \in V_{i-1} : \mathcal{P}_{c_1}^v, \dots, \mathcal{P}_{c_{r(D_{i-1}, v, t)}}^v$ were cycle-free in D_{i-1} , such candidate always exists.

We also need to ensure that $r(D_i, v_i, t)$ different colors are available at node v_i on its out-arcs.

- **Case 2:** After repeating the process in Case 1B we have \mathcal{C} out-arcs of v_i colored. To complete the proof, we need to color $r(D_i, v_i, t) - |\mathcal{C}|$ uncolored out-arcs (if any). Each uncolored out-arc $v_i \rightarrow x$ heads to a node x where $r(D_{i-1}, x, t) \geq r(D_i, v_i, t)$ owing to the split-off order in \mathcal{S} . Thus, x has at least one color \mathcal{T}_j that is not used among the already colored out arcs of v_i . We can color $v_i \rightarrow x$ with \mathcal{T}_j without introducing cycles (the arc is only used in the path from the new node v_i). We can repeat the process until v_i has $r(D_i, v_i, t)$ different colors on its out-arcs.

Finally, in $D_{|V|}$ the colors from the out-arcs of t are removed to make the arc sets $\mathcal{T}_1, \dots, \mathcal{T}_{\delta^{\leftarrow}(t)}$ cycle-free, i.e., routing arborescences. ■

Although we proved Theorem 2 for Eulerian digraphs, our result is directly applicable for fast reroute in all real-world topologies where all communication links are bidirectional.

Corollary 1: Every undirected graph $G = (V, E)$ can be transformed into an equivalent Eulerian digraph $D = (V, A)$ by replacing $\forall e \in E$ edge with two directed arcs, one in each direction. According to Theorem 2 perfect routing arborescences $\mathcal{T}_1, \dots, \mathcal{T}_{\delta^{\leftarrow}(t)}$ exist in $D_{|V|}$ which provide $r(D_{|V|}, v, t)$ pairwise arc-disjoint paths towards any root t in all real-world networks with only bidirectional communication links.

B. Perfect Routing Arborecence Construction Algorithm

The proof of Theorem 2 is constructive, the resulting algorithm calculating perfect routing arborescences in Eulerian digraphs is summarized in Algorithm 1. First, in Step 2 of the graph decomposition (Phase 1) we calculate a split-off order \mathcal{S} of nodes $D_{|V|} = (V, A)$ with increasing local connectivity towards root node t according to Theorem 2, i.e., we split-off the nodes with maximum $r(D_{|V|}, v, t)$ last. Hence, we can ensure that in the reverse order of the list at least $r(D_i, v, t)$ arborescences will be available at node v . In Step 3 we split-off the nodes in the order of \mathcal{S} to construct a CP-sequence, and store the split-off arc-pairs to each node v_{t_j} in \mathcal{L}_{t_j} .

After the decomposition, during arborescence construction (Phase 2) we assign a different arborescence \mathcal{T}_i to the loop arcs of root t in Step 6. We iterate through the reverse split-off order of \mathcal{S} , and put back node v_{t_j} in Step 8 and its corresponding arcs in Step 10 based on \mathcal{L}_{t_j} . In Step 11 of Algorithm 1 we extend the arborescence \mathcal{T}_i which was on $v_{in}^l \rightarrow v_{out}^l$ (if any) in $D_{|V|-j}$ to the in-arc $v_{in}^l \rightarrow v_{t_j}$ in $D_{|V|-j+1}$. This assignment is always valid, except when $v_{in}^l = t$, i.e., it is the out-arc of the root, in which case we simply remove arc $t \rightarrow v_{t_j}$ from \mathcal{T}_i . However, for the out-arcs of v_{t_j} in Step 12 we need to handle the three cases in Theorem 2 to exclude extensions which disrupt connectivity or cause cycles, and to extend arborescences on the remaining uncolored out-arcs. Note that, arborescences are unchanged on the (remaining) arcs of $A_{|V|-j}$, we only extend them to the in- and out-arcs of v_{t_j} in \mathcal{L}_{t_j} based on the arborescence(s) on the arcs $v_{in}^l \rightarrow v_{out}^l$, keeping all modifications local to v_{t_j} .

C. Optimization Options of Path Length in Algorithm 1

Although Theorem 2 gives an exact algorithm to construct perfect routing arborescences, there is some freedom in Algorithm 1 which can be used to improve the path length:

- **(O1) Split-off order in Step 2:** Although the nodes have to be split-off with increasing local connectivity according to Theorem 2, the order of nodes with the same $r(D_{|V|}, v, t)$ value is arbitrary. As a heuristic approach we split-off them in decreasing hop distance $h(v)$ to mimic that the arborescences are growing from the root in the reverse order [11].
- **(O2) Splittable arc-pairs in Step 3:** In order to split-off v_{t_j} we have to find $\delta^{\leftarrow}(v_{t_j})$ splittable pairs among the in-arcs and out-arcs of v_{t_j} . Although intuitively this could require a lot of calculation, in [11] it was demonstrated that it can be selected efficiently. Our proposed heuristic approach in this paper tries to find

Algorithm 1: Perfect Routing Arborecence Construction Algorithm

Input: Eulerian digraph $D_{|V|} = (V, A)$; root node t
Output: Routing arborescences $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_{\delta^{\leftarrow}(t)}$
 // Phase 1: Graph Decomposition
 1 Initialize list \mathcal{S} and sets $\forall v_{t_j} \in V : \mathcal{L}_{t_j}$ empty
 2 Calculate split-off order $\mathcal{S} = (v_{t_1}, v_{t_2}, \dots, v_{t_{|V|-1}})$ for root node t in $D_{|V|}$
 3 **for** $v_{t_j} \in \mathcal{S}$ **do**
 4 Split-off all $l = 1, 2, \dots, \delta^{\leftarrow}(v_{t_j})$ arc-pairs of v_{t_j} from $D_{|V|-j+1}$ to obtain $D_{|V|-j}$
 5 Store arc-pairs $\{v_{in}^l \rightarrow v_{t_j}, v_{t_j} \rightarrow v_{out}^l\}_l$ in \mathcal{L}_{t_j}
 // Phase 2: Arborecence Construction
 6 Assign $\delta^{\leftarrow}(t)$ different \mathcal{T}_i to the loop arcs of t in D_1
 7 **for** $v_{t_j} \in reverse(\mathcal{S})$ **do**
 8 $D_{|V|-j+1} := (V_{|V|-j} \cup \{v_{t_j}\}, A_{|V|-j})$
 9 **for** $l = 1, 2, \dots, \delta^{\leftarrow}(v_{t_j})$ **do**
 10 Add new arcs $\{v_{in}^l \rightarrow v_{t_j}, v_{t_j} \rightarrow v_{out}^l\}_l \in \mathcal{L}_{t_j}$ to $A_{|V|-j+1}$ and remove arc $v_{in}^l \rightarrow v_{out}^l$
 11 Assign arborecence \mathcal{T}_i on $v_{in}^l \rightarrow v_{out}^l$ to the in-arc $v_{in}^l \rightarrow v_{t_j}$
 12 Assign arborescences to the out-arcs of v_{t_j} according to Case 1A, Case 1B and Case 2

node split-offs with least bidirectional arc-pairs to make the decomposition distinct from undirected algorithms, using the least number of loop arcs as tiebreaker. If there are still multiple candidates, a random one is selected.

- **(O3) Extending arborescences in Step 12:** If multiple out-arcs $v_{t_j} \rightarrow v_i$ inherited the same arborecence \mathcal{T}_i from $D_{|V|-j}$, after excluding out-arcs which disrupt connectivity, as a heuristic approach we select the out-arc (if still multiple remain) which results in the shortest path from v_{t_j} to the root in \mathcal{T}_i . Similarly, if v_{t_j} inherited less than $r(D_{|V|-j+1}, v_{t_j}, t)$ arborescences on its in-arcs (Case 2), we extend the arborecence \mathcal{T}_i in which v_{t_j} reaches the root along the shortest path.
- **(O4) Post-Processing:** In order to optimize the path lengths in the arborescences, we can conduct a post-process approach where for each \mathcal{T}_i arborecence, $i = 1, \dots, \delta^{\leftarrow}(t)$ we erase all arcs of $\forall \mathcal{T}_j \neq \mathcal{T}_i$ to ensure arc-disjointness, and replace \mathcal{T}_i with the Shortest Path Tree (SPT) \mathcal{T}' to root t in the residual graph.

D. Complexity Analysis of Algorithm 1

Here, we investigate the time complexity of Algorithm 1. Note that, efficient algorithms are known [26] for finding s - t cuts in undirected graphs leveraging Gomory–Hu tree construction [27]. However, these results only generalize to symmetric digraphs (i.e., for each arc (v, w) there is an arc (w, v) in D). Although we maintain the Eulerian property for digraphs D_i throughout the algorithm, D_i is not necessarily symmetric. Hence, in the current analysis we rely only on

known algorithms with established worst-case bounds. We assume that the maximum in-degree (and out-degree) in Eulerian digraph $D_{|V|}$ (denoted by Δ) is a small fixed value that does not depend on the number of nodes $n = |V|$.

First, we calculate the overall complexity of the graph decomposition (i.e., Phase 1). In order to get \mathcal{S} , we calculate local connectivity of a node to t by using Diniz’s maximum flow algorithm with dynamic trees [28], which gives $O(m \cdot n^2 \cdot \log n)$ for all nodes, where $m = |A|$. Sorting \mathcal{S} in increasing order takes $O(n \cdot \log n)$. We iterate through this list and split-off the nodes by finding splittable arc-pairs among its arcs. Note that the maximum number of arc-pairs at any node is $O(\Delta!)$, as this corresponds to the number of distinct perfect matchings between Δ in-arcs and Δ out-arcs, i.e., the number of bijective mappings between the two sets. In order to check if an arc-pair is splittable, we need to calculate if it preserves local connectivity between all remaining node pairs, which requires n^2 maximum flow calculations, giving $O(m \cdot n^3 \cdot \log n)$. Therefore, the total complexity of calculating a CP-sequence and to find the split-off arc-pairs is $O(\Delta! \cdot m \cdot n^3 \cdot \log n)$.

Next, we compute the complexity of the routing arborecence construction in Phase 2 of Algorithm 1. Extending the digraph with the new nodes and arcs can be done in $O(m + n)$ linear time. In order to exclude invalid arborecence assignments when extending them on the new out-arcs, we have to check cycles on the arcs of \mathcal{T}_j in $O(n)$ time, which gives $O(n^2 \cdot \Delta)$ complexity in total with a Breadth First Search (BFS) algorithm. Therefore, the overall complexity of Algorithm 1 is dominated by Phase 1, resulting in $O(\Delta! \cdot m \cdot n^3 \cdot \log n)$.

Finally, we analyze the additional complexity added by the optimization options in Section IV-C. Computing the hop distance $h(v)$ of node v and root t in O1 can be done in linear time $O(n \cdot \Delta)$ with BFS, and the list can be sorted in $O(n \cdot \log n)$. Evaluating each valid node split-off in O2 requires to check all in- and out-arc pairings of v , which gives $O(n \cdot \Delta!)$ for the whole graph. To extend the arborescences in O3 we need to calculate the shortest paths from v_{t_j} to t in Δ arborescences \mathcal{T}_i in worst case, and selecting the ones with minimum distances, adding $O(n \cdot \Delta)$ complexity. In O4 in each step we remove the arcs of $\Delta - 1$ arborescences and calculate a shortest path tree in the remaining graph with a BFS algorithm in $O(n \cdot \Delta)$ time, repeated for each Δ arborescences, resulting in $O(n \cdot \Delta^2)$ complexity in total. Therefore, we conclude that none of the four optimization options increase the overall complexity of Algorithm 1.

V. SIMULATION RESULTS

In this section we present our experiment results on the performance of the perfect routing arborescences. We conducted our simulations on a MacBook Pro 2022 equipped with an Apple M2 chip and 16 GB RAM, running a Sequoia 15.5 MacOS operating system. Our algorithms were implemented in Python 3.13.2. For performance testing on real-world networks we selected small topologies from the SNDLib

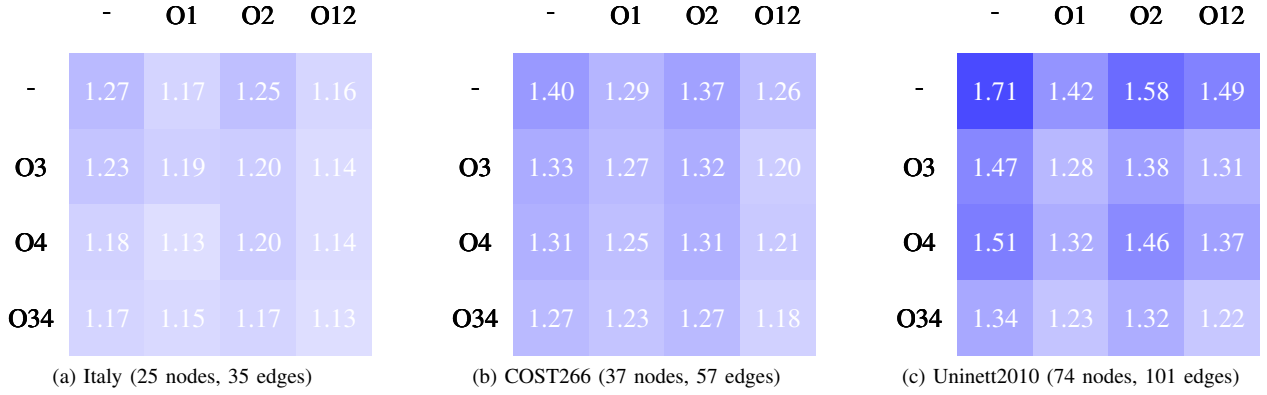


Fig. 3. Path stretch along the shortest arborescence \mathcal{T}_i at root t averaged for all root nodes. Oxy denotes that both Ox and Oy was used in Algorithm 1. Light colors indicate low average path stretch, while the color becomes darker as average path stretch increases.

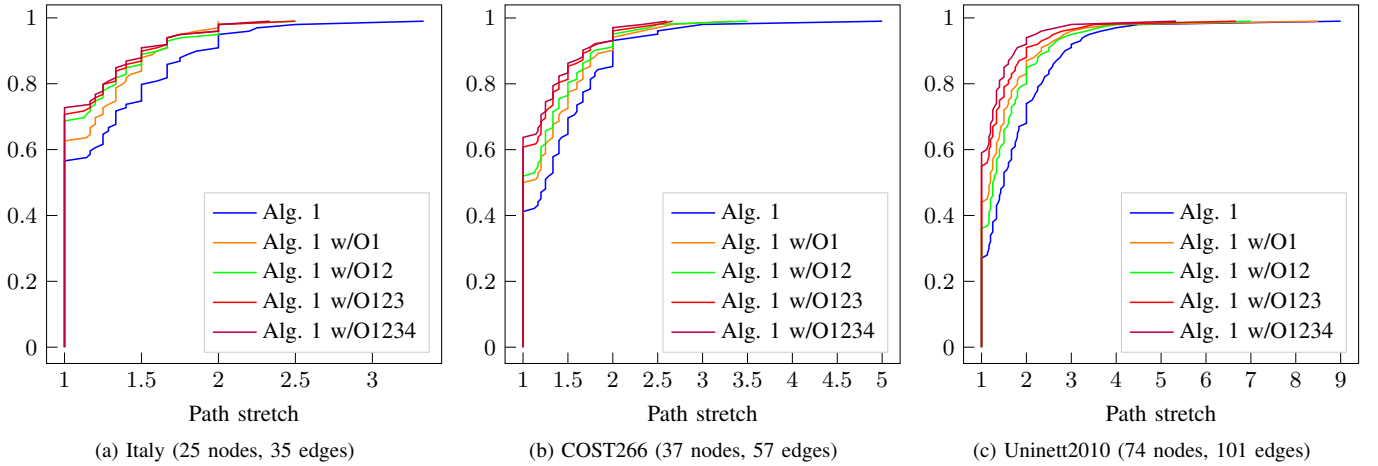


Fig. 4. Cumulative distribution of the path stretch along the shortest arborescence \mathcal{T}_i at root t averaged for all root nodes. Oxy denotes that both Ox and Oy was used in Algorithm 1. When path stretch is 1, the packet is routed along the shortest path.

database [8], and larger networks including ones with high Δ values from the Internet TopologyZoo [29] repository. As the main performance metric we evaluated the *path stretch* [17] in the arborescences compared to the hop distance along the shortest path tree, i.e., the trade-off we have to pay for the scalability of routing tables in FRR [2]. Formally, path stretch for arborescence \mathcal{T} at root t is defined as:

$$\text{stretch}(\mathcal{T}) = \sum_{\substack{v=1 \\ v \neq t}}^{|V|} \frac{h_{\mathcal{T}}(v)}{h_{SPT}(v)}.$$

A. Analyzing Algorithm 1 Optimization Options

We investigated the individual effect of the optimization options O1-O4 listed in Section IV-C on the path stretch. As the network spends most of the time in an operating state without link failures, we were interested in the path stretch of the arborescences \mathcal{T}_i with the minimum hop distance towards root t . The averaged results for all root nodes are given in Figure 3. As the network size increases, the average path stretch of Algorithm 1 without any optimization increases from 1.27 to 1.71. Although the Italy network in Figure 3(a) and the

COST266 topology in Figure 3(b) have similar average nodal degree and shortest path length (refer to Table II), the average path stretch still increases from 1.27 to 1.40 because the arborescences need to make larger detours.

Investigating different combinations of the optimization options, one can observe that all of them improves the performance of Algorithm 1. Individually O1 with the node split-off in decreasing hop distance has the largest gain in path quality compared to the unoptimized solution, while the arborescence extension along the shortest path in O3 and post-processing in O4 give significant improvement as well. The least effective option is the selection of splittable arc-pairs in O2. Note that, we investigated multiple heuristics in O2 before deciding to leverage the directed nature of our problem described in Section IV-C, but all of them performed worse than the other three optimization options and seemed to be the most dependent on the characteristics of the input graph as well. However, we decided to keep it as in all investigated network topologies combining O2 with the other three options, Algorithm 1 w/O1234 reached the best performance according to Figure 3 by improving the average

TABLE II
 SHORTEST PATH LENGTH, PATH STRETCH AND ROUTING PERFORMANCE OF ALGORITHM 1 IN DIFFERENT REAL-WORLD NETWORK TOPOLOGIES

Network topologies				Shortest path length [hop count]	Path stretch									Running time	Circular routing		
Name	$ V $	$ E $	Δ		Arc-disjoint paths			Alg. 1			Alg. 1 w/O1234			Alg. 1 w/O1234 avg per root [sec]	Alg. 1 w/O1234 [root %]		
				min	avg	max	min	avg	max	min	avg	max		\forall global	\exists global	\exists local	
Abilene	10	14	3	2.42	1.009	1.540	2.066	1.151	1.815	2.491	1.031	1.665	2.292	0.022	100	100	100
Italy	25	35	4	3.71	1.039	1.604	2.215	1.278	2.390	3.633	1.136	1.953	2.831	0.252	100	100	100
USA	26	42	5	3.28	1.076	1.677	2.343	1.273	2.421	3.775	1.147	2.068	3.186	0.472	92.3	100	100
COST266	37	57	5	3.73	1.069	1.574	2.100	1.400	2.504	3.697	1.186	2.133	3.169	1.541	83.7	97.3	100
Germany	50	88	5	4.04	1.051	1.569	2.151	1.445	2.858	4.506	1.238	2.363	3.728	7.376	18.0	72.0	90
Uninett2010	74	101	8	4.58	1.024	1.305	1.592	1.713	2.389	3.102	1.227	1.679	2.160	8.687	82.4	97.3	100
TataNId	145	186	6	9.84	1.055	1.407	1.773	1.740	3.076	4.454	1.232	2.047	2.860	37.865	93.7	93.7	94.4
USFibre	170	229	8	9.63	1.061	1.421	1.802	1.848	3.190	4.623	1.286	1.957	2.736	87.256	88.8	88.8	88.8

path stretch of Algorithm 1 by 10-30% without significantly increasing the running time of the algorithm.

B. Path Stretch Distribution in the Shortest Arborescences

Besides the average minimum path stretch, in Figure 4 we investigated the cumulative distribution of the individual path stretches along arborescences \mathcal{T}_i with the minimum hop distance towards root t by applying the optimization options incrementally. One can observe that in the base solution of Algorithm 1 only about 56%, 42%, and 27% of the node-pairs are using the shortest path (i.e., path stretch equals to 1) for communication in the failure-less state in the Italy, COST266 and Uninett2010 topologies, respectively, which leads to inefficient bandwidth usage. However, with the application of O1-O4 not only this percentage was significantly improved, but in the Uninett2010 topology in Figure 4(c) it was even doubled by Algorithm 1 w/O1234 to 59%, resulting in a more efficient bandwidth resource consumption and lower communication latency along the perfect routing arborescences. As Algorithm 1 w/O1234 has the best performance, in the rest of the paper we only focus on Algorithm 1 as the baseline without optimization, and on Algorithm 1 w/O1234 as our recommended method applying all four optimization options to minimize path stretch.

C. Algorithm 1 Performance in Large Real-World Topologies

In Table II we selected real world network topologies with different sizes, densities and maximum nodal degrees Δ to analyze the performance of the perfect routing arborescences created by Algorithm 1. Note that, $|E|$ denotes the number of undirected edges, which are directed in both directions in the input of our algorithm. We added the average shortest path length measured in hop count between each v, t pair to the table as a reference. We also calculated the shortest $r(D, v, t)$ arc-disjoint paths between each v, t pair which provide the minimum bandwidth achievable by any approach without considering the scalability of routing tables in FRR [30]. Hence, in contrast with routing arborescences these paths cannot be used in destination based routing efficiently (see Section V-D for further details).

On one hand, the path stretch increase of the shortest arborescence path (*min* in the table) for Algorithm 1 without any optimization is 74% in worst case compared to the shortest arc-disjoint paths, while even more than double in

average and maximum for the large networks, i.e., TataNId and USFibre. On the other hand, Algorithm 1 w/O1234 lowers the shortest arborescence path increase to 21%, while for the Germany network the worst case average and maximum stretch increases are 51% and 73%, respectively, i.e., our arborescences use 3 more arcs on average than the theoretical minimum, which is the price we have to pay for an efficient routing implementation.

Finally, we analyzed the running time of Algorithm 1 with our Python implementation. Note that, our measurement results confirmed our complexity analysis in Section IV-D, as the optimization options did not increase the running time significantly, thus, we only present the results for Algorithm 1 w/O1234 in Table II. Our results showed that in smaller networks the routing arborescences can be constructed below a second on average, which started to increase in larger networks with large maximum nodal degree Δ and shortest path length. Note that, Δ increases the computation time for splittable arc-pairs in Step 3, while the network diameter affects the cycle check during arborescence construction in Step 12 of Algorithm 1, resulting 87 sec on average for the 170 node USFibre network. However, note that arborescence and routing table construction is a pro-active off-line process [3], i.e., not restricted by the short time-scale of failure recovery.

D. Routing Along the Perfect Arborescences

We also investigated the routing performance of our perfect arborescences. We assumed *circular routing* [5] as the simplest and most used approach for arborescence routing [2], where an order of \mathcal{T}_i is defined to each in-arc per destination, and the packet is forwarded on the same, or upon link failure on the next available arborescence in the list. Although [2] proved that all circular routing is good until $r(G) \leq 3$ and good order exists for $r(G) = 4$ spanning arborescences, unfortunately, these results do not generalize to our (non-spanning) routing arborescences, as forwarding loop can occur even for $r(D, v, t) = 3$ if the order is not selected carefully. Therefore, we simulated all possible circular orders and link failure scenarios³ up to $r(D, v, t)$. The % of root nodes where an arborescence order provided a loop-free FRR solution is given in the last columns of Table II.

³Link failure means the failure of both corresponding directed arcs.

TABLE III
COMPARISON OF ALGORITHM 1 WITH OTHER ROUTING ARBORESCENCES
FOR 20 RANDOM ROOT NODES

Network topologies				Shortest path [hop]	Average path stretch			
Name	V	E	Δ		Arc-disj. paths	Alg. 1 w/O1234	DLCP [11]	DAG-FRR [9]
German	17	26	6	2.69	1.568	1.823	2.212	2.585
ARPA	21	25	4	3.46	1.756	2.123	2.343	2.294
EU	22	45	7	2.44	1.587	2.016	2.505	2.728
USA	26	42	5	3.19	1.622	2.094	2.529	2.768
Nobel	28	41	5	3.39	1.608	2.146	2.232	2.531
Italy	33	56	7	3.74	1.535	1.960	2.147	2.418
COST266	37	57	5	3.69	1.574	2.148	2.431	2.525
US North	39	61	5	4.45	1.577	1.984	2.112	2.304
NFSNet	79	108	5	5.52	1.500	2.567	2.582	3.060

One can observe that except in the 50 node Germany network, for at least 82.4% of roots all global circular order were good, and for further root nodes although not all orders provided loop-free paths upon link failures, there existed a good circular order where the packet reached the destination. We also implemented a simple heuristic⁴ where each node can have a different (i.e., local) circular order of out-arcs, but except for the Germany network the gain was marginal. Therefore, for the scenarios when no circular order exist, similarly to spanning arborescences, e.g., a fixed number of packet header bits and packet header rewrite at intermediate routers might be necessary to deliver the packet to the root [2]. However, the complexity of such packet forwarding is still lower than the routing on the arc-disjoint paths, where even for $r(D, v, t) = 2$ extra packet header bits are required [30] and can not be used in destination based routing for more than two paths.

E. Comparing Algorithm 1 to State-of-the-Art Approaches

The performance of our perfect solution was compared to arborescences given by the DLCP-sequences [11] and DAG-FRR [9], which similarly can provide $r(D, v, t)$ arc-disjoint paths depending on the local connectivity between the nodes. However, the primary goal of these methods is to calculate the maximum number of $r(D, v, t)$ arborescences for all nodes while minimizing path length is only a secondary objective. In contrast, all the solutions of Algorithm 1 are perfect routing arborescences according to Theorem 2, thus, it can focus on finding arborescences with the shortest paths through optimization options O1-O4 as its primary objective. For a fair comparison we used the same small topologies as [11] and similarly we averaged the results for 20 random root nodes, shown in Table III.

One can observe that the average path stretch in the arborescences of Algorithm 1 is 1-20% lower than DLCP (and even more compared to DAG-FRR), which results around 1 less hop per path on average in most topologies than in its counterparts. Note that, Algorithm 1 not only has superior performance in terms of average path stretch compared to the

⁴Although SyPer [18] provides solutions for $r(G) \leq 4$ fast, it is unable to handle the larger $r(D, v, t) - 1$ values we encountered efficiently.

other two approaches, but the arborescences have $r(D, v, t)$ paths for each node in the network towards the root, while DLCP provides sufficient paths only for 99.9% of nodes [11], and DAG-FRR has only 92.6% coverage in worst case in the investigated topologies. Furthermore, the measured running times of Algorithm 1 are in the same magnitude as the total running time of DLCP provided in [11] for the small networks in Table III, e.g. 7.013 sec on average for the NSFNet network compared to ≈ 6 sec for DLCP. However, we expect DLCP to scale worse than Algorithm 1 for larger networks with high Δ nodes shown in Table II owing to the involved split-off operation for odd-degree nodes in undirected graphs and the corresponding ILPs for arborescence construction.

VI. CONCLUSIONS

In this paper, we presented a constructive proof that perfect routing arborescences always exist in Eulerian digraphs, i.e., they provide the theoretical maximum number of arc-disjoint paths from every node v towards the root t along the arborescences (i.e., equals the local connectivity between v and t). Therefore, a fast reroute built on these perfect routing arborescences is able to tolerate k link failures in any real-world network with bidirectional communication links, where $k + 1$ is the local connectivity between the source and destination of the packet. While the study of arborescences in undirected graphs has been ongoing for over four decades, Theorem 2 offers a surprisingly compact new result in this well-established field, beyond its practical relevance to FRR. We see this as a noteworthy example of how contributions from the networking community may advance theoretical computer science.

Building on the constructive proof, we identified four optimization options to minimize path length along the arborescences to improve bandwidth efficiency and minimize communication latency. Note that, previously the existence of perfect routing arborescences was only a de facto conjecture [11], thus, those algorithms could not rely on such strong theoretical background as Theorem 2 and left optimal network performance as a secondary objective while focusing on finding the maximum number of routing arborescences for every node. Therefore, our Algorithm 1 w/O1234 is the first to target path length minimization as its primary objective while providing perfect routing arborescences, achieving substantial improvements in path stretch and hop count in simulations compared to prior approaches.

The authors have provided public access to their code at https://github.com/peterbabarczy/routing_arborescences.

ACKNOWLEDGMENTS

This work was supported in part by the National Research, Development and Innovation Fund of Hungary, financed through the K_23 funding scheme under Project 146347. The authors would like to express their sincere gratitude to András Frank and Lajos Rónyai for the discussions and their valuable insights that contributed to this research work.

REFERENCES

- [1] D. Turner, K. Levchenko, A. C. Snoeren, and S. Savage, "California fault lines: understanding the causes and impact of network failures," in *ACM SIGCOMM*, 2010, pp. 315–326.
- [2] M. Chiesa, I. Nikolaevskiy, S. Mitrović, A. Gurtov, A. Madry, M. Schapira, and S. Shenker, "On the resiliency of static forwarding tables," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1133–1146, 2017.
- [3] M. Caesar, M. Casado, T. Koponen, J. Rexford, and S. Shenker, "Dynamic route recomputation considered harmful," *SIGCOMM CCR*, vol. 40, no. 2, pp. 66–71, Apr. 2010.
- [4] M. Chiesa, A. Kamisinski, J. Rak, G. Retvari, and S. Schmid, "A survey of fast-recovery mechanisms in packet-switched networks," *IEEE Communications Surveys and Tutorials*, pp. 1–50, 2021.
- [5] K.-T. Foerster, J. Hirvonen, Y.-A. Pignolet, S. Schmid, and G. Tredan, "On the feasibility of perfect resilience with local fast failover," in *SIAM Symposium on Algorithmic Principles of Computer Systems (APOCS)*, 2021, pp. 55–69.
- [6] M. Chiesa, I. Nikolaevskiy, S. Mitrović, A. Panda, A. Gurtov, A. Maidry, M. Schapira, and S. Shenker, "The quest for resilient (static) forwarding tables," in *IEEE INFOCOM*, 2016, pp. 1–9.
- [7] J. Edmonds, "Edge-disjoint branchings," *Combinatorial Algorithms*, pp. 91–96, 1973.
- [8] S. Orłowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0—Survivable Network Design Library," *Networks*, vol. 55, no. 3, pp. 276–286, 2010.
- [9] K.-T. Foerster, A. Kamisiński, Y.-A. Pignolet, S. Schmid, and G. Tredan, "Grafting arborescences for extra resilience of fast rerouting schemes," in *IEEE INFOCOM*, 2021, pp. 1–10.
- [10] J. Tapolcai, P. Babarcsi, P.-H. Ho, and L. Rónyai, "Resilient routing table computation based on connectivity preserving graph sequences," in *IEEE INFOCOM*, 2023, pp. 1–10.
- [11] J. Tapolcai, P. Babarcsi, B. Brányi, P.-H. Ho, and L. Rónyai, "Connectivity preserving graph sequences for routing arborescence construction," *IEEE Journal on Selected Areas in Communications*, vol. 43, no. 2, pp. 484–494, 2025.
- [12] M. Chiesa, A. Gurtov, A. Madry, S. Mitrović, I. Nikolaevskiy, M. Schapira, and S. Shenker, "On the resiliency of randomized routing against multiple edge failures," in *International Colloquium on Automata, Languages and Programming (ICALP)*, 2016, pp. 1–15.
- [13] J. Feigenbaum, B. Godfrey, A. Panda, M. Schapira, S. Shenker, and A. Singla, "Brief announcement: On the resilience of routing tables," in *ACM Symposium on Principles of Distributed Computing (PODC)*, 2012, pp. 237–238.
- [14] B. Yang, J. Liu, S. Shenker, J. Li, and K. Zheng, "Keep forwarding: Towards k-link failure resilient routing," in *IEEE INFOCOM*, 2014, pp. 1617–1625.
- [15] K.-T. Foerster, J. Hirvonen, Y.-A. Pignolet, S. Schmid, and G. Tredan, "On the price of locality in static fast rerouting," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2022, pp. 215–226.
- [16] A. Bhalgat, R. Hariharan, T. Kavitha, and D. Panigrahi, "Fast edge splitting and Edmonds' arborescence construction for unweighted graphs," in *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2008, pp. 455–464.
- [17] K.-T. Foerster, Y.-A. Pignolet, S. Schmid, and G. Tredan, "Local fast failover routing with low stretch," *SIGCOMM CCR*, vol. 48, no. 1, pp. 35–41, Apr. 2018.
- [18] C. Györgyi, K. G. Larsen, S. Schmid, and J. Srba, "Syper: Synthesis of perfectly resilient local fast re-routing rules for highly dependable networks," in *IEEE INFOCOM*, 2024, pp. 2398–2407.
- [19] A. Frank and T. Jordán, "Graph connectivity augmentation," *Handbook of Graph Theory, Combinatorial Optimization, and Algorithms*, pp. 313–346, 2015.
- [20] L. Lovász, "On the ratio of optimal integral and fractional covers," *Discrete Mathematics*, vol. 13, no. 4, pp. 383–390, 1975.
- [21] W. Mader, "A reduction method for edge-connectivity in graphs," in *Annals of Discrete Mathematics*. Elsevier, 1978, vol. 3, pp. 145–164.
- [22] H. N. Gabow, "A matroid approach to finding edge connectivity and packing arborescences," *Journal of Computer and System Sciences*, vol. 50, no. 2, pp. 259–273, 1995.
- [23] A. Frank, "On a theorem of Mader," *Discrete Mathematics*, vol. 101, no. 1, pp. 49–57, 1992.
- [24] J. Bang-Jensen, A. Frank, and B. Jackson, "Preserving and increasing local edge-connectivity in mixed graphs," *SIAM Journal on Discrete Mathematics*, vol. 8, no. 2, pp. 155–178, 1995.
- [25] A. Frank, "Connections in combinatorial optimization," *Discrete Applied Mathematics*, vol. 160, no. 12, p. 1875, Aug. 2012.
- [26] L. C. Lau and C. K. Yung, "Efficient edge splitting-off algorithms maintaining all-pairs edge-connectivities," *SIAM Journal on Computing*, vol. 42, no. 3, pp. 1185–1200, 2013.
- [27] R. E. Gomory and T. C. Hu, "Multi-terminal network flows," *Journal of the Society for Industrial and Applied Mathematics*, vol. 9, no. 4, pp. 551–570, 1961.
- [28] E. A. Dinic, "Algorithm for solution of a problem of maximum flow in networks with power estimation," in *Soviet Math. Doklady*, vol. 11, 1970, pp. 1277–1280.
- [29] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, October 2011.
- [30] P. Babarcsi, G. Rétvári, L. Rónyai, and J. Tapolcai, "Routing on the shortest pairs of disjoint paths," in *IFIP Networking*, 2022, pp. 1–9.