Connectivity Preserving Graph Sequences for Routing Arborescence Construction

János Tapolcai, Péter Babarczi, Balázs Brányi, Pin-Han Ho, and Lajos Rónyai

***Abstract*—Fast reroute (FRR) is among the fastest survivable routing approaches in packet-switched networks, because the routers are equipped with a resilient routing table in advance such that the packets can be rerouted instantly upon failures solely relying on local information, i.e., without notification messages. However, designing the routing algorithm for FRR is challenging as the number of possible sets of failed network links can be extremely high, while the algorithm should keep track of which routers are aware of the failure. Therefore, FRR methods often rely on spanning arborescences, which provide multiple disjoint failover paths up to the global connectivity of the network. In this paper, we propose a generic algorithmic framework that theoretically increases the number of failover paths to the local connectivity between each node and the root by extending an efficient connectivity preserving operation from graph theory – called edge splitting-off – to decompose the network topology node-by-node, and use Integer Linear Programs (ILPs) on these partial subproblems to build routing arborescences in the reverse direction for the original topology. Although our practical implementation cannot reach the local connectivity in all instances, we demonstrate through simulations that it still outperforms the state-of-the-art FRR mechanisms and provides better resilience with shorter paths in the arborescences.**

***Index Terms*—fast reroute, routing arborescences, edge splitting-off, survivable routing**

A preliminary version of this paper was presented at the IEEE INFOCOM conference, New York, NY, USA, May 2023.

This work was partly supported by Project no. 134604 and Project no. 146347, which have been implemented with support from the National Research, Development and Innovation Fund of Hungary, financed under the FK_20 and K_23 funding schemes, respectively. The research of L. Rónyai was partly supported by the Hungarian NRDI Office within the Artificial Intelligence National Laboratory Program framework, and in part by NRDI through the Program of Excellence TKP2021-NVA-02 at the Budapest University of Technology and Economics. The authors would like to express their sincere gratitude to András Frank for the discussions and his valuable insights that contributed to this research work. Pin-Han Ho serves as the corresponding author.

J. Tapolcai, P. Babarczi and B. Brányi are with the MTA-BME Information Systems Research Group, 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 (e-mail: {tapolcai, babarczi, branyi}@tmit.bme.hu).

Pin-Han Ho is with the Shenzhen Institute for Advanced Study, UESTC, China; and the Department of Electrical and Computer Engineering, University of Waterloo, Canada (e-mail: p4ho@uwaterloo.ca).

L. Rónyai is with the HUN-REN Institute for Computer Science and Control; and the Department of Algebra and Geometry, Institute of Mathematics, Budapest University of Technology and Economics, Műegyetem rkp. 3., H-1111 Budapest, Hungary (e-mail: ronyai@sztaki.hu).

I. Introduction

Traditional communication networks were prepared to survive through single link and node failures as the chance of having two independent failure events within a short period is very small [1]. Accordingly, it is a common assumption that there is sufficient time to restore a failure before the next one occurs. However, with the increasing complexity of multilayer networks [2], the effect of a failure event in the physical infrastructure often manifests as multiple simultaneous link and node failures in upper layers [3]. As Internet service providers often lease the network from a physical infrastructure provider [4], [5], the correlation between failures might be completely hidden. Thus, they have to prepare their IP networks for an excessive number of simultaneous failures.

Dynamic routing table recomputation immediately after failures might be harmful to critical connections [6] as the control plane struggles to provide strict timing requirements [7], [8]. Therefore, fast reroute (FRR) mechanisms [8]–[11] were proposed, which provide failover paths with precomputed routing tables towards each root node in the data plane against as many failures as possible, purely based on local failure information. Among several FRR implementations, deterministic methods built on spanning trees (or arborescences) are usually proposed for intra-domain IP networks, where the topology is known and well connected [8], [11]. Spanning arborescences can go beyond single failure resilience and exploit the higher connectivity of the networks towards "perfect resilience", i.e., source node s can reach root node t as long as the failure does not isolate s from t. Unfortunately, perfect resilience is not always achievable with pre-computed *static rules* [12], [13].

Spanning arborescences perform well in homogeneous graphs, where the number of link-disjoint paths between an arbitrary source s and root t is close to the *global connectivity*[1] k of the network. However, in heterogeneous graphs where nodes in dense subgraphs have significantly more link-disjoint paths towards t than k, spanning arborescences cannot fully explore the potential of such additional redundancy. Although methods using partial arborescences in these dense components exist [8], it is not clear how the subpaths can be efficiently "glued together" into static routing tables. Therefore, our goal is to design an efficient arborescence-based

[1]Minimum number of links (nodes) whose removal from the network will separate the remaining nodes into at least two isolated components.

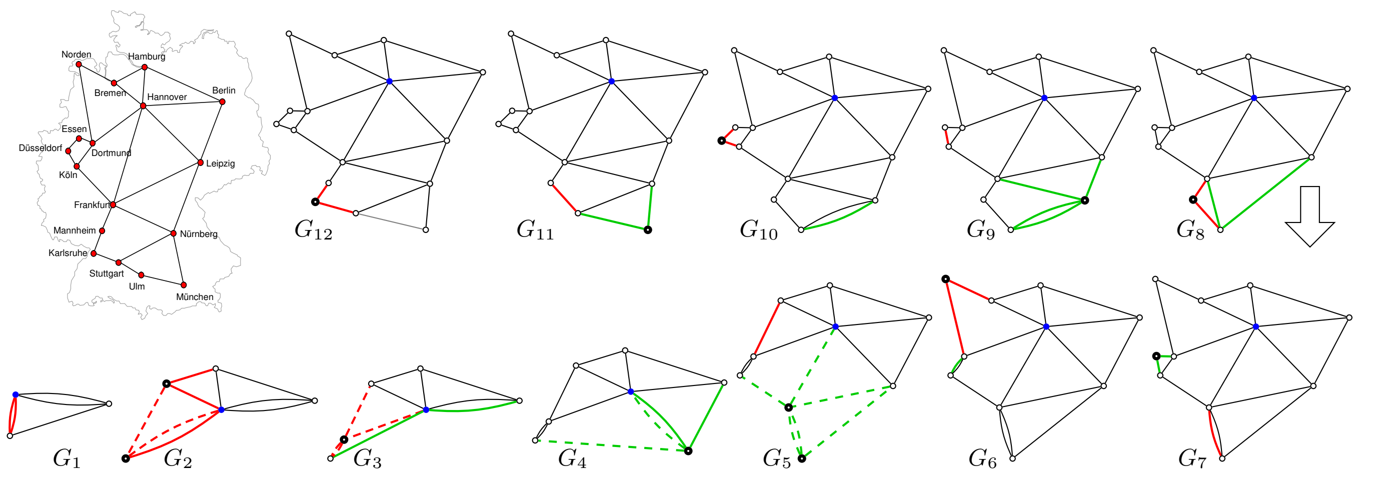


Fig. 1. Degree and local connectivity preserving graph sequence for the 17-node German backbone network [14]. The blue node is the root, the black larger nodes are the new nodes. We draw the edges involved in the transformation between G_{i-1} and G_i either with the same color or the same line style, e.g., the two red edges of G_1 correspond to the red edges (solid and dashed) of G_2, while the dashed (red) edges of G_2 correspond to the dashed edges of G_3.

FRR algorithm that provides resilience by surviving at least $r(s,t)-1$ link (arc) failures between source s and root t if the *local connectivity* (the number of link-disjoint paths between s and t) is $r(s,t)$, even if the global connectivity, i.e., the minimum local connectivity between every s and t pair of the graph is very small, e.g., $2=k<r(s,t)$.

In this paper, we propose a generic algorithmic framework that can handle complex cases with a massive number of simultaneous failures, and is applicable to several routing table computation problems for different local connectivity-based FRR implementations either using arborescences [8], [11], allowing randomized forwarding, or even applying packet header rewrite [9], [10]. Previously, arborescences have been used either for problems involving global connectivity, for networks with even degree nodes, for routing on colored trees and directed acyclic graphs, or as a black box approach. In contrast, the high-level idea of our framework is integrating Integer Linear Program (ILP) formulations with a graph theoretical approach namely *edge splitting-off* – which can be used to prove various properties of graphs with given global [15] and local [16] connectivity, often called *k-connected graph characterization* [17] – for achieving the desired efficiency and flexibility. Note that edge splitting-off was already successfully applied for different network design problems [18], [19], including Edmonds' arborescence construction [20], [21].

The rest of the paper is organized as follows. Section II introduces routing arborescences. Section III contains our general degree and local connectivity preserving graph decomposition algorithm based on edge splitting-off and our novel approach to handle odd degree nodes. The resulting (reverse) graph sequence is used for FRR routing table design, leveraging our general ILP formulations for arborescence construction in Section IV. Section V provides the scalability analysis of the framework. Finally, Section VI contains our simulation results while Section VII concludes our work.

II. Routing Arborescences

Fast reroute mechanisms for destination based hop-by-hop routing rely only on local information such as the destination (and source) of the packet, in-port the packet arrived, and set of failed adjacent links; thus, they provide an instantaneous reaction to failures without control plane messages [11]. Hence, routers not adjacent to a failure will forward packets as normal, as they have no information about the failure. It was already demonstrated that given a k-edge-connected graph, k-arc-disjoint *spanning arborescences* rooted at node t can be found efficiently by splitting-off edges by preserving global connectivity [20], [21], and $k-1$ arc-failures can be tolerated with static routing [9], i.e., with pre-configured routing tables and without changing packet headers, which follows a global circular permutation of $T_1,\ldots,T_k$, where the packet follows the same T_i as it came from unless the next-hop out-arc is failed. In this case, the packet is forwarded along T_{i+1} according to the global order. As the arborescences are arc-disjoint, the in-port uniquely identifies T_i.

Although they are simple, spanning arborescences [9], [21] cannot exploit the available edges in densely connected subgraphs of the topology, and thus, cannot provide $r(v,t)-1$ arc-failure resilience in worst-case $\forall v\neq t$ [8], [12], [13]. In order to give resilience guarantees beyond global connectivity in these subgraphs, directed acyclic graph (DAG) based method DAG-FRR was proposed [8], where in Part 1 rooted (partial) arborescences are built greedily according to the root's nodal degree, referred to as *routing arborescences*. In Part 2, as many unused edges are added to these arborescences as possible to form DAGs.

Our proposed framework handles the complexity of FRR with simultaneous failures and increases the resilience beyond the global connectivity with a scalable routing arborescence design through the following two stages:

(i) *General graph decomposition:* We generate a graph sequence $G_l, G_{l-1}, \ldots, G_2, G_1$ starting with $G_l = G$ (i.e., the network topology graph), where in each iteration we apply one of the following two simple rules:
   (i) Remove an even degree node other than the root node t by splitting off its edges according to [16];
   (ii) Remove two adjacent odd degree nodes other than the root node t by splitting off their edges with a novel approach proposed in Section III-C.

(ii) *Arborescence construction for FRR:* Use graph sequence $G_1, G_2, \ldots, G_{l-1}, G_l = G$ to iteratively build routing arborescences by solving an ILP locally for the new node(s) and edges in each step.

Fig. 1 shows an example of such graph sequence $G_1, \ldots, G_l$ for a real-world network. The graph G_i is one or two nodes larger than G_{i-1}, and the edges of G_{i-1} and G_i differ only around the new node(s). The degree of the nodes both in G_i and G_{i-1} does not change as we step from i to $i-1$ and the connectivity between pairs of such nodes does not decrease, i.e., $r_{i-1}(s,t) \geq r_i(s,t)$ for each G_i which we call *local connectivity preserving* property. Using this graph sequence, we can build up arborescences and the routing tables for FRR by solving the small trivial graph G_1, and in each step for G_{i+1} we compute a routing table with an ILP only for the new node(s), by simply copying the routing tables for the other nodes of G_i without any modifications.

III. Connectivity Preserving Graph Sequences

The main idea of our framework is to leverage the benefits of a graph sequence that can be efficiently generated and flexible enough to be used for resilient routing table computation. Here we formulate the requirements of the graph sequence, which keep the subsequent changes local and thus divides the overall complex design problem into simple local decisions. In Section III-A we formally introduce edge splitting-off, summarize the corresponding results from the literature and define the removal of even degree nodes. Armed with these results, we define a *degree and local connectivity preserving (DLCP)* graph sequence and prove that it always exists in 2-connected graphs in Section III-B. In Section III-C we propose a novel degree and local connectivity preserving edge splitting-off operation for two odd degree nodes, which enables us to generate DLCP graph sequences in Section III-D.

A. Degree and Connectivity Preserving Edge Splitting-Off

We denote an undirected graph as $G = (V, E)$, and use the notation $d(v)$ for the degree of node $v \in V$, and $r(s,t)$ for the edge-connectivity between $s, t \in V$. The edge-connectivity between $s, t \in V$ is the maximum number of edge-disjoint paths connecting s and t. Notations are summarized in Table I.

TABLE I
Notation list

Notation	Description
$G = (V, E)$	Undirected graph with node set V, edge set E.
t	Root node $t \in V$ of the routing arborescences.
$r(s,t)$	Local edge-connectivity between node s, and root t
$d(v)$	Degree of node $v \in V$.
$\delta(v)$	Hop distance of node v from root t.
$G_i = (V_i, E_i)$	The i^{th} graph in the DLCP graph sequence.
χ_i	The number of tear-off edges in G_i.
α_i	The number of edges that become parallel in step i.
β_i	The number of edges become loop in step i.
γ_i	The total Euclidean distance of the new edges in step i (assume node coordinates given).

Definition 1: If we remove edges (x,u), (x,v) and add the edge (u,v), we say that edge pair $(x,u),(x,v)$ has been *split-off* from node x. We say that the edge pair $(x,u),(x,v)$ is *splittable* if in the above resulting graph there are still at least $r(s,t)$ edge-disjoint paths between s and t, $\forall s,t \in V \setminus \{x\}$.

The edge splitting-off operation makes a little change in the graph [22], [23], all the nodes have the same nodal degree apart from the node x. Note that it may add a parallel edge to the graph. We are interested in edge splitting-off that not only preserves the global edge-connectivity k of the graph [24, Problem 6.53] but also does not change the local edge-connectivity between any pair of nodes (apart from x). We will use the following related theorem.

Theorem 1 (Mader [16]): Let $G = (V, E)$ be an undirected graph that has at least $r(s,t) \geq 2$ edge-disjoint paths between s and t for all $s,t \in V \setminus \{x\}$, and x is not incident to a cut-edge. If $d(x) \neq 3$, then some edge pair $(x,u),(x,v)$ can be split off so that in the resulting graph there are still at least $r(s,t)$ edge-disjoint paths between s and t, $\forall s,t \in V \setminus \{x\}$.

In our case, we would like to *remove a node*; thus, we split off all of its adjacent edges as edge pairs, one after the other. As a resilient topology G is at least 2-connected, there are no cut-edges in the graph; thus, if the node has an even degree, then by the above theorem, this can always be done.

For example, in Fig. 2a a possible way is shown to remove node v_i from G_i by splitting off its edges while the degree and local connectivity of the remaining nodes in G_{i-1} are preserved. Unfortunately, based on Theorem 1 for odd degree nodes, we can split off the edges only until $d(x) = 3$. Therefore, node x with an odd degree cannot be removed with edge splitting-off operations. This is a barrier to the practical applicability of this powerful theoretical result, as network topologies often have nodes with odd degrees, too.

B. Degree and Local Connectivity Preserving Graph Sequence

We formally define DLCP graph sequences and prove that such a sequence always exists in 2-connected graphs. The DLCP graph sequence starts with a small graph and finishes with G, thus it is the result of listing graphs in reverse order after removing nodes one by one and splitting off all their adjacent edges. Due to the reverse order, the graph transformation now involves pinching edges and adding a node on them.

Definition 2: A graph sequence $G_1, G_2, \ldots, G_{l-1}, G_l$ is *degree and local connectivity preserving (DLCP)* if it satisfies the following properties:

1) **First graph** $G_1 = (V_1, E_1)$ has two or three nodes, i.e., $|V_1| = 2$ or $|V_1| = 3$,
2) **Subsequent graphs** $\forall i = 2, \ldots, l : G_i = (V_i, E_i)$ is constructed from $G_{i-1} = (V_{i-1}, E_{i-1})$:
   - **Add one or two nodes:** either $V_i = V_{i-1} \cup \{v_i\}$, where v_i denotes the new node, or $V_i = V_{i-1} \cup \{v_i, w_i\}$, where v_i and w_i denote the new nodes.
   - **Add common, and split-off or tear-off edges:** every edge $(u, v) \in E_{i-1}$ is either part of $(u, v) \in E_i$ (called *common edges*), or (u, v) is replaced by two edges which are incident with one of the new nodes, formally $(z, u) \in E_i$ and $(z', v) \in E_i$, where $z \in \{v_i, w_i\}$ and $z' \in \{v_i, w_i\}$. If $z = z'$ we call it a *split-off edge*; otherwise as a *tear-off edge*. The number of tear-off edges in G_{i-1} is denoted by χ_{i-1}.
   - **Add between edges:** if two new nodes v_i and w_i are added to G_i, then they are connected by a new *between edge* (v_i, w_i) as well. If $\chi_{i-1} \geq 3$, then there might be multiple new parallel (v_i, w_i) edges added to G_i, but at most $\chi_{i-1} - 1$.
   - **Preserve connectivity:** for any pair of nodes $s, t \in V_{i-1}$ the local connectivity is $r_{i-1}(s, t) \geq r_i(s, t)$.
3) **Last graph** $G_l = (V, E) = G$ is the network topology.

The graphs in Fig. 1 form a DLCP graph sequence. Note that if a single node v_i is added, it will have an even degree. If two nodes are added, they will be adjacent, and both have odd degrees. Furthermore, if v_i and w_i are adjacent odd nodes, the number of edges incident with either v_i or w_i (but not both) is always even[2].

Next, we show that a DLCP graph sequence always exists in 2-connected network topologies:

Theorem 2: For any 2-connected undirected graph G, a DLCP graph sequence $G_1, \ldots, G_l = G$ always exists.

Proof: We construct the graph sequence in reverse order $G_l, \ldots, G_1$ applying one of the following two operations in each iteration, see Fig. 2:

1) *Remove even degree node v_i of G_i:* we can apply Theorem 1 and split off every edge incident with v_i such that the local connectivity between every node pair $s, t \in V \setminus \{v_i\}$ does not decrease. Note that G is 2-connected; thus, there is no cut-edge in the graph.
2) *Remove two adjacent odd degree nodes v_i and w_i of G_i:* we can add a single edge (v_i, w_i) which obviously does not decrease the local connectivity between any node pair $s, t \in V \setminus \{v_i\}$. Now there are at least two edges between v_i and w_i, and both node v_i and w_i has even degree. Thus, we can apply Theorem 1 for each, and split-off every adjacent edge such that the local connectivity between every node pair $s, t \in V \setminus \{v_i, w_i\}$ does not decrease.

Note that, the adjacency requirement of odd degree nodes is not a serious restriction, as even degree nodes can be removed until two odd nodes become adjacent. ■

Note that, in Case 2) we have added an edge between the two odd nodes. After the graph decomposition, in our FRR framework we will use the obtained graph sequence in a reverse order for routing arborescence construction. Thus, adding an edge in the decomposition is equivalent of removing an edge during the construction. If the erased edge is part of a routing arborescence in G_{i-1}, then we will not be able to extend the arborescences to graph G_i. However, the argument in the proof of Theorem 2 holds even if we remove edge (v_i, w_i) from the graph G_i between the two adjacent odd nodes, instead of adding one extra edge. We refer to this case as *no extra tear-off edge* removal, which is related to the following definition.

Definition 3: Let $G = (V, E)$ be an undirected multigraph. An edge (u, v) is *local*, if erasing the edge from G the local connectivity between s and t remains the same for $\forall s, t \in V \setminus \{u, v\}$.

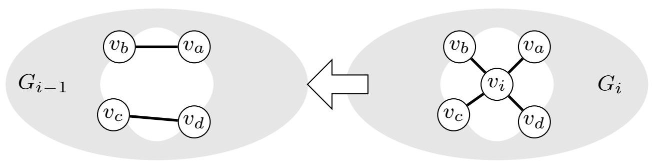
Roughly speaking a local edge is only important to achieve the connectivity between its end nodes, and it has no global effect. If we delete a local edge (u, v), then it has no effect on the local connectivity between any node pair that is disjoint from $\{u, v\}$. Based on this observation, we formulate two lemmas. Before that, we introduce a few important notions used in their proofs. We call a set of nodes X *tight* if there is a node pair $s \in X$ and $t \in V \setminus X$ such that $r(s, t) = d(X)$, where $d(X)$ denotes the number of edges incident with X, i.e., between X and $V \setminus X$. Similarly, we say that a subset $X \subseteq V$ is near-tight if $d(X) = r(s, t) + 1$, and *dangerous* if X is tight or near-tight. For a graph G and $X, Y \subseteq V$, $d(X, Y)$ denotes the number of edges between $X \setminus Y$ and $Y \setminus X$. If X is tight then $V \setminus X$ is also tight. We call a tight set X *non-trivial* if $|X| \geq 2$ and $|V \setminus X| \geq 2$, otherwise it is a *trivial cut.*

Lemma 1: Let $G = (V, E)$ be a 3-connected undirected graph with a node t, where every node has degree 3 in $V \setminus \{t\}$, and $|V| \geq 2$. There is a local edge e in G.

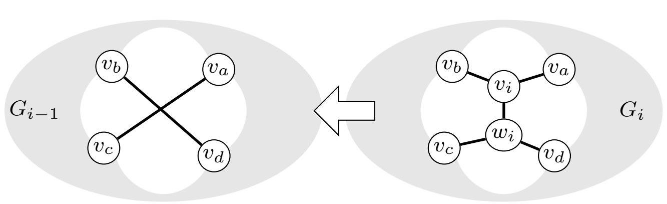
Proof: The poof is a direct consequence of Proposition 8.1.5 of [25]. For the sake of completeness, let us repeat the proof here.

Since every node $V \setminus \{t\}$ has nodal degree 3, and G is 3 connected, all the tight sets are 3-cuts. By Theorem 7.1.2 of [25], these 3-cuts are cross-free. If every 3-cut is a trivial cut, then e can be chosen arbitrarily. Suppose now that there is a non-trivial tight set $Z \subseteq V$ where $t \notin Z$ and assume that $|Z|$ is minimal. Here, non-trivial tight set means, $d(Z) = 3$, and $2 \leq |Z| \leq |V| - 2$. Since the 3-cuts are cross-free, every

[2]The sum of two odd degrees is even, and we need to subtract two times the number of edges between v_i and w_i, which is even again.



(a) The rule removing a single node v_i with even degree



(b) The rule removing two nodes v_i and w_i, both with odd degree

Fig. 2. Degree preserving edge splitting-off operations. The gray area contains the common edges, only the edges around the removed node(s) are changed.

3-cut containing an arbitrary edge induced by Z is a trivial cut. ■

Lemma 2: Let $G = (V, E)$ be a 2-connected undirected graph with a root node t, where every node has degree 3 in $V \setminus \{t\}$, and $|V| \geq 2$. There is a local edge e in G that is not adjacent with t.

Proof: By Theorem 7.1.8 of [25], every 2-cut of $G = (V, E)$ can be represented with a cactus graph C. In a cactus graph each edge belongs to exactly one circuit. Each node v of C corresponds to a 3-connected component of G, called the preimage of v.

Let v be a node of the cactus, which is either a leaf node or part of a single circuit, and its preimage does not contain t. Let $X \subseteq G$ be the preimage of v. X is a 2-cut of G, where (v_1, u_1) and (v_2, u_2) denote the edges of the 2-cut, with $v_1, v_2 \in X$ and $u_1, u_2 \notin X$.

Construct G', which has the nodes X and all the induced edges in G, with an additional edge (v_1, v_2). G' is 3-regular and 3-connected. By Lemma 1, G' has a local edge e that is not adjacent to v_1. Edge e is also local in G because it cannot be part of any 2-cut or 3-cut. ■

C. Heuristic Approach to Select Edge Pairs for Splitting-Off

Theorem 2 proves the existence of DLCP graph sequences. In our observation, the number of different DLCP graph sequences is huge, and in the following subsections, we discuss which sequence is the most suitable for our needs. First, we present the multiple ways the incident edges can be split off to remove node(s), and how these suitable pairs can be computed. Then, in the next section, we focus on the order of the nodes for removal.

Let B denote the edges incident with v_i (or w_i, but not both), which we call *border edges*. If we remove a single node, each perfect matching among border edges is called a *candidate* set of edges for splitting off, which is *valid* if it preserves the local connectivity. If there are no parallel edges in B, then the number of perfect matchings among them equals a double factorial, i.e., $\frac{(2b)!}{2^b b!}$, where $b = \frac{|B|}{2}$ (remember that $|B|$ is even). In typical network topologies, the nodal degree and thus b is small and the number of candidates is reasonable:

b	1	2	3	4	5
$\frac{(2b)!}{2^b b!}$	1	3	15	105	945

. (1)

When we remove two nodes, not all perfect matchings are candidates, only those where the number of tear-off edges is at most the number of parallel edges $(v_i, w_i) \in E_i$ (plus one if (v_i, w_i) is not local). If there are parallel edges among the border edges, the number of candidates will be even less. Overall, it is a reasonable assumption that we can list all the valid candidates and pick the most suitable one for our needs.

To compute the valid candidates we need to identify the dangerous sets. Note that, there is a polynomial time algorithm [23] to find the dangerous sets, and thus, the splittable edge pairs based on Gomory-Hu trees [26]. For example, if a node has 6 adjacent nodes, $v_1, \ldots, v_6$, then 3 new edges must be added, and according to the table in Eq (1) the number of such candidates is 15. Although intuitively there are not many splittable pairs that maintain the DLCP property, we demonstrate that surprisingly *a significant amount of edge pairs are valid.* For example, if a dangerous set separates them into two sets, say v_1, v_2, v_3 and v_4, v_5, v_6, then it is sufficient that the 3 new edges are between the two sides of this cut. In our example it would result in six valid candidates:

$$\begin{array}{ll}(v_1, v_4), (v_2, v_5), (v_3, v_6) & (v_1, v_4), (v_2, v_6), (v_3, v_5) \\ (v_1, v_5), (v_2, v_4), (v_3, v_6) & (v_1, v_5), (v_2, v_6), (v_3, v_4) \\ (v_1, v_6), (v_2, v_4), (v_3, v_5) & (v_1, v_6), (v_2, v_5), (v_3, v_4)\end{array}$$

Among these candidates we proceed with the one which achieves the highest of the following heuristic metrics that we obtained through hyperparameter optimization:

$$-10\alpha_i + 100\beta_i - \frac{\gamma_i}{c_{avg}},$$

where α_i denotes the number of edges that become parallel in step i, β_i the number of edges become loop, γ_i the total Euclidean distance of the new edges and c_{avg} is the average physical length of the links in the topology.

D. Constructing DLCP Graph Sequences

In Section III-C, we described how to remove any node with an even degree or two adjacent nodes with odd degrees. According to Theorem 1, we can select any single even-degree node for removal; however, for adjacent odd-degree nodes, we prefer to choose those ones that are connected by a local edge. Note that such a local edge will always exist in 3-regular graphs; see Lemmas 1 and 2. This still allows significant freedom in the order of node removal. In this subsection, we propose heuristic approaches to select the next node(s) to be removed in each iteration to generate the DLCP graph sequence $G_l, \ldots, G_1$. Our aim is that, in reverse order, this sequence will help us build the best routing tables towards a root node t. Therefore, our design goals are the following.

Goal 1: During the node removal process, we can remove adjacent odd-degree nodes only if they are connected by a local edge. Thus, we prefer to remove the even-degree nodes first, and then focus on the removal of the odd-degree nodes. If the only local edges left in the graph are those adjacent to the root t, we remove the local edges.

Goal 2: We want to focus first on nodes with high local connectivity to the root, because they must be connected to the root by more arborescences. Later, we extend some of these trees to nodes that have smaller local connectivity to the root. In reverse order, this means we first remove the nodes that are 2-connected to the root t, then those that are 3-connected, and so on.

Goal 3: We aim to mimic the process of growing the trees from the root t. In other words, if the area around t is already built, then we do not touch it anymore; instead, we add nodes only to its periphery. Therefore, in each step, we try to remove the node or node pair that is farthest from the root t.

Let $\delta(v)$ denote the hop distance of node v from root t. Let v_{max}^{even} be a node with an even degree that has the maximum hop distance from t. We define the hop distance of an adjacent node pair as the average hop distance of both terminal nodes. Let $(v_{max}^{odd}, w_{max}^{odd})$ be a pair of odd adjacent node pair with maximum average hop distance from t. The root is never removed, and the algorithm terminates when the graph has three (or two) nodes; see Fig. 1.

We propose the following approaches, ranging from the simplest to the one that incorporates all of our design goals:

Random: is our baseline approach, which selects the next node(s) for removal randomly. If the selected node has an odd degree, then a second node is chosen randomly among its neighbors with an odd degree if such exists; otherwise, another node is selected randomly.

Grow: focuses on Goal 3 and removes a single node $v_{\max}^{\text{even}}$ in the next step if

$$\delta(v_{\max}^{\text{even}}) \geq \min\{\delta(v_{\max}^{\text{odd}}), \delta(w_{\max}^{\text{odd}})\}, \tag{2}$$

otherwise, it removes the node pair $(v_{\max}^{\text{odd}}, w_{\max}^{\text{odd}})$.

Even-first: focuses on Goal 3 and partially on Goal 1 by selecting the even nodes first with descending hop distance to the root. When only odd nodes remain (apart from the root), it selects the odd node pairs with descending hop distance to the root. It only partially follows our first design goal as it allows the removal of adjacent odd nodes that are not connected by a local edge.

Advanced: covers all three design goals. It divides the nodes into sets based on local connectivity and, starting with $r(s,t) = 2$, it applies the even-first approach for the sets in increasing order. Furthermore, it does not remove odd pairs that are not connected by a local edge.

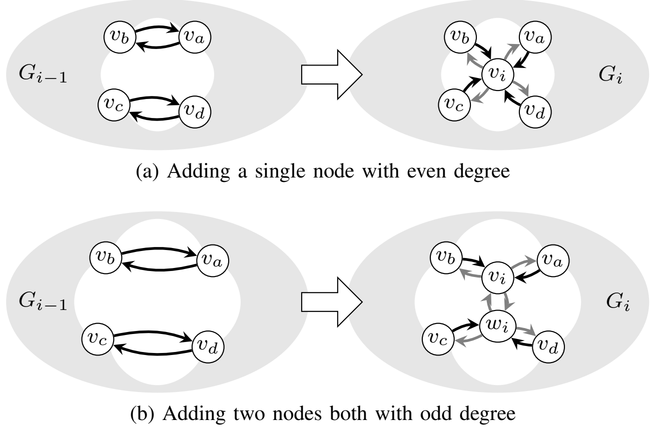

v_b
v_a
G_{i-1}
v_c
v_d
v_i
G_i
(a) Adding a single node with even degree
w_i
(b) Adding two nodes both with odd degree

Fig. 3. FRR arborescence construction in $G_1, \ldots, G_l$. Gray arcs are the new arcs, while the head of modified arcs are changed towards v_i (and w_i).

IV. Routing Arborescence Construction

In this section, we illustrate the applicability of our DLCP graph sequence in arborescence (i.e., routing table) construction for FRR. As FRR routing table computation can be performed independently for each root, *the task is to find a set of directed trees, called arborescences, such that each tree is directed towards a given root t*. Hence, we consider the edges of $G = (V, E)$ as *two directed arcs*, one in each direction. Arborescences in FRR usually span all nodes of the graph, but there might be partial directed trees, too [8]. In any case, $\forall v \neq t$ let $T_1, \ldots, T_{l_v}$ denote the arborescences in which v can reach the root. Let P_i denote the unique path from v to the root in tree T_i, for $i = 1, \ldots, l_v$. The task in FRR arborescence routing is to design a set of arborescences, such that for each node v the corresponding paths $P_1, \ldots, P_{l_v}$ are *pairwise arc-disjoint* (thus, the arborescences are arc-disjoint). Although it is easy to verify if a set of arborescences meet the arc-disjointness property, it is hard to design a generic algorithm that computes such arborescences. We will demonstrate that our DLCP graph sequence can simplify this process.

A. Arborescence Construction Algorithm

In the directed representation of the graph sequence $G_1, \ldots, G_l$ the inverse split-off or tear-off transformation can be handled by changing the head of some arcs towards the new node(s) v_i (and w_i), called *modified arcs*, and adding several *new arcs* $\mathcal{E}_i^{new}$ with their tail at the new node(s), see Fig. 3. Hence, we can define a *one-to-one mapping* of the arcs of G_{i-1} to the common and modified arcs of G_i.

As Algorithm 1 shows, the proposed arborescence construction heuristic prepares a DLCP graph sequence $G_1, \ldots G_l$ in Phase 1, and iterates through them in Phase 2. The first graph G_1 has 2 or 3 nodes, e.g., the root t and two other nodes in Fig. 4 (arborescences are denoted with a different color). The construction of the solution to such a small graph is simple. For example, for the 3-node network,

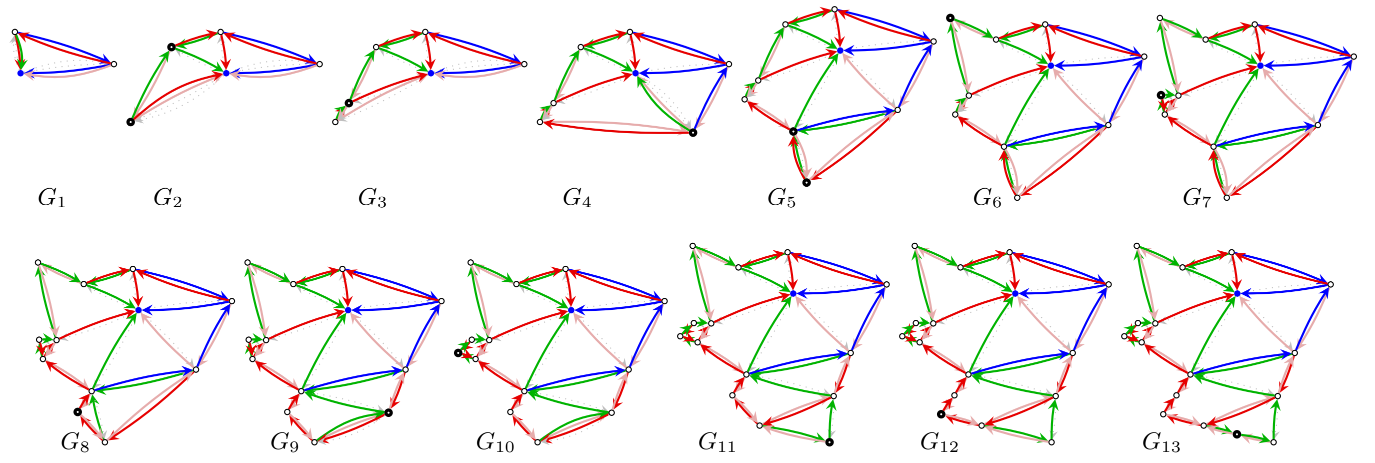


Fig. 4. Four arborescences are built up iteratively through graph sequence shown in Fig. 1. As the arborescences are arc-disjoint, a unique color is used to denote the arcs of each T_i, while non-arborescence arcs are colored black (dotted).

Algorithm 1: Routing Arborescence Construction

```
   Input: Graph $G$; root node $t$
   Output: Routing arborescences $\mathcal{T}_1, \mathcal{T}_2, \dots \mathcal{T}_{|\mathcal{C}|}$
   // Phase 1: Graph Decomposition
1  Create a DLCP graph sequence $G_1, \dots, G_l = G$
     according to Section III-D;
   // Phase 2: Arborescence
      Construction
2  Assign unique color to each in-arc of $t$ in $G_1$
3  If $|V_1| = 3$, then extend the trees for arcs between
     the two nodes $V_1 \setminus \{t\}$
4  for $i = 2, 3, \dots, l$ do
5      Transfer colors from $E_{i-1}$ to $E_i$
6      if A local edge $(t, v)$ added adjacent with $t$ then
7          Color arc $v \to t$ with a new color.
8      else if single node $v_i$ is added to $G_{i-1}$ then
9          Color arcs $\mathcal{E}_i^{new}$ by solving ILP of Sec. IV-B
10     else
11         Color arcs $\mathcal{E}_i^{new}$ by solving ILP of Sec. IV-C
   // Phase 3: Post-processing
12 for $i = 1, 2, \dots, |\mathcal{C}|$ do
13     $\forall \mathcal{T}_{j \neq i}$: erase all arcs or $\mathcal{T}_j$
14     Compute an SPT $\mathcal{T}'$ towards $t$ in the residual
         graph
15     $\mathcal{T}_i = \mathcal{T}'$
```

we color each in-arc of the root differently. In such a way, each arborescence is composed of one arc. Next, we assign colors to the arcs between v_1 and v_2 such that they extend the arborescences to have two arcs, respectively. Finally, we assign a unique color to each loop edge of the root (in both directions). In Step 4 we iterate through $G_2, \dots G_l$, and in the i^{th} step, we take the arborescences of G_{i-1} and map their common and modified arcs to G_i, i.e., use the same color for them. Finally, for the new arcs $\mathcal{E}_i^{new}$ of v_i (and w_i) we extend the previous arborescences if possible with the ILPs in Section IV-B and Section IV-C, resulting in spanning and partial arborescences in G_i. In this way, with the application of the DLCP graph sequence, we solved the complex FRR routing table computation problem by dealing with small local ILPs around the new nodes.

In order to further improve the quality of the routing arborescences, we have added a simple *post-processing* step to Algorithm 1 in Phase 3, where we re-optimize each arborescence T_j independently by erasing the arcs of all the other arborescences and computing the shortest path tree (SPT) towards t in the residual graph. With post-processing, we can reduce the *loss in coverage*, i.e., when adding node v_j to the graph in the i^{th} step the ILP is not able to connect it to $r(v_j, t)$ arborescences owing to the restriction of local modifications around v_j. Note that, the number of trees cannot be more than the local connectivity.

B. Integer Linear Program Formulation: Adding a Node

This section defines the ILP for adding a single node v_i. The task is to assign colors to the new arcs $\mathcal{E}_i^{new}$ which are the out-arcs of node v_i. The variables are:

$$x_a^c = \begin{cases} 1 & \text{if arc } a \text{ has color } c \\ 0 & \text{otherwise.} \end{cases} \qquad \forall c \in \mathcal{C}, \forall a \in \mathcal{E}_i^{new}$$

Here $\mathcal{C}$ is the set of colors (arborescences). We also have a variable for each color:

$$y^c = \begin{cases} 1 & \text{if } v_i \text{ is involved in arborescence } c \\ 0 & \text{otherwise.} \end{cases} \qquad \forall c \in \mathcal{C}$$

Constraint (3) says that each new arc has at most one color.

$$\sum_{c \in \mathcal{C}} x_a^c \leq 1 \qquad \forall a \in \mathcal{E}_i^{new}. \tag{3}$$

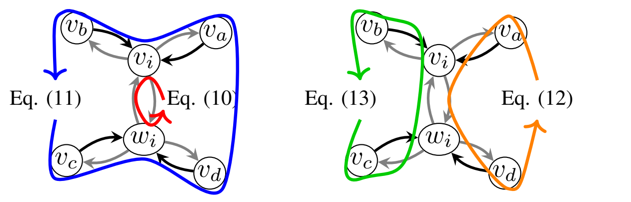


Fig. 5. The constraints of the ILP to avoid loops when two nodes are added.

Constraint (4) ensures that for each color c there is an out-arc from v_i only if node v_i is involved in arborescence c.

$$\sum_{a\in\mathcal{E}_i^{new}} x_a^c = y^c \qquad \forall c\in\mathcal{C}. \tag{4}$$

Next, we ensure there are no loops over the same edge. The colors of the in-arcs of v_i are inherited (known) from G_{i-1}:

$$x^c_{v_i\to v}=0 \quad \forall(v_i\to v)\in\mathcal{E}_i^{new},\ \text{if } v\to v_i \text{ has color } c. \tag{5}$$

Moreover, we ensure that there is no loop over multiple arcs either. In other words, if v is upstream of v_i in arborescence c then arc $v_i\to v$ should not be colored to c. Here upstream means there is a directed path in tree c from v to v_i, which we denote by $v\overset{c}{\rightsquigarrow}v_i$. Although the tree in color c is inherited from G_{i-1}, it has no out-arcs at node v_i; thus, it is currently not necessarily a valid rooted tree at t. Formally,

$$x^c_{v_i\to v}=0 \quad \forall(v_i\to v)\in\mathcal{E}_i^{new},\ \text{if } v\overset{c}{\rightsquigarrow}v_i\ . \tag{6}$$

We also need to ensure that every arborescence reaches the root; thus, for each color c, we need to avoid forwarding a packet to a node with no outgoing arc in color c. Formally,

$$x^c_{v_i\to v}=0 \quad \forall(v_i\to v)\in\mathcal{E}_i^{new},\forall c\in\mathcal{C},$$
$$\text{if } v\neq t \text{ and node } v \text{ has no out-arc in color } c. \tag{7}$$

Note that to guarantee that an arborescence is a t rooted tree, we need to ensure that every node involved in it (except t) has an out-arc and there are no loops.

Finally, the objective function is to maximize the weighted sum of the colors we can assign to the new node:

$$\max \quad \sum_{\forall c\in\mathcal{C}} \omega_c y^c\ . \tag{8}$$

We use weight $\omega_c=1$ for color c if node v_i has no in-arc of color c; otherwise, ω_c is the number of upstream nodes of v_i in color c plus 1. The intuition behind this is that the importance of having a color c path from node v_i is the number of nodes it will be used by, i.e., weighting defines that if we have a loss in coverage, then which is less painful.

C. Integer Linear Program Formulation: Adding Two Nodes

In this section, we formulate the ILP for adding two nodes v_i and w_i. We generate the constraints of the ILP for adding both nodes v_i and w_i just like adding them as single nodes described in the previous section, i.e., Constraints (3)–(7). The variables x_a^c corresponding to edges are the same for both nodes, while the distinct variables for color are denoted as y^c and $\hat{y}^c$ for v_i and w_i, respectively. We merge the two ILPs by summing up their objectives in Eq. (8), formally:

$$\max \quad \sum_{\forall c\in\mathcal{C}} (\omega_c y^c+\omega_c\hat{y}^c)\ . \tag{9}$$

Furthermore, to complete the ILP, we need to add some extra constraints to avoid loops that traverse both nodes v_i and w_i. When adding two nodes, Constraint (6) becomes weaker because of the upstream condition. Roughly speaking, the tree in color c may fall into more parts than when a single node is added because initially it has no out-arcs at two nodes v_i and w_i. In other words, node v_a may not be upstream to v_i in the trees inherited from G_{i-1}; however, once we add the out-arcs of w_i, it may become upstream, causing a loop. Fig. 5 illustrates all possible loops we must avoid. First, we ensure no loop in any tree along the arcs between the two new nodes. Note that the two new nodes might be connected with parallel edges. In this case, we need to add the following constraint for every combination of arc pairs with opposite directions.

$$x^c_{v_i\xrightarrow{m}w_i}+x^c_{w_i\xrightarrow{\hat{m}}v_i}\le 1 \quad \forall c\in\mathcal{C},$$
$$\forall m\in M_{v_i,w_i},\forall\hat{m}\in M_{w_i,v_i}\ . \tag{10}$$

Here, M_{v_i,w_i} denotes the set of parallel arcs from v_i to w_i, while m and $\hat{m}$ denotes a given instance. We add a similar constraint for larger loops that traverse both v_i and w_i:

$$x^c_{v_i\xrightarrow{m}v_b}+x^c_{w_i\xrightarrow{\hat{m}}v_d}\le 1 \quad \forall c\in\mathcal{C},\text{if } v_b\overset{c}{\rightsquigarrow}w_i \text{ and}$$
$$\text{if } v_d\overset{c}{\rightsquigarrow}v_i,\forall m\in M_{v_i,v_b},\forall\hat{m}\in M_{w_i,v_d}\ . \tag{11}$$

Constraint (12) is needed to avoid loops traversing an arc from v_i to w_i:

$$x^c_{v_i\xrightarrow{m}w_i}+x^c_{w_i\xrightarrow{\hat{m}}v_d}\le 1 \quad \forall c\in\mathcal{C},\text{if } v_d\overset{c}{\rightsquigarrow}v_i,$$
$$\forall m\in M_{v_i,w_i},\forall\hat{m}\in M_{w_i,v_d}\ . \tag{12}$$

We also add this in the opposite direction to avoid loops traversing an arc from w_i to v_i:

$$x^c_{w_i\xrightarrow{\hat{m}}v_i}+x^c_{v_i\xrightarrow{m}v_b}\le 1 \quad \forall c\in\mathcal{C},\text{if } v_b\overset{c}{\rightsquigarrow}w_i,$$
$$\forall\hat{m}\in M_{w_i,v_i},\forall m\in M_{v_i,v_b}\ . \tag{13}$$

We need to ensure that the arc between w_i and v_i does not take a color that has no arborescence from v_i:

$$x^c_{w_i\xrightarrow{m}v_i}\le y^c \quad \forall c\in\mathcal{C},\forall m\in M_{w_i,v_i}\ , \tag{14}$$

and the same constraint in the opposite direction is:

$$x^c_{v_i\xrightarrow{m}w_i}\le \hat{y}^c \quad \forall c\in\mathcal{C},\forall m\in M_{v_i,w_i}\ . \tag{15}$$

Finally, we need to ensure that if there is an arc in c from w_i to v such that v is upstream to v_i in c then we have a valid path in color c from node w_i only if v_i is involved in arborescence c. This can be formulated in both directions as:

$$y^c \geq x^c_{w_i \xrightarrow{\hat{m}} v} \quad \forall c \in \mathcal{C}, \text{if } v \overset{c}{\rightsquigarrow} v_i, \forall \hat{m} \in M_{w_i,v}, \tag{16}$$

$$\hat{y}^c \geq x^c_{v_i \xrightarrow{m} v} \quad \forall c \in \mathcal{C}, \text{if } v \overset{c}{\rightsquigarrow} w_i, \forall m \in M_{v_i,v} \ . \tag{17}$$

D. Path Length

To illustrate the flexibility of the proposed algorithmic framework, in this section we show how to extend the ILP with additional requirements, e.g., to decrease the lengths of the paths P_i. We extend the objective function to minimize the length of the paths in each tree as follows:

$$\max \quad \sum_{\forall c \in \mathcal{C}} \omega_c y^c - \sum_{\forall c \in \mathcal{C}} \sum_{\forall v_i \to v \in \mathcal{E}_i^{new}} \frac{h^c_{v_i \to v}}{100} \cdot x^c_{v_i \to v} \ , \tag{18}$$

where $h^c_{v_i \to v}$ is a constant and denotes the hop length of the path in arborescence c from node v, if such exists, otherwise, it is the maximal path length in G_{i-1}. We have divided the path lengths by 100 because *providing $r(v,t)-1$ resilience in worst-case is our primary objective* in the optimization, and we assumed that the maximum length is smaller than 100.

V. Scalability of the Proposed Algorithms

In this section, we briefly investigate the running time of the proposed algorithms. We assume the maximum degree in graph G, denoted Δ, is a constant value that does not depend on the number of nodes $n = |V|$. Note that the number of candidate edge sets in Eq. (1) has a bound of $(2\Delta)^\Delta$ because the number of border edges is at most $|\mathcal{E}_i^{new}| \leq 2 \cdot \Delta$. We call an algorithm *scalable* if its expected running time is a polynomial function of n for constant Δ.

We run Gomory-Hu algorithm to find the maximum flows between every pair of nodes, which has an expected running time $\tilde{O}(m \cdot r_{max})$ [27], where r_{max} is the maximum local connectivity, that is $\tilde{O}(n \cdot \Delta^2_{max})$ because $r_{max} \leq \Delta$ and $|E| = m \leq n \cdot \Delta$. Computing the hop distance of node v and root t can be done in linear time $O(n \cdot \Delta)$ with a Breadth First Search (BFS) algorithm. The number of graphs is $l \leq n$; thus the expected running time to construct DLCP graph sequence is at most $O(poly(n, \Delta) \cdot (2\Delta)^\Delta)$.

To compute the routing arborescences in Phase 2 of Algorithm 1 we need to solve l ILPs, where each ILP has $|\mathcal{C}| + |\mathcal{C}| \cdot |\mathcal{E}_i^{new}| \leq \Delta + 4\Delta^2 \leq 5\Delta^2$ binary variables. To formulate the ILP we need to deal with the paths in the arborescences from the border nodes in each color, that is $|\mathcal{C}| \cdot |\mathcal{E}_i^{new}| \leq 2\Delta^2$ paths in total. We need to perform membership queries for each of these paths (whether it traverses the modified arc or not). We can use Bloom filters [28] as a probabilistic data structure for constant time membership testing with the possibility of false positives. We can solve the ILP in $O(poly(n, \Delta) \cdot 2^{5\Delta^2})$ steps because every variable is binary.

Finally, in the post-process phase in each step we remove the edges of $|\mathcal{C}| - 1$ arborescences and calculate an SPT in the remaining graph with a BFS algorithm in $O(n \cdot \Delta)$ time, repeated for each $|\mathcal{C}|$ arborescences. Thus, Phase 3 requires $O(n \cdot \Delta^2)$ steps altogether, which means the overall complexity of Algorithm 1 is $O(poly(n, \Delta) \cdot 2^{5\Delta^2})$.

VI. Evaluation

In this section, we present numerical results that demonstrate the effectiveness of the proposed framework on real network topologies. We investigate the DLCP graph sequence generator heuristics (proposed in Section III-D) of Phase 1 in Section VI-A, while we focus on Phase 2 of Algorithm 1 for constructing arc-disjoint routing arborescences with the ILPs (introduced in Section IV) in Section VI-B. Finally, in Section VI-C we analyse the improvement of the post-process in Phase 3. In the evaluation, we have investigated 9 network topologies [29], [30], see the first three columns of Table II for the network names and the number of nodes and edges.

A. DLCP Graph Sequence Generator Heuristics

First, we focus on the heuristic approach in selecting the edge pairs for splitting-off. The results shown in the middle part of Table II were generated with the node selection approach Grow according to Eq. (2). The average number of removed nodes ($|V_i \setminus V_{i-1}|$) depends on the number of odd and even degree nodes in the input topology. These backbone network topologies are not very dense; thus, the number of border edges is 3.2 on average and, at most 10 (it is when removing two nodes). The average number of candidate split-off edge sets among these border edges is 2.2, and the maximum was 240. The average number of valid ones among these candidates is 1.5, and the maximum is 36.

The heuristic approaches select one according to some parameters such as the number of parallel edges (α), which was 0.6 on average and a maximum of 5, the number of loop edges (β) which was 0.1 on average and a maximum of 2, and the number of tear-off edges (χ) which was 0.6 on average and maximum 4. Loop edges are only needed to keep the nodal degree, but they have no role in the network connectivity. In other words, a loop edge means we can erase an edge from the network, which will not change the local connectivity. We also evaluated the number of valid candidates with at least a node pair with increased local connectivity $r(s,t)$ (better % in Table II), which only happened when we removed two nodes.

We also added the running time of the DLCP graph sequence generator in Phase 1 of Algorithm 1 to Table II, which was measured on a commodity laptop with a Core i5 CPU at 1.8 GHz with 4 GB of RAM running the Python code[3],

[3]Code and data available at https://github.com/jtapolcai/graph-sequences.

TABLE II
STATISTICS OF THE INPUT NETWORK TOPOLOGIES, SELECTING THE EDGE PAIRS FOR SPLITTING-OFF IN THE DLCP GRAPH SEQUENCE GENERATOR ALGORITHM, THE RUNTIMES, AND THE COVERAGE OF THE ROUTING ARBORESCENCES.

Network topologies			DLCP graph sequences															Arborescence coverage [%]				
			$\lvert V_i \setminus$	borders		candidates		valid		α		β		χ		better	runtime	Algorithm 1				Partial
name	$\lvert V\rvert$	$\lvert E\rvert$	$V_{i-1}\rvert$	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max	[%]	[sec]	random	grow	even-first	advanced	arb. [8]
German	17	26	1.2	3	6	1.4	8	1.2	4	0.6	2	0.2	1	0.2	1	1.8	0.074	99.4	99.9	**100**	**100**	**100**
ARPA	21	25	1.1	2.4	4	1.3	4	1.2	4	0.1	1	0.1	1	0.3	2	0	0.13	99.9	99.4	99.8	**100**	97.6
EU	22	45	1.2	4.4	10	2.9	21	2.1	14	1.2	4	0.1	2	0.7	4	13.4	0.19	99.4	99.7	99.5	**99.9**	98.3
USA	26	42	1.3	3.4	6	1.9	5	1.3	3	0.6	2	0.1	2	0.4	2	16.6	0.25	96.7	99.3	99.8	**100**	99.0
EU (Nobel)	28	41	1.4	3.2	6	2.2	8	1.5	4	0.4	2	0.1	1	0.6	2	5.2	0.23	92.0	99.5	98.6	**100**	99.9
Italy	33	56	1.2	3.5	10	2.8	240	1.5	36	1.4	5	0.1	2	0.9	4	0.7	0.48	98.6	99.3	99.9	**100**	99.2
EU (COST266)	37	57	1.4	3.4	6	2.5	18	1.6	10	0.4	2	0.1	2	0.7	2	7.1	0.71	98.5	98.5	98.6	**100**	97.0
N.-America	39	61	1.2	3.4	6	2.3	18	1.5	7	0.4	2	0.1	2	0.5	2	3.4	0.69	96.6	99.3	99.8	**100**	98.0
US (NFSNet)	79	108	1.3	2.9	6	2.2	18	1.6	10	0.2	2	0.1	1	0.6	3	5.4	5.03	96.6	96.4	98.5	**100**	92.6

where the graph algorithms were implemented in `networkx`. Note that, the runtime is dominated by Phase 1 and performs similarly for each heuristic presented in the paper. The total running time of the arborescence construction with the ILPs in Phase 2 and the post-process in Phase 3 of Algorithm 1 were below 0.5 sec in all cases, except the US network, where they took almost a second; however, it is still much less than the running time of the DLCP construction. The results back up the complexity analysis in Section V, demonstrating that the heuristics scale well with increasing network size.

Fig. 6 shows the number of nodes removed (one or two) in each step with the heuristic approaches described in Section III-D. We take the graph sequence $G_1, \dots, G_l = G$ and evaluate $|V_i \setminus V_{i-1}|$. As expected, even-first removes the single nodes first, then the odd node pairs, while others provide a relatively balanced way of removing single or two nodes.

B. Evaluating the Arborescence Construction Algorithm

We compared the performance of Algorithm 1 to the state-of-the-art partial arborescences used in Part 1 of the DAG-FRR heuristic [8] (called *Partial arb.* in the figures), which generates arc-disjoint routing arborescences with a greedy approach: starts with $d(t)$ arborescences on the in-arcs of t, and grows them greedily one after the other until possible. Our main performance metric for arborescences is coverage, which is 100% for a node $s \neq t$ if there are $r(s,t)$ arborescences, and thus there are $r(s,t)$ arc-disjoint paths from s to the root t which provide $r(s,t) - 1$ resilience in worst-case. Hence, 100% coverage for graph G is $\sum_{\forall s \in V \setminus \{t\}} r(s,t)$, while it is lower if some nodes have fewer trees available.

The last part of Table II shows the coverage of the arborescences constructed by our ILP formulations in Section IV on the obtained DLCP graph sequences. One can observe that there is a significant improvement for grow, even-first and advanced compared to the baseline approach, which randomly selects the next node(s) for removal. Furthermore, the **advanced method has near-optimal performance**. It removes the farthest even degree node with the smallest local connectivity and, in case of an odd node-pair, the one that produces the fewest tear-off edges. We believe this (reverse) order helps the ILPs in Phase 2 to build trees in dense subgraphs first; thus, no loss in coverage occurs due to previously lost trees. Such loss inevitably happens when a high degree node v is connected to low degree neighbors which already do not have $d(v)$ trees together.

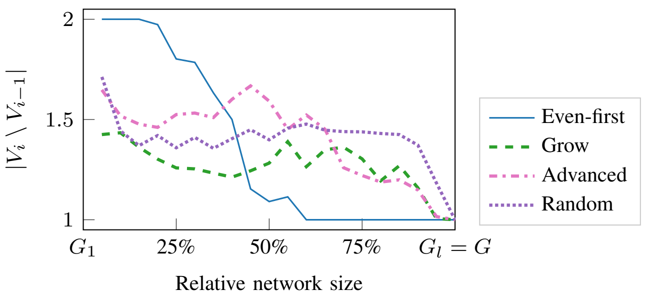


Fig. 6. The number of nodes removed in each step of the DLCP graph sequence $G_l, \dots, G_1$ (from right to left).

C. Post-Processing

The impact of loss in coverage can be reduced with the post-process approach in Phase 3. Fig. 7 shows the average coverage versus the path stretch in the arborescences for all networks. The *path stretch* is the hop length of each path divided by the minimum hop distance between the end nodes of that path. In the ILPs of Phase 2 we use Eq. (18) as a secondary objective to minimize path length. In Fig. 7 we present two points connected with a line: the left one is the result of Algorithm 1 (with post-processing), and the right one is without the post-processing in Phase 3. Post-processing can only increase the coverage; thus, it is always the point with better coverage among the two. Overall, as expected the advanced method outperforms all other approaches both in terms of coverage and path stretch.

VII. CONCLUSIONS

In this paper, we proposed an algorithmic framework to efficiently solve the routing problems corresponding to fast reroute. The framework is scalable and has great flexibility in defining multiple routing problems at the same time, as we strongly build on several optimization and graph-theoretical

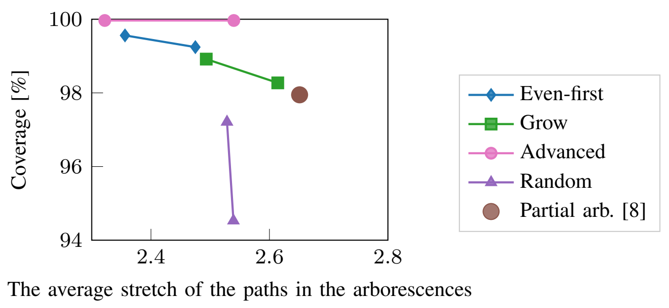


Fig. 7. Comparison of the FRR algorithms averaged for paths between 20 randomly selected root nodes and all sources in each 9 network topologies.

approaches, such as constructive graph characterization of k-connected graphs, dynamic programming, and integer linear programming. We showed how to compute arc-disjoint routing arborescences for FRR with the help of a degree and local-connectivity preserving graph sequence. In the simulations, we observed that our DLCP graph sequence generated with the Advanced heuristic outperformed the other FRR algorithms and provided 100% coverage in almost all topologies with low path stretch. We envision this flexible and efficient algorithmic framework as a first step that can pave the way to deal with more complex routing algorithms needed for an ideal FRR mechanism that allows local circular permutation of the arborescences, allows packet header rewrite, or even stores different routing table for each set of adjacent link failures.

References

[1] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. Chuah, Y. Ganjali, and C. Diot, "Characterization of failures in an operational IP backbone network," *IEEE/ACM Trans. Netw.*, vol. 16, no. 4, pp. 749–762, 2008.

[2] M. S. Javed, K. Thulasiraman, and G. Xue, "Logical topology design for ip-over-wdm networks: A hybrid approach for minimum protection capacity," in *2008 Proceedings of 17th International Conference on Computer Communications and Networks*, 2008, pp. 1–7.

[3] J. Rak and D. Hutchison, *Guide to disaster-resilient communication networks*. Springer Nature, 2020.

[4] B. Vass, J. Tapolcai, and E. R. Bérczi-Kovács, "Enumerating maximal shared risk link groups of circular disk failures hitting k nodes," *IEEE/ACM Transactions on Networking*, vol. 29, no. 4, pp. 1648–1661, 2021.

[5] "Shaping Europe's digital future, Actors in the broadband value chain," European Commission, Available: https://digital-strategy.ec.europa.eu/en/policies/broadband-actors-value-chain, 2019, Accessed: 2022-07-19.

[6] 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.

[7] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," in *Proceedings of the ACM SIGCOMM 2011 Conference*, 2011, pp. 350–361.

[8] K.-T. Foerster, A. Kamisiński, Y.-A. Pignolet, S. Schmid, and G. Tredan, "Grafting arborescences for extra resilience of fast rerouting schemes," in *IEEE INCOCOM*, 2021, pp. 1–10.

[9] 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.

[10] M. Chiesa, A. Gurtov, A. Madry, S. Mitrovic, I. Nikolaevskiy, M. Shapira, and S. Shenker, "On the resiliency of randomized routing against multiple edge failures," in *Int. Colloquium on Automata, Languages, and Programming (ICALP)*, 2016.

[11] 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.

[12] J. Feigenbaum, B. Godfrey, A. Panda, M. Schapira, S. Shenker, and A. Singla, "Brief announcement: On the resilience of routing tables," in *Proceedings of the 2012 ACM symposium on Principles of distributed computing*, 2012, pp. 237–238.

[13] 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)*, January 2021.

[14] S. Verbrugge, D. Colle, P. Demeester, R. Huelsermann, and M. Jaeger, "General availability model for multilayer transport networks," in *Proc. Int. Workshop on Design of Reliable Communication Networks (DRCN)*. IEEE, 2005, pp. 1–8.

[15] L. Lovász, "On the ratio of optimal integral and fractional covers," *Discrete Mathematics*, vol. 13, no. 4, pp. 383 – 390, 1975.

[16] W. Mader, "A reduction method for edge-connectivity in graphs," in *Annals of Discrete Mathematics*. Elsevier, 1978, vol. 3, pp. 145–164.

[17] A. Frank and T. Jordán, "Graph connectivity augmentation," *Handbook of Graph Theory, Combinatorial Optimization, and Algorithms*, pp. 313–346, 2015.

[18] Y. H. Chan, W. S. Fung, L. C. Lau, and C. K. Yung, "Degree bounded network design with metric costs," *SIAM Journal on Computing*, vol. 40, no. 4, pp. 953–980, 2011.

[19] T. Jordán, "On minimally k-edge-connected graphs and shortest k-edge-connected steiner networks," *Discrete applied mathematics*, vol. 131, no. 2, pp. 421–432, 2003.

[20] 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.

[21] A. Bhalgat, R. Hariharan, T. Kavitha, and D. Panigrahi, "Fast edge splitting and Edmonds' arborescence construction for unweighted graphs," in *Proc. ACM-SIAM symposium on Discrete algorithms*, 2008, pp. 455–464.

[22] H. N. Gabow, "Efficient splitting off algorithms for graphs," in *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing (STOC)*, 1994, pp. 696–705.

[23] L. C. Lau and C. K. Yung, "Efficient edge splitting-off algorithms maintaining all-pairs edge-connectivities," in *Integer Programming and Combinatorial Optimization*, F. Eisenbrand and F. B. Shepherd, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 96–109.

[24] L. Lovász, *Combinatorial problems and exercises*. American Mathematical Soc., 1993, vol. 361.

[25] A. Frank, *Connections in combinatorial optimization*. Oxford University Press Oxford, 2011, vol. 38.

[26] 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.

[27] R. Hariharan, T. Kavitha, D. Panigrahi, and A. Bhalgat, "An o (mn) gomory-hu tree construction algorithm for unweighted graphs," in *Proc. ACM symposium on Theory of computing*, 2007, pp. 605–614.

[28] J. Tapolcai, J. Biro, P. Babarczi, A. Gulyás, Z. Heszberger, and D. Trossen, "Optimal false-positive-free bloom filter design for scalable multicast forwarding," *IEEE/ACM Transactions on Networking*, vol. 23, no. 6, pp. 1832–1845, Dec 2015.

[29] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design Library," in *Proc. INOC*, 2007.

[30] 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.

Péter Babarczi received the M.Sc. and Ph.D. (*summa cum laude*) degrees in computer science from the Budapest University of Technology and Economics (BME), Hungary, in 2008 and 2012, respectively. From 2017 to 2019, he was an Alexander von Humboldt Post-Doctoral Research Fellow with the Chair of Communication Networks at the Technical University of Munich, Germany. He is currently working as an Associate Professor with the Department of Telecommunications and Artificial Intelligence at BME. He received the János Bolyai Research Scholarship of the Hungarian Academy of Sciences in 2013, and from 2020 to 2024 he was the lead researcher of an OTKA FK Young Researchers' Excellence Programme supported by the National Research, Development and Innovation Fund of Hungary. His current research interests include self-adapting networks, multi-path routing, cloud gaming, network coding, and combinatorial optimization in softwarized networks.

Balázs Brányi received the M.Sc. degree in applied mathematics from the Budapest University of Technology and Economics in 2022. He has worked on multiple papers with the High-Speed Network Laboratories. His research interests are network optimizations and network simulations.

Pin-Han Ho is currently a Full Professor in the Department of Electrical and Computer Engineering, University of Waterloo. He is the author/coauthor of over 400 refereed technical papers, several book chapters, and the coauthor of two books on Internet and optical network survivability. His current research interests cover a wide range of topics in broadband wired and wireless communication networks, including wireless transmission techniques, mobile system design and optimization, and network dimensioning and resource allocation.

János Tapolcai received the M.Sc. degree in technical informatics and the Ph.D. degree in computer science from the Budapest University of Technology and Economics (BME), Budapest, in 2000 and 2005, respectively, and the D.Sc. degree in engineering science from the Hungarian Academy of Sciences (MTA) in 2013. He is currently a Full Professor with Department of Telecommunications and Artificial Intelligence, BME. He is a winner of the MTA Lendület Program and the Google Faculty Award in 2012, Microsoft Azure Research Award in 2018. He is a TPC member of leading conferences, e.g. IEEE INFOCOM 2012-, and the general chair of ACM SIGCOMM 2018.

Lajos Rónyai is a research professor with the Artificial Intelligence Laboratory of the HUN-REN Institute of Computer Science and Control, Budapest, Hungary. He leads a research group there which focuses on theoretical computer science and discrete mathematics. He is also a full professor at the Mathematics Institute of the Budapest University of Technology and Economics. He received his PhD in 1987 from the Eötvös Loránd University Budapest. His research interests include efficient algorithms, complexity of computation, algebra, and discrete mathematics. He is a member of the Hungarian Academy of Sciences and a recipient of the Count Széchenyi Prize.