Resilient Routing Table Computation Based on Connectivity Preserving Graph Sequences

János Tapolcai*, Péter Babarczi*, Pin-Han Ho†, and Lajos Rónyai‡
*Department of Telecommunications and Media Informatics, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, Műegyetem rkp. 3., H-1111 Budapest, Hungary
†Deptartment of Electrical and Computer Engineering, University of Waterloo, Canada
‡Institute for Computer Science and Control, Eötvös Loránd Research Network; and Department of Algebra, Institute of Mathematics, Budapest University of Technology and Economics, Műegyetem rkp. 3., H-1111 Budapest, Hungary
E-mail: {tapolcai, babarczi}@tmit.bme.hu, p4ho@uwaterloo.ca, ronyai@sztaki.hu

***Abstract*—Fast reroute (FRR) mechanisms that can instantly handle network failures in the data plane are gaining attention in packet-switched networks. In FRR no notification messages are required as the nodes adjacent to the failure are prepared with a routing table such that the packets are re-routed only based on local information. However, designing the routing algorithm for FRR is challenging because the number of possible sets of failed network links and nodes can be extremely high, while the algorithm should keep track of which nodes are aware of the failure. In this paper, we propose a generic algorithmic framework that combines the benefits of Integer Linear Programming (ILP) and an effective approach from graph theory related to constructive graph characterization of k-connected graphs, i.e., edge splitting-off. We illustrate these benefits through arborescence design for FRR and show that (i) due to the ILP we have great flexibility in defining the routing problem, while (ii) the problem can still be solved very fast. We demonstrate through simulations that our framework outperforms 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**

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 multi-layer 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 pre-computed 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 arborescence-based methods work 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 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 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]. In the absence of a generic algorithmic framework that can handle complex cases with a massive number of simultaneous failures, 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. The high-level idea of

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

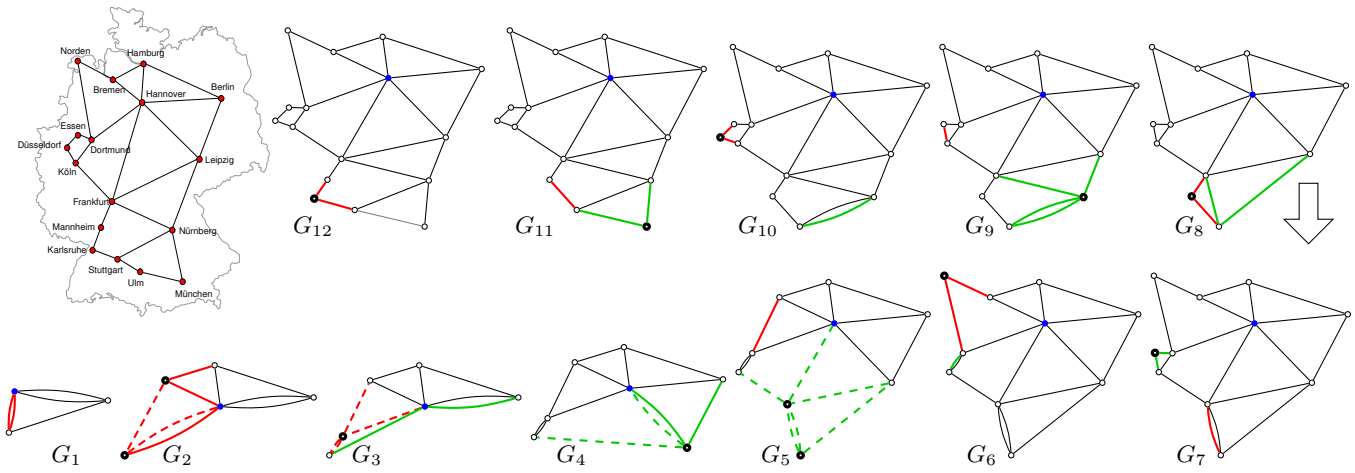


Fig. 1. Degree and local connectivity preserving graph sequence for the 17-node German backbone network [14]. The blue node is the root, the black larger nodes are the new nodes. We draw the edges involved in the transformation between G_{i-1} and G_i either with the same color or the same line style, 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.

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].

Our proposed framework handles the complexity of FRR with simultaneous failures and consists of 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 II-C.

(ii) *Arborescence construction for FRR:* Use graph sequence $G_1, G_2, \ldots, G_{l-1}, G_l = G$ to iteratively build 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. In Fig. 1 the farthest node(s) from root t were removed in each step of the decomposition; thus, we expect that the reverse graph sequence will mimic that the arborescences are "growing" from the root.

The rest of the paper is organized as follows. Section II 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 applied for routing table design for FRR in Section III, where our general ILP formulation for growing arborescences is presented. Section IV provides the runtime analysis of the framework. Finally, Section V contains our simulations results while Section VI concludes our work.

II. Constructing 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 II-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 II-B. In Section II-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 II-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.

Definition 1: The *edge splitting-off* operation in undirected graphs is that incident edges (x, u) and (x, v) are removed while edge (u, v) is added.

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, $\forall s, t \in V \setminus \{x\}$.

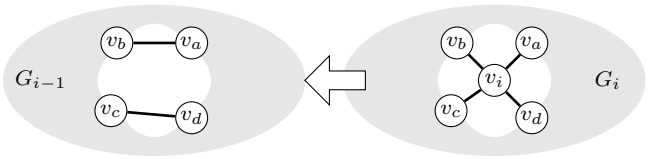
In our case, we would like to *remove a node*; thus, we split off all of its edges. As a resilient topology G is at least 2-connected, there are no cutting edges in the graph; thus, if the node has an even degree, then by the above theorem, this can always be done[2]. For example, in Fig. 2a a possible 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

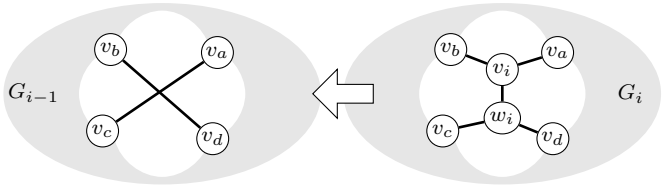
We formally define DLCP graph sequences and prove that such a sequence always exists in 2-connected graphs.

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, split-off and 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 *split-off edge*; otherwise, it is 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 $\chi_{i-1} - 1$ (see proof of Theorem 2).
   - **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.



(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.

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[3].

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 cutting 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,

[2]Frank proved in [25] a slight strengthening of Theorem 1, that the incident edges of node x with $d(x) \neq 3$ can be partitioned into $\lfloor d(x)/2 \rfloor$ disjoint splittable pairs. As a direct corollary, by splitting-off these edge pairs of an even degree node x we get a graph $G' = (V', E')$ where $V' = V \setminus \{x\}$, $E' = E \setminus \{(x, u_1), (x, v_1), \ldots, (x, u_{\lfloor d(x)/2 \rfloor}), (x, v_{\lfloor d(x)/2 \rfloor})\} \cup \{(u_1, v_1), \ldots (u_{\lfloor d(x)/2 \rfloor}, v_{\lfloor d(x)/2 \rfloor})\}$ with $\forall s, t \in V' : r'(s, t) \geq r(s, t)$ and $\forall v \in V' : d'(v) = d(v)$, i.e., edge splitting-off is a degree-preserving and local edge-connectivity preserving operation.

[3]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.

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.

In Case 2) there will be χ_{i-1} tear-off edges in G_i. Each tear-off edge is the result of two splitting-off operations: first, splitting-off an edge incident to v_i and an edge (v_i, w_i), and second, splitting-off this edge and an edge incident to w_i, e.g., in Fig. 2b (v_a, v_c) is a tear-off edge, because it is the result of splitting off edges (v_i, v_a) and (w_i, v_c) in G_i. Hence, $\chi_{i-1} - 1$ is at most the number of parallel edges $(v_i, w_i) \in E_i$ of G_i.

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. ∎

Theorem 2 states that a DLCP graph sequence always exists; in the next two subsections, we investigate how to find one. Our first observation is that **a huge number of different DLCP graph sequences exist**, and we can find a suitable one for our needs efficiently. Note that the proof of the above theorem showed that any even degree node could be removed, and also any two adjacent odd degree nodes can be removed as well. Thus, we have great freedom in selecting the order of nodes during the removal, see Section II-D. As there are multiple ways the incident edges can be split off to remove node(s), we first discuss how suitable pairs can be found in Section II-C.

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

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 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 the candidates and evaluate their validity. Once we have valid candidates, we can pick the most suitable one for our needs.

Note that there is a polynomial time algorithm [23] to find 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 Table (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 minimum cut 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:

$$(v_1, v_4), (v_2, v_5), (v_3, v_6) \quad (v_1, v_4), (v_2, v_6), (v_3, v_5)$$
$$(v_1, v_5), (v_2, v_4), (v_3, v_6) \quad (v_1, v_5), (v_2, v_6), (v_3, v_4)$$
$$(v_1, v_6), (v_2, v_4), (v_3, v_5) \quad (v_1, v_6), (v_2, v_5), (v_3, v_4)$$

In our implementation, we generate all the candidate splittable edge pairs and filter out the invalid ones, i.e., those that do not preserve the local connectivity. For each valid candidate set of edges for splitting off, denoted by $E_{i-1}^{spl} = E_{i-1} \setminus E_i$, we evaluate the following metrics:

- $\alpha(E_{i-1}^{spl})$ The number of edges that would become parallel in the following graph, i.e., among $E_i \cup E_{i-1}^{spl}$.
- $\beta(E_{i-1}^{spl})$ The number of loop edges among E_{i-1}^{spl}.
- $\gamma(E_{i-1}^{spl})$ The total Euclidean distance of the new edges. Here we assume the node coordinates are given.
- $\chi(E_{i-1}^{spl})$ The number of tear-off edges. If a single node is added, it is 0.

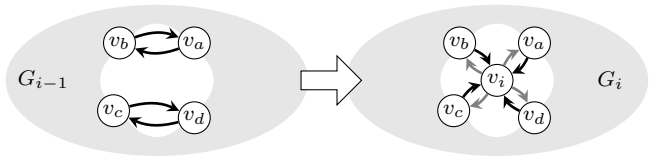
Finally, select the E_{i-1}^{spl} valid candidate set for the splitting off whose weighted sum of the above metric is maximal. The weights depend on heuristic design principles such as avoiding parallel edges [18] or introducing loop edges, discussed in Section V.

D. Constructing DLCP Graph Sequences

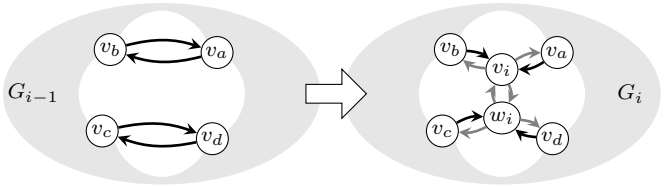
In Section II-C we described how to remove any node with an even degree or two adjacent nodes with odd degrees. According to Theorem 2, we can select any single even degree node or any adjacent odd node pairs for removal. Here we propose heuristic approaches to select the node(s) to be removed in each iteration to generate the DLCP graph sequence $G_l, \ldots, G_1$, which in the reverse order will help us to build up the best routing tables towards a root node t. We prefer that if the closed area around t is already built, then we do not touch it anymore, but instead, we add nodes only to its periphery. In other words, 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. We propose the following heuristic approaches:

- *Grow* removes single node v_{max}^{even} in the next step if
$$\delta(v_{max}^{even}) \geq \min\{\delta(v_{max}^{odd}), \delta(w_{max}^{odd})\}, \tag{2}$$
otherwise it removes the node pair $(v_{max}^{odd}, w_{max}^{odd})$. The root is never removed, and the algorithm terminates when the graph has three (or two) nodes; see Fig. 1.
- *Even-first* selects the even nodes first with descending hop distance to the root, and when there are only odd nodes remaining (apart from the root), it selects the odd node pairs with descending hop distance to the root.



(a) Adding a single node with even degree



(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 the modified arcs are changed towards v_i (and w_i).

- *Advanced* divides the nodes into sets based on local connectivity and starting with $r(s,t) = 2$ it applies even-first for the sets in increasing order. However, it does not remove odd pairs, which would produce the maximum number of tear-off edges[4] until another choice exists.
- As a baseline, we evaluate a so-called *random* approach, which selects a random node for removal. 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.

III. 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-Based Fast Reroute

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]. Therefore, 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. The static routing 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.

Note that 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 the network's global connectivity in these subgraphs, in [8], several novel fast rerouting algorithms were proposed which extend and combine multiple arborescences to overcome the limitations of spanning trees. The best algorithm is called *DAG-FRR*, composed of two steps called Part 1 and 2. In Part 1 as many rooted (partial) arborescences are built greedily as the root's nodal degree. In Part 2, as many unused edges are added to these trees as possible to form DAGs. We will use the partial arborescences in Part 1 of DAG-FRR as a benchmark in Section V.

Algorithm 1: Iterative Arborescence Construction

Input: DLCP graph sequence $G_1, \ldots, G_l = G$, a root t
Output: Routing arborescences $\mathcal{T}_1, \mathcal{T}_2, \ldots \mathcal{T}_{|\mathcal{C}|}$ for graph G

```
Assign unique color to each in-arc of t in G_1
If |V_1| = 3, then extend the trees for arcs between the two
  nodes V_1 \ {t}
for i = 2, 3, ..., l do
    Transfer colors from E_{i-1} to E_i
    if single node v_i is added to G_{i-1} then
        Color arcs E_i^new by solving ILP of Section III-C
    else
        Color arcs E_i^new by solving ILP of Section III-D
```

B. 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* $\mathcal{E}_i^{mdf}$, 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 iterates through a DLCP graph sequence $G_1, \ldots G_l$. The first graph G_1 has 2 or 3 nodes, e.g., the root t and nodes v_1 and v_2 in Fig 4 (arborescences are denoted with

[4]It is removed only if $\chi_{i-1} \leq$ the number of parallel edges $(v_i, w_i) \in E_i$.

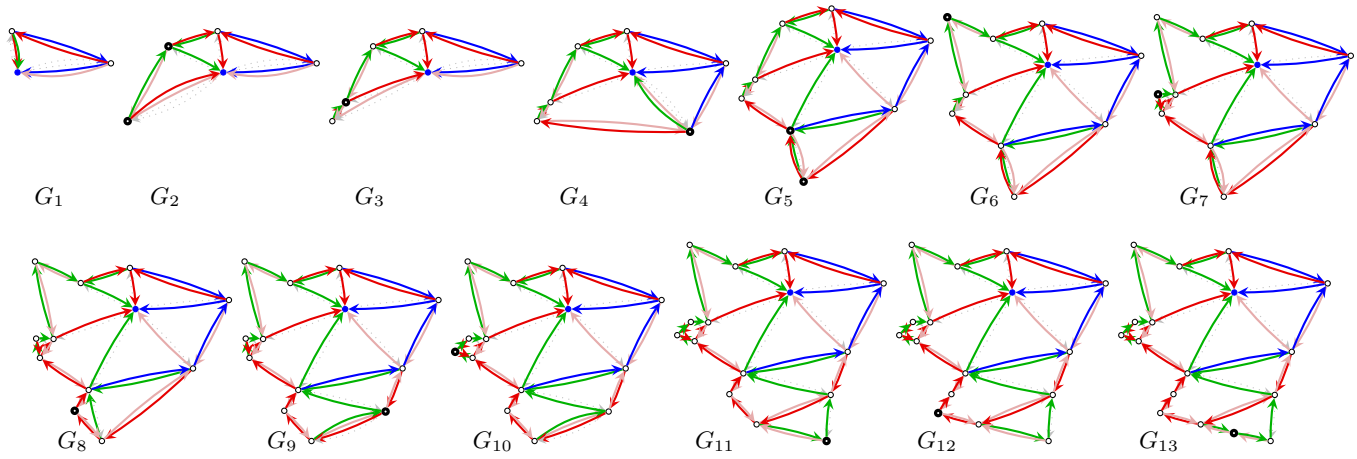


Fig. 4. Four arborescences are built up iteratively through graph sequence shown on Fig. 1. As the arborescences are arc-disjoint, a unique color is used to denote the arcs of each T_i, while non-arborescence arcs are colored black (dotted).

a different color). The construction of the solution to such a small graph is simple. For example, for the 3-node network, 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 3 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 edges 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 solve an ILP which extends the previous arborescences if possible, 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.

C. 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.

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

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:

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

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. The number of trees cannot be more than

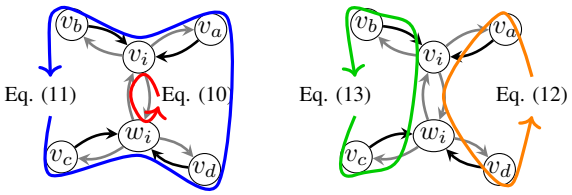


Fig. 5. The constraints of the ILP to avoid loops when two nodes are added.

the local connectivity. We say there was some *loss* in the i^{th} step if the ILP could not extend one (or multiple) trees for the new nodes. In this case, the weighting defines the preference among the trees. Roughly speaking, weighting defines that if we have a loss, then what is less painful.

D. 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:

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

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.

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

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:

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

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

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

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

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

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:

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

and the same constraint in the opposite direction is:

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

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:

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

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

E. 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:

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

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.

IV. 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 $\tilde{O}(n \cdot \Delta_{max}^2)$ 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

TABLE I

STATISTICS OF THE INPUT NETWORK TOPOLOGIES, SELECTING THE EDGE PAIRS FOR SPLITTING-OFF DESCRIBED IN SECTION II-C OF THE DLCP GRAPH SEQUENCE GENERATOR ALGORITHM, THE RUN TIMES, AND THE COVERAGE OF THE ROUTING ARBORESCENCES.

Network topologies			DLCP graph sequences														runtime		Arborescence coverage [%]							
																			IterativeILP (DLCP + Alg. 1)							Partial
			$\lvert V_i \setminus$	borders		candidates		valid		α		β		χ		better	DLCP	Alg. 1	grow		even-first		random		adv.	arb.
name	$\lvert V \rvert$	$\lvert E \rvert$	$V_{i-1}\rvert$	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max	[%]	[sec]	[sec]	post		post		post			[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	0.16	99.9	99.9	**100**	**100**	98.3	99.4	**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	0.16	99.1	99.4	99.7	99.8	99.9	99.9	**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	0.23	99.3	99.7	98.7	99.5	98.0	99.4	**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	0.22	96.6	99.3	99.7	99.8	90.8	96.7	**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	0.29	0.23	98.5	99.5	98.6	98.6	91.1	92.0	**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	0.26	99.3	99.3	99.8	99.9	97.3	98.6	**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	0.29	97.8	98.5	98.6	98.6	96.0	98.5	**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	0.30	98.8	99.3	99.7	99.8	92.7	96.6	**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	0.67	95.3	96.4	98.5	98.5	91.7	96.6	**100**	92.6

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 with Alg. 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 or these paths (whether it traverses the modified arc or not). We can use Bloom filters 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. Thus, the overall complexity for the DLCP graph sequence generation and Alg. 1 is $O(poly(n, \Delta) \cdot 2^{5\Delta^2})$.

V. EVALUATION

In this section, we present numerical results that demonstrate the use of the proposed framework on some real network topologies. We investigate the DLCP graph sequence generator algorithm (proposed in Section II) in Section V-A, and we focus on Algorithm 1 for constructing arc-disjoint routing arborescences (introduced in Section III) in Section V-B. In the evaluation, we have investigated 9 network topologies [28], [29], see the first three columns of Table I 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 described in Section II-C. The results are shown in the middle part of Table I, which 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 described in Section II-C selects 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" %), which only happened when we removed two nodes.

We also added the running time of the DLCP graph sequence generator algorithm to Table I, 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, where the graph algorithms were implemented in `networkx`. The Alg. 1 column corresponds to the total runtime of the arborescence construction. The results back up the runtime analysis in Section IV, demonstrating that the algorithms scale well with increasing network size. We refer to the DLCP graph sequence generation and Alg. 1 together as *IterativeILP*.

Finally, we investigated the four heuristic approaches described in Section II-D in selecting the next node or node pair for removal. Fig. 6 shows the number of nodes removed in each step. We take the graph sequence $G_1, \ldots, G_l = G$ 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

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.

In our evaluation the DLCP graph sequence generator selects the set of split-off edges E_{i-1}^{spl} based on the following fitness function:

$$-10 \cdot \alpha(E_{i-1}^{spl}) + 100 \cdot \beta(E_{i-1}^{spl}) - \frac{\gamma(E_{i-1}^{spl})}{c_{avg}} - 1000 \cdot \chi(E_{i-1}^{spl})$$

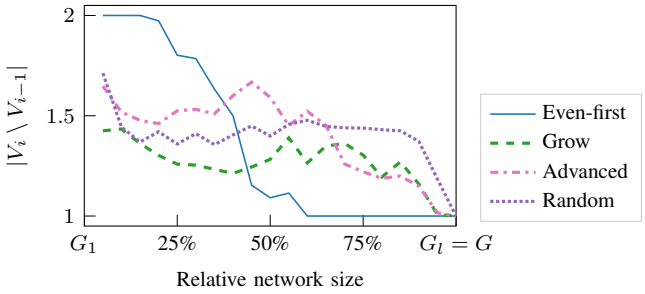


Fig. 6. The number of nodes removed in each step of the DLCP graph sequence $G_l, \ldots, G_1$ (from right to left).

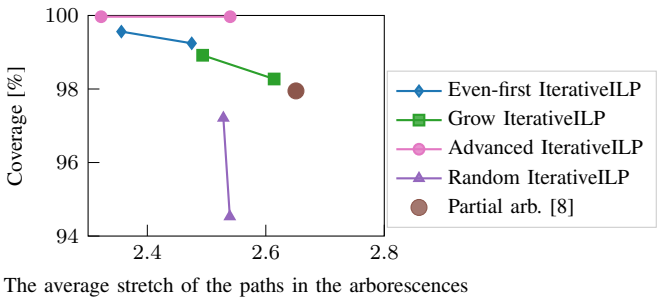


Fig. 7. Comparison of the FRR algorithms averaged for paths between 20 randomly selected root nodes and all sources in each 9 network topologies.

where c_{avg} is the average physical length of the links in the topology. However, we note that the approach is robust as we have observed similar results with most parameter settings.

We have evaluated how IterativeILP performs compared to the state-of-the-art partial arborescences used in Part 1 of the DAG-FRR heuristic [8] (called *Partial arb.* in the figures). Part 1 of DAG-FRR 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. For comparison, we have added a simple *post-processing* step to our IterativeILP, where we re-optimize each arborescence T_i independently by erasing the arcs of all the other arborescences and computing the shortest path tree towards t in the residual graph. With post-processing, we can reduce the "loss" in coverage (i.e., when adding node v_i to the graph, the ILP is restricted only to local modifications around v_i, and thus it is not able to connect it to $r(v_i, t)$ arborescences), discussed in the last paragraph of Section III-C.

The last part of Table I shows the obtained results for the proposed strategies of selecting the next node(s) for removal. 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 IterativeILP 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. For the other strategies, the loss effect of the IterativeILP can be reduced with the post-process approach, which is beneficial to increase coverage based on our observations. The runtime is dominated by the graph decomposition stage (DLCP) and performs similarly for each heuristic presented in the paper.

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. For IterativeILP minimizing path length as a secondary objective with Eq. (18), we present two points connected with a line: the left one is the result with, and the right one is without post-processing. Post-processing can only increase the coverage; thus, it is always the point with better coverage among the two. Overall, the advanced IterativeILP outperforms all other approaches in terms of coverage and path stretch.

VI. Conclusions

In this paper, we proposed a novel 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 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.

The authors have provided public access to their code and data at https://github.com/jtapolcai/graph-sequences.

Acknowledgements

This work was partly supported by Project no. 134604 and Project no. 128062, which have been implemented with support from the National Research, Development and Innovation Fund of Hungary, financed under the FK_20 and K_18 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.

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. [Online]. Available: http://eprints.cs.univie.ac.at/6529/

[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 *Proceedings of the nineteenth annual 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, p. 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, "On a theorem of Mader," *Discrete Mathematics*, vol. 101, no. 1, pp. 49–57, 1992.

[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 *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, 2007, pp. 605–614.

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

[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.