Scalable and Efficient Multipath Routing via Redundant Trees

János Tapolcai, Gábor Rétvári, Péter Babarczi, Erika R. Bérczi-Kovács

Abstract—**Nowadays, a majority of the Internet Service Providers are either piloting or migrating to Software-Defined Networking (SDN) in their networks. In an SDN architecture a central network controller has a top-down view of the network and can directly configure each of their physical switches. It opens up several fundamental unsolved challenges, such as deploying efficient multipath routing that can provide disjoint end-to-end paths, each one satisfying specific operational goals (e.g., shortest possible), without overwhelming the data plane with a prohibitive amount of forwarding state. In this paper, we study the problem of finding a pair of shortest (node- or edge-) disjoint paths that can be represented by only two forwarding table entries per destination. Building on prior work on minimum length redundant trees, we show that the complexity of the underlying mathematical problem is NP-complete and we present fast heuristic algorithms. By extensive simulations we find that it is possible to very closely attain the absolute optimal path length with our algorithms (the gap is just 1–5%), eventually opening the door for wide-scale multipath routing deployments. Finally, we show that even if a primary tree is already given it remains NP-complete to find a minimum length secondary tree concerning this primary tree.**

Index Terms—**redundant trees, independent spanning trees, not-all-equal 3SAT, minimum length disjoint paths**

I. Introduction

In traditional hop-by-hop routing, packets are forwarded along a single path, such that each router associates a default next hop with each destination address in its forwarding table. In multipath routing, however, routers maintain multiple next hops for each destination, each one corresponding to a different path towards the destination, and packets are mapped to one of these paths using header hashing, packet tagging, etc. There are many practical motivations towards multipath routing, such as to improve end-to-end reliability, security, and latency, allow users to avoid congested links, and provide some control to applications to meet their performance requirements [1]–[5]. The most common implementation is equal-cost multi-path routing (ECMP) where multipath routing is performed among some specific node pairs to improve load balancing. In this paper our goal is to make a step forward and *enable multipath routing among every node pair*.

We argue that the major ingredients of a multipath routing are, by and large, in place, like a flexible data plane (OpenFlow, SDN, network function virtualization) [6], [7], multipath rate-control protocols (MPTCP) [4], sufficient path-diversity [1], [8], with some even having reached large-scale test phase [9], [10]. One important blocker is the scalability barrier; provisioning multiple custom end-to-end paths would cause forwarding state to grow quadratically with the number of endpoints [11], while the data plane is already struggling to scale with a much slower growth rate in the first place [12], [13].

A recent study by Verizon [14] estimated that 57% of the enterprises will deploy SDN in their networks within two years. In contrast to traditional IP where the user cannot influence the routing of the packets, SDN makes multi-path routing mechanisms easily deployable. The SDN controller has an exact view of the network topology and for each physical switch it computes two next-hops towards each destination: one is called the red next-hop, the other is the blue next-hop. Fig. 1d shows an example of red and blue next-hops at each switch towards destination node r. The hosts (or the egress switches) include the destination address and set a single bit in the header to mark whether the packet should be forwarded along the red or the blue next-hops. Packets then travel hop-by-hop to the destination along one of the two paths assigned by the single bit, according to the red and blue next-hops stored at the intermediate nodes. We will refer to them as red and blue paths, see Fig. 1c for the route of packets from v_5 towards r along the red and blue next-hops. Note that this scheme is fully compatible with all SDN standards (P4 [15], OpenFlow [6], PoF [16]); for instance, it would be easy to implement this behavior in a couple of lines of P4 [15]. Furthermore, hosts can use this scheme to adopt a multipath rate control algorithm to actively balance their load along their paths in an end-to-end fashion [4].

In this paper, we propose a *multipath routing* scheme with the following design objectives, based on the above SDN implementation model:

Scalable: the paths are such that nodes need at most two forwarding table entries per destination. Users will be able to select between these two (red and blue) next-hops by tagging

A preliminary version of this article appeared in the IEEE International Conference on Network Protocols (ICNP) '15, San Francisco, CA, November 2015. János Tapolcai, Gábor Rétvári, and Péter Babarczi are with the MTA-BME Future Internet Research Group and MTA-BME Information Systems Research Group, Dept. of Telecommunications and Media Informatics, Budapest University of Technology and Economics (BME), Hungary (e-mail: {tapolcai, retvari, babarczi}@tmit.bme.hu). Erika R. Bérczi-Kovács is with the Department of Operations Research, Eötvös University, Budapest, Hungary and with the MTA-ELTE Egerváry Research Group on Combinatorial Optimization (e-mail: koverika@cs.elte.hu).

The authors would like to thank Panna Kristóf for her constructive comments. The research leading to these results was partially supported by the High Speed Networks Laboratory (HSNLab). Project no. 123957, 129589, 124171, 128062 and 124171 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the FK_17, KH_18, K_17, K_18 and K_17 funding schemes respectively. The research report in this paper was also supported by the BME-Artificial Intelligence FIKP grant of EMMI (BME FIKP-MI/SC). The work of P. Babarczi was supported in part by the Post-Doctoral Research Fellowship of the Alexander von Humboldt Foundation.

their packets appropriately.
Disjoint: the red and blue paths from any source node towards the common node in the network are maximally disjoint (i.e., do not share common edges or nodes if possible [17]). Note that the red and blue next hops of Fig. 1d meet this objective for destination node r. This contributes to better availability and resilience against single failures [1], [3] and eliminates adverse interference between the subflows carried by those paths [4].
Fast: the algorithm for computing the next hops has the same computational complexity as traditional shortest path routing (i.e., Dijkstra's algorithm), in order to amortize the cost of multipath routing in comparison to standard control plane operations.
Short paths: the length of the paths are close to the absolute theoretical minimum. An adequately small average path length would improve forwarding delay and reduce the performance gap as compared to traditional single-path routing to a tolerable level [9], [10], [18].

This paper is dedicated to find algorithmic techniques for disjoint multipath routing. We demonstrate that the Suurballe-Tarjan algorithm [19] – which was originally proposed to find minimum length disjoint path-pairs – can be effectively used to provide efficient solutions with the above requirements. In particular, we concentrate on the following fundamental question:
What is the price for the simplest possible forwarding scheme implementable both in SDN and destination based hop-by-hop forwarding, in terms of (i) computational complexity and (ii) the gap between the length of the disjoint paths representable by two next hops per destination compared to the minimum for two disjoint paths?

This question essentially boils down to find a pair of rooted trees under the constraint that the paths in the trees must be disjoint. Such trees are called *redundant trees* (or colored trees or independent trees) in the literature and enjoy wide-scale use, ranging from reliable forwarding in wireless [20] and wired networks [21], [22], robust multicasting [23], general multipath routing [11], [24] and load-balancing [25], to Fast ReRoute (FRR) protection [17], [26], [27]. In contrast to these works, however, our main concern is the length of the paths within the redundant trees, as this is crucial for disjoint multipath routing.

Building on our own [28] and independent [11], [21], [24], [29], [30] prior work on this subject, in this paper *we carry out the first systematic study of the performance penalty related to scalable multipath routing*. In particular, we make the following main contributions.

- We settle the computational complexity of the mathematical problems related to minimum length redundant trees, including a problem variant where a primary (e.g., shortest path) tree is given.
- We classify the heuristic techniques to solve the problem, we point out the limitations of each, and we propose a new design concept yielding several new heuristics.
- We improve the best-known heuristic complexity from cubic to the same as that of Dijkstra's algorithm without major performance hit, and we exercise the time–efficiency trade-off to gain considerable performance improvements at the cost of a slight running time overhead.
- In numerical evaluations, we show that our algorithms find near-optimal solutions even for large networks that cannot be solved by integer linear programs [11], and they provide shorter paths [17] with less computation time [24] than their existing heuristic counterparts.

The rest of this paper is structured as follows. In Section II, we present some background on redundant trees and we pose the minimum length redundant tree problem. In Section III we show that the problem is NP-complete. In Section IV we present the algorithmic framework of redundant trees and discuss its relation to the related work. In Section V we present our new heuristics for the node-redundant tree problem for a root node with degree two. The necessary modifications to solve the general node- or edge-redundant case are summarized in Section VI along with some discussion on the problem when the primary tree is fixed. In Section VII we present an extensive numerical study, extending to hundreds of network topologies and edge length settings to evaluate the performance gap between the optimal and the obtained path lengths. Finally, in Section VIII we draw the conclusions.

II. Background and Problem Formulation

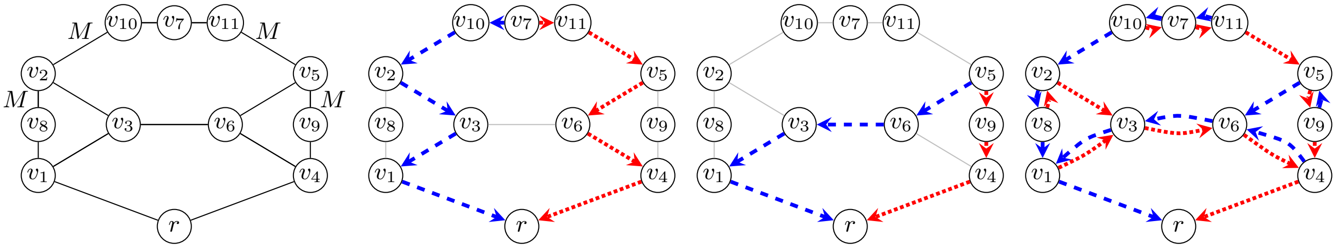
Suppose we are given a 2-connected[1], undirected graph $G = (V, E)$, where V denotes the set of nodes ($|V| = n$) and E denotes the set of edges ($|E| = m$), with an edge length function $l : E \to \mathbb{R}^+$ set according to some traffic engineering considerations. A *path* P in G is then an ordered set of nodes and edges $P = s\text{-}v_1\text{-}v_2\text{-}\dots\text{-}v_{k-1}\text{-}r$, where $(s, v_1), (v_1, v_2), \dots, (v_{k-1}, r) \in E$. Nodes s and r are called terminal nodes. For easier presentation we often assign a direction to the path $P = s \to v_1 \to \dots \to r$ and call s the source and r the destination node. We call two paths *(node-)disjoint* if they do not have any common nodes except the terminal nodes. Two paths are *edge-disjoint* when they have no common edges. This is a weaker property.

A. Minimum Length Disjoint Paths

Suppose we want to find two short disjoint paths between each pair of nodes. Easily, the problem can be decomposed into independent sub-problems for each destination node r as follows: given a *root node* r, find a pair of disjoint paths from each $s \neq r$ to r of minimum total length over all s.

Consider the example graph topology Fig. 1a, let r be the root and let M be some arbitrary positive length. Then, a pair of disjoint paths with minimum aggregate (total) length from node v_7 is given in Fig. 1b and from node v_5 in Fig. 1c. Such a pair of paths from each source to a given root can be computed by a single pass of the Suurballe-Tarjan algorithm, with two iterations of the Dijkstra shortest path algorithm (yielding a complexity of $O(n \log n + m)$ for all nodes to r) [19]. Hence, it seems this algorithm would then readily lend itself as a multipath routing algorithm.

[1]A graph is 2-connected (2-edge-connected) if the removal of any single node (edge) does not disconnect the graph, which is a necessary condition for the second (*Disjoint*) design objective.



(a) Topology graph G_1 (b) Optimal paths for v_7 (c) Optimal paths for v_5 (d) Shortest redundant trees

Fig. 1: An illustrative example, with root r and a large enough positive edge length M. All unmarked edges are of unit length.

Unfortunately, it does not. The reason is that this algorithm would not satisfy all the requirements for deployability set out above, as the resultant forwarding tables would scale superlinearly with the number of nodes. This is demonstrated in Fig. 1: as the red path starting at v_5 diverges from the red path starting at v_7, node v_5 would need to allocate a separate forwarding table entry corresponding to v_7 and for itself to correctly route to r. Swapping the red and the blue paths for, say, v_5, would not help either, as now a similar extra forwarding table entry would arise at node v_6. Unfortunately, there does not seem to be a simple way out of this trap [19].

Henceforth, we shall use the Suurballe-Tarjan algorithm to produce an optimal pair of *minimum length disjoint paths* (ones we could use if forwarding state were not of concern) and we shall compare our heuristic paths (now representable by just 2 forwarding table entries per destination) to these ideal paths. Notation-wise, given some root r let the optimal $v \to \ldots \to r$ paths (as computed by the Suurballe-Tarjan algorithm) be denoted by $P_1^*(G, v, r)$ and $P_2^*(G, v, r)$ for each $v \neq r$. We shall denote the length of this "ideal" path-pair as

$$L^2_{v,r}(G) = \sum_{e \in P_1^*(G,v,r)} l_e + \sum_{e \in P_2^*(G,v,r)} l_e \ .$$

Easily $L^2_{v,r}(G) \geq 2L^1_{v,r}(G)$, where $L^1_{v,r}(G)$ denotes the length of the shortest path $P^*(G, v, r)$ from v to r:

$$L^1_{v,r}(G) = \sum_{e \in P^*(G,v,r)} l_e \ .$$

B. Redundant Trees

An (undirected) rooted spanning tree $\mathcal{T}_r$ in graph $G = (V, E)$ for some root r is a tree rooted at r in which from any node $v \in V, v \neq r$ there is exactly one path from v to r. For easier presentation, often a direction is assigned to the edges of $\mathcal{T}_r$ and directed towards the root [11], [21], i.e., from any node $v \in V, v \neq r$ there is exactly one directed path from v to r. For a tree $\mathcal{T}_r$ rooted at r and any $v \neq r$, denote by $P(\mathcal{T}_r, v)$ the unique path in $\mathcal{T}_r$ from v to r. Then, redundant trees are a pair of spanning rooted trees with certain disjointness requirements on their paths towards the root [11].

Definition 1: A pair of (spanning) trees $\mathcal{T}_r^1, \mathcal{T}_r^2$ with common root r is called (a pair of) *node-redundant trees* for r if for each $v \in V$ paths $P(\mathcal{T}_r^1, v)$ and $P(\mathcal{T}_r^2, v)$ are node-disjoint.

We also define a weaker form as follows.

Definition 2: A pair of (spanning) trees $\mathcal{T}_r^1, \mathcal{T}_r^2$ with common root r is called (a pair of) *edge-redundant trees* for r if for each $v \in V$ paths $P(\mathcal{T}_r^1, v)$ and $P(\mathcal{T}_r^2, v)$ are edge-disjoint.

Consider the *red* tree $\mathcal{T}_r^1$ and the *blue* tree $\mathcal{T}_r^2$ in Fig. 1d with directions assigned to their edges. Even though the edge (v_3, v_6) is used in both trees, the paths themselves from each node to the root are edge-disjoint (node-disjoint) and hence $\mathcal{T}_r^1$ and $\mathcal{T}_r^2$ qualify as edge-redundant (node-redundant) trees.

The graph theoretical problem related to redundant trees was widely investigated in the last decades. For 2-edge-connected undirected graphs, a pair of edge-redundant trees for any root is guaranteed to exist, and it can be found in polynomial time [21], [23]. This was later reduced to linear time [31] and linear time algorithms for finding maximally edge-redundant trees were also given for other than 2-connected case [32].

C. Implementing Multipath SDN Packet Forwarding

We have seen that a trivial implementation of packet forwarding along minimum length disjoint paths might require a next hop for each source per destination. This would violate the stringent scalability requirements of multipath SDN forwarding: in SDN switches flow-table entries are a scarce resource due to the limited amount of TCAM/SRAM space programmable switch ASICs provide (RMT [33], dRMT [34], FlexPipe [35], Cavium [36], etc), and software-based SDN switches seem to have similar operational limitations (see a recent study in [37]). The routing algorithms are running on the SDN controller; if necessary, distributed versions can then be bolted on the centralized algorithms [11], [20], [26]. Below, we sketch a centralized scheme that requires only two next hops per destination, this way greatly enhancing the scalability of SDN multipath routing.

First, the SDN controller computes a pair of redundant trees concerning each destination node as a root r, and sets two forwarding table entries in each physical switch v corresponding to every pair of such trees. One entry gives the next-hop for destination r along the red tree $\mathcal{T}_r^1$ and the other along the blue tree $\mathcal{T}_r^2$ (for instance, node v_5 in Fig. 1d would set v_9 as the next-hop for destination r along $\mathcal{T}_r^1$ and v_6 as next-hop along $\mathcal{T}_r^2$). Packets then travel hop-by-hop to the destination along the tree assigned by the single bit, according to the next-hops stored in the SDN flow tables at the intermediate nodes. This scheme is easy to implement in a couple of lines of P4 [15]. As the forwarding paths are lined up into trees, at every node a "single" outgoing edge per tree is assigned as a next-hop for each destination, which was not the case with the minimum length disjoint paths. Next, for packet forwarding, either the hosts or the egress switch include the destination address and set a single bit in the header to mark whether the packet should be forwarded along $\mathcal{T}_r^1$ or $\mathcal{T}_r^2$. In the former, hosts can use this scheme to adopt a multipath rate control algorithm to actively balance their load along their paths in an end-to-end fashion [4].

The delivery of packets to their respective destination should be guaranteed even if the topology changes. Hence, we need to avoid generating incorrect network updates, caused by the asynchrony of the communication channels between the controller(s) and the switches. In our proposed multi-path routing scheme the packets are forwarded along two trees to every destination node. Therefore, the forwarding rules along each tree can be updated independently, where each tree is a spanning tree with a single destination node. Here we are facing a well-studied version of the so-called loop-free flow migration problem in SDN networks [38]. For example, the $O(n)$-round scheduler by [39] ensures strong loop-free network updates, when at any point in time, the forwarding rules stored at the switches should be loop-free. Another option is to use the deterministic update scheduling algorithm by [40], [41], which completes in $O(\log n)$-round in the worst case for relaxed loop-freedom. In this case a small number of "old packets" may temporarily be forwarded along loops.

D. Minimum Length Redundant Trees

The length of the (unique) path in a tree $\mathcal{T}_r$ from source node v towards tree root r is calculated as $L_{v,r}(\mathcal{T}_r) = \sum_{e \in P(\mathcal{T}_r, v, r)} l_e$, while the length of a tree can be obtained as $L_r(\mathcal{T}_r) = \sum_{v \neq r} L_{v,r}(\mathcal{T}_r)$. What we are concerned with in this paper is finding a pair of redundant trees $\mathcal{T}_r^1, \mathcal{T}_r^2$ of "minimum length" for a given root r. This metric implicitly corresponds to the case where all nodes are equally likely to send packets to the root. The *(total) length of the redundant tree pair* is denoted by

$$L_r(\mathcal{T}_r^1, \mathcal{T}_r^2) = L_r(\mathcal{T}_r^1) + L_r(\mathcal{T}_r^2).$$

Our task is now to find a pair of trees that minimize the total path length. It turns out that the trees in Fig. 1d are such minimum length redundant trees for the running example of Fig. 1a. Observe that each node maintains only two forwarding table entries (one for the red tree and one for the blue), which gives excellent scalability. Formally, the problem is stated as follows.

Definition 3: Minimum Length Redundant Trees problem (MLRT): given an undirected graph G, lengths l, root node $r \in V$, and positive integer k, determine whether there exists a pair of redundant trees $\mathcal{T}_r^1$ and $\mathcal{T}_r^2$ so that $L_r(\mathcal{T}_r^1, \mathcal{T}_r^2) \leq k$.

Our main concern here is the optimization version of this problem, where the task is to minimize $L_r(\mathcal{T}_r^1, \mathcal{T}_r^2)$. For this version, an optimal Integer Linear Program (ILP) with exponential worst-case solution time along with a heuristic with $O(n^3)$ running time were given in [11], [24]. In Section IV, we shall improve the running time to the same as Dijkstra's algorithm and the Suurballe-Tarjan algorithm, $O(n \log n + m)$.

E. Performance Metric for Redundant Trees

Regrettably, the coupling between the paths brought about by the requirement that we need these paths to make up two trees yields that the total length will increase somewhat. In general, it holds that the path-lengths for any pair of redundant trees $\mathcal{T}_r^1, \mathcal{T}_r^2$ will be higher than the optimum:

$$L_r(\mathcal{T}_r^1, \mathcal{T}_r^2) \geq \sum_{v \neq r} L_{v,r}^2(G) \ . \tag{1}$$

This is the price we pay for scalability.

For a graph G with edge lengths l and root node r, the path length gap of node v is defined as $L_{v,r}(\mathcal{T}_r^1) + L_{v,r}(\mathcal{T}_r^2) - L_{v,r}^2(G)$. We say that a node v is *perfectly covered* by the redundant trees $\mathcal{T}_r^1$ and $\mathcal{T}_r^2$ if $L_{v,r}(\mathcal{T}_r^1) + L_{v,r}(\mathcal{T}_r^2) - L_{v,r}^2(G) = 0$. Hence, the *path length ratio* of a redundant tree pair provided by an arbitrary heuristic algorithm for root node r is defined as:

$$\eta(G, r) = \frac{1}{n-1} \sum_{v \in V : v \neq r} \left(\frac{L_{v,r}(\mathcal{T}_r^1) + L_{v,r}(\mathcal{T}_r^2)}{L_{v,r}^2(G)} - 1 \right) \ , \tag{2}$$

where n is the number of nodes in the network. Hence, $\eta(G, r) = 0$ means all nodes are perfectly covered for root node r, while positive values of $\eta(G, r)$ represents the penalty we pay for scalability.

Besides the length ratio of a single root node r, when pairs of redundant trees are given for each node we are also interested in the *average path length ratio* of graph G, that describes the possible performance hit of scalable multipath routing using redundant trees for network operators:

$$\eta(G) = \frac{1}{n} \cdot \sum_{r \in V} \eta(G, r). \tag{3}$$

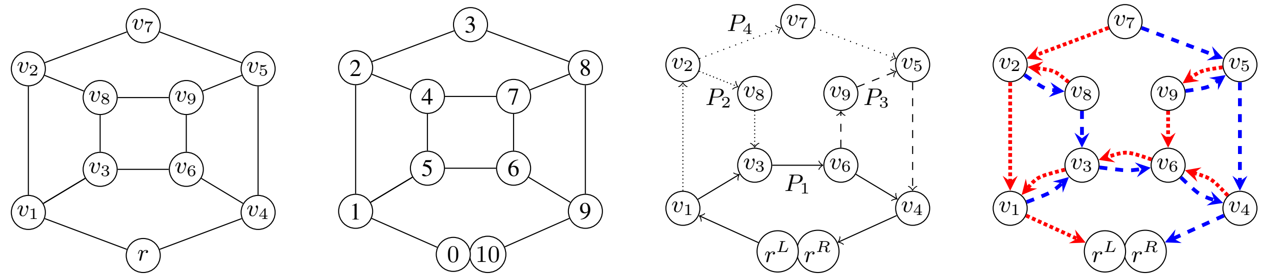
Our aim in this paper is to analyze the price we pay for scalability, as measured by the (average) path length ratio.

III. Computational Complexity of the Minimum Length Redundant Trees Problem

There is a substantial body of literature on various forms of length-minimization for redundant trees, yet, as far as we are aware of, for the exact formulation above no complexity characterization is available. The authors in [29], [30] study the task to find two edge-disjoint spanning trees of a minimum stretch, but for the all-pairs case (i.e., when the trees are not rooted). Another version where the total length of the edges in the redundant trees (in contrast to the length of the paths) is to be minimized is examined in [22], [31]. Although such trees are optimal in total link length, some paths towards the root might be sub-optimal. The exact problem formulation for *MLRT* appears in [11], [24], but no complexity analysis ensued. Next, we settle this long-standing question.

Theorem 1: MLRT is NP-complete.

Refer to the Appendix for the full proof [28]. The argument is based on a Karp-reduction from a special form of the Boolean Satisfiability problem called "not-all-equal" 3SAT (*NAE-3SAT*). Given a Boolean expression in conjunctive normal form with 3 literals per clause, *NAE-3SAT* asks for an



(a) A sample graph (b) st-numbering (c) st-orientation (d) Redundant trees

Fig. 2: A simple undirected graph demonstrating the different techniques used in the MLRT algorithms.

assignment of variables so that in every clause at least one literal is set to true and at least one literal is set to false [42].

IV. Algorithmic Framework for Constructing Node-Redundant Trees

As most of the redundant tree algorithms revolve around the concepts of ear-decompositions and st-numberings [11], [17], [21]–[26], [31], [32], we will introduce them in the following sub-sections. For simpler presentation, we often split r into two nodes r^L and r^R and the red tree terminates in r^L, and the blue tree in r^R (see Fig. 2).

For the sake of explanation, we describe the design concepts and introduce our heuristic algorithms in Section V on a special version of the node-redundant tree problem, where *the root node has exactly two adjacent links*. Later (in Section VI-A) we will explain how to extend these algorithms for the general node-redundant and edge-redundant problems.

A. Ear-Decomposition

Ear-decomposition is a graph reduction technique to decompose any 2-connected graph G into a sequence of 2-connected subgraphs $G_0 \subset G_1 \subset \cdots \subset G_k$. $G_0 = (V_0, E_0)$ is composed of a single root r and G_k has all nodes of G, i.e., $V_k = V$. For each $i = 1, \ldots, k : G_i := G_{i-1} \cup P_i$, where $P_i = x_i - \ldots - v - \ldots - y_i$ is a path between nodes $x_i, y_i \in V_{i-1}$, where $P_i \cap V_{i-1} = \{x_i, y_i\}$. Such a P_i is called an *ear*.

In the node-redundant tree design, ear P_i is either a simple path between *two distinct nodes* $x_i \neq y_i$, or a simple cycle traversing the root node if $x_i = y_i = r$. For the graph in Fig. 2a, a possible ear-decomposition would consist of the following ears: $P_1 = r - v_1 - v_3 - v_6 - v_4 - r$, $P_2 = v_1 - v_2 - v_8 - v_3$, $P_3 = v_6 - v_9 - v_5 - v_4$, and $P_4 = v_2 - v_7 - v_5$ (shown in Fig. 2c). In order to obtain node-redundant trees from the ear-decomposition, the "forward" directed path (visiting the nodes from left to right) of an ear P_i^f excluding x_i is added to $\mathcal{T}_r^2$ (or $\mathcal{T}_r^1$) and the "backward" directed path (traversing nodes from right to left) P_i^b excluding y_i is added to $\mathcal{T}_r^1$ (or $\mathcal{T}_r^2$), respectively. In our running example, tree $\mathcal{T}_r^1$ in Fig. 2d is constructed from $P_1^b = v_4 \to v_6 \to v_3 \to v_1 \to r$, $P_2^b = v_8 \to v_2 \to v_1$, $P_3^b = v_5 \to v_9 \to v_6$, and $P_4^b = v_7 \to v_2$, while tree $\mathcal{T}_r^2$ contains $P_1^f = v_1 \to v_3 \to v_6 \to v_4 \to r$, $P_2^f = v_2 \to v_8 \to v_3$, $P_3^f = v_9 \to v_5 \to v_4$, and $P_4^f = v_7 \to v_5$.

Although ears can be selected basically arbitrarily [21] in a redundant tree design, constructing the redundant trees from the ears should be done carefully, because a wrong decision influences the length of every path connected to these ears later on. What is even worse, we may end up in a situation where none of the directions of an ear can be selected to provide node-redundant trees. For example, in Fig. 3 an ear P_i is selected which traverses the node with smallest ID in $V \setminus V_i$, while P_i^f and P_i^b are randomly added to the two trees. In the first iteration ear $P_1 = r - v_8 - v_1 - v_2 - v_9 - r$ is selected (traverses v_1) and P_1^f is added to $\mathcal{T}_r^2$ and P_1^b is added to $\mathcal{T}_r^1$. In the next steps, ear $P_2 = v_8 - v_4 - v_3 - v_9$ traversing v_3 is selected between $v_8, v_9 \in V_1$, and P_2^f is added to $\mathcal{T}_r^2$ and P_2^b is added to $\mathcal{T}_r^1$; ear $P_3 = v_3 - v_5 - v_1$ is selected between $v_3, v_1 \in V_2$ traversing v_5 and $\mathcal{T}_r^2$ is extended with P_3^b and $\mathcal{T}_r^1$ with P_3^f; and $P_4 = v_2 - v_6 - v_4$ is selected traversing v_6 between $v_2, v_4 \in V_3$ while $\mathcal{T}_r^2$ is extended with P_4^f and $\mathcal{T}_r^1$ with P_4^b. Until this point, $\mathcal{T}_r^1$ and $\mathcal{T}_r^2$ are a pair of node-redundant trees for the nodes in V_4. However, when the ear $P_5 = v_5 - v_7 - v_6$ is selected traversing v_7 between $v_5, v_6 \in V_4$, either adding P_5^f to $\mathcal{T}_r^2$ and P_5^b to $\mathcal{T}_r^1$ or vice versa would result in a solution where the paths from v_7 in $\mathcal{T}_r^1$ and $\mathcal{T}_r^2$ are not node-disjoint (either having nodes v_3, v_4 or v_1, v_2 and the edge between them in common). Note that if P_3^f would have been added to $\mathcal{T}_r^2$ and P_3^b to $\mathcal{T}_r^1$, adding either direction of P_5 to the trees will end up in two node-redundant trees for the nodes $V_5 = V$.

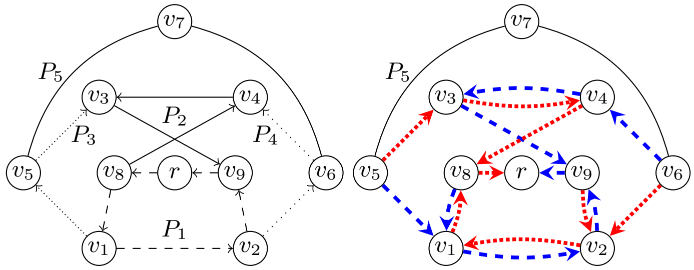
To avoid the above situation, the concept of st-numbering can be applied, which provides a sufficient condition to construct redundant trees from an ear-decomposition.

B. Sufficient Conditions for Redundant-Trees

An *st-numbering* is a complete order (a.k.a. linear, or strict total order) defined on the nodes in G, where r^L is the smallest and r^R is the largest element and each remaining node v is adjacent to two nodes x and y such that $x < v < y$ (see Fig. 2b). For the sake of simplicity, we assume that an st-numbering assigns a real number for each node v denoted by π_v, and two nodes u and v are in relation $<$ if and only if $\pi_u < \pi_v$.

Lemma 1: Given an st-numbering of G, two redundant trees in G always exist.

Proof: We orient the edges of G such that each edge (u, v) is directed $u \to v$ if $\pi_u < \pi_v$, and $u \leftarrow v$ otherwise. Let



(a) The ear decomposition (b) Redundant trees in G_4

Fig. 3: An example where wrong selection of ear direction results an invalid solution. Every edge has a unit length.

Algorithm 2: Compute st-orientation (partial order)

Procedure `earDirection`$(P_i = x_i\text{-}\dots\text{-}v\text{-}\dots\text{-}y_i)$

1 **if** $L_{x_i,r}(\mathcal{T}_r^1) + L_{y_i,r}(\mathcal{T}_r^2) < L_{x_i,r}(\mathcal{T}_r^2) + L_{y_i,r}(\mathcal{T}_r^1)$ **then**
2   **if** *no path* $y_i \to x_i$ *in* G^D **then**
3   Set $x_i \to \dots \to v \to \dots \to y_i$; **return** `true`
  else
4   Set $x_i \leftarrow \dots \leftarrow v \leftarrow \dots \leftarrow y_i$; **return** `false`
else
5   **if** *no path* $x_i \to y_i$ *in* G^D **then**
6   Set $x_i \leftarrow \dots \leftarrow v \leftarrow \dots \leftarrow y_i$; **return** `false`
  else
7   Set $x_i \to \dots \to v \to \dots \to y_i$; **return** `true`

$\mathcal{T}_r^1$ be an out-tree from r^L and $\mathcal{T}_r^2$ be an in-tree towards r^R. Clearly, every node can be reached from r^L and every node can reach r^R. The path $P(\mathcal{T}_r^1, v, r)$ will traverse nodes with label at most π_v, while $P(\mathcal{T}_r^2, v, r)$ will traverse nodes with label at least π_v; therefore they are disjoint paths. ■

As a consequence of Lemma 1 an st-numbering is sufficient to find redundant trees. Hence, the question is how to find an st-numbering efficiently. One possibility is to obtain it from an ear-decomposition by maintaining a complete order of the nodes through the graph sequence $G_0 \subset G_1 \subset \dots \subset G_k$ when ear $P_i = x_i\text{-}\dots\text{-}v\text{-}\dots\text{-}y_i$ is added to G_{i-1}. As we are dealing with node-redundant trees and we split the root node into r^L and r^R, ears are simple paths (i.e., $\forall i : x_i \neq y_i$). Hence, if $\pi_{x_i} > \pi_{y_i}$ we can label the nodes as $\pi_{x_i} > \dots > \pi_v > \dots > \pi_{y_i}$ by arbitrarily assigning values to them from the range (π_{y_i}, π_{x_i}), otherwise we label them as $\pi_{x_i} < \dots < \pi_v < \dots < \pi_{y_i}$. The pseudocode is summarized in Algorithm 1.

Algorithm 1: Compute st-numbering (complete order)

Procedure `earDirection`$(P_i = x_i\text{-}\dots\text{-}v\text{-}\dots\text{-}y_i)$

if $\pi_{x_i} > \pi_{y_i}$ **then**
  Set π_v for all internal nodes v such that $\pi_{x_i} > \dots > \pi_v > \dots > \pi_{y_i}$; **return** `true`
else
  Set π_v for all internal nodes v such that $\pi_{x_i} < \dots < \pi_v < \dots < \pi_{y_i}$; **return** `false`

A *st-orientation* (or bipolar orientation) is a different technique to maintain the order of nodes, which assigns an orientation to each edge of G. The resultant graph G^D is a directed acyclic graph, r^L is the only node with zero in-degree, and r^R is the only node with zero out-degree (see Fig. 2c).

Lemma 2: Given an st-orientation of G, two redundant trees in G always exist.

Proof: Use a topological order of G as an st-numbering for Lemma 1. ■

When an ear is added, we need to add it as a directed path to G^D as described in Algorithm 2.

After every ear is processed (i.e., $V_k = V$), the directed paths in G^D define a partial order between the nodes. Finally, any topological order of G^D provides an st-numbering (i.e., complete order) of the nodes, which is sufficient to compute two redundant trees by Lemma 1, as one path will traverse only nodes with a label at least π_v, while the other will traverse nodes only with labels at most π_v.

In the implementations instead of an st-numbering often the complete order is represented using a simple node-potential [31] (see also [17]), while $V \setminus V_{i-1}$ is maintained with simple node marking (observe that each node is visited at most once). In contrast with the complete order used in st-numbering, in st-orientation we might be able to choose the direction of the ear P_i, e.g., based on the current path lengths in the sub-trees in G_{i-1}, checked in Step (1). Hence, using an st-orientation the path lengths can be improved. However, it has a higher computational complexity owing to check the existence of paths between the node pairs in Step (2) and Step (5).

C. Related Work

Essentially all existing heuristic algorithms use (some variant of) the above algorithmic framework: build an ear-decomposition and maintain an st-orientation thereof to obtain a feasible redundant tree-pair. However, neither of these works are based on the Suurballe-Tarjan algorithm, which will drive the heuristics introduced in this paper.

The heuristic in [21] selects the ears basically randomly, which was later improved to a greedy strategy to minimize some intuitive path-length metrics in [25] and [24]. In particular, the BR algorithm [24] selects an ear so as to minimize the path length after the ear is added, which requires an all-pairs shortest path calculation each time an ear is added, in worst case $O(n^3)$ steps overall. However, it yields the best performance in the investigated performance metrics. Another common approach is to use Depth First Search (DFS) tree to construct the ears [31], which is also used in the IETF RFC 7811 Maximally Redundant Trees (MRT) [17].

As for redundant tree construction st-numbering (or complete order) is used in [17], [22], [31], while st-orientations (partial orders) in [24]–[26]. The trade-off between the two is the usual time vs. performance type; complete orders yield somewhat longer paths but can be computed in $O(1)$ per node in a path (although correct implementation is far from trivial [17]); while partial orders allow more liberty for the st-orientation and deliver shorter paths, but can be maintained only in $O(n)$ [11], [24].

Algorithm 3: *MLRT* Algorithmic Framework

Input: Undirected graph $G = (V, E)$, root node r, edge lengths l
begin
1 Run Suurballe-Tarjan algorithm from root r
2 $i := 1$; $V_0 := \{r\}$ `// Ear decomposition`
3 **while** $v \leftarrow$ `selectNode(`$V \setminus V_{i-1}$`)` **do**
4 $P_1^*(G, v, r), P_2^*(G, v, r)$:= min length disjoint path-pair from node v
5 Segment $\{v - a_1 - \ldots - a_k - x_i\} \subseteq P_1^*(G, v, r)$: $a_1, \ldots, a_k \notin V_{i-1}$, $x_i \in V_{i-1}$
6 Segment $\{v - b_1 - \ldots - b_l - y_i\} \subseteq P_2^*(G, v, r)$: $b_1, \ldots, b_l \notin V_{i-1}$, $y_i \in V_{i-1}$
7 **if** `earDirection(`$P_i = x_i - a_k - \ldots - a_1 - v - b_1 - \ldots - b_l - y_i$`)` **then**
8 add $x_i \leftarrow a_k \leftarrow \ldots \leftarrow b_l$ to $\mathcal{T}_r^1$ and $a_k \rightarrow \ldots \rightarrow b_l \rightarrow y_i$ to $\mathcal{T}_r^2$
else
9 add $x_i \leftarrow a_k \leftarrow \ldots \leftarrow b_l$ to $\mathcal{T}_r^2$ and $a_k \rightarrow \ldots \rightarrow b_l \rightarrow y_i$ to $\mathcal{T}_r^1$
10 $G_i := G_{i-1} \cup P_i$; $i := i + 1$

V. Minimum Length Redundant Tree Algorithms

In this section we propose the design concept of the heuristics that take the minimum length disjoint paths obtained by Suurballe-Tarjan algorithm [19] and convert them into redundant trees after a series of modifications. The approach was inspired by the following observation, a direct consequence of the data structure used in the Suurballe-Tarjan algorithm [19].

Observation 1: Let A be the union of the directed edges in the minimum length disjoint paths from every node to a single root. These paths can be chosen in a way that every node other than the root has out-degree 2 in A.

Proof: It is a consequence of the explicit construction to obtain the pair of paths from any given node described in [19]. The explicit construction contains two steps: in the first, some nodes are marked in the graph, while in the second a "traversal step" is defined to construct the path edge-by-edge from any given node. In the traversal step either the link of the shortest path towards the destination node is selected, or (if the node is marked) a specific link, denote by $p(x)$ in [19]. As a result any obtained pair of paths is composed of edges of the shortest path tree towards the destination node, denote by T in [19], or the edges of $p(x)$ for $\forall x \in V$, where each $p(x)$ is an edge with a source node x by definition. ■

This intuitively means A determined by the Suurballe-Tarjan algorithm is a subgraph of G with very few edges and so there is a hope that it can be partitioned into two redundant trees with low $\eta(G, r)$ value.

A. MLRT Algorithmic Framework

The main idea of our approach is to let the Suurballe-Tarjan algorithm drive the augmentation of the ear-decomposition. Algorithm 3 describes the pseudocode of the general framework we built our MLRT heuristics on. Each of our heuristics differs in two functions `selectNode()` and `earDirection()`, discussed in Section V-B.

Suppose we are given a weighted undirected graph G and a root r. First, for each node $v \neq r$ we compute the minimum length disjoint path-pair $P_1^*(G, v, r), P_2^*(G, v, r)$ from r using a single run of the (node-disjoint) Suurballe-Tarjan algorithm in Line 1. This can be done in $O(n \log n + m)$ steps [19]. Next, we generate an ear-decomposition $G_0 \subset G_1 \subset \cdots \subset G_k$ based on the data available after the run of the Suurballe-Tarjan algorithm (see Section V-B). The first subgraph consists of the root $V_0 = \{r\}$, and i denotes the number of ears processed (Line 2). We will select the next ear P_i as segments of the disjoint path-pair towards a node $v \in V \setminus V_{i-1}$ defined by the `selectNode()` function in Line 3. As we add an ear P_i traversing $v \in V \setminus V_{i-1}$, it has at least two edges. Let x_i be the first node along $P_1^*(G, v, r)$ that is already part of V_{i-1} and likewise let y_i be the first node of V_{i-1} along $P_2^*(G, v, r)$ (Lines 5 and 6). Construct the ear P_i as the concatenation of path segments $x_i \rightarrow \ldots \rightarrow v$ of $P_1^*(G, v, r)$ and $v \rightarrow \ldots \rightarrow y_i$ of $P_2^*(G, v, r)$ and denote this ear by $P_i = x_i \text{-} \ldots \text{-} v \text{-} \ldots \text{-} y_i$.

At this point the function `earDirection(`P_i`)` will decide the orientation for the new ear (Line 7) and either adds P_i^b to $\mathcal{T}_r^1$ and P_i^f to $\mathcal{T}_r^2$ (Line 8), or vice versa (Line 9). Finally, we add P_i to G_{i-1} to obtain G_i (Line 10) and the process is repeated until every node is covered by V_i.

B. MLRT Heuristic Algorithms

Here we summarize the different `selectNode()` and `earDirection()` functions which might be of interest.

As for `earDirection()` first we compute the length of selecting the ear in each direction, which is $L_{x_i,r}(\mathcal{T}_r^1) + L_{y_i,r}(\mathcal{T}_r^2)$ for the `true` branch of the condition of Line (7), and $L_{x_i,r}(\mathcal{T}_r^2) + L_{y_i,r}(\mathcal{T}_r^1)$ for the `false` branch. Then we return the smallest length direction that is a valid solution validated by one of the following two options:

- ST_{stn} st-numbering (complete order), see Algorithm 1, and
- ST_{po} st-orientation (partial order), see Algorithm 2.

The function `selectNode()` is implemented by selecting the nodes from a list sorted in the ascending order of the following lengths available after a single run of the Suurballe-Tarjan algorithm:

- ST^0 minimal total length of the disjoint path-pair $L_{v,r}^2(G)$,
- ST^α minimal $L_{v,r}^2(G) - \alpha L_{v,r}^1(G)$ length, where $0 \leq \alpha \leq 2$. Note that with $\alpha = 0$ we get ST^0, while $\alpha = 2$ corresponds to the order the Suurbale-Tarjan algorithm labels the nodes (i.e., reduced edge lengths $L_{v,r}^2(G) - 2L_{v,r}^1(G)$). Hence, α represents a trade-off between the two orders.

Based on the above classification we focus on the following four heuristics:

- ST_{stn}^0 which runs in $O(n \log n + m)$ steps. One could hardly expect to go beyond that point, as at least one shortest path calculation is needed to direct the algorithm towards short paths.
- ST_{stn}^α which solves the problem with 10 different $\alpha = 0, 0.2, 0.4, \ldots 2$ values and the solution with smaller $\eta(G, r)$ is selected. Asymptotically it has the same running time $O(n \log n + m)$.
- ST_{po}^0 has slightly shorter paths by maintaining an st-orientation (partial order) at the price of increasing the running time to $O(n^2)$.

ST_{po}^{α} which solves the problem with 10 different $\alpha = 0, 0.2, 0.4, \ldots 2$ values and the solution with smaller $\eta(G, r)$ is selected. Asymptotically it has the same running time $O(n^2)$.

If the lengths of the paths in the redundant trees equal the minimum length disjoint path pairs (i.e., $\eta(G, r) = 0$), BR algorithm [24] produces similar results as ST^0. Note that our methods restricted to ears generated by the Suurballe-Tarjan algorithm improve the running time of BR from $O(n^3)$ to $O(n^2)$ with ST_{po} or even to $O(n \log n + m)$ with ST_{stn} without major performance hit, shown in Section VII.

Claim 1: The worst case running time of the heuristic ST_{po} is $O(n^2)$ and ST_{stn} is $O(n \log n + m)$.

VI. Discussion on Redundant Tree Algorithms

This section is devoted to summarizing the generalizations, strengths, and limitations of redundant tree algorithms. Section VI-A shows the necessary modifications of the algorithmic framework of Section IV to handle the general node- and edge-redundant tree problems. In Section VI-B we provide our conjecture on the theoretical upper bound for $\eta(G, r)$. Although redundant tree approaches revolve around the concept of st-numbering, we give an example that optimal redundant trees are not always obtainable with this approach (Section VI-C). Finally, in Section VI-D we show that even if a primary tree is already given it is NP-complete to find a minimum length secondary tree concerning this primary tree.

A. General Edge- and Node-Redundant Trees

Now we show how the previously presented methods Algorithm 1 and Algorithm 2 can be modified to find two edge-redundant trees in 2-edge-connected graphs or to find node-redundant trees when the root degree is more than two. The common characteristic of these cases is the appearance of closed ears, which can be tackled by representing a node $v \in V$ by two copies v^L and v^R in the auxiliary graph G_D.

Lemma 3: Let $G_D = (V', E')$ be a directed auxiliary graph of G with the following properties. Node set $V' = \{v^L, v^R | v \in V\}$. Edges of the form $v^L \to v^R$ are in E' and each edge $(u, v) \in E$ has at most one corresponding edge in G_D with endpoints u^L or u^R and v^L or v^R. If for each node $v \in V - r$ there is an edge entering v^L and there is an edge leaving v^R such that G_D is a Directed Acyclic Graph (DAG), then there are two edge-redundant spanning trees in G.

Proof: Note that r^L is the only source node in G_D and similarly r^R is the only sink. For each node $v \in V - r$ we pick an arbitrary edge entering v^L. Together with edges $v^L \to v^R$ these edges form a spanning out-tree in G_D rooted at r^L. Similarly, can we define an in-tree rooted at r^R by fixing an outgoing edge for each node v^R.

Note that by contracting edges $v^L \to v^R$ we get two r-rooted spanning trees $\mathcal{T}_r^1$ and $\mathcal{T}_r^2$ in G which are edge-redundant, since each edge in G has at most one corresponding pair in G_D. ■

Based on the above a pair of edge-redundant trees can be obtained from ear-decompositions as follows. First, we need to run the edge-disjoint version of Suurballe-Tarjan algorithm and obtain the ears as shown in Algorithm 3. The key difference compared to the node redundant case is that besides simple paths we may have "closed ears", i.e., simple cycles owing to $x_i = y_i$. It can be handled in the same way as the graph transformation used in Lemma 3. In case of st-orientation for each node v we need to assign two nodes v^L, v^R, in G_D, and add directed edges $v^L \to v^R$ for $\forall v \in V$. Before an ear $P_i = x_i\text{-}a_k\text{-} \ldots \text{-}b_l\text{-}y_i$ is added in Line 2 of Alg. 2 we search for a path $x_i^L \to \ldots \to y_i^R$ in G_D. If such path exists (the return value is `true` in Lines 3 and 7) we add directed edges $x_i^L \to a_k^L$, $a_k^R \to a_{k-1}^L$, …, $b_{l-1}^R \to b_l^L$, $b_l^R \to y_i^R$. When the return value is `false` (Lines 4 and 5), we add directed edges $y_i^L \to b_l^L$, $b_l^R \to b_{l-1}^L$, …, $a_{k-1}^R \to a_k^L$, $a_k^R \to x_i^R$. It is easy to check that the resulting graph G_D fulfills the properties of the one described in Lemma 3, and the spanning trees built in the analog of Algorithm 3 are like the ones constructed in the proof, thus giving two edge-redundant trees.

For general node redundant trees the only closed ears we may face is in the root node r, for which we assign two nodes r^L and r^R in G^D, and follow the same approach.

In the case of st-numbering, we assign two numbers for each node with similar logic as we did above for the partial order and in Lemma 3.

B. Conjecture on the Average Path Length Ratio

For the case of our sample graph G_1, the redundant trees on Fig. 1d yield the total length of $L_r(\mathcal{T}_r^1, \mathcal{T}_r^2) = 16M + 62$, while $\sum_{v \neq r} L_{v,r}^2(G_1) = 10M + 68$ and so $\eta(G_1, r) \simeq 0.6$. A simple calculation will lead to the following observation.

Observation 2: Let graph G_M be a $4M + 8$ node graph that is constructed from Fig. 1a by replacing each of v_8, v_{10}, v_{11}, and v_9 by a chain of M new nodes. Then, $\lim_{M \to \infty} \eta(G_M, r) = 2/3$ for root node r.

See the proof in the Appendix. Curiously, so far we have not been able to find any graph for which the path length ratio was larger. In all our theoretical investigations, evaluations on famous difficult graph instances, and all our simulations on hundreds of graphs with widely varying length functions (see Section VII), we have not found even a single example and root node where the ratio was above $2/3$.

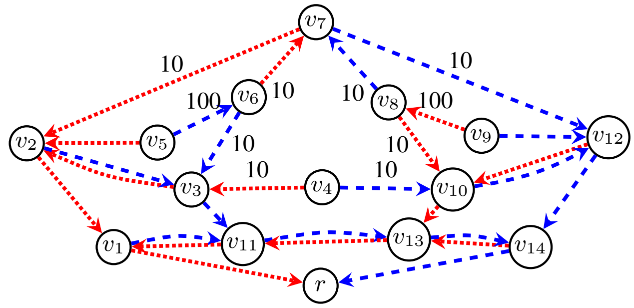
Conjecture 1: For an undirected graph G, lengths l and root node r, there is a pair of redundant trees with average path length ratio at most $\eta(G, r) \leq 2/3$.

It is highly unexpected as, at first sight, the restriction that paths must reside in two trees looks daunting. It seems that in reality, the penalty for scalable multipath routing might not be that large.

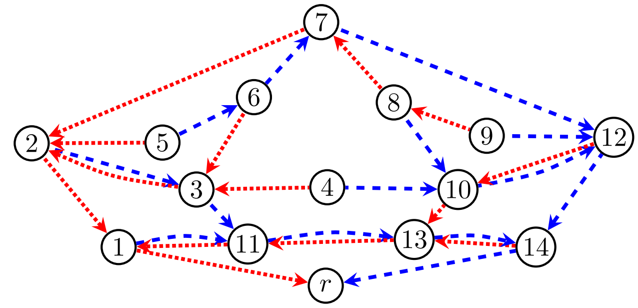
C. Minimum Length Redundant Trees with st-Numbering

In Lemma 1 we proved that an st-numbering is sufficient to find redundant trees. Hence, previous approaches [11], [17], [22], [24]–[26], [31] and our proposed algorithms are built on ear-decomposition and st-numbering to tackle the redundant tree problem.

Observation 3: Not every minimum-length redundant tree pair can be obtained by an st-numbering (or st-orientation).



(a) Optimal redundant trees. Unlabeled edges are of zero length. Every node is perfectly covered. There is no equivalent st-numbering because of the cycle v_7, v_8, v_{10}, v_4, v_3 and v_6.



(b) Optimal solution over st-orientations. Nodes are labeled as of the st-numbering. Observe that at nodes v_9 and v_5 the path length gap is 10.

Fig. 4: An example network where the optimal redundant trees cannot be obtained by an st-numbering.

See Fig. 4 for a counter-example. However, in Section VII we show that ears based on the Suurballe-Tarjan algorithm are efficient in practice and can approximate the minimum length redundant trees by a factor of $1.01 - 1.05$.

D. Secondary Tree for a Fixed Spanning Tree

We show that the redundant tree problem remains NP-complete when a spanning tree, without loss of generality $\mathcal{T}_r^1$ (e.g., the shortest path tree) is already given, and the task is to find a redundant tree-pair $\mathcal{T}_r^2$ for it. Note that if we do not require the secondary next-hops to line up into a spanning tree, then a linear algorithm exists to compute the secondary next-hops, i.e., backup forwarding table [43]. However, the backup paths contain huge detours in this case, which questions their practical applicability. Furthermore, as the primary and backup paths are not fully disjoint, this method guarantees only that the failed link can be bypassed, and can not be used for e.g., load-balancing or multipath routing purposes. Next, we show that even deciding the existence of a secondary node-redundant tree for a fixed tree is already a complex problem.

Definition 4: Minimum Length Secondary Tree problem (MLST): given an undirected graph $G = (V, E)$ with fixed root node $r \in V$ and spanning tree $\mathcal{T}_r^1$ in G, determine whether there is a spanning tree $\mathcal{T}_r^2$ in G that is node-redundant with $\mathcal{T}_r^1$.

Theorem 2: MLST is NP-complete.

Refer to the Appendix for the full proof. The argument is based on a Karp-reduction from a special form of the Boolean satisfiability problem called "SAT with non-mixed clauses" (*NM-SAT*). Given a Boolean expression in conjunctive normal form, *NM-SAT* asks for an assignment of variables so that all clauses have a type "unnegated" (contain only unnegated literals) or "negated" (contain only negated literals).

VII. Numerical Evaluation

We carried out an extensive numerical evaluation to investigate the following questions:

Q1 How much penalty do we pay for scalability in disjoint multipath routing?

Q2 What is the performance of the prosed heuristics compared to their existing counterparts?

In order to give high statistical significance to our experimental results, we have examined a broad family of graphs, from real ISP network topologies from [44] (Table I) and small-world random graphs [45] to random planar graphs, over widely varying edge length settings including inferred lengths [46] and uniform random lengths. We examined both the node- and the edge-disjoint case. All in all, we evaluated more than $20,000$ individual problem instances, including $4,000$ small-world and $11,000$ random planar problem instances. Note that a problem instance is composed of a network topology, a root node, a cost function, and whether we consider the edge- or node-disjoint case. We used 75 random planar graphs with 50–400 nodes and random edge costs, and 14 small world graph with 50–300 nodes and uniform edge costs.

A. Q1: We Observe $\eta(G) \approx 0.043$

Eq. (3) defines the measure of the penalty of the scalability in disjoint multipath routing, which is the *average path length ratio* metric $\eta(G)$. In other words, we evaluate how close redundant trees can approximate the length of the shortest possible disjoint paths. For the sake of easier comparison, the results are shown in percentages on some charts. Note that an $\eta(G)$ of 1% also means that the measured average path length is at most 1.01-times larger than the optimal value because of Eq. (1).

We evaluated all versions of our novel MLRT heuristics of Algorithm 3 for both the edge- and node-disjoint cases, with complete and partial order and with different distance functions for `selectNode()`. In particular, we had the two linear time algorithms ST^0_{stn} and ST^α_{stn}, and the versions ST^0_{po} and ST^α_{po} producing shorter paths for the price of increased computational complexity.

Strikingly, amongst the $20,000$ instances examined by ST^α_{po} not in a single case we found the average path length ratio to grow beyond 32% (with a mean of 4.3%), which is still less than half of the hypothesized maximum 66% as of Lemma 2.

1) Performance on Real-World Topologies: Table I shows the results obtained on several real world topologies, and Table II shows the averages over all investigated real-world topologies and instances (avg. $\eta(G)$). The average path lengths

TABLE I: Results on real-world network topologies. Edge costs are taken from [44]. The presented values are averages over every root node in the network. The columns mark the parameters of the input graphs (name, number of nodes and edges); the average path length ratio for both the edge- and node-disjoint case and for different algorithms; the average length of the shortest path, and the average length of the shorter path among the edge-disjoint path-pair for Surrballe's algorithm and the proposed heuristics. Note that the ratios are in percentages! For instance, a result of 1% means that the measured parameter is 1.01-times larger than the base parameter.

Network topology			The average path length ratio $\eta(G)$								The average path length					
			Edge-disjoint				Node-disjoint				Shortest path	Shorter the edge-disjoint path				
Name	$\|V\|$	$\|E\|$	ST^{α}_{po}	ST^{α}_{stn}	ST^{0}_{po}	ST^{0}_{stn}	ST^{α}_{po}	ST^{α}_{stn}	ST^{0}_{po}	ST^{0}_{stn}		Suurballe	ST^{α}_{po}	ST^{α}_{stn}	ST^{0}_{po}	ST^{0}_{stn}
Germany	17	25	0.46%	3.97%	2.55%	5.53%	0.86%	0.99%	2.10%	2.10%	2.60	2.62	2.74	2.71	2.76	2.78
BtEurope	17	30	0.02%	3.53%	0.02%	3.57%	0.00%	0.00%	0.00%	0.00%	40.53	49.14	55.68	60.18	55.68	59.74
InternetMCI	18	32	0.70%	1.53%	0.75%	1.65%	0.70%	0.88%	0.70%	0.95%	27.46	27.93	28.30	28.36	28.38	28.41
AS1755	18	33	1.46%	3.31%	2.58%	6.73%	0.76%	1.83%	1.61%	4.01%	67.99	69.20	70.68	70.52	70.06	73.12
ChinaTelc	20	44	0.10%	0.18%	0.23%	0.27%	0.09%	0.14%	0.22%	0.23%	48.34	56.13	57.16	57.80	56.65	58.18
AS3967	21	36	0.75%	3.50%	1.15%	4.76%	0.48%	2.76%	0.51%	3.60%	137.51	139.24	142.40	143.76	143.66	147.98
NSF	26	43	2.21%	4.63%	2.98%	5.44%	3.04%	3.56%	3.33%	4.17%	3.16	3.19	3.31	3.31	3.31	3.36
BICS	27	42	1.43%	5.69%	2.63%	14.58%	0.21%	1.10%	1.99%	5.41%	102.36	110.46	114.92	119.57	114.24	134.57
AS1239	30	69	0.93%	2.29%	2.51%	4.29%	0.72%	2.12%	2.14%	4.22%	123.68	126.74	128.79	129.83	130.78	131.37
BtNAmerica	36	76	0.50%	8.45%	0.73%	10.27%	2.30%	3.25%	3.07%	4.76%	44.88	48.38	51.34	59.98	51.36	61.19
Germany	50	88	2.44%	6.66%	3.34%	9.46%	4.09%	5.99%	5.24%	8.55%	3690	3762	3846	3934	3878	4049
Deltacom	103	151	1.97%	7.39%	4.08%	10.17%	4.67%	6.15%	6.96%	9.32%	1986	2449	2626	2754	2714	2863

in the redundant trees are on average 1–7% longer than the optimal minimum length disjoint paths. We have also added the largest $\eta(G)$ among these topologies (max. $\eta(G)$) to Table II, which shows that even in the worst topology the average path length gap is below 20%.

Besides the most important metric $\eta(G)$, we plotted the maximum penalty (i.e., largest path length increase) a particular node suffers towards an arbitrary root node r in graph G, called the *maximum path length gap*:

$$\lambda(G) = \frac{1}{n} \cdot \sum_{r \in V} \left(\max_{\forall v \in V : v \neq r} \frac{L_{v,r}(\mathcal{T}_r^1) + L_{v,r}(\mathcal{T}_r^2)}{L^2_{v,r}(G)} - 1 \right).$$

The $\lambda(G)$ values shown in Table II are averaged over all investigated real-world topologies, which show that the maximum path length gap $\lambda(G)$ suffered by any node for different algorithms is on average 11.6% for the edge- and 15% for the node-disjoint case. Finally, to demonstrate that longer paths along the redundant trees result in moderate-length detours [17], [47], we measured their length compared to the shortest path (computed with Dijsktra's algorithm), formally:

$$\mu_{max}(G) = \frac{1}{n} \sum_{r \in V} \left[\frac{1}{n-1} \sum_{\forall v \in V : v \neq r} \left(\frac{\max\{L_{v,r}(\mathcal{T}_r^1), L_{v,r}(\mathcal{T}_r^2)\}}{L^1_{v,r}(G)} - 1 \right) \right].$$

Similarly, the average path length $\mu_{min}(G)$ can be defined for the shorter path along $\mathcal{T}_r^1, \mathcal{T}_r^2$. The average of $\mu_{min}(G)$ among all investigated real-world topologies shows that the shorter paths along the redundant trees provided by the MLRT algorithms are 18.8% longer than the possible shortest paths, while the longer paths in the redundant trees are typically twice as long as the shortest paths (134%).

Fig. 5 shows the cumulative distribution function of the maximum path length gap for both the edge- and node-redundant trees. One can observe that in more than 50% of the problem instances the redundant paths in the trees were equal to the minimum length disjoint path-pair, which means that the path length increase caused by redundant trees does not have any effect on more than half of the node-pairs.

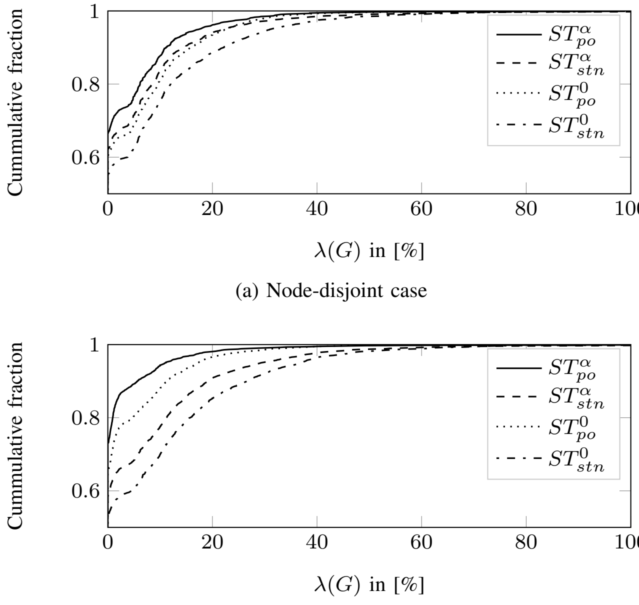
TABLE II: MLRT performance metrics averaged (maximized) over all investigated real-world topologies. Columns mark the average path length ratio $\eta(G)$; the maximum path length gap $\lambda(G)$ suffered by any node; and the average length of the shorter $\mu_{min}(G)$ and the longer path $\mu_{max}(G)$ in the redundant trees.

	Heuristic	avg. $\eta(G)$	max. $\eta(G)$	avg. $\lambda(G)$	avg. $\mu_{min}(G)$	avg. $\mu_{max}(G)$
Edge	ST^{α}_{po} (≈ ILP)	1.34%	13.50%	11.55%	18.47%	134.89%
	ST^{α}_{stn}	4.98%	28.72%	30.83%	23.89%	142.54%
	ST^{0}_{po} (≈ BR)	2.34%	21.04%	11.18%	19.81%	136.75%
	ST^{0}_{stn}	7.30%	71.15%	28.69%	28.54%	145.73%
Node	ST^{α}_{po} (≈ ILP)	2.32%	16.11%	14.47%	27.32%	145.89%
	ST^{α}_{stn}	3.29%	24.84%	24.02%	28.60%	147.71%
	ST^{0}_{po} (≈ BR)	3.34%	29.80%	13.09%	29.69%	147.12%
	ST^{0}_{stn}	5.17%	33.87%	21.07%	32.34%	150.35%

2) Performance on Random Topologies: For small world and for random planar graphs the average path length ratio and running time for the different algorithms are given in Fig. 6 and Fig. 7, respectively.

The MLRT algorithm ST^0_{po} produced almost the same results as BR but proved to be much faster in practice (theoretically too, by, recall, a factor of $O(n \log n)$) and both produce incredibly high quality paths with about 1–8% length ratio (0–3% in real networks). Even the very fast (linear average time) ST^0_{stn} heuristic was within 10–25% of the absolute optimum regarding the average path length ratio $\eta(G)$, suggesting that this algorithm is very well suited for performance-oriented applications. Finally, we found that the results are robust against parameter settings, as the ratio does not seem to vary with, say, the average nodal degree or the edge lengths.

The maximum path length gap of a single node was 100%, which means that there was a node whose two paths along the redundant trees were twice as long as the shortest disjoint path-pair. Furthermore, the optimal disjoint paths ($L^2_{v,r}(G)$)



(a) Node-disjoint case

(b) Edge-disjoint case

Fig. 5: Cumulative distribution function (CDF) of the maximum path length gap $\lambda(G)$ for node- and edge-redundant trees.

were in turn only about one and a half times longer as the absolute shortest paths ($L^1_{v,r}(G)$) obtained by Dijkstra's algorithm, indicating that *the penalty for disjoint multipath routing itself is also small.*

B. Q2: The Proposed Heuristic ST^{α}_{po} Outperforms the Existing Approaches

To compare with the prior art we have implemented the BR algorithm [24], the ILP [11], and the MRT algorithm [17].

1) The MRT algorithm: It is currently under standardization at the IETF to drive the MRT IP Fast ReRoute scheme. It is proposed only for the node-redundant version of the problem, and does not optimize for the path length. In our experience it has a huge $\eta(G)$ (the average was 58.1%); thus, it is not shown on the charts.

2) The BR algorithm: It gave the exact same results as ST^0_{po}, thus we omit BR from the charts, too. The most significant difference between the two approaches is the computational complexity, as the BR algorithm requires an all-pairs shortest path computation, thus, much slower than ST^0_{po}. In our experience BR algorithm is in average 1000 times slower than ST^0_{po}; therefore, the running times are also omitted (Fig. 7).

3) The ILP: For small problem instances the ILP always produced the same result as ST^{α}_{po}. However, even for medium-size problem instances (more than 50 nodes) the ILP could not be solved to optimality. Correspondingly, we omit the results for the ILP henceforth.

Note that ST^{α}_{po} (and ST^{α}_{stn}) are launched with 10 different $\alpha = 0, 0.2, 0.4, \ldots 2$ values. The average path length gaps corresponding to different values of the α parameter are shown in Fig. 8 (averaged for all the above real-world and random topologies). We observe that the algorithms built on partial order provide paths 5% shorter than for complete order. Furthermore, the range $[1.1 - 1.8]$ is the best range setting for the parameter α to minimize $\eta(G)$. Hence, exploring different α values and selecting the best one in ST^{α}_{stn} and ST^{α}_{po} result in performance improvement compared to ST^0_{stn} and ST^0_{po} methods, respectively, which fix α to 0.

VIII. Conclusions

With the spread of SDN, a transition to multipath routing became feasible which would fix many long-standing issues related to end-to-end reliability, security, and latency, and might also bring unexpected benefits like solving network-scale load-balancing or location-independent addressing [1]–[5], [18]. In the paper we focused on a fundamental open question [18] related to multipath routing: *How to provide path diversity with destination-based hop-by-hop forwarding (like in the OpenFlow SDN standard)?*

Our approach is inspired by Suurballe-Tarjan algorithm [19] that delivers the shortest disjoint path pairs from a single root within the same complexity as Dijkstra's shortest path algorithm. Can this algorithm be used for *scalable disjoint two-path* routing, where only two next hops (associated with red and blue trees) need to be stored in the forwarding table? In this paper, we sought answers for this crucial question.

Complexity-wise, the immediate answer is negative: we have shown that scalable disjoint multipath routing is intractable, even when the shortest paths have to be included. And performance-wise, the straightforward answer would also be negative; why would minimum length disjoint paths align into trees after all? Surprisingly, our results seem to refute these expectations; we have shown both theoretical and experimental evidence that disjoint multipath routing is viable within the same complexity as standard control plane operations (like Dijkstra's shortest path algorithm). Furthermore, with a slight increase in complexity the average path length ratio between redundant trees and optimal disjoint paths can be reduced below 8% in the vast majority of the cases. Hence, our results suggest that *scalable multipath routing might not cause a significant performance hit for operators*.

References

[1] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall, "Improving the reliability of internet paths with one-hop source routing," in *OSDI*, 2004, pp. 13–13.

[2] D. Wischik, M. Handley, and M. B. Braun, "The resource pooling principle," *SIGCOMM CCR*, vol. 38, no. 5, pp. 47–52, Sep. 2008.

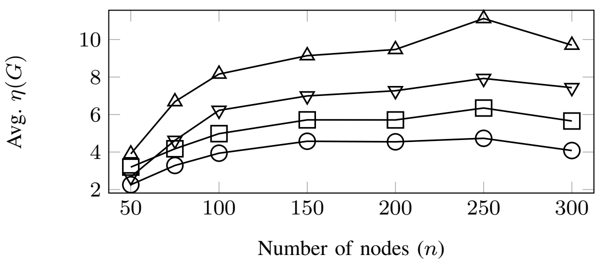
[3] M. Caesar, M. Casado, T. Koponen, J. Rexford, and S. Shenker, "Dynamic route recomputation considered harmful," *SIGCOMM CCR*, vol. 40, no. 2, pp. 66–71, Apr. 2010.

[4] O. Bonaventure, M. Handley, and C. Raiciu, "An Overview of Multipath TCP," *Usenix ;login: magazine*, vol. 37, no. 5, Oct. 2012.

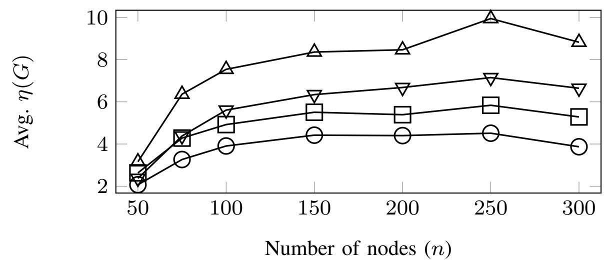
[5] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," in *CoNEXT*, December 2013.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM CCR*, vol. 38, no. 2, pp. 69–74, 3 2008.

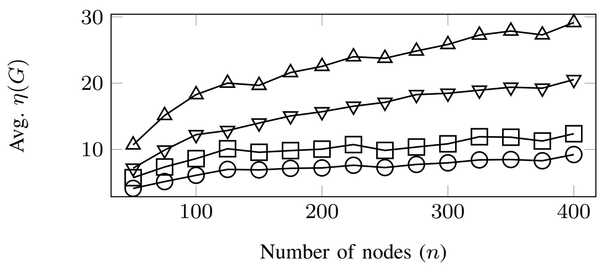
[7] Q. Wang, G. Shou, Y. Liu, Y. Hu, Z. Guo, and W. Chang, "Implementation of multipath network virtualization with SDN and NFV," *IEEE Access*, 2018.



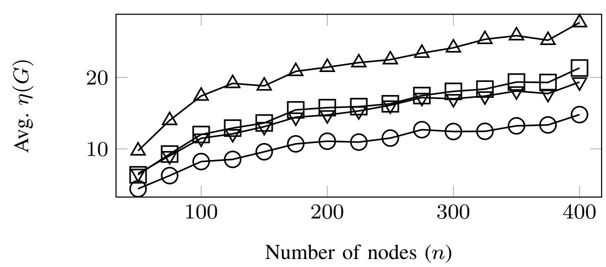
(a) Edge-redundant trees in small-world graphs.



(b) Node-redundant trees in small-world graphs.

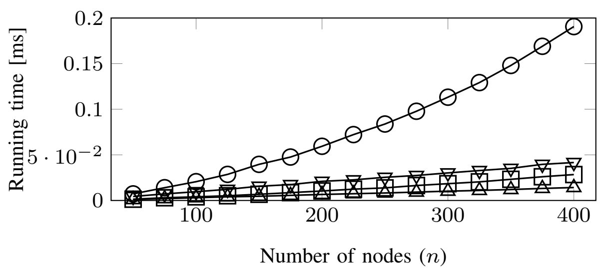


(c) Edge-redundant trees in random planar graphs.

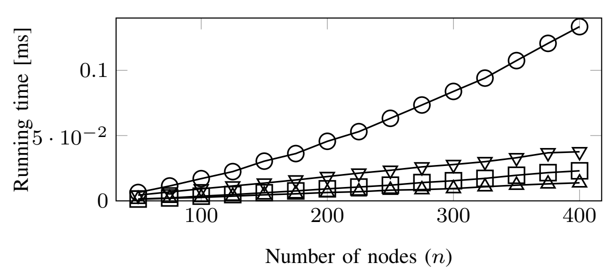


(d) Node-redundant trees in random planar graphs.

Fig. 6: Average path length ratio in random small-world and planar graphs for ST^{α}_{po} ($\circ$),ST^{α}_{stn} ($\triangledown$), ST^{0}_{po} ($\square$), ST^{0}_{stn} ($\triangle$).



(a) Small-world graphs



(b) Planar graphs

Fig. 7: Running time of MLRT algorithms ST^{α}_{po} ($\circ$),ST^{α}_{stn} ($\triangledown$), ST^{0}_{po} ($\square$), ST^{0}_{stn} ($\triangle$). The running time of BR algorithm was way outside of the visible range of the figure thus it was omitted.

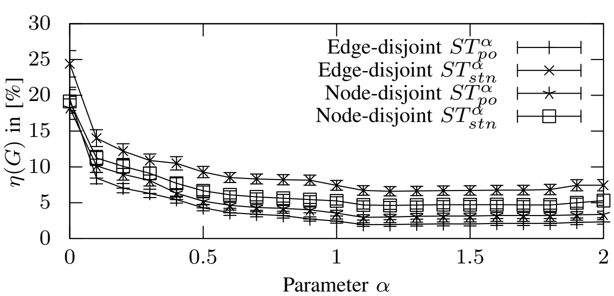


Fig. 8: Solutions depending on the parameter α.

[8] R. Teixeira, K. Marzullo, S. Savage, and G. M. Voelker, "In search of path diversity in ISP networks," in *IMC*, 2003, pp. 313–318.

[9] O. Bonaventure, "NorNet moving to Multipath TCP," 2014, http://blog.multipath-tcp.org/blog/html/2014/05/30/nornet.html.

[10] E. G. Gran, T. Dreibholz, and A. Kvalbein, "NorNet Core – a multi-homed research testbed," *Computer Networks*, vol. 61, pp. 75 – 87, 2014, special issue on Future Internet Testbeds – Part I.

[11] S. Ramasubramanian, H. Krishnamoorthy, and M. Krunz, "Disjoint multipath routing using colored trees," *Computer Networks*, vol. 51, no. 8, pp. 2163–2180, 2007.

[12] X. Zhao, D. J. Pacella, and J. Schiller, "Routing scalability: an operator's view," *IEEE JSAC*, vol. 28, no. 8, pp. 1262–1270, 2010.

[13] G. Huston, "BGP in 2013," http://www.potaroo.net/ispcol/2014-01/bgp2013.html.

[14] V. Lonker, "Thinking beyond the box – how Software Defined Networks are changing the future of connectivity," blog post, Verizon, Tech. Rep., 2018.

[15] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM CCR*, vol. 44, no. 3, pp. 87–95, 2014.

[16] H. Song, "Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane," in *ACM HotSDN*, 2013, pp. 127–132.

[17] G. Enyedi, A. Csaszar, A. Atlas, C. Bowers, and A. Gopalan, "An algorithm for computing IP/LDP fast reroute using maximally redundant trees (MRT-FRR)," Internet Requests for Comments, RFC Editor, RFC 7811, June 2016.

[18] J. He and J. Rexford, "Toward internet-wide multipath routing," *Network, IEEE*, vol. 22, no. 2, pp. 16–21, 2008.

[19] J. W. Suurballe and R. E. Tarjan, "A quick method for finding shortest pairs of disjoint paths," *Networks*, vol. 14, no. 2, pp. 325–336, 1984.

[20] P. Thulasiraman, S. Ramasubramanian, and M. Krunz, "Disjoint multipath routing in dual homing networks using colored trees," in *IEEE GLOBECOM*, Nov 2006, pp. 1–5.

[21] M. Médard, S. G. Finn, R. A. Barry, and R. G. Gallager, "Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs," *IEEE/ACM ToN*, vol. 7, no. 5, pp. 641–652, Oct. 1999.

[22] G. Xue, L. Chen, and K. Thulasiraman, "Quality-of-Service and Quality-of-Protection issues in preplanned recovery schemes using redundant trees," *IEEE JSAC*, vol. 21, no. 8, pp. 1332–1345, 2003.

[23] A. Itai and M. Rodeh, "The multi-tree approach to reliability in distributed networks," *Inf. and Comput.*, vol. 79, no. 1, pp. 43–59, 1988.

[24] R. Balasubramanian and S. Ramasubramanian, "Minimizing average path cost in colored trees for disjoint multipath routing," in *IEEE ICCCN*, 2006, pp. 185–190.

[25] S. Cho, T. Elhourani, and S. Ramasubramanian, "Independent directed acyclic graphs for resilient multipath routing," *IEEE/ACM Transactions on Networking (TON)*, vol. 20, no. 1, pp. 153–162, 2012.

[26] G. Enyedi, P. Szilágyi, G. Rétvári, and A. Császár, "IP fast reroute: Lightweight Not-Via without additional addresses," in *INFOCOM'09 Mini-Conference*, 2009, pp. 2771–2775.

[27] D. Merling, W. Braun, and M. Menth, "Efficient data plane protection for SDN," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018, pp. 10–18.

[28] J. Tapolcai, G. Rétvári, P. Babarczi, E. R. Bérczi-Kovács, P. Kristóf, and G. Enyedi, "Scalable and efficient multipath routing: Complexity and algorithms," in *Proc. IEEE ICNP*, Nov 2015, pp. 376–385.

[29] F. Annexstein, K. Berman, and R. Swaminathan, "Independent spanning trees with small stretch factors," Center for Discrete Mathematics, Theoretical Computer Science, Tech. Rep., 1996.

[30] T. Hasunuma, "On edge-disjoint spanning trees with small depths," *Information Processing Letters*, vol. 75, no. 1–2, pp. 71–74, 2000.

[31] W. Zhang, G. Xue, J. Tang, and K. Thulasiraman, "Faster algorithms for construction of recovery trees enhancing QoP and QoS," *IEEE/ACM Transactions on Networking*, vol. 16, no. 3, pp. 642–655, 2008.

[32] G. Enyedi and G. Rétvári, "Finding multiple maximally redundant trees in linear time," *Periodica Polytechnica*, vol. 54, pp. 29–40, 2010.

[33] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *ACM SIGCOMM*, 2013, pp. 99–110.

[34] S. Chole, A. Fingerhut, S. Ma, A. Sivaraman, S. Vargaftik, A. Berger, G. Mendelson, M. Alizadeh, S.-T. Chuang, I. Keslassy, A. Orda, and T. Edsall, "dRMT: Disaggregated programmable switching," in *ACM SIGCOMM*, 2017, pp. 1–14.

[35] R. Ozdag, "Intel® Ethernet switch FM6000 series - Software Defined Networking," *Intel Corporation*, 2012.

[36] Cavium, "Cavium's XPliant Ethernet switch supports the emerging open ecosystems."

[37] T. Levai, G. Pongracz, P. Megyesi, P. Voros, S. Laki, F. Nemeth, and G. Retvari, "The price for programmability in the software data plane: The vendor perspective," *IEEE Journal on Selected Areas in Communications*, pp. 1–1, 2018.

[38] K.-T. Foerster, S. Schmid, and S. Vissicchio, "Survey of consistent Software-Defined Network updates," *IEEE Communications Surveys & Tutorials*, 2018.

[39] R. Mahajan and R. Wattenhofer, "On consistent updates in Software Defined Networks," in *ACM Workshop on Hot Topics in Networks*, 2013, p. 20.

[40] A. Ludwig, J. Marcinkowski, and S. Schmid, "Scheduling loop-free network updates: It's good to relax!" in *ACM Symposium on Principles of Distributed Computing*, 2015, pp. 13–22.

[41] A. Basta, A. Blenk, S. Dudycz, A. Ludwig, and S. Schmid, "Efficient loop-free rerouting of multiple SDN flows," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 948–961, April 2018.

[42] A. Avidor, I. Berkovitch, and U. Zwick, "Improved approximation algorithms for Max NAE-SAT and Max SAT," in *Approximation and Online Algorithms*. Springer, 2006, pp. 27–40.

[43] H. Ito, K. Iwama, Y. Okabe, and T. Yoshihiro, "Polynomial-time computable backup tables for shortest-path routing," in *SIROCCO*, 2003, pp. 163–177.

[44] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," http://www.topology-zoo.org.

[45] J. Kleinberg, "The small-world phenomenon: An algorithmic perspective," in *ACM STOC*, 2000, pp. 163–170.

[46] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, "Inferring link weights using end-to-end measurements," in *IMC*, 2002, pp. 231–236.

[47] A. Atlas, R. Kebler, M. Konstantynowicz, G. Enyedi, A. Császár, and M. Shand, "An architecture for IP/LDP fast-reroute using maximally redundant trees," IETF 81, Quebec, Canada, 2011, http://www.ietf.org/proceedings/81/slides/rtgwg-2.pdf.

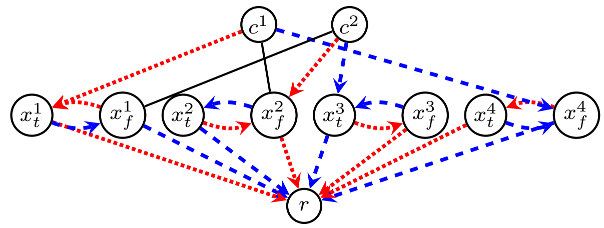


Fig. 9: The polynomial-time transformation of the NAE-3SAT instance $(X = \{x_1, x_2, x_3, x_4\}, C = \{c_1 = \{x_1, \overline{x}_2, \overline{x}_4\}, c_2 = \{\overline{x}_1, \overline{x}_2, x_3\}\})$. A NAE truth assignment is $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1$.

Appendix

A. Proof of Theorem 1

Proof: Let X, C denote an instance of a NAE-3SAT problem with variables $X = \{x_1, \ldots, x_n\}$ and clauses $C = \{c_1, \ldots, c_m\}$. We build an undirected graph $G = (V, E)$ belonging to this instance as follows:

$$V := \{r\} \cup \{x_t^i, x_f^i | x_i \in X\} \cup \{c^j | c_j \in C\},$$
$$E := \{(r, x_t^i), (r, x_f^i), (x_t^i, x_f^i) | x_i \in X\} \cup \{(x_t^i, c^j) | c_j \in C, x_i \in X, x_i \in c_j\} \cup \{(x_f^i, c^j) | c_j \in C, x_i \in X, \overline{x}_i \in c_j\},$$

where x_t^i and x_f^i correspond to the true and false assignment of variable x_i, respectively. The length $l_e = 1, \forall e \in E$. This polynomial-time transformation is shown in Fig. 9. The minimal length of the two disjoint $v \to \ldots \to r$ paths are $L^2_{v,r}(G) = 3$ for nodes $v = x_t^i$ or $v = x_f^i$, and 4 for nodes $v = c^j$. Note that the shortest pair of disjoint paths is unique for every x_t^i and x_f^i node.

Lemma 4: There is a not-all-equal truth assignment of the NAE-3SAT instance (X, C) if and only if G has two minimum length redundant trees with $L_r(\mathcal{T}_r^1, \mathcal{T}_r^2) = \sum_{v \in V} L^2_{v,r}(G)$.

Proof: $(\rightarrow)$ Let $a : X \rightarrow \{t, f\}$ be a not-all-equal truth assignment of the instance and let $\overline{a}$ denote the opposite assignment. For a clause $c_i \in C$ let $x_{t(i)}$ be a variable that gives true-valued literal in c_i (that is, either $x_{t(i)} \in c_i$ and $a(x_{t(i)}) = t$ or $\overline{x}_{t(i)} \in c_i$ and $a(x_{t(i)}) = f$). Similarly can we pick a literal $x_{f(i)}$ which evaluates to false in c_i. Now we are ready to construct trees $\mathcal{T}_r^1$ and $\mathcal{T}_r^2$:

$$\mathcal{T}_r^1 = \{(x^j_{a(x_j)}, r), (x^j_{\overline{a}(x_j)}, x^j_{a(x_j)}) | x_j \in X\} \cup \{(c^i, x^{t(i)}_{a(x_{t(i)})}) | c_i \in C\},$$
$$\mathcal{T}_r^2 = \{(x^j_{\overline{a}(x_j)}, r), (x^j_{a(x_j)}, x^j_{\overline{a}(x_j)}) | x_j \in X\} \cup \{(c^i, x^{f(i)}_{\overline{a}(x_{f(i)})}) | c_i \in C\}.$$

These are minimum length redundant trees, as $(c^i, x^{t(i)}_{a(x_{t(i)})}), (x^{t(i)}_{a(x_{t(i)})}, r) \in \mathcal{T}_r^1$ and $(c^i, x^{f(i)}_{\overline{a}(x_{f(i)})}), (x^{f(i)}_{\overline{a}(x_{f(i)})}, r) \in \mathcal{T}_r^2$, hence nodes c^i have $L_{c^i,r}(\mathcal{T}_r^1) + L_{c^i,r}(\mathcal{T}_r^2) = 2 + 2 = 4$, which is minimal. The trees $\mathcal{T}_r^1$ and $\mathcal{T}_r^2$ are clearly minimum length for nodes x_t^i and x_f^i, too.

$(\leftarrow)$ To prove the other direction let $\mathcal{T}_r^1$ and $\mathcal{T}_r^2$ be two minimum length redundant trees. Hence, for every variable $x_i \in X$, (directed) paths $(x_t^i, x_f^i), (x_f^i, r)$ and $(x_f^i, x_t^i), (x_t^i, r)$

are part of different trees, so we can define the following evaluation of X:

$$a(x_i) := \begin{cases} t & \text{, if } (x_t^i, r) \in \mathcal{T}_r^1 \\ f & \text{, if } (x_f^i, r) \in \mathcal{T}_r^1 \end{cases}$$

From the assumption on minimum length, we get that c^i have $L_{c^i,r}(\mathcal{T}_r^1) = L_{c^i,r}(\mathcal{T}_r^2) = 2$, that is there exists a variable x_j with either $x_j \in c_i$ and $(x_t^j, r) \in \mathcal{T}_r^1$ or $\overline{x}_j \in c_i$ and $(x_f^j, r) \in \mathcal{T}_r^1$. Both are equivalent to that there is a literal that is evaluated to true in clause c_i. Similarly we can derive from $L_{c^i,r}(\mathcal{T}_r^2) = 2$ that there is also a literal which is evaluated to false in c_i. ■

Since NAE-3SAT is NP-complete, the lemma proves the theorem. We note here that this proof applies both for the node-redundant and for the edge-redundant problem. ■

B. Proof of Observation 2

Proof: First, let G_1 denote the graph in Fig. 1a and we show that $\eta(G_1, r) \to 0.6$ if M grows large enough. Then, for the modified graph G_M of Observation 2 (where v_8, v_{10}, v_{11}, and v_9 are replaced by a chain of M new nodes) the same argument will result $\eta(G_M, r) = \frac{20M^2+26M+32}{12M^2+30M+36} - 1$ and so $\eta(G_M, r) \to \frac{2}{3}$ as M tends to ∞. The details are omitted for brevity.

Consider the graph G_1 in Fig. 1a, let edge lengths be 1 except on edges (v_2, v_8), (v_{10}, v_2), (v_{11}, v_5), (v_5, v_9) that have length M. We show that for any $\epsilon > 0$ there exists a value M_ϵ such that if $M > M_\epsilon$, the length ratio of G_1 is greater than $0.6 - \epsilon$. It is easy to check that the sum of shortest pair of paths is 5 for nodes v_1, v_3, v_6, v_4 and $M + 6$ for nodes v_8, v_2, v_5, v_9, finally $2M + 8$ for nodes v_{10}, v_7, v_{11}, giving a total sum of $10M + 68$.

Fig. 1d shows the pair of optimal redundant trees $\mathcal{T}_r^1$ and $\mathcal{T}_r^2$. Assume indirectly that there exist shorter redundant trees $\mathcal{F}_r^1$ (blue) and $\mathcal{F}_r^2$ (red). Without loss of generality we can assume that arc $r \to v_1$ is blue. Note that then the blue tree can only reach nodes v_{10}, v_7, v_{11} through node v_2, otherwise path $r \to v_1 \to v_3 \to v_6 \to v_5 \to v_{11}$ should be all blue, cutting nodes v_{10}, v_7, v_{11} from the red tree. Also, since in $\mathcal{T}_r^1$ and $\mathcal{T}_r^2$ only nodes v_{10}, v_7, v_{11} have longer paths from r than in G_1, the sum of the length of their corresponding path must be shorter in $\mathcal{F}_r^1$ and $\mathcal{F}_r^2$. Assume that the blue path is shorter in $\mathcal{F}_r^1$ than in $\mathcal{T}_r^1$. It can be checked that the only alternative is path $r \to v_1 \to v_3 \to v_2 \to v_{10}$, decreasing at most $3M - 3$ on the total sum. However, the red paths to nodes v_8 and v_2 must go through $v_5 \to v_{11} \to v_7 \to v_{10} \to v_2$, increasing the total sum with at least $4M$, which is bigger than $3M - 3$, giving a contradiction. ■

C. Proof of Theorem 2

Proof: First, we show that NM-SAT is NP-complete.

Lemma 5: NM-SAT is NP-complete.

Proof: We prove the lemma by reducing any SAT instance to a NM-SAT problem. Let X, C denote an instance of a SAT problem with variables $X = \{x_1, \dots, x_n\}$ and clauses $C = \{c_1, \dots, c_m\}$. If a clause c_i contains literals x_k and $\overline{x}_l$, we consider the following, equivalent problem: we add a new variable $z_{i,l}$ and instead of clause c_i we add two clauses $c_i' := c_i - \overline{x}_l + z_{i,l}$ and $c_{i,l} := \overline{z}_{i,l} \vee \overline{x}_l$. If the original SAT instance has a solution, setting $z_{i,l} = \overline{x}_l$ gives a solution of the corresponding problem and the other way round, deleting $z_{i,l}$ from a true evaluation of the second problem gives a solution of the original SAT problem. ■

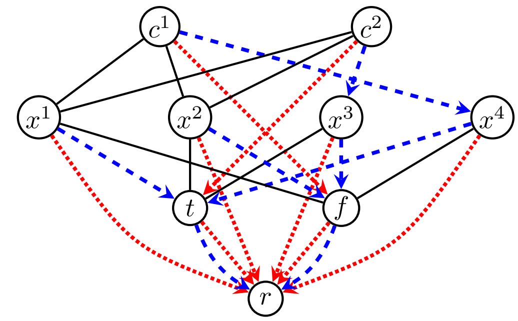


Fig. 10: The polynomial-time transformation of the NM-SAT instance with $\mathcal{T}_r^1$ fixed. ($X = \{x_1, x_2, x_3, x_4\}, C = \{c_1 = \{x_1, x_2, x_4\}, c_2 = \{\overline{x}_1, \overline{x}_2, \overline{x_3}\}\}, g(c_1) = U, g(c_2) = N$). A truth assignment is $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1$.

Now let X, C denote an instance of a NM-SAT problem with variables $X = \{x_1, \dots, x_n\}$ and clauses $C = \{c_1, \dots, c_m\}$ and let $g : C \to \{U, N\}$ be the function defining the type of the clauses (i.e., unnegated clauses have type U while negated clauses have N). We build an undirected graph G corresponding to this instance:

$$\begin{aligned} V :=& \{r\} \cup \{t, f\} \cup \{x^i | x_i \in X\} \cup \{c^j | c_j \in C\}, \\ \mathcal{T}_r^1 :=& \{(t, r)_1, (f, r)_1\} \cup \{(x^i, r) | x_i \in X\} \cup \\ & \{(c^j, f) | c_j \in C, g(c_j) = U\} \cup \\ & \{(c^j, t) | c_j \in C, g(c_j) = N\}, \\ E \setminus \mathcal{T}_r^1 :=& \{(t, r)_2, (f, r)_2\} \cup \{(x^i, t), (x^i, f) | x_i \in X\} \cup \\ & \{(c^j, x^i) | c_j \in C, x_i \text{ or } \overline{x}_i \in c_j\}. \end{aligned}$$

The polynomial-time transformation is shown in Fig. 10, where the edges in $\mathcal{T}_r^1$ are directed towards r. Note that $(t, r), (f, r)$ are multi-edges.

Lemma 6: There is a good evaluation of the instance (X, C) if and only if there is a spanning tree $\mathcal{T}_r^2$ in G which is node-redundant with $\mathcal{T}_r^1$.

Proof: ($\to$) Let $a : X \to \{t, f\}$ be a good evaluation of the NM-SAT instance. For a clause $c_i \in C$ let $x_{t(i)}$ be a variable that gives true-valued literal in c_i (that is, either $x_{t(i)} \in c_i$ and $a(x_{t(i)}) = t$ or $\overline{x}_{t(i)} \in c_i$ and $a(x_{t(i)}) = f$). Now, we are ready to construct $\mathcal{T}_r^2$:

$$\begin{aligned} \mathcal{T}_r^2 :=& \{(t, r)_2, (f, r)_2\} \cup \{(x^j, a(x_j)) | x_j \in X\} \cup \\ & \{(c^i, x^{t(i)}) | c_i \in C\}. \end{aligned}$$

To show that $\mathcal{T}_r^1$ and $\mathcal{T}_r^2$ are node-redundant, only nodes c^j have to be checked because all other nodes have a one-edge path in $\mathcal{T}_r^1$ to r. Let $c_j \in C$ be a clause with $g(c_j) = U$. The path $P(\mathcal{T}_r^2, c^j)$ is c^j-$x^{t(j)}$-t-r, which is indeed internally node-disjoint from the $P(\mathcal{T}_r^1, c^j)$ path c^j-f-r. The negated case can be shown similarly.

($\leftarrow$) To prove the other direction, let $\mathcal{T}_r^2$ be a spanning tree in E node-redundant with $\mathcal{T}_r^1$. Since only edges $(t,r)_2$ and $(f,r)_2$ are incident to r in $E \setminus \mathcal{T}_r^1$, each path in $\mathcal{T}_r^2$ to any node passes exactly one of them, defining a straightforward evaluation of X. All is left to prove is that this is a good evaluation of the clauses, that is, every clause contains a variable evaluating to true. Indeed, the first variable on path $P(\mathcal{T}_r^2, c^j)$ from a clause c^j to root r is such a variable. ■

Since NM-SAT is NP-complete from Lemma 5, Lemma 6 proves the theorem. ■

The multi-edges $(t,r),(f,r)$ can be removed by adding an intermediate node to one of the parallel edges in the transformation in Fig. 10. Thus, the same reasoning works for simple graphs as well. Furthermore, NP-completeness can be proved similarly to the *edge-redundant* MLST problem with a slightly modified polynomial-time transformation to the NM-SAT problem.

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 the High-Speed Networks Laboratory, Department of Telecommunications and Media Informatics, BME. He has authored over 150 scientific publications. His current research interests include applied mathematics, combinatorial optimization, optical networks and IP routing, addressing, and survivability. He was a recipient of several Best Paper Awards, including ICC'06, DRCN'11, HPSR'15, and NaNa'16. He is a winner of the MTA Lendölet 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 2012-2017, and the general chair of ACM SIGCOMM 2018.

Gábor Rétvári received the M.Sc. and Ph.D. degrees in electrical engineering from the Budapest University of Technology and Economics in 1999 and 2007. He is now a Senior Research Fellow at the Department of Telecommunications and Media Informatics. His research interests include all aspects of network routing and switching, the programmable data plane, and the networking applications of computational geometry and information theory. He maintains several open source scientific tools written in Perl, C, and Haskell.

Péter Babarczi (M'11) 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. He is an Assistant Professor with the Department of Telecommunications and Media Informatics at BME and currently also an Alexander von Humboldt Post-Doctoral Research Fellow with the Chair of Communication Networks at the Technical University of Munich, Germany. His current research interests include multi-path Internet routing, network coding in transport networks, and combinatorial optimization in softwarized networks. He received the János Bolyai Research Scholarship of the Hungarian Academy of Sciences and the Post-Doctoral Research Fellowship of the Alexander von Humboldt Foundation.

Erika Bérczi-Kovács received the M.Sc. Degree in Mathematics and the Ph.D. degree in Applied Mathematics from the Eötvös Loránd University (ELTE), Budapest, in 2007 and 2015, respectively. She is currently an assistant professor at the Department of Operations Research, ELTE, and she is a member of the MTA-ELTE Egerváry Research Group on Combinatorial Optimization. Her research interests are discrete mathematics, combinatorial optimization and network coding. She was a recipient of NaNA 2016 Best Paper Award.