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 multipath 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 scheme with

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, babarcbzi}@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 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 traditional 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 as compared to traditional single-path routing to a tolerable level* [9], [10], [18].

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 \( l : E \rightarrow \mathbb{R}^+ \) set according to some traffic engineering considerations. A path \( P \) in \( G \) is then an ordered set of nodes and edges \( P = s \rightarrow v_1 \rightarrow v_2 \ldots \rightarrow v_{k-1} \rightarrow r \), where \((s, v_1), (v_1, v_2), \ldots, (v_{k-1}, r) \in E\). Nodes \( s \) and \( r \) are called terminal nodes. For easier presentation we often assign a direction to the path \( P = s \rightarrow v_1 \rightarrow \ldots \rightarrow r \) 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.
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 v5 diverges from the red path starting at v7, node v5 would need to allocate a separate forwarding table entry corresponding to v7 and for itself to correctly route to r. Swapping the red and the blue paths for, say, v5, would not help either, as now a similar extra forwarding table entry would arise at node v6. 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, forwarding table entry would arise at node v, and the optimal forwarding table entry corresponding to v would need to allocate a separate forwarding table entry for each destination node as a root.

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

We also define a weaker form as follows.

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

Consider the red tree T^1_r and the blue tree T^2_r in Fig. 1d with directions assigned to their edges. Even though the edge (v3, v6) is used in both trees, the paths themselves from each node to the root are edge-disjoint (node-disjoint) and hence T^1_r and T^2_r 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 T^1_r and the other along the blue tree T^2_r (for instance, node v5 in Fig. 1d would set v9 as the next-hop for destination r along T^1_r and v6 as next-hop along T^2_r). 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

![Fig. 1: An illustrative example, with root r and a large enough positive edge length M. All unmarked edges are of unit length.](image)
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 \( T_1 \) or \( T_2 \). 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 \( T_r \) from source node \( v \) towards tree root \( r \) is calculated as \( L_{v,r}(T_r) = \sum_{e \in P(T_r,v,r)} l_e \), while the length of a tree can be obtained as \( L_r(T_r) = \sum_{v \neq r} L_{v,r}(T_r) \). What we are concerned with in this paper is finding a pair of redundant trees \( T_1^r, T_2^r \) 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

\[
L_r(T_1^r, T_2^r) = L_r(T_1^r) + L_r(T_2^r).
\]

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 \( T_1^r \) and \( T_2^r \) so that \( L_r(T_1^r, T_2^r) \leq k \).

Our main concern here is the optimization version of this problem, where the task is to minimize \( L_r(T_1^r, T_2^r) \). 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 \( T_1^r, T_2^r \) will be higher than the optimum:

\[
L_r(T_1^r, T_2^r) \geq \sum_{v \in T_r} L_{v,r}^2(G).
\]

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 \( L_{v,r}(T_1^r) + L_{v,r}(T_2^r) - L_{v,r}(G) \). We say that a node \( v \) is perfectly covered by the redundant trees \( T_1^r \) and \( T_2^r \) if \( L_{v,r}(T_1^r) + L_{v,r}(T_2^r) - L_{v,r}(G) = 0 \). Hence, the path length ratio of a redundant tree pair provided by an arbitrary heuristic algorithm for root node \( r \) is defined as:

\[
\eta(G, r) = \frac{1}{n-1} \sum_{v \in V: v \neq r} \left( \frac{L_{v,r}(T_1^r) + L_{v,r}(T_2^r)}{L_{v,r}(G)} - 1 \right),
\]

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:

\[
\eta(G) = \frac{1}{n} \sum_{r \in V} \eta(G, r).
\]

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
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 $G_0 \subset G_1 \subset \cdots \subset G_k$, $G_0 = (V_0, E_0)$ 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 $P_i = x_1 \rightarrow \cdots \rightarrow y_\ell$ 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: $P_1 = r \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_7 \rightarrow r$, $P_2 = v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4$, $P_3 = v_5 \rightarrow v_8 \rightarrow v_9 \rightarrow v_6$ and $P_4 = v_2 \rightarrow v_7 \rightarrow v_6$ (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 $T_r^2$ (or $T_r^3$) and the “backward” directed path (traversing nodes from right to left) of $P_i^b$ excluding $y_i$ is added to $T_r^1$ (or $T_r^4$), respectively. In our running example, tree $T_r^4$ in Fig. 2d is constructed from $P_1^b = v_9 \rightarrow v_8 \rightarrow v_3 \rightarrow v_2 \rightarrow v_1$, $P_2^b = v_8 \rightarrow v_2 \rightarrow v_1$, $P_3^b = v_5 \rightarrow v_9 \rightarrow v_6$, and $P_4^b = v_7 \rightarrow v_6$, while tree $T_r^2$ contains $P_1^f = v_1 \rightarrow v_3 \rightarrow v_6 \rightarrow v_4 \rightarrow r$, $P_2^f = v_2 \rightarrow v_8 \rightarrow v_3$, $P_3^f = v_9 \rightarrow v_5 \rightarrow v_4$, and $P_4^f = v_7 \rightarrow v_5$.

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 $V \setminus V_1$, while $P_i^f$ and $P_i^b$ are randomly added to the two trees. In the first iteration ear $P_1 = r \rightarrow v_8 \rightarrow v_3 \rightarrow v_2 \rightarrow v_9 \rightarrow v_7 \rightarrow r$ is selected (traverses $v_1$) and $T_r^2$ and $T_r^4$ are added to $T_r^3$. In the next steps, ear $P_2 = v_8 \rightarrow v_4 \rightarrow v_3 \rightarrow v_9 \rightarrow v_7 \rightarrow v_2$ is selected between $v_8, v_9 \in V_1$, and $P_3^f$ is added to $T_r^2$ and $P_3^b$ is added to $T_r^4$; ear $P_3 = v_3 \rightarrow v_5 \rightarrow v_1$ is selected between $v_3, v_1 \in V_2$ traversing $v_2$ and $T_r^3$ is extended with $P_3^b$ and $T_r^4$ with $P_3^f$; and $P_4 = v_2 \rightarrow v_6 \rightarrow v_4$ is selected traversing $v_6$ between $v_2, v_4 \in V_3$ while $T_r^3$ is extended with $P_4^f$ and $T_r^4$ with $P_4^b$. Until this point, $T_r^3$ and $T_r^4$ are a pair of node-redundant trees for the nodes in $V_4$. However, when the ear $P_5 = v_5 \rightarrow v_7 \rightarrow v_6$ is selected (traversing $v_7$ between $v_5, v_6 \in V_4$, either adding $P_5^f$ to $T_r^3$ and $P_5^b$ to $T_r^4$ or vice versa) would result in a solution where the paths from $v_7$ in $T_r^3$ and $T_r^4$ 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 $P_3^b$ has been added to $T_r^4$ and $P_5^b$ to $T_r^3$, 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 $\pi_v$, and two nodes $u$ and $v$ are in relation $< \pi$ 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 $u \rightarrow v$ if $\pi_u < \pi_v$, and $u \leftarrow v$ otherwise. Let
Algorithm 1: Compute st-numbering (complete order)

Procedure earDirection \((P_i = x_i \ldots v \ldots y_i)\)

1. If \(\pi_{x_i} > \pi_{y_i}\), then
2. Set \(\pi_v\) for all internal nodes \(v\) such that
   \(\pi_{x_i} > \pi_v > \ldots > \pi_{y_i}\); return true
3. Else, set \(\pi_v\) for all internal nodes \(v\) such that
   \(\pi_{x_i} < \pi_v < \ldots < \pi_{y_i}\); return false

Algorithm 2: Compute st-orientation (partial order)

Procedure earDirection \((P_i = x_i \ldots v \ldots y_i)\)

1. If \(L_{x_i,v}(T_i^1) + L_{y_i,v}(T_i^2) < L_{x_i,v}(T_i^2) + L_{y_i,v}(T_i^1)\) then
2. If no path \(y_i \rightarrow x_i\) in \(G^D\) then
3. Set \(x_i \rightarrow \ldots \rightarrow v \rightarrow \ldots \rightarrow y_i\); return true
4. Else, set \(x_i \leftarrow \ldots \leftarrow v \leftarrow \ldots \leftarrow y_i\); return false
5. If no path \(x_i \rightarrow y_i\) in \(G^D\) then
6. Set \(x_i \leftarrow \ldots \leftarrow v \leftarrow \ldots \leftarrow y_i\); return false
7. Else, set \(x_i \rightarrow \ldots \rightarrow v \rightarrow \ldots \rightarrow y_i\); return true

A \textit{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).

\textbf{Lemma 2}: Given an st-orientation of \(G\), two redundant trees in \(G\) always exist.

\textbf{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 \(\pi_v\), while the other will traverse nodes only with labels at most \(\pi_v\).

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 checking the existence of paths between the node pairs in Step (2) and Step (5).

\section*{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].

\begin{figure}[h]
  \centering
  \includegraphics[width=\textwidth]{image.png}
  \caption{An example where wrong selection of ear direction results an invalid solution. Every edge has a unit length.}
  \label{fig:example}
\end{figure}
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. \begin{align*}
        & P^1_r(G, v, r), P^2_r(G, v, r) := \text{min length disjoint path-pair from node } v \\
        & \text{Segment } \{v-a_1-\ldots-a_k-x_1\} \subseteq P^1_r(G, v, r) : a_1, \ldots, a_k \notin V_{i-1}, x_1 \in V_{i-1} \\
        & \text{Segment } \{v-b_1-\ldots-b_l-y_1\} \subseteq P^2_r(G, v, r) : b_1, \ldots, b_l \notin V_{i-1}, y_1 \in V_{i-1} \\
        & \text{if earDirection } (P_i = x_1-a_1-\ldots-a_k-v-b_1-\ldots-b_l-y_1) \\
        & \text{then}
4. \begin{align*}
        & \quad \text{add } x_1-a_k-\ldots-\rightarrow b_l \rightarrow T^1_i \text{ and add } a_k-\ldots-\rightarrow y_l \rightarrow T^2_i \\
4. \begin{align*}
        & \quad \text{else}
4. \begin{align*}
        & \quad \text{add } x_1-a_k-\ldots-\rightarrow b_l \rightarrow T^2_i \text{ and add } a_k-\ldots-\rightarrow y_l \rightarrow T^1_i \\
4. \begin{align*}
        & \end{align*}
4. \begin{align*}
        & G_i := G_{i-1} \cup P_i; \ i := i + 1
4. \begin{align*}
        & \end{align*}

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 $P^*_r(G, v, r), P^*_r(G, v, r)$ 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 $x_1$ be the first node along $P_i(G, v, r)$ that is already part of $V_{i-1}$ and likewise let $y_i$ be the first node of $V_{i-1}$ along $P^*_i(G, v, r)$ (Lines 5 and 6). Construct the ear $P_i$ as the concatenation of path segments $x_1 \rightarrow \ldots \rightarrow v$ of $P^*_i(G, v, r)$ and $v \rightarrow \ldots \rightarrow y_i$ of $P^*_i(G, v, r)$ and denote this ear by $P_i = x_1 \ldots v \ldots y_i$.

At this point the function earDirection($P_i$) will decide the orientation for the new ear (Line 7) and either adds $P_i$ to $T^1_i$ and $P^*_{i}$ to $T^2_i$ (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 $L_{x,v,r}(T^1_i) + L_{y,v,r}(T^2_i)$ for the true branch of the condition of Line (7), and $L_{x,v,r}(T^1_i) + L_{y,v,r}(T^2_i)$ 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:

\begin{align*}
ST^0 & \text{ minimal total length of the disjoint path-pair } L^2_r(G), \\
ST^\alpha & \text{ minimal } L^2_r(G) - \alpha L^1_r(G) \text{ length, where } 0 \leq \alpha \leq 2. \text{ Note that with } \alpha = 0 \text{ we get } ST^0, \text{ while } \\
\alpha = 2 \text{ corresponds to the order the Suurballe-Tarjan algorithm labels the nodes (i.e., reduced edge lengths } L^2_r(G) - 2L^1_r(G)). \text{ Hence, } \alpha \text{ represents a trade-off between the two orders.}
\end{align*}

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

$ST^0_{stn}$ 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.

$ST^\alpha_{stn}$ 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 $O(n \log n + m)$.

$ST^0_{po}$ has slightly shorter paths by maintaining an st-orientation (partial order) at the price of increasing the running time to $O(n^2)$. 


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 $v_L \rightarrow v_R$ for $v \in V$. Before an ear $P_i = x_i - a_k \cdots - b_l - y_i$ is added in Line 2 of Alg. 2 we search for a path $x_i \rightarrow \cdots \rightarrow y_i$ in $G_D$. If such path exists (the return value is true in Lines 3 and 7) we add directed edges $x_i \rightarrow a_i, a_l \rightarrow a_{l-1}, \ldots, b_{l-1} \rightarrow b_l, b_R \rightarrow y_i$. When the return value is false (Lines 4 and 5), we add directed edges $y_l \rightarrow b_l, b_R \rightarrow b_{l-1}, \ldots, a_{k-1} \rightarrow a_k, a_R \rightarrow x_i$. 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 $L_r(T_1^1, T_2^2) = 16M + 62$, while $\sum_{v \neq r} L_{v,r}(G_1) = 10M + 68$ 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).
complete when a spanning tree, without loss of generality.

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 $T_r$ (e.g., the shortest path tree) is already given, and the task is to find a redundant tree-pair $T_r^1$ 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 $T_r$ in $G$, determine whether there is a spanning tree $T_r^1$ in $G$ that is node-redundant with $T_r$.

**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 $\text{selectNode}(\cdot)$. In particular, we had the two linear time algorithms $ST_{po}$ and $ST_{po}$, and the versions $ST_{po}$ and $ST_{po}$, producing shorter paths for the price of increased computational complexity.

Strikingly, amongst the 20,000 instances examined by $ST_{po}$, 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 Random 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
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:

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

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 against parameter settings, as the ratio does not seem to vary incredibly high quality paths with about \( O(n \log n) \) 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) \( ST_{str}^{ilp} \) 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.

![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.](image)

<table>
<thead>
<tr>
<th>Network topology</th>
<th>The average path length ratio ( \eta(G) )</th>
<th>The average path length ratio ( \eta(G) )</th>
<th>The average path length ratio ( \eta(G) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Edge-disjoint</td>
<td>Node-disjoint</td>
<td>Shortest path</td>
</tr>
<tr>
<td></td>
<td>( ST_{po}^{ilp} )</td>
<td>( ST_{po}^{ilp} )</td>
<td>Surrballe</td>
</tr>
<tr>
<td>Germany</td>
<td>0.46% 3.97% 2.35% 5.33%</td>
<td>0.00% 0.00% 0.00% 0.00%</td>
<td>2.60</td>
</tr>
<tr>
<td>BiEurope</td>
<td>0.02% 3.53% 0.02% 3.57%</td>
<td>0.00% 0.00% 0.00% 0.00%</td>
<td>4.67% 4.70%</td>
</tr>
<tr>
<td>InternetMCI</td>
<td>0.70% 1.53% 0.75% 1.65%</td>
<td>0.00% 0.00% 0.00% 0.00%</td>
<td>6.15% 6.96%</td>
</tr>
<tr>
<td>AS1755</td>
<td>1.46% 3.31% 2.58% 6.73%</td>
<td>0.76% 1.83% 1.61% 4.01%</td>
<td>9.32%</td>
</tr>
<tr>
<td>ChinaTel</td>
<td>0.10% 0.18% 0.23% 0.27%</td>
<td>0.09% 0.14% 0.22% 0.23%</td>
<td>2.23% 4.63%</td>
</tr>
<tr>
<td>AS3967</td>
<td>0.75% 3.50% 1.15% 4.76%</td>
<td>0.48% 2.76% 0.51% 3.60%</td>
<td>5.41%</td>
</tr>
<tr>
<td>NSF</td>
<td>2.21% 4.63% 2.98% 5.44%</td>
<td>3.04% 3.56% 3.33% 4.17%</td>
<td>6.73%</td>
</tr>
<tr>
<td>BICS</td>
<td>1.43% 5.69% 2.63% 14.58%</td>
<td>0.21% 1.10% 1.99% 5.41%</td>
<td>2.55%</td>
</tr>
<tr>
<td>AS1239</td>
<td>0.93% 2.29% 2.51% 4.29%</td>
<td>0.72% 2.12% 2.14% 4.22%</td>
<td>1.97%</td>
</tr>
<tr>
<td>BtAmerica</td>
<td>0.50% 8.45% 0.73% 10.27%</td>
<td>2.50% 3.25% 3.07% 4.76%</td>
<td>3.16%</td>
</tr>
<tr>
<td>Germany</td>
<td>2.44% 6.66% 3.34% 9.46%</td>
<td>4.09% 5.99% 5.24% 8.55%</td>
<td>103.48</td>
</tr>
<tr>
<td>Deltacom</td>
<td>1.97% 7.39% 4.08% 10.17%</td>
<td>4.67% 6.15% 6.96% 9.32%</td>
<td>113.85</td>
</tr>
</tbody>
</table>

![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.](image)

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>avg. ( \eta(G) )</th>
<th>max. ( \eta(G) )</th>
<th>avg. ( \lambda(G) )</th>
<th>avg. ( \mu_{min}(G) )</th>
<th>avg. ( \mu_{max}(G) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( ST_{po}^{ilp} ) (≈ ILP)</td>
<td>1.34% 13.50% 11.55% 18.47%</td>
<td>134.89%</td>
<td>4.98% 28.72% 30.83% 23.89%</td>
<td>142.54%</td>
<td></td>
</tr>
<tr>
<td>( ST_{str}^{ilp} ) (≈ BR)</td>
<td>2.34% 21.04% 11.18% 19.81%</td>
<td>136.75%</td>
<td>7.30% 71.15% 28.69% 28.54%</td>
<td>145.73%</td>
<td></td>
</tr>
<tr>
<td>( ST_{po}^{ilp} ) (≈ ILP)</td>
<td>2.52% 16.11% 14.47% 27.32%</td>
<td>145.89%</td>
<td>3.29% 24.84% 24.02% 28.60%</td>
<td>147.71%</td>
<td></td>
</tr>
<tr>
<td>( ST_{str}^{ilp} ) (≈ BR)</td>
<td>3.34% 29.80% 13.09% 29.69%</td>
<td>147.12%</td>
<td>5.17% 33.87% 21.07% 32.34%</td>
<td>150.35%</td>
<td></td>
</tr>
</tbody>
</table>

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 \( ST_{po}^{ilp} \) 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) \( ST_{str}^{ilp} \) 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 (\( L^2_{v,r}(G) \))
were in turn only about one and a half times longer as the absolute shortest paths \(L_{c,r}(G)\) obtained by Dijkstra’s algorithm, indicating that the penalty for disjoint multipath routing itself is also small.

B. Q2: The Proposed Heuristic \(ST_{\alpha}^{\text{po}}\) 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 \(ST_{\alpha}^{\text{po}}\), 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 \(ST_{\alpha}^{0}\). In our experience BR algorithm is in average 1000 times slower than \(ST_{\alpha}^{0}\), therefore, the running times are also omitted (Fig. 7).

3) The ILP: For small problem instances the ILP always produced the same result as \(ST_{\alpha}^{\text{po}}\). 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 \(ST_{\alpha}^{\text{po}}\) and \(ST_{\alpha}^{0}\) 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 \(\alpha\) 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 \(\alpha\) to minimize \(\eta(G)\). Hence, exploring different \(\alpha\) values and selecting the best one in \(ST_{\alpha}^{\text{stn}}\) and \(ST_{\alpha}^{\text{po}}\) result in performance improvement compared to \(ST_{\alpha}^{\text{stn}}\) and \(ST_{\alpha}^{0}\) methods, respectively, which fix \(\alpha\) 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


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

Fig. 7: Running time of MLRT algorithms $ST_{po}^\alpha (\circ), ST_{stn}^\alpha (\bigtriangledown), ST_{po}^0 (\square), ST_{stn}^0 (\triangle)$. 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 $\alpha$.


[36] Cavium, “Cavium’s XPliant Ethernet switch supports the emerging open ecosystem.”


---

**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:

$$
V := \{r\} \cup \{(x_i^1, x_i^2) | x_i \in X\} \cup \{(c^f, c^b) | c \in C\},
$$

$$
E := \{(r, x_i^1), (r, x_i^2), (x_i^1, x_i^2) | x_i \in X\} \cup \{(x_i^1, c^f), (c^b, x_i) | c \in C, x_i \in X, x_i \in c\} \cup \{(x_i^1, c^f), (c^b, c) | c \in C, x_i \in X, x_i \in c\},
$$

where $x_i^1$ and $x_i^2$ 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 $v \rightarrow \ldots \rightarrow r$ paths are $L_{v,r}(G) = 3$ for nodes $v = x_i^1$ or $v = x_i^2$, and 4 for nodes $v = c$. Note that the shortest pair of disjoint paths is unique for every $x_i^1$ and $x_i^2$ 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 $L_r(T_1, T_2) = \sum_{e \in E} L_{v,r}(G)$.

*Proof:* ($\rightarrow$) Let $a : X \rightarrow \{t, f\}$ be a not-all-equal truth assignment of the instance and let $\pi$ denote the opposite assignment. For a clause $c_i \in C$ let $x_i(t)$ be a variable that gives true-valued literal in $c_i$ (that is, either $x_i(t) \in c_i$ and $a(x_i(t)) = t$ or $\pi(x_i(t)) \in c_i$ and $a(x_i(t)) = \pi(x_i(t))$). Similarly can we pick a literal $x_i(t)$ which evaluates to false in $c_i$. Now we are ready to construct trees $T_1$ and $T_2$:

$$
T_1 = \{(x_i(a(x_i(t)))) \cup \{(c^f, x_i(a(x_i(t)))) | c = C\},
$$

$$
T_2 = \{(x_i(a(x_i(t)))) \cup \{(c^f, x_i(a(x_i(t)))) | c = C\}.
$$

These are minimum length redundant trees, as $(x_i(a(x_i(t)))) \cup \{(c^f, x_i(a(x_i(t)))) | c = C\} \subseteq T_1$ and $(x_i(a(x_i(t)))) \cup \{(c^f, x_i(a(x_i(t)))) | c = C\} \subseteq T_2$, hence nodes $c_i$ have $L_{c_i}(T_1) + L_{c_i}(T_2) = 2 + 2 = 4$, which is minimal. The trees $T_1$ and $T_2$ are clearly minimum length for nodes $x_i(t)$ and $x_i(t)$, too.

($\leftarrow$) To prove the other direction let $T_1$ and $T_2$ be two minimum length redundant trees. Hence, for every variable $x_i \in X$, (directed) paths $(x_i^1, x_i^2), (x_i^1, r)$ and $(x_i^2, x_i^1), (x_i^2, r)$
are part of different trees, so we can define the following evaluation of $X$:

$$a(x_i) := \begin{cases} 
  t, & \text{if } (x_i, r) \in T_r^1 \\
  f, & \text{if } (x_i, r) \in T_r^2 
\end{cases}$$

From the assumption on minimum length, we get that $c^i$ have $L_{c^i,r}(T_r^1) = L_{c^i,r}(T_r^2) = 2$, that is there exists a variable $x_j$ with either $x_j \in c_i$ and $(x_j, r) \in T_r^1$ or $\overline{x}_j \in c_i$ and $(x_j, r) \in T_r^2$. Both are equivalent to that there is a literal that is evaluated to true in clause $c_i$. Similarly we can derive from $L_{c^i,r}(T_r^2) = 2$ 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 $\eta(G_1, r) \rightarrow 0.6$ 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 $\eta(G_M, r) = \frac{20M^2 + 26M + 32}{12M^3 + 30M^2 + 36}$ and so $\eta(G_M, r) \rightarrow \frac{2}{3}$ as $M$ tends to $\infty$. 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 $M_\epsilon$ 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 $v_8, v_2, v_9, v_9$ 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 $T_r^1$ and $T_r^2$. Assume indirectly that there exist shorter redundant trees $F_r^1$ (blue) and $F_r^2$ (red). Without loss of generality we can assume that arc $r \rightarrow v_1$ is blue. Note that then the blue tree can only reach nodes $v_{10}, v_7, v_{11}$ through node $v_2$, otherwise path $r \rightarrow v_1 \rightarrow v_3 \rightarrow v_6 \rightarrow v_5 \rightarrow v_{11}$ should be all blue, cutting nodes $v_{10}, v_7, v_{11}$ from the red tree. Also, since in $T_r^1$ and $T_r^2$ 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 $F_r^1$ and $F_r^2$. Assume that the blue path is shorter in $F_r^1$ than in $T_r^1$. It can be checked that the only alternative is path $r \rightarrow v_1 \rightarrow v_3 \rightarrow v_2 \rightarrow v_{10}$, decreasing at most $3M - 3$ on the total sum. However, the paths to nodes $v_8$ and $v_2$ must go through $v_5 \rightarrow v_1 \rightarrow v_7 \rightarrow v_{10} \rightarrow v_2$, 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 $X = \{x_1, \ldots, x_n\}$ and clauses $C = \{c_1, \ldots, c_m\}$. If a clause $c_i$ contains literals $x_k$ and $\overline{x}_k$, we consider the following, equivalent problem: we add a new variable $z_{i,l}$ and instead of clause $c_i$ we add two clauses $c'_i := c_i - z_{i,l}$ and $c_{i,l} := \overline{z}_{i,l} \lor \overline{c}_i$. If the original SAT instance has a solution, setting $z_{i,l} = 1$ 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.

Now let $X, C$ denote an instance of a NM-SAT problem with variables $X = \{x_1, \ldots, x_n\}$ and clauses $C = \{c_1, \ldots, c_m\}$ and let $g : C \rightarrow \{U, N\}$ 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:

$$V := \{r\} \cup \{t, f\} \cup \{x^i | x_i \in X\} \cup \{c^l | c_j \in C\},$$

$$T_r^1 := \{(t, r_1), (f, r_1)\} \cup \{(x^i, r) | x_i \in X\} \cup \{(c^l, t) | c_j \in C, g(c_j) = U\} \cup \{(c^l, f) | c_j \in C, g(c_j) = N\},$$

$$E \setminus T_r^1 := \{(t, r_2), (f, r_2)\} \cup \{(x^i, t), (x^i, f) | x_i \in X\} \cup \{(c^l, x^i) | c_j \in C, x_i \lor \overline{x}_i \lor r_i \in c_j\}.$$
(→) To prove the other direction, let \( T'_2 \) be a spanning tree in \( E \) node-redundant with \( T'_1 \). Since only edges \((t, r)_2\) and \((f, r)_2\) are incident to \( r \) in \( E \setminus T'_1 \), each path in \( T'_2 \) 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 \( P(T'_2, c) \) from a clause \( c \) 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 Lendőlet 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.

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.

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.

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.