Scalable and Efficient Multipath Routing: Complexity and Algorithms

János Tapolcai*, Gábor Rétvári*, Péter Babarczi*, Erika R. Bérczi-Kovács†, Panna Kristóf†, Gábor Enyedi‡
*MTA-BME Future Internet Research Group, Dept. of Telecommunications and Media Informatics,
Budapest University of Technology and Economics (BME), {tapolcai, retvari, babarczi}@tmit.bme.hu
†Department of Operations Research, Eötvös University, Budapest, {koverika, kristof}@cs.elte.hu
‡TrafficLab, Ericsson Research, Hungary, gabor.sandor.enyedi@ericsson.com

***Abstract*—A fundamental unsolved challenge in multipath routing is to provide disjoint end-to-end paths, each one satisfying certain operational goals (e.g., shortest possible), without overwhelming the data plane with prohibitive amount of forwarding state. In this paper, we study the problem of finding a pair of shortest 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 underlying mathematical problem is NP-complete and we present heuristic algorithms that improve the known complexity bounds from cubic to the order of a single shortest path search. Finally, 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.**

***Index Terms*—protection routing, redundant trees, independent spanning trees, not-all-equal 3SAT, minimal path length**

I. Introduction

How to provide path diversity with destination-based hop-by-hop forwarding is among those fundamental open questions in network research [1]. The practical motivations are 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 [2]–[6]. Most of the major ingredients of a multipath Internet are, by and large, in place, like sufficient path-diversity [2], [7], multipath rate-control protocols [5], a flexible data plane [8], and inter-domain multipath routing protocols [9], [10], with some even having reached large-scale test phase [11], [12]. 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 [13], while the data plane is already struggling to scale with much slower growth rate in the first place [14], [15].

In traditional IP forwarding packets are forwarded over 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.

We argue that the most important requirements a deployable *multipath routing* scheme must fulfill are the following:
Scalable: the paths are such that nodes need at most two forwarding table entries per destination. These two next-hops will be called red and blue next hops and users will be able to select between them by tagging their packets appropriately (e.g., using the Differentiated Services Code Poing (DSCP) bits in the IPv4 header).
Disjoint: the red and blue paths between any pair of nodes in the network are maximally disjoint (i.e., do not share common edges or nodes if possible [16]). This contributes to better availability and resilience against single failures [2], [4] and eliminates adverse interference between the subflows carried by those paths [5].
Efficient centralized implementation: 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[1].
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 [1], [11], [12].

This paper is dedicated to finding algorithmic techniques for disjoint multipath routing. In particular, we concentrate on the following fundamental question:
What is the price of implementing disjoint routing in a destination based hop-by-hop forwarding architecture, 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 finding 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

Research partially supported by the High Speed Networks Laboratory (HSN*Lab*) and the Hungarian Scientific Research Fund (OTKA 108947, K 109240, PD-104939). P. B. was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences (MTA). We thank our shepherd Roch Guerin for providing useful feedback that improved this paper.

[1]For simplicity, our algorithms will be centralized, which better serves the purpose of finding the best solution eliminating the potential sub-optimality introduced by distributedness; distributed versions can then be bolted on centralized algorithms [13], [17], [18] if at all necessary [8].

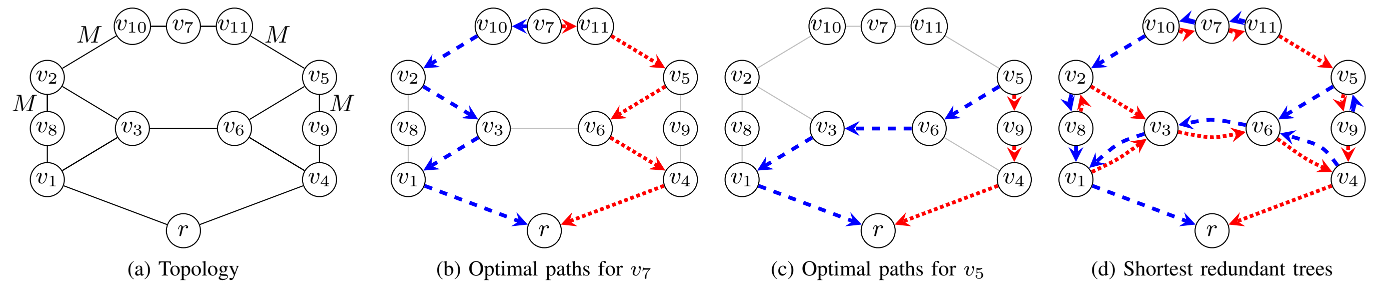

(a) Topology
(b) Optimal paths for v_7
(c) Optimal paths for v_5
(d) Shortest redundant trees

Fig. 1: An illustrative example, with root r and an arbitrary positive edge weight M. All unmarked edges are of zero weight.

wide-scale use, ranging from reliable forwarding in wireless [17] and wired networks [19], [20], robust multicasting [21], general multipath routing [13], [22] and load-balancing [23], to IP Fast ReRoute (IPFRR) protection[2] [18], [26]. 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 prior work on this subject [13], [19], [22], [27], [28], 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.

- For the first time in the literature, we settle the computational complexity of the mathematical problems related to minimum length redundant trees.
- 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.

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, so in Section IV we turn to heuristics algorithms. In Section V we present an extensive numerical study, extending to hundreds of network topologies and edge cost settings, to evaluate the performance gap. Finally, in Section VI we draw the conclusions.

II. Background

Suppose we are given a 2-connected[3], 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 weight function $w : E \to \mathbb{R}^+$ set according to some traffic engineering considerations. A *path* P in G is then an ordered set of k edges, $P = \{(s, v_1), (v_1, v_2), \ldots, (v_{k-1}, r)\}$, where s and r are called terminal nodes. For easier presentation we often assign a direction to the path 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. A weaker property if two path are *edge-disjoint* when they have no common edges. An *orientation* of the graph is an assignment of a direction to each edge, turning the initial graph into a directed graph.

A. Pairs of Shortest Disjoint Paths

Our task is now 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.

Consider the sample graph topology Fig. 1a, let r be the root and let M be some arbitrary positive weight. Then, a pair of minimum length disjoint paths 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 $O(n \log n + m)$) [29] and, so it seems, this algorithm would then readily lend itself as a multipath routing algorithm.

Alas, 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 [29].

Henceforth, we shall use the Suurballe-Tarjan algorithm to produce an "ideal" 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. Notationwise, given some root r let the optimal $v \to r$ paths (as computed by the Suurballe-Tarjan algorithm)

[2]IPFRR lends special timeliness to our work. Currently, there is a redundant-tree-based IPFRR scheme under standardization at the IETF [24] that is often criticized due to the length of the resultant detours [25]. A viable algorithm to minimize the length of the paths within the redundant trees would greatly foster standardization efforts.

[3]The removal of any single node/link does not disconnect the graph.

be denoted by $P_1^*(v)$ and $P_2^*(v)$ for each $v \neq r$. We shall denote the length of this "ideal" path-pair as

$$d_2(v) = \sum_{e \in P_1^*(v)} w_e + \sum_{e \in P_2^*(v)} w_e \ .$$

Easily $d_2(v) \geq 2d_1(v)$, where $d_1(v)$ denotes the length of the shortest path $P^*(v)$ from v to r:

$$d_1(v) = \sum_{e \in P^*(v)} w_e \ .$$

B. Packet Forwarding Along Disjoint Paths

A trivial implementation for packet forwarding along the minimum-length disjoint paths would require a next hop per source-destination pair. Next, we sketch a centralized scheme that requires only two next hops per destination.

The control plane computes a pair of rooted trees with respect to each destination node as a root and sets two forwarding table entries in each node corresponding to every pair of such trees. One entry gives the next-hop along the red tree and the other along the blue tree (for instance, node v_5 in Fig. 1d would set v_9 as the next-hop for destination r along the red tree and v_6 as next-hop along the blue tree). Within the context of this paper, we shall use the term *redundant trees* to refer to such a pair of trees. This is possible owing to the fact that both trees assign a "single" outgoing link[4] as a next-hop for each destination, which would not be the case if the forwarding paths are not lined up into trees. Note that, this scheme is fully compatible with the current IP hop-by-hop routing practice (allocating e.g., one of the DSCP bits in the header) and can also be implemented with MPLS, OpenFlow, etc. Users simply 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 tree. Packets then travel hop-by-hop to the destination along the tree assigned by the user, according to the next-hops stored at the intermediate hops. Note also that hosts can adapt a multipath rate control algorithm to actively balance their load along their paths in an end-to-end fashion [5].

C. Redundant Trees

A (spanning) arborescence $\mathcal{T}$ for some root r is a directed tree rooted[5] at r in which from any node $v \in V, v \neq r$ there is exactly one directed path from v to r. With a slight abuse of terminology, we shall henceforth refer to such arborescences simply as (rooted) trees. For a tree $\mathcal{T}$ rooted at r and any $v \neq r$, denote by $P(\mathcal{T}, v)$ the unique path in $\mathcal{T}$ from v to r. Then, redundant trees are a pair of spanning rooted trees with certain strong disjointness properties [13].

Definition 1: A pair of (spanning) trees $(\mathcal{T}_1, \mathcal{T}_2)$ with common root r is called (a pair of) *node-disjoint redundant trees* for r if for each $v \in V, v \neq r$: $i \in P(\mathcal{T}_1, v), i \neq r \quad \Rightarrow i \notin P(\mathcal{T}_2, v)$.

[4]We ignore multiple outgoing links and Directed Acyclic Graph (DAG) routing. Note that if the problem formulation is generalized to allow DAGs the optimal solutions could still be trees.

[5]Often in the literature the direction is just the opposite.

We also define a weaker form as follows.

Definition 2: A pair of (spanning) trees $(\mathcal{T}_1, \mathcal{T}_2)$ with common root r is called (a pair of) *edge-disjoint redundant trees* for r if for each $v \in V, v \neq r$: $(i,j) \in P(\mathcal{T}_1, v) \Rightarrow (i,j) \notin P(\mathcal{T}_2, v)$ and $(j,i) \notin P(\mathcal{T}_2, v)$.

Consider the *red* tree $\mathcal{T}_1$ and the *blue* tree $\mathcal{T}_2$ in Fig. 1d. Here, as $\mathcal{T}_1$ and $\mathcal{T}_2$ share a common edge $v_3 - v_6$ they are not node-disjoint redundant trees per definition. However, even though the edge $v_3 - v_6$ is used in both trees (in opposite directions), the paths themselves from each node to the root are still edge-disjoint and hence $\mathcal{T}_1$ and $\mathcal{T}_2$ qualify as edge-disjoint 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-disjoint redundant trees for any root is guaranteed to exist and it can be found in polynomial time [19], [21]. This was later reduced to linear time [30] and linear time algorithms for finding maximally edge-disjoint redundant trees were also given for the non-2-connected case [31].

D. Minimum Length Redundant Trees

What we are concerned with in this paper is for a given root r finding a pair of redundant trees $(\mathcal{T}_1, \mathcal{T}_2)$ of "minimum length". Here, length is defined as follows. For each source node v the redundant trees $\mathcal{T}_1$ and $\mathcal{T}_2$ define two disjoint paths towards the root, denote these as $P(\mathcal{T}_1, v)$ and $P(\mathcal{T}_2, v)$. The length of each path is then $c_{\mathcal{T}}(v) = \sum_{e \in P(\mathcal{T}, v)} w_e$ and the *(total) length of the redundant tree pair* is denoted by $c(\mathcal{T}_1, \mathcal{T}_2) = c(\mathcal{T}_1) + c(\mathcal{T}_2)$, where $c(\mathcal{T}) = \sum_{v \neq r} c_{\mathcal{T}}(v)$.

Our task is now to find a pair of trees that minimize the total path length. It turns out that the trees given 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 (see also [22] and [13]).

Definition 3: Minimum Length Redundant Trees (MLRT): given a graph G, weights w, root r, and positive integer k, is there a pair of redundant trees $\mathcal{T}_1$ and $\mathcal{T}_2$ so that $c(\mathcal{T}_1, \mathcal{T}_2) \leq k$?

Our main concern here is the optimization version of this problem, where the task is to minimize $c(\mathcal{T}_1, \mathcal{T}_2)$. For this version, an optimal Integer Linear Program (ILP) with exponential worst-case solution time along with a heuristics with $O(n^3)$ running time were given in [13], [22]. In Section IV, we shall improve the running time to the same as Dijkstra's algorithm and the Suurballe-Tarjan algorithm, $O(n \log n + m)$.

So far, we have deliberately avoided to fix the disjointness requirement for the redundant trees, i.e., whether we want the node-disjoint (as of Definition 1) or the edge-disjoint (as of Definition 2) property. The reason is that, as we show below, the latter can be formulated in terms of the former[6].

Lemma 1: The problem of finding edge-disjoint redundant trees can be reduced to the problem of finding node-disjoint

[6]Surprisingly, the situation is just the reverse for directed graphs, where the edge-disjoint case is the more generic one.

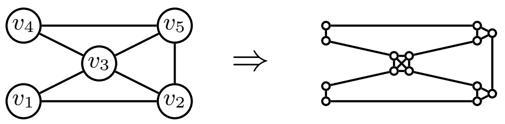


Fig. 2: An example of the graph transformation to transform an edge-disjoint redundant trees problem onto an equivalent node-disjoint redundant trees problem.

redundant trees such that the length of the paths in the trees remains the same.

Proof: Given an instance of the edge-disjoint redundant trees problem, we convert it to an equivalent instance of the node-disjoint redundant trees problem by the following graph transformation. Every node v is replaced with a complete subgraph of $N(v)$ nodes, where $N(v)$ denotes the number of adjacent nodes to v. Furthermore, each adjacent node of v is connected to a different node of the complete subgraph corresponding to v (see Fig. 2). The cost of the new edges is zero. Pick an arbitrary node as the root in the transformed graph that corresponds to r in the original graph. One can verify that the cost $c(\mathcal{T}_1, \mathcal{T}_2)$ remains the same in the transformed graph, while the paths in the trees $P(\mathcal{T}_1, v)$ and $P(\mathcal{T}_2, v)$ may not share common nodes other than v and r. ■

For the rest of the paper, we focus on the more generic node-disjoint case.

E. Performance Penalty

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 $(\mathcal{T}_1, \mathcal{T}_2)$ will be higher than the optimum: $c(\mathcal{T}_1, \mathcal{T}_2) \geq \sum_{v \neq r} d_2(v)$. This is the price we pay for scalability. Our aim in this paper is to analyze this price, as measured by the following metrics.

For a graph G with weights w and root node r, the *(path) length gap* is defined as $c_{\mathcal{T}_1}(v) + c_{\mathcal{T}_2}(v) - d_2(v)$. We say that a node v is *perfectly covered* by $\mathcal{T}_1$ and $\mathcal{T}_2$ if $c_{\mathcal{T}_1}(v) + c_{\mathcal{T}_2}(v) - d_2(v) = 0$. Furthermore, define the *(path) length ratio* as:

$$\eta(G, r) = \min \frac{c(\mathcal{T}_1, \mathcal{T}_2)}{\sum_{v \neq r} d_2(v)} \text{ over all choices } (\mathcal{T}_1, \mathcal{T}_2). \quad (1)$$

For the case of our sample graph, the redundant trees on Fig. 1d yield the total cost of $16M + 62$, while $\sum_{v \neq r} d_2(v) = 10M + 68$ and so $\eta(G, r) \simeq 1.6$. A simple calculation will lead to the following observation.

Lemma 2: Consider the graph G obtained from the sample graph in Fig. 1a by replacing each of v_8, v_{10}, v_{11}, and v_9 by a chain of M new nodes. Then, $\lim_{M \to \infty} \eta(G, r) = {}^5\!/_3$.

See the proof in the Appendix.

Curiously, so far we have not been able to find any graph for which the length ratio were larger. In all our theoretical investigations, evaluations on famous difficult graph instances, and all our simulations on hundreds of graphs with widely varying weight functions (see Section V), we have not found even a single instance where the ratio were above ${}^5\!/_3$. This is highly unexpected as, for the first sight, the restriction that paths must reside in two trees looks daunting. It seems, however, that in reality the penalty for scalable multipath routing might not be that large. The rest of the paper is dedicated to making this claim explicit.

III. Complexity

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 [27], [28] study the task to find two redundant spanning trees of minimum stretch, but for the all-pairs case (i.e., when the trees are not rooted). Another version where the total cost of the edges in the redundant trees (in contrast to the length of the paths) is to be minimized is examined in [20], [30]. The exact problem formulation for *MLRT* appears in [13], [22], 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. 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 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 [32].

IV. Heuristic Algorithms

In this section we provide several heuristics for solving the problem. We take a novel design concept for the heuristics that takes the optimal minimum cost disjoint paths obtained by Suurballe-Tarjan algorithm [29] and converts 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 Suurballe-Tarjan algorithm [29].

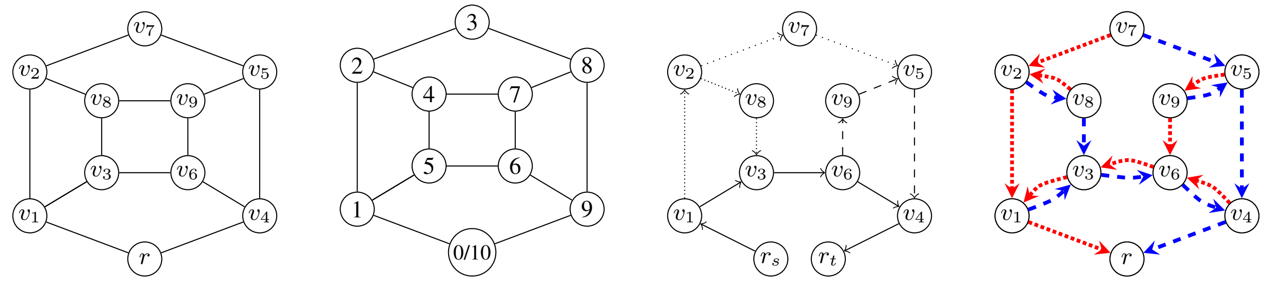
Observation 1: Let A be the union of the directed edges in the minimum cost disjoint paths from every node to a single root. Every node other than the root is of out-degree 2 in A.

This means A is a sparse subgraph of G and so there is a hope that it can be partitioned into two redundant trees.

In [22] a simple heuristic, called the *BR algorithm*, was proposed for solving *MLRT*. The heuristic requires $O(n^2 S)$ steps, where n is the number of nodes and m is the number of edges in G and $S = O(n \log n + m)$ is the complexity of a shortest path search with Dijkstra's algorithm. In this section, we improve the computation time to $O(n^2)$ and we also give an even faster $O(S)$ algorithm without major performance hit.

A. St-Numberings and Ear-Decompositions

Most redundant tree algorithms revolve around the concepts of st-numbering and ear-decomposition [13], [16], [18]–[23], [30], [31]. An *st-numbering* is a numbering of the nodes in G by $1, \ldots, n$, such that the node numbered with "1" and "n" are adjacent and each remaining node v is adjacent to two nodes



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

Fig. 3: A simple undirected graph demonstrating the different techniques used in the MLRT algorithms.

x and y such that for the respective st-numbers $x < v < y$ (see Fig. 3b).

Lemma 3: Given an st-numbering of G, two redundant trees in G can be computed in linear time.

Proof: First we split the root node r into r_s and r_t, such that node r_s is labeled "0" and it is adjacent to only the node having label "1", while r_t is labeled "n" and it is adjacent to the rest of the neighbors of r. Next, we orient the edges of G such that each (u, v) edge is directed $u \to v$ if $u < v$, and $u \leftarrow v$ otherwise. Let $\mathcal{T}_1$ be an out-tree from r_s and $\mathcal{T}_2$ be an in-tree towards r_t. These can be found in linear time. Clearly, every node can be reached from r_s and every node can reach r_t. The path $P(\mathcal{T}_1, v)$ will traverse nodes with label at most v, while $P(\mathcal{T}_2, v)$ will traverse nodes with label at least v; therefore they are disjoint paths. ■

This means that an st-numbering is sufficient to find two redundant trees. The question is, how to find such an st-numbering. This can be done by a method called *ear-decomposition*, which is a graph reduction technique to decompose any 2-connected graph G into a sequence of 2-connected graphs $G_0 \subset G_1 \subset \cdots \subset G_k$. G_0 is composed of a single root r and G_k has all nodes of G, i.e., $V(G_k) = V(G)$. For each $i = 1, \dots, k : G_i \equiv G_{i-1} \cup P_i$, where P_i is a path of length at least 2 edges, called an *ear*, between the root or two distinct[7] nodes $x_i, y_i \in V(G_{i-1})$, $V(P_i) \cap V(G_{i-1}) = \{x_i, y_i\}$ and $x_i \neq y_i$ except if $x_i = y_i = r$. For the graph in Fig. 3a, a possible ear-decomposition would consist of the following ears: $P_1 = r\text{-}v_1\text{-}v_3\text{-}v_6\text{-}v_4\text{-}r$, $P_2 = v_1\text{-}v_2\text{-}v_8\text{-}v_3$, $P_3 = v_6\text{-}v_9\text{-}v_5\text{-}v_4$, and $P_4 = v_2\text{-}v_7\text{-}v_5$.

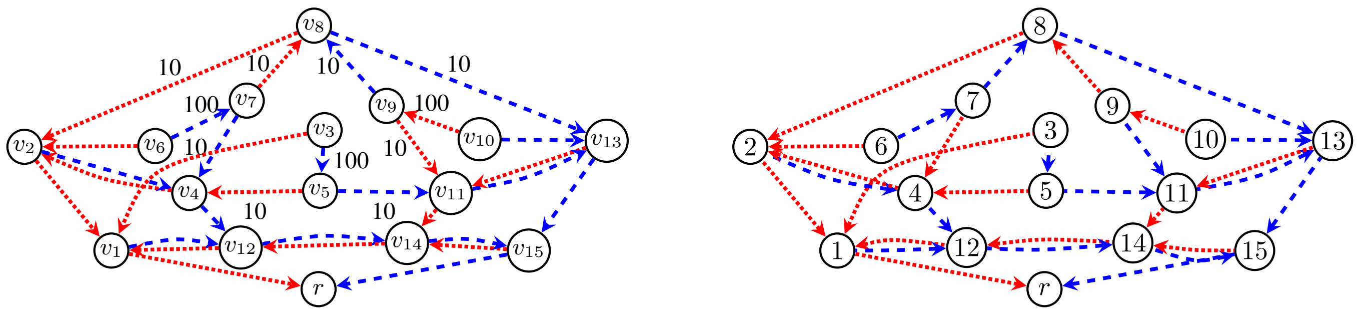
The idea is to maintain an st-numbering through the graph sequence $G_0 \subset G_1 \subset \cdots \subset G_k$. The induction is that we have an st-numbering for G_{i-1} and we add an ear $P_i = x_i\text{-}v_1\text{-}\dots\text{-}v_j\text{-}y_i$. If $x_i > y_i$ we label the nodes as $x_i > v_1 > \cdots > y_i$, otherwise $x_i < v_1 < \cdots < y_i$. Here a common technique is to allow fractional labels, and thus no relabeling is needed in the inductive step.

A slightly weaker notion than st-numbering is an *st-orientation* (or bipolar orientation) of G, an assignment of an orientation to each edge of G, so that after splitting the root node into r_s and r_t such that node r_s has all the out-arcs and r_t all the in-arcs of the root r, the resultant graph G^D is a DAG, r_s is the only node with zero in-degree, and r_t is the only node with zero out-degree (see Fig. 3c). Note that any topological order of G^D is an st-numbering, which is sufficient to compute two redundant trees by Lemma 3. Interestingly, *not every redundant tree pair can be obtained by an st-orientation* (see Fig. 4 for a counter-example).

This observation means that the optimal redundant trees may not be found with searching for the best st-numbering and ear-decomposition. Essentially all existing heuristic algorithms use (some variant of) the above algorithmic framework: build an ear-decomposition and maintain an st-orientation thereof. The heuristic in [19] selects the ears basically randomly, which was later improved to a greedy strategy to minimize some intuitive path-length metrics in [23] and [22]. In particular, the BR algorithm [22] selects an ear so as to minimize the path length after the ear is added, which in worst case requires an all-pairs shortest path calculation, in $O(n^2 S)$ steps overall. St-numbering (or complete order) is used in [20], [30], [31], while st-orientations (or the equivalent notion of partial orders) in [18], [22], [23]. The trade-off between the two is the usual time vs. performance type; complete orders yield somewhat longer paths but can be maintained in $O(1)$ (although correct implementation is far from trivial [16]); while partial orders allow more liberty for the st-orientation and deliver shorter paths, but can be maintained only in $O(n)$ [13], [22].

Based on the above classification in the rest of the section we propose a series of heuristics that take the optimal minimum cost disjoint paths obtained by Suurballe-Tarjan algorithm [29] and convert them into redundant trees. The first heuristic does this conversion in linear time by maintaining an st-numbering. Such an approach has overall time complexity same as a single shortest path calculation. 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. The second heuristic, in Section IV-C, has slightly shorter paths by maintaining an st-orientation (partial order) at the price of increasing the running time to $O(n^2)$. Finally, in Section IV-D we give a near-optimal algorithm that, despite its exponential worst-case running-time, gives very high quality redundant trees in very short time even for large graphs for which the ILP of [22] does not terminate at all. For simplicity, in the rest of this section we concentrate on input graphs G that are 2-node-connected.

[7]Recall, it is sufficient to deal with the node-disjoint case by Lemma 1.



(a) Optimal redundant trees. Unlabeled edges are of unit cost. Every node is perfectly covered, except v_9 and v_7 with path length gap 1. There is no equivalent st-numbering because of the cycle v_8, v_9, v_{11}, v_5, v_4 and v_7.

(b) Optimal solution over st-orientations. Nodes are labeled as of the st-numbering. Observe that at nodes v_{10} and v_6 the path length gap is 8.

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

Algorithm 1: A fast *MLRT* algorithm

```
Input: Undirected graph $G(V, E)$, root node $r$, edge weights $w$
begin
1    Find disjoint paths $P_1^*(v)$ and $P_2^*(v)$ for $v \in V$
2    Sort $v \in V$ in ascending order of distance $d_2(v)$ into $V^*$
3    $i := 1$; $G_0 := \{r\}$; $G_0^D := \{s, t\}$
4    for $v \in V^*$ do
5        $P_i = x_i\text{-}v_1\text{-}\ldots\text{-}v_j\text{-}y_i$ where $x_i$ ($y_i$) is the first node
                 of $P_1^*(v)$ ($P_2^*(v)$, resp.) in $G_{i-1}$
6        if $x_i < y_i$ then
7            add $x \leftarrow v_1 \leftarrow \cdots \leftarrow v_j$ to $\mathcal{T}_1$ and
                 $v_1 \rightarrow \cdots \rightarrow v_j \rightarrow y$ to $\mathcal{T}_2$
         else
8            add $y \leftarrow v_j \leftarrow \cdots \leftarrow v_1$ to $\mathcal{T}_1$ and
                 $v_j \rightarrow \cdots \rightarrow v_1 \rightarrow x$ to $\mathcal{T}_2$
9        $V^* = V^* \setminus \{u \in P_i\}$; $G_i = G_{i-1} \cup P_i$
10       $i = i + 1$
```

B. *A Fast* MLRT *Algorithm*

Here, the main idea is to let the Suurballe-Tarjan algorithm drive the augmentation of the ear-decomposition. Suppose we are given a weighted undirected graph G and a root r. First, for each node $v \neq r$ we compute a pair of minimum length disjoint paths $P_1^*(v)$ and $P_2^*(v)$ to r using a single run of the Suurballe-Tarjan algorithm. This can be done in $O(S)$ steps [29]. Next, we sort the nodes in V in ascending order of their $d_2(v)$ distance. This can be done in $O(n \log n)$ steps. Denote the sorted list by V^*. In the third step, we iterate through V^* and we generate the ear-decomposition $G_0, \ldots, G_k$, the complete order, and the redundant trees along the way.

The first ear consists of the root: $G_0 = \{r\}$. Then, in the i-th iteration we select the node v in $G \setminus G_{i-1}$ that has the smallest distance $d_2(v)$ and we add the ear through v to G_{i-1}^D. Let x_i be the first node along $P_1^*(v)$ that is already part of G_{i-1} and likewise let y_i be the first node of G_{i-1} along $P_2^*(v)$. Note that if $x_i \neq r$ then $x_i \neq y_i$. Construct the ear P_i as the concatenation of path segments $x_i \rightarrow v$ of $P_1^*(v)$ and $v \rightarrow y_i$ of $P_2^*(v)$ and denote this ear by $P_i = x_i\text{-}v_1\text{-}\ldots\text{-}v_j\text{-}y_i$.

We still need to decide the orientation for the new ear and update G_i^D and the complete order. There are two cases: if $x_i < y_i$ then set the complete order accordingly, i.e., $x_i < v_1 < \ldots < v_j < y_i$, and finally add the "forward" chain to $\mathcal{T}_1$ and the "backward" chain to $\mathcal{T}_2$. If $x_i > y_i$, on the other hand, the procedure is executed in the opposite direction with nodes y_i and x_i.

Algorithm 1 gives the pseudo-code. Here, instead of an st-numbering the complete order is represented using a simple node-potential [30] (see also [16]), while V^* can be maintained with simple node marking (observe that each node is visited at most once). The complexity is dominated by the Suurballe-Tarjan pass performed in the first phase, fixing the running time at $O(S)$ as required.

C. *An Improved* MLRT *Algorithm over Partial Orders*

It has been observed a number of times previously that a complete order is an overkill for maintaining the st-orientation. In fact, if we allow certain node-pairs *not* to be ordered with respect to each other then this will open up more options to set the orientation of a new ear in some cases, which we can often exploit to reduce the length of the paths within the trees. Unfortunately, the running time of the algorithm will be somewhat higher in return.

In this case the operation "set $x < y$" means adding an arc from x to y and the query "$x < y$" is substituted with checking whether there is a directed path from x to y in G^D. But what if we find – when trying to decide the orientation of a newly found ear – that the endpoints x_i and y_i are not ordered (i.e., no directed path exists between them)? In such cases, we are free to choose the orientation arbitrarily. Consider we add a new node v_{10} connected to v_3 and v_7 in Fig. 3, and suppose we add an ear v_3-v_{10}-v_7. Note that there is no $v_3 \rightarrow v_7$ directed path in the DAG of Fig. 3c nor the other way around, so any orientation will yield an acyclic graph. If we set $v_3 \rightarrow v_{10} \rightarrow v_7$ then the paths of v_{10} would be $3 + 4 = 7$ hops long while $v_7 \rightarrow v_{10} \rightarrow v_3$ would result $4 + 4 = 8$ hop paths, so we would rather go with the first choice.

Unfortunately, the complete order would encode just the wrong orientation in this case. Hence, we apply a simple optimization here; calculate $c(P(\mathcal{T}_1, x_i)) + c(P(\mathcal{T}_2, y_i))$ and $c(P(\mathcal{T}_2, x_i)) + c(P(\mathcal{T}_1, y_i))$ and choose the orientation that minimizes this metric. This can be done by adding a third

TABLE I: Results on real network topologies. Columns mark the parameters of the input graphs (name, number of nodes and edges); for the edge disjoint case the average path length ratio (η) for different algorithms; the maximum path length gap suffered by any node ($\max_{\forall v}\{c_{\mathcal{T}_1}(v)+c_{\mathcal{T}_2}(v)-d_2(v)\}$); and the average length of the shorter and the longer path along the redundant trees compared to the absolute shortest path length for the ML$_{PO}$ algorithm ($\sum_{\forall v}\frac{\min\{c_{\mathcal{T}_1}(v),c_{\mathcal{T}_2}(v)\}}{(|V|-1)d_1(v)}$,$\sum_{\forall v}\frac{\max\{c_{\mathcal{T}_1}(v),c_{\mathcal{T}_2}(v)\}}{(|V|-1)d_1(v)}$); and for the node-disjoint case the average path length ratio for the ML$_{PO}$ and MRT algorithms (η) . Note that the results 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			Edge-disjoint								Avg. path length		Node-disjoint	
			Average path length ratio				Max length gap at a node				Edge-disjoint ML$_{PO}$		Avg. path length ratio	
Name	$\|V\|$	$\|E\|$	ML	ML$_{PO}$	ML$_{NO}$	BR [22]	ML	ML$_{PO}$	ML$_{NO}$	BR [22]	the shorter	the longer	ML$_{PO}$	MRT [24]
Germany	17	25	3.39%	0.82%	0.51%	1.83%	12.6%	5.0%	4.7%	7.7%	7.41%	59.2%	6.32%	34.06%
BtEurope	17	30	3.57%	0.01%	0.17%	0.01%	15.5%	0.01%	4.9%	0.1%	37.20%	302.3%	3.24%	71.70%
AS6461	17	37	4.32%	1.93%	0.71%	1.53%	17.7%	8.7%	9.1%	10.0%	10.90%	78.2%	5.09%	46.63%
InternetMCI	18	32	1.48%	0.83%	0.67%	0.83%	10.2%	7.2%	9.5%	7.2%	3.45%	44.8%	1.98%	63.27%
AS1755	18	33	5.32%	2.51%	1.35%	3.52%	16.5%	9.2%	12.8%	12.0%	4.23%	73.2%	8.09%	42.62%
ChinaTelc	20	44	0.19%	0.15%	0.10%	0.15%	3.6%	2.9%	5.1%	2.9%	20.00%	202.0%	1.79%	118.90%
AS3967	21	36	4.90%	1.20%	1.53%	1.15%	15.1%	3.6%	10.8%	3.5%	4.17%	65.3%	6.14%	32.89%
AT&T	22	38	2.87%	2.37%	1.02%	2.06%	12.9%	10.5%	12.3%	6.8%	4.00%	52.0%	6.45%	38.36%
BICS	27	42	13.90%	1.96%	1.96%	2.69%	41.3%	9.8%	15.5%	11.0%	15.10%	129.2%	9.37%	94.82%
AS3257	27	64	3.35%	0.65%	0.25%	0.69%	17.9%	6.4%	4.7%	6.5%	6.80%	74.7%	5.14%	22.11%
AS1239	30	69	3.39%	2.01%	0.92%	2.29%	19.6%	9.5%	13.9%	10.0%	4.72%	41.7%	9.20%	61.02%
Arnes	31	47	1.91%	0.15%	0.15%	0.15%	17.5%	2.9%	7.2%	2.8%	11.90%	90.1%	0.41%	58.73%
Geant	31	49	4.05%	0.90%	0.84%	0.97%	49.6%	19.0%	20.4%	21.0%	3.45%	31.0%	5.15%	33.33%
Italy	33	56	2.80%	1.65%	1.50%	1.77%	14.5%	8.6%	17.4%	9.1%	5.56%	47.2%	4.81%	27.58%
BtNAmerica	36	76	9.68%	1.43%	1.32%	1.38%	43.0%	12.0%	57.5%	12.0%	17.40%	193.5%	4.68%	126.51%
BellCanada	39	55	8.93%	2.71%	1.07%	4.24%	32.0%	12.2%	10.9%	18.0%	75.30%	257.9%	14.47%	38.33%
Germany	50	88	8.77%	3.33%	2.62%	3.10%	30.4%	17.6%	25.8%	17.0%	5.21%	39.3%	15.00%	71.03%
Deltacom	103	151	10.90%	3.42%	3.32%	3.74%	41.2%	18.6%	23.8%	19.0%	44.00%	146.6%	10.96%	64.88%

branch to the conditional expression on line 6 of Algorithm 1 to handle the above "endpoints of the new ear are not ordered" case. The optimization incurs a factor-$O(n)$ complexity tax, setting the overall complexity to $O(n^2)$.

D. A Near-optimal MLRT *Algorithm*

Finally, we set off to bring the time–efficiency trade-off to the other extreme and find the best possible redundant tree pair (i.e., the one with shortest length). Unfortunately, during the numerical evaluations we found that the optimal *MLRT* ILP [22] very often fails to produce meaningful results in reasonable time. Therefore, our final algorithm, while still a heuristics, will aim for a near-optimal solution at the price of increased (although in practice still manageable) running time.

Inspired by Lemma 3, we formulate *MLRT* as an optimization problem to find an st-orientation where the average depth of the in-tree and out-tree from the root is minimal. The idea is to cut down the search space to such an extent that we can use brute-force search to find the best solution.

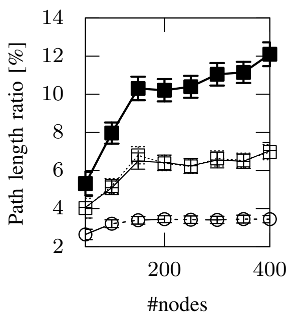
In the first step, we orient the edges according to the Suurballe-Tarjan algorithm. After the first step, we will have a mixed graph, where some edges oriented and others are not (they are oriented in both directions). The idea is to run Algorithm 1 but orienting the edges with a different concept. We decide a direction for each ear, but all edges along paths $P_1^*(v)$ and $P_2^*(v)$ are oriented in a single pass, not just those along the new ear. Note that some edges might be oriented in both directions eventually; we call such edges *conflicted edges*. In each step the direction of the ear is selected such that a minimal number of edges becomes conflicted.

At the end of this process we set the conflicted edges undirected to obtain a mixed graph with some directed and undirected edges. Let E_C denote the number of undirected edges; note that the number of such undirected edges $|E_C|$ is typically small. Next, we execute a brute-force search to obtain the redundant trees: we orient the undirected edges in all $2^{|E_C|}$ combinations and we run Dijkstra's shortest path algorithm to find the shortest in-tree and out-tree from the root node that has a minimal average depth to every other node.

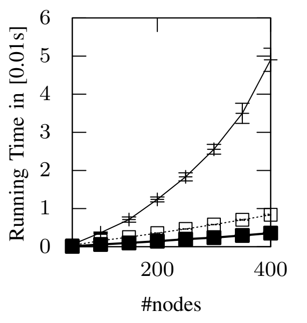
V. Numerical evaluation

We carried out an extensive numerical evaluation in order to gain further insights into the focal question of this paper: How close redundant trees can approximate the length of the shortest possible disjoint paths, as obtained by the Suurballe-Tarjan algorithm? In other words, how much penalty do we pay for scalability in disjoint multipath routing, in terms of the *path length ratio* metric η as of (1)? In the rest of this section we concentrate on the edge-disjoint case, as this is more relevant to our subject of interest, multipath-routing. Note that, the algorithms themselves are for the node-disjoint case, but we used them on the transformed graphs (as defined in Lemma 1) to obtain results for the edge-disjoint version.

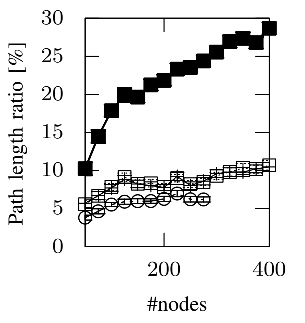
The results were computed with the following algorithms. First, we used the BR algorithm (BR) from [22]. To cut down the complexity of the algorithm we applied the optimization that, when adding a new ear, we set the distances from the previous iteration to warm-start the all-pairs shortest path computation. We also experimented with the new algorithms presented in Section IV, in particular, the fast MLRT heuristics



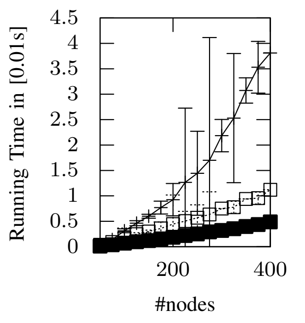
(a) Length ratio in small world graphs



(b) Running time in small world graphs



(c) Length ratio in random planar graphs



(d) Running time in planar graphs

Fig. 5: Results on random small-world and planar graphs for the BR-algorithm ($+$), the fast MLRT algorithm ML with complete order (■) and with partial order (ML$_{PO}$, □), and the (assumed) baseline with the heuristic ML$_{NO}$ ($\circ$).

as of Algorithm 1 with complete order (ML) and the one with partial order (ML$_{PO}$). We also implemented the ILP from [13] to seed the comparisons with a base value. However, even in medium size problem instances the ILP could not be solved to optimality, thus we applied the algorithm described in Section IV-D instead (ML$_{NO}$). Note that the ML$_{NO}$ heuristic found the optimal solution for those problem instances whenever the ILP successfully terminated. We also implemented the MRT algorithm [26], currently under standardization at the IETF to drive the MRT IP Fast ReRoute scheme. Note that this algorithm is intended only for the node-disjoint version of the problem, so we shall use it as a reference below for this case.

We have examined a broad family of graphs, from real ISP network topologies [33], [34] and small-world random graphs [35] to random planar graphs, over widely varying edge cost settings including inferred costs [34] and uniform random costs. All in all, we evaluated more than $20,000$ individual problem instances, including $4,000$ small-world and $11,000$ random planar problem instances. Hence, the results are of high statistical significance. Strikingly, amongst these $20,000$ instances not in a single case we found the path length ratio to grow beyond 32% (with a mean of 4.3%), which is still less than half of the hypothesized maximum as of Lemma 2.

Table I shows the results obtained on real world topologies (note that the results are given in percentages). For small worlds and for random planar graphs the path length ratio and running time for the different algorithms are given in Fig. 5.

The most important observations are as follows. First, the path length ratio between redundant trees and optimal disjoint paths seems very small: just a couple of percents for real graphs (half of the real networks have ratio below 1%), roughly 3% for small-worlds, and about 5% for random planar graphs. This was a really unanticipated outcome, as initially we expected the requirement that paths must form a pair of trees to be overly restrictive. This did not end up being the case, suggesting that *scalable multipath routing might not cause major performance hit for operators*. Interestingly, the optimal disjoint paths ($d_2(v)$) were in turn only about one and a half times as long as the absolute shortest paths ($d_1(v)$) obtained by Dijkstra's algorithm (see, e.g., the results

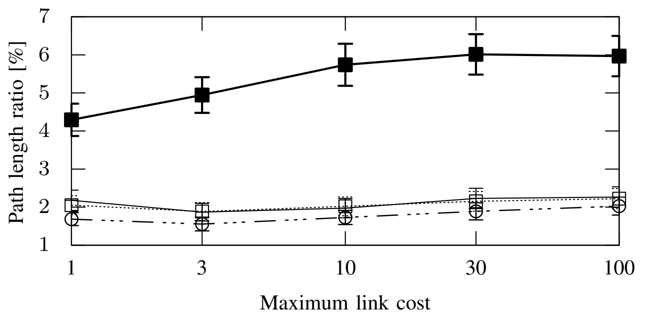


Fig. 6: Path length gap near random edge cost settings of increasing variance in real topologies with the BR ($+$), the ML (■), ML$_{PO}$ (□) algorithms and the heuristic ML$_{NO}$ ($\circ$).

for real instances in Table I), indicating that *the penalty for disjoint multipath routing itself is also small*. The maximum performance hit suffered by a single node was 100%, meaning that there was a node whose two paths along the redundant trees were twice as long as the shortest disjoint path-pair.

Of the algorithms, ML$_{PO}$ produced very much the same results as BR but proved much faster in practice (theoretically too, by, recall, a factor of $O(n \log n)$) and both produce extremely high quality paths with about 1–8% length ratio (0–3% in real networks). Even the very fast (linear average time) ML heuristic was within 10–25% of the absolute optimum in terms of the path length ratio, suggesting that this algorithm is very well suited for performance-oriented applications. Finally, we found that the results are very robust against parameter settings, in that the ratio does not seem to vary with, say, the average nodal degree (results not included here) or the edge costs (see Fig. 6).

VI. Conclusions

Transition to multipath routing in the Internet 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]–[6]. In the paper, we laid down four major requirements against a disjoint multipath routing algorithm, namely scalability, path disjointness, computational complexity, and, last but not least, path length.

An execution of the Suurballe-Tarjan algorithm [29] 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 clearly negative: we have shown that scalable disjoint multipath routing is intractable. And performance-wise, the naive 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 clearly viable within the same complexity as standard control plane operations (like Dijkstra's shortest path algorithm) for about 25% performance penalty. Furthermore, if we are willing to take on a minor complexity hit (jumping to $O(n^2)$ complexity, the same as that of a less clever Dijkstra implementation) then the path length penalty reduces to just 1–8% of the optimal case (or 1–5% with yet another jump of complexity)! We believe that these results clearly support our main conclusion that, in contrast to expectations, disjoint multipath routing is indeed feasible and beneficial.

A last missing point seems to be an analytical bound on the path length ratio. Is the ratio $5/3$ observed in Lemma 2 maximal? Our future plans also include decentralising the scheme and adapting and plugging our algorithms into real-world applications, most probably replacing [26] in the MRT IP Fast ReRoute framework [24] with our implementations. Note that extending disjoint multipath routing for multiple failures is very challenging, because it requires solving the following long-standing open algorithmic problems: find redundant trees with more than four colors [36], and compute more than two disjoint paths with the same complexity as shortest path search [29].

References

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

[2] 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.

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

[4] 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.

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

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

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

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

[9] W. Xu and J. Rexford, "MIRO: Multi-path interdomain routing," in *SIGCOMM*, 2006, pp. 171–182.

[10] I. Ganichev, B. Dai, P. B. Godfrey, and S. Shenker, "YAMR: Yet another multipath routing protocol," *SIGCOMM CCR*, vol. 40, no. 5, pp. 13–19, 2010.

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

[12] 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.

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

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

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

[16] G. Enyedi, G. Rétvári, and A. Császár, "On finding maximally redundant trees in strictly linear time," in *IEEE ISCC*, 2009, pp. 206–211.

[17] 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.

[18] 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.

[19] M. Médard, S. G. Finn, and R. A. Barry, "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.

[20] 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.

[21] 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.

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

[23] 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.

[24] A. Atlas, R. Kebler, M. Konstantynowicz, G. Enyedi, A. Császár, and M. Shand, *An Architecture for IP/LDP Fast-Reroute Using Maximally Redundant Trees*, Internet-Draft, Standards Track status Std., 2014.

[25] ——, "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.

[26] G. Enyedi, A. Császár, A. Atlas, C. Bowers, and A. Gopalan, *Algorithms for computing Maximally Redundant Trees for IP/LDP Fast-Reroute*, Internet-Draft, Standards Track status Std., 2014.

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

[28] T. Hasunuma, "On edge-disjoint spanning trees with small depths," *Information Processing Letters*, vol. 75, no. 12, pp. 71–74, 2000.

[29] 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.

[30] 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.

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

[32] 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.

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

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

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

[36] A. Bhalgat, R. Hariharan, T. Kavitha, and D. Panigrahi, "Fast edge splitting and edmonds' arborescence construction for unweighted graphs," in *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2008, pp. 455–464.

Appendix

Proof of Lemma 2: First, we consider the graph G in Fig. 1a and we show that $\eta(G,r) \to 1.6$ if M grows large enough. Then, for the modified graph of Lemma 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 $\eta(G,r) = \frac{20M^2+26M+32}{12M^2+30M+36}$ and so $\eta \to \frac{5}{3}$ as M tends to ∞. The details are omitted for brevity.

Consider the graph G in Fig. 1a, let edge weights be 1 except on edges (v_2,v_8), (v_{10},v_2), (v_{11},v_5), (v_5,v_9) that have weight 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 is greater than $1.6-\epsilon$. 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 $\mathcal{T}_1$ and $\mathcal{T}_2$. Assume indirectly that there exist shorter redundant trees F_1 (blue) and F_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 $\mathcal{T}_1$ and $\mathcal{T}_2$ only nodes v_{10}, v_7, v_{11} have longer paths from r than in G, the sum of the length of their corresponding path must be shorter in F_1 and F_2. Assume that the blue path is shorter in F_1 than in $\mathcal{T}_1$. 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. ■

Proof of Theorem 1: Let X, C denote an instance of a NAE-3SAT problem with variables $X = \{x_1, \dots, x_n\}$ and clauses $C = \{c_1, \dots, c_m\}$. We build a graph $G = (V,E)$ belonging to this instance as follows:

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

where x_t^i and x_f^i correspond to the true and false assignment of variable x_i, respectively. The weight $w(e) := 1, \forall e \in E$. For every node v, let $d(v)$ denote the minimal length of two edge-disjoint $v-r$-paths. Note that $d(v) = 3$ for nodes $v = x_t^i$ or $v = x_f^i$, and 4 for nodes $v = c^j$ and this shortest pair of edge-disjoint paths is unique for every node.

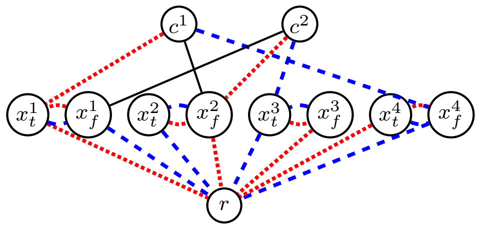


Fig. 7: The polynomial-time tranformation of the NAE-3SAT instance $(X = \{x_1, x_2, x_3, x_4\}, C = \{c_1 = \{x_1, \overline{x}_2, \overline{x}_4\}, c_2 = \{\overline{x}_1, \overline{x}_2, x_3\}\}, k = 2)$. A NAE truth assignment is $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1$.

Claim 1: 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 $c(\mathcal{T}_1, \mathcal{T}_2) = \sum_{v \in V} d(v)$.

Proof: Let $a : X \to \{t,f\}$ 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 literal that evaluates to true 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 $\mathcal{T}_1$ and $\mathcal{T}_2$:

$$\mathcal{T}_1 = \{(x^j_{a(x_j)}, r), (x^j_{\overline{a}(x_j)}, x^j_{a(x_j)}) | x_j \in X\} \cup \{(c^i, x^{t(i)}_{a(x_{t(i)})}) | c_i \in C\}$$

$$\mathcal{T}_2 = \{(x^j_{\overline{a}(x_j)}, r), (x^j_{a(x_j)}, x^j_{\overline{a}(x_j)}) | x_j \in X\} \cup \{(c^i, x^{f(i)}_{\overline{a}(x_{f(i)})}) | c_i \in C\}$$

These trees are redundant and minimum length, as $(c_i, x^{t(i)}_{a(x_{t(i)})}), (x^{t(i)}_{a(x_{t(i)})}, r) \in \mathcal{T}_1$ and $(c_i, x^{f(i)}_{\overline{a}(x_{f(i)})}), (x^{f(i)}_{\overline{a}(x_{f(i)})}, r) \in \mathcal{T}_2$, hence nodes c_i have $c(P(\mathcal{T}_1, c_i)) + c(P(\mathcal{T}_2, c_i)) = 2 + 2 = 4$, which is minimal. The trees $\mathcal{T}_1$ and $\mathcal{T}_2$ are clearly minimum length for nodes x_t^i and x_f^i, too.

To prove the other direction let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two minimum length redundant trees. Hence, for every variable $x_i \in X$, 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:

$$a(x_i) := \begin{cases} t & \text{, if } (x_t^i, r) \in \mathcal{T}_1 \\ f & \text{, if } (x_f^i, r) \in \mathcal{T}_1 \end{cases}$$

From the assumption on minimum length, we get that c_i have $c(P(\mathcal{T}_1, c_i)) = c(P(\mathcal{T}_2, c_i)) = 2$, that is there exists a variable x_j with either $x_j \in c_i$ and $(x_j^t, r) \in \mathcal{T}_1$ or $\overline{x}_j \in c_i$ and $(x_j^f, r) \in \mathcal{T}_1$. Both are equivalent to that there is a literal that is evaluated to true in clause c_i. Similarly we can derive from $c(P(\mathcal{T}_2, c_i)) = 2$ that there is also a literal which is evaluated to false in c_i. ■

Since NAE-3SAT is NP-complete, the claim follows. ■