CG4SR: Near Optimal Traffic Engineering for Segment Routing with Column Generation

Mathieu Jadin
ICTEAM, UCLouvain
Belgium
mathieu.jadin@uclouvain.be

Francois Aubry
ICTEAM, UCLouvain
Belgium
f.aubry@uclouvain.be

Pierre Schaus
ICTEAM, UCLouvain
Belgium
pierre.schaus@uclouvain.be

Olivier Bonaventure
ICTEAM, UCLouvain
Belgium
olivier.bonaventure@uclouvain.be

Abstract—**Segment Routing (SR) is a powerful tool to solve traffic engineering in large networks. It enables steering the traffic along any arbitrary network path while limiting scalability issues as routers do not need to maintain a global state. Mathematical programming approaches proposed so far for SR either do not scale well with the size of topology or impose a strong limit on the number of possible detours (typically at most one). Moreover they do not support Segment Routing fully by ignoring the adjacency segments. This paper leverages column generation, a widely used technique for solving large scale linear programs, combined with a novel dynamic program for solving the pricing problem. Our approach reaches near optimal solutions with gap guarantees by also computing a strong lower-bound tighter than the multi-commodity flow relaxation. It scales even on large topologies and exploits the full expressiveness of SR including adjacency segments. Our experiments compared with existing traffic engineering techniques on various topologies and demand matrices demonstrate the advantages of our approach in terms of scalability, any-time behavior and quality of the solutions.**

I. Introduction

Wide area networks are a key element of our Internet-centric society. Most of them are managed by Internet Service Providers (ISPs) and large entreprises that use them to interconnect their datacenters. Building and operating these networks is costly and their owners rely on a variety of traffic engineering techniques [1], [2] to optimise them. These techniques range from adjusting link weights based on traffic patterns [3], using a centralized controller in a Software Defined Network [4] to using MPLS to control routing paths [5], [6].

During the last years, networking vendors and network operators have designed [7], [8], implemented [9] and deployed [10], [11] a new routing architecture called Segment Routing (SR). Segment Routing is a modern variant of source routing which can be used in either MPLS or IPv6 networks. In a nutshell, Segment Routing allows the source of a packet or the ingress node in a network to easily specify the path that the packet needs to follow inside the network. This path is specified as a series of labels (MPLS labels or IPv6 addresses in a special IPv6 header extension) that are added to each packet. The added routing flexibility of SR makes it possible to better utilise the network to its full capacity without incurring the overhead that one gets by using MPLS as there is no need to maintain a global state or use specialised signalling protocols such as LDP or RSVP-TE. We describe Segment Routing in more details in Section II.

Segment Routing has been designed with various use cases in mind [12], [13]. In this paper, we focus on the development of efficient traffic engineering algorithms that leverage the unique characteristics of Segment Routing. More precisely, we propose a new scalable approach that relies on column generation [14]: CG4SR. Moreover it can provide strong guarantees on the quality of the solutions by computing a lower-bound that is in most of the cases tighter than the one obtained with a multi-commodity flow relaxation.

In developing CG4SR, we make the following contributions.

CG framework for SR: We propose the first column generation framework for solving TE with Segment Routing.

Fully leverage SR: Our solution is able to leverage Segment Routing to the full extent of its potential as we have full control over the number of segments that are used and we support both node and adjacency segments.

Improved lower bound: We improve the lower bound provided by traditional Multi-Commodity Flow approaches by up to 15% for TE with Segment Routing.

Min cost path SR-problem: As a by-product of our solution, we developed a general and efficient algorithm for computing Segment Routing paths of minimum cost.

The remainder of this paper is organized as follows. In Section II, we give an overview of Segment Routing and describe the main traffic engineering techniques that have been proposed. Then, in Section III, we formalize the traffic engineering problem and propose a column generation formulation to solve it. In Section IV, we provide efficient algorithms to solve the subproblems involved in our formulation and a detailed description of our column generation algorithm. In Section V, we provide a thorough evaluation of our solution, comparing it to the state of the art traffic engineering algorithms for Segment Routing.

II. Background

Segment Routing [8] allows to add a stack of labels to the IP header of each packet with the address of each segment. These labels are used as temporary destination addresses making the packets follow a set of detours before reaching the final destination. There are two types of segments: (i) node and (ii) adjacency segments. Node segments represent a router.

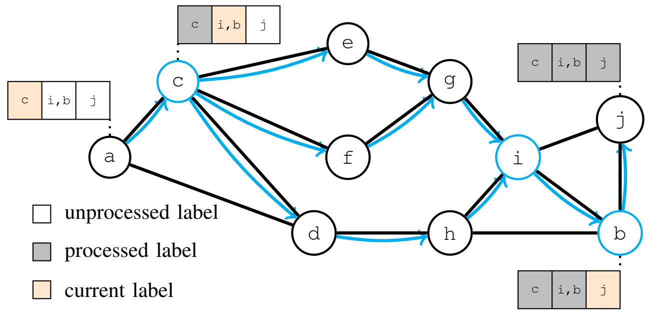


Fig. 1: Segment Routing illustration. The figure assumes unit IGP weights.

When a node segment is at the top of the stack, the packet is routed towards this node using standard shortest path routing. On the other hand, adjacency segments represent an outgoing interface on a node. To simplify the explanation, we think of it as representing a link in the network. Thus, in this case, the packet is routed to the source of the link via the shortest paths and then it will be forwarded to the destination of the link via that specific link. Figure 1 illustrates this process. In this example, the ingress node is node a and the labels consist of a node segment representing router c, an adjacency segment representing link (i, b), and the final destination j. The arrows represent the shortest paths, having multiple arrows represent Equal Cost Multi-Path (ECMP). If there is ECMP between two consecutive segments, that is, several shortest paths exist, we assume that the load is equally split among them.

A. Traffic Engineering for Segment Routing

A simple formulation of the Traffic Engineering (TE) problem is the Multi-Commodity Flow (MCF) linear programming formulation [15]. Unfortunately, formulating TE as a MCF has several drawbacks. First, the formulation uses $O(|V|^2 \cdot |\mathcal{D}|)$ variables making it unusable for large topologies. Second, this formulation is not designed to be used with Segment Routing and can thus output any set of network paths. In practice, deployed routers have hardware limitations on the number of segments that they can forward [16].

In the recent years, a few SR centric approaches for optimising TE have been proposed [17]–[20]. Complete approaches to optimize exactly traffic engineering using Segment Routing are unable to tackle the general problem as they suffer from the same scaling problem as the MCF formulation mentioned above. Bhatia et al. propose in [20] a solution addressing the scalability issue by considering a subset of the possible Segment Routing paths (sr-paths) with one intermediate segment only. Trimponias et al. propose to restrict the set of allowed segments to a subset of the network [21]. Other researchers have proposed heuristic techniques. Hartert et al. proposed in DEFO [17], [18] a Large Neighborhood Search technique combined with Constraint Programming. Later, Gay et al. proposed in [19] to use standard local search to iteratively improve the current solution. These heuristic approaches are rather efficient but provide no way of knowing how far from the optimal value they end up. Moreover, all the existing approaches only consider node segments in their formulation. They are thus not able to fully exploit the flexibility of Segment Routing with adjacency segments.

III. Formalization

We model the network as a directed graph $G = (V, E, igp, c)$ where $igp : E \to \mathbb{Z}^+$ represents the IGP weights configured on the links and $c : E \to \mathbb{Z}^+$ represents the link capacities. We define the size of G as $|G| = |V| + |E|$.

The demands are simply modeled as a set $\mathcal{D}$ of triples (s, t, v) meaning that we have a volume v of traffic to route from s to t. For a given demand $d \in \mathcal{D}$ we write $s(d) = s$, $t(d) = t$ and $v(d) = v$.

We model sr-paths as sequences $\langle x_1, x_2, \ldots, x_n \rangle$ where each x_i is either an element of V, representing a node segment, or an element of E, representing an adjacency segment. If $x_i = (u, v)$ is an adjacency segment we denote $x_i^1 = u$ and $x_i^2 = v$. It is convenient to have a uniform notation to avoid having to distinguish between node and adjacency segments later on. For this reason, we also define for a node segment x_i the notations $x_i^1 = x_i^2 = x_i$. We denote the set of indexes of node segments by $node(p)$ and the set of indices of adjacency segments by $adj(p)$. Sr-paths actually represent a subgraph of the network since several shortest paths may exist between consecutive node segments. More specifically, $SP(x_{i-1}^2, x_i^1)$ represents the set of edges on a shortest path between nodes x_{i-1}^2 and x_i^1. We denote the set of edges that belong to the subgraph represented by a sr-path p by $E(p) = adj(p) \cup \left(\bigcup_{i=2}^{n} SP(x_{i-1}^2, x_i^1)\right)$.

Some routers support only a limited number of segments in the packet header [16]. Therefore, our TE algorihms should be able to limit the number of segments. Adjacency segments are local segments which can only be parsed by the link origin. They require to store both the node segment to reach the link origin and the actual adjacency segment identifying the router interface. Hence, they cost twice as much as node segments.

The segment cost of a sr-path is thus the number of node segments plus twice the number of adjacency segments. We denote the segment cost of a sr-path p from s to t by $seg(p) = |node(p) \setminus \{s\}| + 2 \cdot |adj(p)|$ and the set of all sr-paths with segment cost at most k by $\mathcal{P}(k) = \{p \mid seg(p) \leq k\}$. Note that the source node is not counted as it needs not to be added to the segment stack.

We say that a sr-path $p = \langle x_1, \ldots, x_n \rangle$ is compatible with a demand d if and only if $x_1^1 = s(d)$ and $x_n^2 = t(d)$. That is, p starts and ends at the source and destination nodes of d, respectively. We denote the set of demands compatible with p by $\mathcal{D}(p)$.

When a demand is routed over a sr-path, it is split equally whenever there is ECMP between two consecutive segments. As illustrated in Figure 2, the traffic is first sent from node a to c without any split as there is a single shortest path. Then, between segments c and i there are three shortest paths, so the demand is split equally between them. The edge labels represent the ratio of the demand that will traverse each edge.

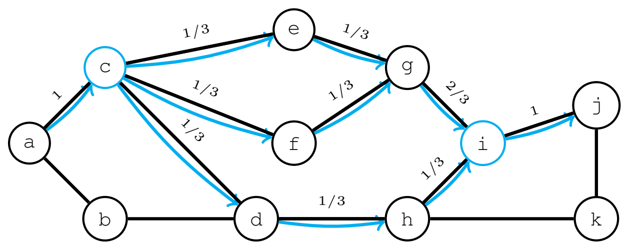


Fig. 2: Illustration of the split ratio for sr-path $\langle a, c, i, j \rangle$. The figure assumes unit IGP weights.

This motivates the definition of $r(x, y, e)$ which represents the load generated on edge e per unit of traffic forwarded on the shortest paths between nodes x and y. If e does not belong to $SP(x, y)$ then $r(x, y, e) = 0$. More generally we define utilisation ratio on an edge for a given sr-path $p = \langle x_1, \ldots, x_n \rangle$ as

$$r(p, e) = \sum_{i=2}^{n} r(x_{i-1}^2, x_i^1, e) + |\{i \in adj(p) \mid x_i = e\}|.$$

The first part of the expression evaluates the ratio on e for the shortest path subgraphs between consecutive segments, while the second part counts how many times e is used as an adjacency segment on the sr-path p. This summation is valid because when an adjacency segment occurs, the full unit demand is routed through the edge without split.

A. Problem formulation

The Segment Routing Traffic Enginnering (SRTE) problem is to discover a sr-path p for each demand such that the worst link utilization (total traffic/capacity) is minimized. This problem was demonstrated to be NP-hard in [17].

We propose a scalable mathematical programming approach for solving the general SRTE problem with segments up to cost k also allowing adjacency segments. Our approach relies on the so-called Column Generation (CG) decomposition approach [22] for solving huge linear programs. CG introduces one decision variable per column and defines the objective function to maximize as a linear combination of those column variables. For SRTE, columns correspond to candidate sr-paths that can be selected for each demand. The minimization of maximum link utilisation is not directly a good fit for a CG formulation as our experiments showed that it had difficulties to converge. Therefore we instead solve a close variant of this problem seeking to assign demands to sr-paths so that the amount of demand volume routed is maximal without exceeding the link capacities above a given threshold and later adapt it to minimize the maximum link utilisation. We call this problem SRTED.

Considering the whole set of sr-paths $\overline{\mathcal{P}}(k)$ upfront would not scale as it would require solving a linear program with an huge number of candidate paths and variables. Therefore a restricted set of sr-paths $\mathcal{P} \subseteq \overline{\mathcal{P}}(k)$ is initially considered (called the columns in the CG framework) and then is iteratively grown by carefully selecting new candidate paths from $\overline{\mathcal{P}}(k)$ until we obtain a proof that no matter what new paths would be added, the solution would not be further improved. The linear program working on the restricted set of sr-paths $\mathcal{P}$ is called the *master problem*. Once the master problem is solved optimally, new interesting paths can be added incrementally to the restricted set $\mathcal{P}$ by solving a sub-problem called the *pricing problem*. This pricing problem amounts to discovering new columns (sr-paths) with negative reduced costs in the master problem, meaning that if added to the path set, they could potentially improve the current solution. The process of interleaving the master problem solving and the pricing problem continues until the pricing problem fails to discover negative reduced cost columns. At that point, the master problem, although working on a restricted set of sr-paths is proven optimal with respect to every possible path in $\overline{\mathcal{P}}(k)$. The advantage of CG is thus to solve tight linear program formulations with an exponential number of variables without even considering them all explicitly in practice.

The ILP formulation of the SRTED problem which maximizes the volume of the demands, that are routed through the set of sr-paths $\mathcal{P}$, depends on three parameters: the sr-path set $\mathcal{P}$, the demand set $\mathcal{D}$ and a capacity factor $\lambda \in \mathbb{R}^+$. This capacity factor is used to control by which factor we allow the edge capacities to be exceeded. A binary variable is introduced for each pair of compatible sr-path p and demand d:

$$\mathbf{x}_{dp} = \begin{cases} 1 & \text{if demand } d \text{ is routed over sr-path } p \in \mathcal{P} \\ 0 & \text{otherwise} \end{cases}$$

The ILP is then formally defined as:

SRTED-ILP($\mathcal{P}$,$\mathcal{D}$,λ)

$$\begin{aligned} \max_{\mathbf{x}} \quad & \sum_{p\in\mathcal{P}} \sum_{d\in\mathcal{D}(p)} v(d) \cdot \mathbf{x}_{dp} \\ \text{s.t.} \quad & \sum_{p\in\mathcal{P}} \mathbf{x}_{dp} \leq 1 \quad \forall d\in\mathcal{D} \\ & \sum_{p\in\mathcal{P}} \sum_{d\in\mathcal{D}(p)} r(e,p) \cdot v(d) \cdot \mathbf{x}_{dp} \leq \lambda \cdot c(e) \quad \forall e\in E \\ & \mathbf{x}_{dp} \in \{0,1\} \quad \forall p\in\mathcal{P}, \forall d\in\mathcal{D}(p) \end{aligned}$$

The objective function seeks to route a maximum demand volume over the sr-paths $\mathcal{P}$. The first set of constraints ensures that at most one sr-path is selected for each demand and the second set of constraints makes sure that the capacity of an edge is not exceeded by a factor greater than λ. If $\lambda = 1$ then the problem routes the maximum amount of demand volume without exceeding any link capacities. Our model may thus discard some demands if the parameter λ is too restrictive. Discovering the minimal value for λ, such that all the demands are routed, can be achieved with a dichotomic search on top of the CG framework as discussed later.

In order to be able to use LP duality theory and ultimately column generation, we consider the LP relaxation of $\mathsf{SRTED\text{-}ILP}(\mathcal{P}, \mathcal{D}, \lambda)$ which we name $\mathsf{SRTED\text{-}LP}(\mathcal{P}, \mathcal{D}, \lambda)$. It has the exact same formulation except that the binary variables $\mathbf{x}_{dp}$ are allowed to have fractional values in $[0, 1]$.

It is known that any LP model has a dual problem which can be obtained by following a systematic procedure [23]. Doing so, we obtain the following dual problem:

$\mathsf{SRTE\text{-}DUAL}(\mathcal{P},\mathcal{D},\lambda)$

$$\min_{\mathbf{y},\mathbf{z}} \quad \lambda \cdot \sum_{e \in E} c(e) \cdot \mathbf{y}_e + \sum_{d \in \mathcal{D}} \mathbf{z}_d$$

s.t.

$$\mathbf{z}_d + \sum_{e \in E(p)} r(e,p) \cdot v(d) \cdot \mathbf{y}_e \ \geq v(d) \quad \forall p \in \mathcal{P}, \forall d \in \mathcal{D}(p)$$

$$\mathbf{y}_e, \mathbf{z}_d \geq 0 \quad \forall p \in \mathcal{P}, \forall d \in \mathcal{D}(p)$$

To simplify the notations we use the names of the problems defined above both to refer to the problem itself as well as to refer to the value of their optimal solution.

As a consequence of the strong duality theorem for linear programming we have that

$$\mathsf{SRTED\text{-}LP}(\mathcal{P}, \mathcal{D}, \lambda) = \mathsf{SRTE\text{-}DUAL}(\mathcal{P}, \mathcal{D}, \lambda).$$

and that whenever we solve $\mathsf{SRTED\text{-}LP}(\mathcal{P}, \mathcal{D}, \lambda)$ we obtain a solution $\mathbf{x}$ together with an optimal solution of $\mathsf{SRTE\text{-}DUAL}(\mathcal{P}, \mathcal{D}, \lambda)$ that we denote by $(\mathbf{y}, \mathbf{z})_{\mathbf{x}}$.

The next theorem is at the heart of the pricing problem. It gives the necessary condition for a sr-path to be considered as interesting for inclusion in the restricted set $\mathcal{P}$.

Theorem 1. *Let $\mathcal{P} \subseteq \overline{\mathcal{P}}(k)$, $\mathbf{x}$ be an optimal solution of $\mathsf{SRTED\text{-}LP}(\mathcal{P}, \mathcal{D}, \lambda)$ and $(\mathbf{y}, \mathbf{z})_{\mathbf{x}}$ the corresponding solution of $\mathsf{SRTE\text{-}DUAL}(\mathcal{P}, \mathcal{D}, \lambda)$. Then, $\mathbf{x}$ is optimal for $\mathsf{SRTED\text{-}LP}(\overline{\mathcal{P}}(k), \mathcal{D}, \lambda)$ if and only if for all $p \in \overline{\mathcal{P}}(k), d \in \mathcal{D}(p)$,*

$$\mathbf{z}_d + \sum_{e \in E(p)} r(e,p) \cdot v(d) \cdot \mathbf{y}_e \ \geq v(d). \tag{1}$$

Proof. Suppose that for all $p \in \overline{\mathcal{P}}(k), d \in \mathcal{D}$ inequality (1) holds. Then this means that $(\mathbf{y}, \mathbf{z})_{\mathbf{x}}$ is also an admissible solution of $\mathsf{SRTE\text{-}DUAL}(\overline{\mathcal{P}}(k), \mathcal{D}, \lambda)$. Since $\mathcal{P} \subseteq \overline{\mathcal{P}}(k)$ we have that $\mathsf{SRTE\text{-}DUAL}(\mathcal{P}, \mathcal{D}, \lambda)$ is a relaxation of $\mathsf{SRTE\text{-}DUAL}(\overline{\mathcal{P}}(k), \mathcal{D}, \lambda)$. Therefore $\mathsf{SRTE\text{-}DUAL}(\mathcal{P}, \mathcal{D}, \lambda) \leq \mathsf{SRTE\text{-}DUAL}(\overline{\mathcal{P}}(k), \mathcal{D}, \lambda)$. Hence, since $(\mathbf{y}, \mathbf{z})$ is an admissible solution of both problems, we have that $\mathsf{SRTE\text{-}DUAL}(\mathcal{P}, \mathcal{D}, \lambda) = \mathsf{SRTE\text{-}DUAL}(\overline{\mathcal{P}}(k), \mathcal{D}, \lambda)$. Hence, by strong duality,

$$\begin{aligned} \mathsf{SRTED\text{-}LP}(\mathcal{P}, \mathcal{D}, \lambda) &= \mathsf{SRTE\text{-}DUAL}(\mathcal{P}, \mathcal{D}, \lambda) \\ &= \mathsf{SRTE\text{-}DUAL}(\overline{\mathcal{P}}(k), \mathcal{D}, \lambda) \\ &= \mathsf{SRTED\text{-}LP}(\overline{\mathcal{P}}(k), \mathcal{D}, \lambda). \end{aligned}$$

□

In practice, what this means is that if we solve SRTED-LP to optimality for a given subset of sr-paths $\mathcal{P}$, demands $\mathcal{D}$ and λ, then we can decide whether this solution is optimal over the set of *all* sr-paths $\overline{\mathcal{P}}(k)$ by checking whether there exist some sr-path p and demand $d \in \mathcal{D}(p)$ that violate inequality (1). This amounts to finding a pair (p, d) such that

$$\mathbf{z}_d + \sum_{e \in E(p)} r(e,p) \cdot v(d) \cdot \mathbf{y}_e \ < v(d). \tag{2}$$

Given $(\mathbf{y}, \mathbf{z})$, the Pricing problem is to find a sr-path $p \in \overline{\mathcal{P}}(k)$ and a demand $d \in \mathcal{D}$ such that (2) holds. Note that solving this problem by iterating over all sr-paths $p \in \overline{\mathcal{P}}(k)$ is unrealistic since $|\overline{\mathcal{P}}(k)| = O(|G|^k)$.

Next section introduces an efficient algorithm to solve the Pricing problem. We show how to integrate it into an algorithm that solves $\mathsf{SRTED\text{-}LP}(\overline{\mathcal{P}}(k), \mathcal{D}, \lambda)$.

IV. Algorithm

A. Solving the Pricing problem

In order to solve the Pricing problem, we developed a general Dynamic Programming (DP) algorithm for solving the following, more general, Segment Routing path finding problem.

Problem 1. ***(SR-Min cost path problem)***[1] *Compute a sr-path of minimum cost between two nodes with a given segment budget. Let $G = (V, E)$ be a graph. Given two cost functions $c_{sp} : V \times V \to \mathbb{R}^+$ and $c_{adj} : E \to \mathbb{R}^+$, a constant k and $s, t \in V$. Find a sr-path $p = \langle x_1, \ldots, x_n \rangle$ from s to t such that $seg(p) \leq k$ and*

$$c(p) = \sum_{i=2}^{n} c_{sp}(x_{i-1}^2, x_i^1) + \sum_{i \in adj(p)} c_{adj}(x_i)$$

is minimum. The cost function c_{sp} represents the cost of using the shortest path DAG between any two given nodes and the c_{adj} represents the cost on the edges used to evaluate adjacency segments.

In order to solve this problem, let us define the following DP state space

$$dp(i, x) = \text{the minimum weight (w.r.t. } c\text{) sr-path from } s \text{ to } x \text{ with segment cost at most } i.$$

The solution to the SR-min cost path problem is $dp(k, t)$. For $i = 0$ the only possible sr-path of segment cost 0 from s to any node is $\langle s \rangle$. Thus, we have that $dp(0, s) = 0$ and $dp(0, x) = \infty$ for $x \neq s$. For $i \geq 1$ it can be shown that

$$dp(i,x) = \min \begin{cases} dp(i-1,x) & & & & \\ dp(i-1,y) & + & c_{sp}(y,x) & s.t & y \in V \\ dp(i-2,r) & + & c_{sp}(r,z) & s.t & r \in V, \\ & + & c_{adj}(z,x) & & (z,x) \in E \end{cases}$$

[1]In addition to solving our Pricing problem, this general problem could also be used for instance to compute a minimum latency sr-path between two nodes in a network.

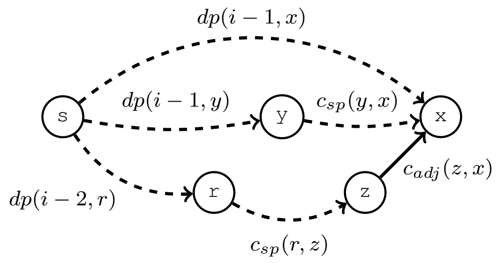


Fig. 3: Illustration of the dp recurrence

where the third value is only defined for $i \geq 2$ (set to ∞) otherwise.

The intuition behind this recurrence is that there are three possibilities to reach a node x with a sr-path of segment cost at most i. These three possibilities are illustrated in Figure 3. The first way is to simply not use the extra segment and reach x in the best way using a sr-path of segment cost at most $i-1$. The second possibility is to reach some node $y \in V$ using a sr-path of segment cost at most $i-1$ and then use one extra node segment to reach x incurring a cost of $c_{sp}(y,x)$. Finally, we can use a sr-path of cost at most $i-2$ to any node r and then append to it an adjacency segment (z,x) where z is a neighbor of x. Note that nothing prevents r from being equal to z. In this case it simply means that we append the adjacency (z,x) to a path that already ends at node z.

Since the values of $dp(i,*)$ depend only on $dp(i-1,*)$, we can compute all these values by iterating in increasing order of i. There are $O(k \cdot |V|)$ states to compute and evaluating each of them takes $O(|V|)$ so the total complexity of the algorithm is $O(k \cdot |V|^2)$.

We now show how to cast the Pricing problem as the one of solving one minimum cost sr-path problem per demand.

Let $d \in \mathcal{D}$ be a demand and $(\mathbf{y}, \mathbf{z})_{\mathbf{x}}$ be the dual values corresponding to a primal solution $\mathbf{x}$. Define $c_{sp}(x,y) = \sum_{e \in E} r(e,x,y) \cdot \mathbf{y}_e$ (if $x = y$ we set $c_{sp}(x,y) = 0$) and $c_{adj}(e) = \mathbf{y}_e$. The above dynamic program computes a minimum cost sr-path $p = \langle x_1, \ldots, x_n \rangle \in \overline{\mathcal{P}}(k)$ from $s(d)$ to $t(d)$ with respect to c_{sp} and c_{adj} that minimizes

$$\begin{aligned} c(p) &= \sum_{i=2}^{n} c_{sp}(x_{i-1}^2, x_i^1) + \sum_{i \in adj(p)} c_{adj}(x_i) \\ &= \sum_{i=2}^{n} \sum_{e \in E} r(e, x_{i-1}^2, x_i^1) \cdot \mathbf{y}_e + \sum_{i \in adj(p)} \mathbf{y}_{x_i} \\ &= \sum_{e \in E} \sum_{i=2}^{n} r(e, x_{i-1}^2, x_i^1) \cdot \mathbf{y}_e + \sum_{e \in E} \sum_{i \in adj(p) : x_i = e}^{n} \mathbf{y}_e \\ &= \sum_{e \in E} \left(\sum_{i=2}^{n} r(e, x_{i-1}^2, x_i^1) + |\{i \in adj(p) \mid x_i = e\}| \right) \mathbf{y}_e \\ &= \sum_{e \in E} r(e,p) \cdot \mathbf{y}_e. \end{aligned}$$

Thus, there exists a sr-path p that satisfies inequality (2) with respect to demand d if and only if $c(p) < \frac{v(d) - \mathbf{z}_d}{v(d)}$. This means that we can solve the Pricing problem by solving the minimum cost sr-path problem for each demand d. The total time complexity of this algorithm is $O(k \cdot |\mathcal{D}| \cdot |V|^2 + |V|^2 \cdot |E|)$ where the second term represents the cost of precomputing the cost function c_{sp}. This shows that we can solve the pricing problem in polynomial time.

B. The column generation algorithm

Algorithm 1 describes the column generation algorithm solving optimally SRTED-LP$(\mathcal{P}, \mathcal{D}, \lambda)$. The set $\mathcal{P}$ contains any set of initial paths. We use $\mathcal{P} = \{\langle s(d), t(d) \rangle \mid d \in \mathcal{D}\}$ so that the algorithm preferably uses IGP shortest paths whenever possible. The main loop of the algorithm consists in calling the iterate Algorithm 2 in charge of the path-generation process until no new interesting path can be generated. By Theorem 1, the stopping criterion ensures that our solution is optimal for the global sr-path set $\overline{\mathcal{P}}(k)$. The iterate Algorithm 2 performs two main steps:

1) Optimizing SRTED-LP$(\mathcal{P}, \mathcal{D}, \lambda)$ with an LP solver and collecting the final dual values $(\mathbf{y}, \mathbf{z})_{\mathbf{x}}$ (a by-product of the simplex algorithm).
2) Considering each demand d and computing the minimum cost sr-path p from $s(d)$ to $t(d)$ as described above. Every minimum cost sr-path p such that $c(p) < (v - \mathbf{z}_d)/v$ is added to P' that eventually is added to the restricted path set $\mathcal{P}$.

To ensure that SRTED-LP is solved optimally, we can choose in step 2 to add any number of paths satisfying $c(p) < (v - \mathbf{z}_d)/v$. At one extreme, one could stop as soon as one such path is found. The path-finding process is then fast but with the drawback of possibly increasing substantially the number of iterations needed to converge. At the opposite extreme, we could generate for each demand every possible path satisfying the property (and not only the optimal one). Doing so would probably reduce the number of iterations but would increase the computation cost of each. We chose a trade off by adding up to *maxp* paths generated at step 2. Those paths are generated by considering the demands by decreasing the value of $(v(d) - \mathbf{z}_d)/v(d)$ in order to maximize our chances of finding a path satisfying inequality (2).

In our implementation, we use a parallel execution for the last loop of Algorithm 2. One main worker appends to a bounded buffer the demands by decreasing value of $(v(d) - \mathbf{z}_d)/v(d)$. Consumer workers execute the mincost-srpath algorithm and add the path to P' if the inequality (2) is satistfied. This provides significant speedup because no communication between the workers is required during the mincost-srpath algorithm. We assume that the function LP-solve solves a linear program and returns a tuple containing the values for the primal variables, the values of the dual variables and the value of the objective function.

C. Minimizing the worst link utilization

Traditionally, TE aims at routing a whole set of demands while minimizing the worst link utilization. There are two

Algorithm 1 colgen $(\mathcal{P}, \mathcal{D}, \lambda, maxp)$

```
1: while $P' \leftarrow$ iterate$(\mathcal{P}, \mathcal{D}, \lambda, maxp) \neq \emptyset$ do
2:     $\mathcal{P} \leftarrow \mathcal{P} \cup P'$
3: return LP-solve(SRTED-LP$(\mathcal{P}, \mathcal{D}, \lambda)$)
```

Algorithm 2 iterate $(\mathcal{P}, \mathcal{D}, \lambda, maxp)$

```
1: $\mathbf{x}, (\mathbf{y}, \mathbf{z})_{\mathbf{x}}, vol \leftarrow$ LP-solve(SRTED-LP$(\mathcal{P}, \mathcal{D}, \lambda)$)
2: for $x, y \in V$ do
3:     $c_{sp}(x, y) \leftarrow \sum_{e \in E(\langle x,y \rangle)} r(e, x, y) \cdot \mathbf{y}_e$
4: for $e \in E$ do
5:     $c_{adj}(e) \leftarrow \mathbf{y}_e$
6: $P' \leftarrow \emptyset$
7: for $d \in \mathcal{D}$ in decreasing order of $(v(d) - \mathbf{z}_d)/v(d)$ do
8:     $p \leftarrow$ mincost-srpath$(s(d), t(d), c_{sp}, c_{adj})$
9:     if $c(p) < (v(d) - \delta_d)/v$ then
10:        $P' \leftarrow P' \cup \{p\}$
11:    if $|P'| \geq maxp$ then
12:        break
13: return $P'$
```

reasons why Algorithm 1 cannot be used directly for solving this problem:

1) It will select a subset of demands to route under the constraint of a given maximum overload λ.
2) The decision variables may take fractional value in the optimal solution to SRTED-LP.

Algorithm 3 CG4SR addresses these two issues by making use of the colgen Algorithm 1 as a subroutine.

The first issue is addressed by performing a binary search for the parameter $\lambda \in [0, \lambda_M]$ (where λ_M is a large enough value) to discover the minimum value for λ such that the full set of demands is routed. At each step of the binary search, we check whether all the volume is routed. If so, we decrease λ, otherwise, we increase it. From one step to the next, the initial path set provided to Algorithm 1 is the one generated at the previous step so that Algorithm 1 only needs to perform a few calls to iterate (Algorithm 2) in practice.

The second issue is addressed by heuristically solving an ILP problem denoted SRTE-UTIL-ILP to select for each demand one path $\mathcal{P}$ while minimizing the worst link utilization.

$$\begin{array}{llcll}
\text{SRTE-UTIL-ILP} \\
\min\limits_{\lambda} & \lambda \\
\text{s.t.} & \sum\limits_{p \in \mathcal{P}} \mathbf{x}_{dp} & = & 1 & \forall d \in \mathcal{D} \\
& \sum\limits_{p \in \mathcal{P}} \sum\limits_{d \in \mathcal{D}(p)} r(e,p) \cdot v(d) \cdot \mathbf{x}_{dp} & \leq & \lambda \cdot c(e) & \forall e \in E \\
& \mathbf{x}_{dp} & \in & \{0,1\} & \forall p \in \mathcal{P}, \forall d \in \mathcal{D}(p)
\end{array}$$

This model is very similar to the SRTED-ILP model. It changes the objective function to minimize the capacity factor λ and replaces the inequality in the first set of constraints with an equality to guarantee that each demand is satisfied. It

Algorithm 3 GC4SR $(\mathcal{D}, \lambda_M, k, \epsilon, maxp)$

```
1: $\mathcal{P} \leftarrow \{\langle s, t \rangle \mid (s, t, d) \in \mathcal{D}\}$
2: _, _, target-vol $\leftarrow$ colgen$(\mathcal{P}, \mathcal{D}, \lambda_M, maxp)$
3: $lb \leftarrow 0$
4: $ub \leftarrow \lambda_M$
5: while $|lb - ub| > \epsilon$ do
6:     $\lambda \leftarrow (lb + ub)/2$
7:     $\mathbf{x}, (\mathbf{y}, \mathbf{z})_{\mathbf{x}}, vol \leftarrow$ colgen$(\mathcal{P}, \mathcal{D}, \lambda, maxp)$
8:     if $vol \geq$ target-vol then
9:         $ub \leftarrow \lambda$
10:    else
11:        $lb \leftarrow \lambda$
12: return ILP-solve(SRTE-UTIL-ILP$(\mathcal{P}, \mathcal{D})$)
```

is guaranteed to be feasible since by initialization of $\mathcal{P}$, it contains at least one path for each demand. Unfortunately it has no guarantee to solve optimally the worst link utilization problem. Indeed, $\mathcal{P}$ may not contain all the necessary paths to reach optimality. Forcing discrete decisions starting heuristically from the set of columns produced by a column generation process is a standard approach (see for instance [14]). We can also report the optimality $gap =$ ILP-solve(SRTE-UTIL-ILP$(\mathcal{P}, \mathcal{D})) - lb$ providing a guarantee of how sub-optimal the integer solution might be. As shown in the experimental section, the solutions computed by Algorithm 3 are very close to optimality in practice.

As before, we assume that ILP-solve is a funtion that solved an integer linear program.

Our framework, similarly to [17], is flexible enough to accommodate other constraints on the paths by only adapting the mincost-srpath algorithm. For example, by computing only paths of segment cost at most k, we implicitely limit the maximum segment cost of any path in the solution. Other types of constraints could for instance be a limit on the latency or forbidden/mandatory intermediate nodes (for service chaining constraints [24]).

V. Evaluation

This section describes the results using the column generation approach described in Algorithm 3 CG4SR. All the experiments were conducted on available RocketFuel topologies [25] for reproducibility. Their characteristics are described in Table I. We use Repetita [26] to run all the other solvers: DEDO [17], [18], Bhatia [20] and SRLS [19]. We reuse the demand matrices generated for Repetita [26]. For each topology, they generated 5 demand matrices through the gravity model described in [27]. Demands were normalized so that MCF can merely force all link loads to be below or equal to 90%. The number of nodes ranges from 80 to 315 and the number of demands from 7482 to 98910.

All our experiments are easily reproducible[2]. Our experiments were run on a computer with 32 CPUs at 2.60GHz, 128GB of RAM and Java 1.8 JVM. CG4SR does not actually need 128GB of RAM but it is able to take advantage of the 32

[2]The code of our solver is integrated to Repetita [26] which is available at https://github.com/svissicchio/Repetita

TABLE I: Dataset summary

ID	# nodes	# links	# demands
AS 1221	104	302	10712
AS 1239	315	1944	98910
AS 1755	87	322	7482
AS 3257	161	656	25760
AS 3967	79	294	6162
AS 6461	138	744	18906

CPUs thanks to the multithreading of the pricing computation. We used the same version of Gurobi [28], v8.0, for all the solvers.

A. Near optimum evaluation

CG4SR provides a better lower bound for TE over SR than MCF. Traditionally, the value of an optimal MCF solution is used as a lower bound for minimum maximum link utilisation that one can achieve for routing a traffic matrix. However, as mentioned above, this bound is unrealistic as MCF is oblivious to SR. Figure 4a studies the quality of the lower bound provided by CG4SR, SRTED-LP, compared to MCF. The load predicted by MCF is always of about 90% because the demand matrices of Repetita [26] were generated artificially to be at this value. However, CG4SR shows that it is strictly impossible to escape network congestion for 5 demand matrices. Moreover, the difference between CG4SR and MCF lower bounds can be as high as 15% in the predicted maximal load. Increasing the number of segments does not get CG4SR lower bound much closer to the MCF.

CG4SR provides solutions whose maximum load is at most 4% more than the optimal solution. We ran CG4SR without enabling adjacency segments and without time limit. The experiment was repeated with limits of 2, 4 and 6 segments to observe the impact of the segment limit on the quality of the solution. Figure 4b shows CDFs of the gap (in percentage) between the CG4SR upper and lower bounds on all the 30 instances (i.e., 5 demand matrices for each of the 6 topologies) and increasing the limit on the number of segments. This gap is the maximum distance to the actual optimum. We can see that increasing the limit from 2 to 4 segments impacts the quality of the solution while increasing the limit from 4 to 6 has little impact. Paths with 4 or 6 segments add more flexibility to SRTE-UTIL-ILP than paths with 2 segments. We can see that this gap is most of the time below 1% of the load and at worst 4% of the load if 4 segments are allowed.

CG4SR is more efficient than Bhatia and MCF. We compared CG4SR speed to Bhatia, MCF and MCFP. MCFP is an efficient variant of MCF (described in [26]) that is only able to compute the optimal objective value of MCF, not the actual paths comprising the solution. Figure 4c describes how fast the different solvers can find their best solution. During these runs, the limit on the number of generated paths at each column generation iteration, *maxp* in Algorithm 2, is fixed to 10. This figure shows a CDF of the execution time on the different topologies and demand matrices for CG4SR, Bhatia and MCF. Because Bhatia only allows two segments, CG4SR is also limited to two segments in this figure. MCF is the slowest one and it runs out of memory for all the demand matrices of the largest topology despite the 120GB available. Bhatia only considers paths that can be expressed with two segments. This significantly reduces the problem size and Bhatia can always get an answer. CG4SR can run with any number of two segments because of the lazy generation of the paths. And this is so effective than we are actually faster than Bhatia with a two-segment limit.

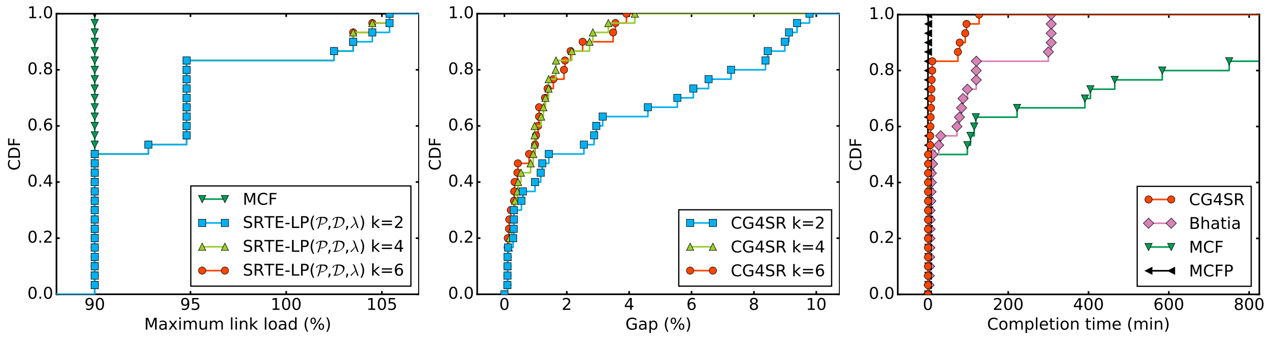
Figure 4c also shows that the MCFP variant of MCF can actually compute the optimal value of the MCF quicker than CG4SR. But, as mentioned above, MCFP only provides the maximum link utilisation of the MCF formulation but not a set of paths satisfying it. Hence, in practice, MCFP can only be used to provide lower bounds which, as we showed in Figure 4a, are worse than the ones provided by our algorithm.

CG4SR scales better than MCF and Bhatia. Our model has fewer variables than the MCF and Bhatia formulations. The size of our model is the number of paths that were generated. The size of MCF considers how much of each demand can be placed on each edge. Therefore, the number of variables is the number of edges multiplied by the number of demands. Bhatia considers for each demand, one detour (i.e., two segments) through a single node of the graph. Its number of variables is thus the number of nodes multiplied by the number of demands. We observe that CG4SR is more scalable because it considers at worst 60 times fewer possibilities than Bhatia and 200 times fewer than MCF. This explains why CG4SR is faster than Bhatia and MCF. This difference does not change significantly when varying the limit on the number of segments. As can be seen in Figure 5, the number of generated columns seems to grow linearly with the number of demands. Given that the restricted path set is initialized with all the direct paths for every demand, the path-finding process (Algorithm 2) only creates a limited number of additional paths to reach optimality. This also explains why the column generation approach is so efficient in practice, as it only needs to solve the linear program with a number of variables only slightly above the number of demands.

B. Any-time behavior

The previous section shows that we can produce quality solutions with illimited time budget. This section evaluates the quality of CG4SR solutions over time. We compare CG4SR to the heuristic approaches DEFO, SRLS and also to Bhatia.

CG4SR finds a good solution even if only allowed to run for a short amout of time. Figures 6a, 6b and 6c show, for each of the cited solvers, a CDF of the gap to the SRTED-LP solution for the cited solvers after, respectively 1 minute, 5 minutes and 10 minutes. During these runs, the *maxp* parameter (see Algorithm 2) of CG4SR is fixed to 10. The limit of segments is set to 6, except for Bhatia which limits itself to 2. The quality of a solution is the difference between its current solution and the CG4SR lower bound that



(a) Lower bound

(b) Gap

(c) Execution time (CG4SR is limited to two segments)

Fig. 4: Evaluation of MCF, MCFP, Bhatia and CG4SR without time limit

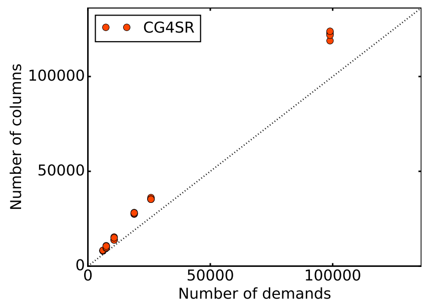


Fig. 5: Number of generated columns over the size of the demand matrices

was computed without time limit and with the same limit on the number of segments.

We see that we are always faster than Bhatia even with limited time spans. SRLS and DEFO are heuristic approaches and therefore are able to quickly find good solutions. Figure 6a shows that CG4SR is already comparable to SRLS and better than DEFO for half of the instances after 1 minute. We see that DEFO initially finds better solutions but CG4SR catches up for most of instances by increasing the timeout in Figure 6b and Figure 6c. The largest instance is not yet solved after 10 minutes and that explains why DEFO is still better.

CG4SR is more robust than SRLS over different sets of demands. SRLS produces good results but this solution is based on local search and can be stuck in a local optimum. We did not observe it on the demand matrices generated by the gravity model. Indeed, the demand volumes are generally much lower than link capacities. This also means that there are many possible ways to reach good solutions, even if the best solution is hard to find. The gravity model is a good match to the traffic engineering problem in ISPs but demand volumes are likely higher in inter-datacenter communication [4]. This also means that there are fewer good solutions. We generated one additional demand matrix for each RocketFuel topology with a low number of large demands requiring 95% of the bandwidth available between their source and destination. Figure 7 shows a CDF of the quality of the solution with a time limit of 10 minutes and a limit of six segments as in Figure 6c. These results confirm that SRLS can be worse than CG4SR when fewer good solutions are available.

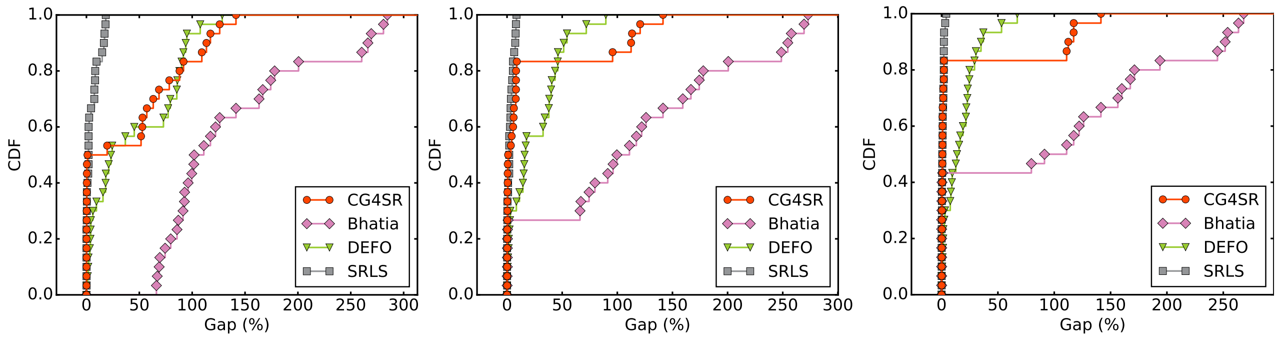
C. *Adjacency segment benefits*

Adjacency segments are important in TE and CG4SR is the first to use them. CG4SR is the first SR traffic engineering model to allow adjacency segments. We evaluate the benefits of adjacency segments on the inter-datacenter network topology of OVH in Europe (described in [29]). This topology has more parallel links than RocketFuel topologies and thus, the OVH topology can really benefit from adjacency segments. We do not have access to the link IGP weights and capacities of OVH. Therefore, for each link bundle we set the capacity of half of the links to 10Gbps and the other half to 2.5Gbps. This simulates the link upgrades on the network. For pairs of nodes with a single link between them, we set the capacity to be ten times bigger. We fix all IGP weights to 1. Five demand matrices were generated for the OVH topology with the gravity model [27].

Figure 8 shows CDFs of the gap (in percentage) between the CG4SR upper and lower bounds over the demand matrices of the OVH topology. We do not limit the execution time and we limit the number of segments to 4. This means that we allow at most one detour through a specific link because one segment is needed for the destination and an adjacency segment has a cost of 2. Even allowing only one link detour halves the load of the maximally loaded link because it utilizes better the parallel links of this topology.

VI. Conclusion

In this paper, we propose CG4SR, a scalable approach that leverages column generation to efficiently solve the traffic engineering problem in networks using Segment Routing. CG4SR enables network operators to exploit both node and adjacency segments with full control over the maximum number of segments while existing techniques only support node segments. Our experiments demonstrate that CG4SR is faster than previously proposed ILP approaches. Compared to heuristics, CG4SR provides a tight bound that ensures the near



(a) Timeout at 1 minute

(b) Timeout at 5 minutes

(c) Timeout at 10 minutes

Fig. 6: Gaps between SRTED-LP$(\mathcal{P}, \mathcal{D}, \lambda)$ and Bhatia, DEFO, SRLS or CG4SR. All solvers are limited to 6 segments.

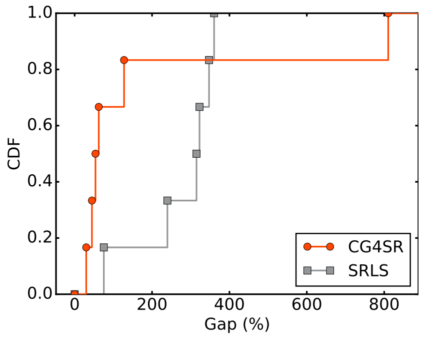


Fig. 7: Gaps between SRTED-LP and SRLS or CG4SR after 10 min with a limit of 6 segments

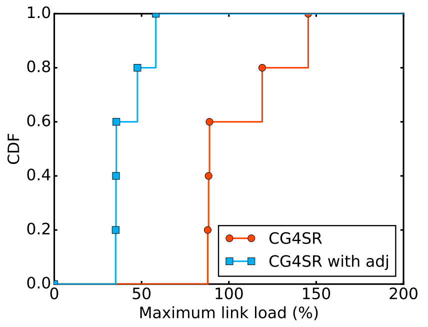


Fig. 8: The CG4SR solutions with or without adjacency segments. The limit of segments is fixed to 4.

optimality of its solution while heuristics are fast but cannot provide any guarantee.

Acknowledgements

This work was supported by the ARC grant 13/18-054 (ARC-SDN) funded by Communaute francaise de Belgique. M. Jadin is supported by a grant from F.R.S.-FNRS FRIA.

References

[1] D. Awduche *et al.*, "Overview and Principles of Internet Traffic Engineering," RFC 3272, 2002.
[2] N. Wang *et al.*, "An overview of routing optimization for internet traffic engineering," *IEEE Communications Surveys & Tutorials*, 2008.
[3] B. Fortz *et al.*, "Traffic engineering with traditional IP routing protocols," *IEEE communications Magazine*, 2002.
[4] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," in *ACM SIGCOMM CCR*, ACM, 2013.
[5] X. Xiao *et al.*, "Traffic Engineering with MPLS in the Internet," *IEEE network*, 2000.
[6] P. Aukia *et al.*, "RATES: A server for MPLS traffic engineering," *IEEE Network*, 2000.
[7] C. Filsfils *et al.*, "The segment routing architecture," in *GLOBECOM*, IEEE, 2015.
[8] C. Filsfils *et al.*, "Segment Routing Architecture," RFC 8402, 2018.
[9] C. Filsfils *et al.*, "IPv6 Segment Routing," in *SIGCOMM demo*, 2017.
[10] T. Schuller *et al.*, "Traffic engineering using segment routing and considering requirements of a carrier IP network.," in *Networking*, 2017.
[11] J. Davidson, "Simplifying Networks through Segment Routing." https://blogs.cisco.com/news/simplifying-networks-through-segment-routing.
[12] J. Brzozowski *et al.*, "Use Cases for IPv6 Source Packet Routing in Networking (SPRING)," RFC 8354, 2018.
[13] C. Filsfils *et al.*, "Resiliency Use Cases in Source Packet Routing in Networking (SPRING) Networks," RFC 8355, 2018.
[14] J. Desrosiers and M. E. Lübbecke, *A Primer in Column Generation*. Boston, MA: Springer US, 2005.
[15] R. K. Ahuja *et al.*, *Network Flows: Theory, Algorithms, and Applications*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.
[16] J. Tantsura, "The critical role of Maximum SID Depth (MSD) hardware limitations in Segment Routing ecosystem and how to work around those," in *NANOG71*, 2017.
[17] R. Hartert *et al.*, "A Declarative and Expressive Approach to Control Forwarding Paths in Carrier-Grade Networks," *SIGCOMM CCR*, 2015.
[18] R. Hartert *et al.*, "Solving segment routing problems with hybrid constraint programming techniques," in *CP*, Springer, 2015.
[19] S. Gay *et al.*, "Expect the unexpected: Sub-second optimization for segment routing," in *IEEE INFOCOM*, 2017.
[20] R. Bhatia *et al.*, "Optimized network traffic engineering using segment routing," in *IEEE INFOCOM*, 2015.
[21] G. Trimponias *et al.*, "On traffic engineering with segment routing in sdn based wans," *arXiv preprint arXiv:1703.05907*, 2017.
[22] G. Desaulniers *et al.*, *Column generation*. Springer Science & Business Media, 2006.
[23] S. Gass, *Linear Programming: Methods and Applications*. Dover Books on Computer Science Series, Dover Publications, 2003.
[24] Y. Zhang *et al.*, "Steering: A software-defined networking for inline service chaining," in *IEEE ICNP*, IEEE, 2013.
[25] N. Spring *et al.*, "Measuring ISP topologies with Rocketfuel," *ACM SIGCOMM CCR*, 2002.
[26] S. Gay *et al.*, "REPETITA: Repeatable Experiments for Performance Evaluation of Traffic-Engineering Algorithms," *arXiv preprint arXiv:1710.08665*, 2017.
[27] M. Roughan, "Simplifying the synthesis of Internet traffic matrices," *ACM SIGCOMM CCR*, 2005.
[28] G. O. LLC, "Gurobi optimizer reference manual," 2018.
[29] F. Aubry *et al.*, "SCMon: Leveraging segment routing to improve network monitoring," in *IEEE INFOCOM*, IEEE, 2016.