HAL
open science

GMPLS Label Space Minimization through Hypergraph Layouts

Jean-Claude Bermond, David Coudert, Joanna Moulierac, Stéphane Pérennes, Ignasi Sau, Fernando Solano Donado

To cite this version:

Jean-Claude Bermond, David Coudert, Joanna Moulierac, Stéphane Pérennes, Ignasi Sau, et al.. GMPLS Label Space Minimization through Hypergraph Layouts. Theoretical Computer Science, 2012, 444, pp.3-16. 10.1016/j.tcs.2012.01.033 . hal-00706260

HAL Id: hal-00706260

https://inria.hal.science/hal-00706260v1

Submitted on 9 Jun 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

GMPLS Label Space Minimization through Hypergraph Layouts

Jean-Claude Bermond[a], David Coudert[a], Joanna Moulierac[a], Stéphane Pérennes[a], Ignasi Sau[b], Fernando Solano Donado[c]

[a] *MASCOTTE, INRIA, I3S, CNRS, University Nice Sophia-Antipolis, France*
[b] *CNRS, LIRMM, Montpellier, France*
[c] *Institute of Telecommunications, Warsaw University of Technology, Poland*

Abstract

All-Optical Label Switching (AOLS) is a new technology that performs packet forwarding without any optical-electrical-optical conversions. In this paper, we study the problem of routing a set of requests in AOLS networks using GMPLS technology, with the aim of minimizing the number of labels required to ensure the forwarding. We first formalize the problem by associating to each routing strategy a logical hypergraph, called a *hypergraph layout*, whose hyperarcs are dipaths of the physical graph, called *tunnels* in GMPLS terminology. We define a cost function for the hypergraph layout, depending on its *total length* plus its *total hop count*. Minimizing the cost of the design of an AOLS network can then be expressed as finding a minimum cost hypergraph layout. We prove hardness results for the problem, namely for general directed networks we prove that it is NP-hard to find a $C \log n$-approximation, where C is a positive constant and n is the number of nodes of the network. For symmetric directed networks, we prove that the problem is APX-hard. These hardness results hold even if the traffic instance is a partial broadcast. On the other hand, we provide approximation algorithms, in particular an $\mathcal{O}(\log n)$-approximation for symmetric directed networks. Finally, we focus on the case where the physical network is a directed path, providing a polynomial-time dynamic programming algorithm for a fixed number k of sources running in $\mathcal{O}(n^{k+2})$ time.

Key words: GMPLS, optical networks, label stacking, hypergraph layout, approximation algorithms, dynamic programming.
2000 MSC: 90B10, 90B18, 68M10, 94C15, 05C85, 90C35.

1. Introduction

All-Optical Label Switching (AOLS) [16] is an approach to route packets transparently and all-optically, thus allowing a speed-up of the forwarding. This very promising technology for the future Internet applications also brings new constraints and new

Email addresses: `Jean-Claude.Bermond@inria.fr` (Jean-Claude Bermond), `David.Coudert@inria.fr` (David Coudert), `Joanna.Moulierac@inria.fr` (Joanna Moulierac), `Stephane.Perennes@inria.fr` (Stéphane Pérennes), `ignasi.sau@lirmm.fr` (Ignasi Sau), `fs@tele.pw.edu.pl` (Fernando Solano Donado)

problems. Due to its flexibility as a control plane and to the fact that it handles traffic forwarding, the Generic MultiProtocol Label Switching (GMPLS) is the most promising architecture to be applied in AOLS-driven networks.

In GMPLS, traffic is forwarded through logical connections called Label Switched Paths (LSPs). When GMPLS is used in packet-based networks, packets are associated with LSPs by means of a label, or tag, placed on top of the header of the packet. In this way, routers - called Label Switched Routers (LSRs) - can treat a flow of packets identified with the same label as a single stream. Labels are stored in the LSR forwarding table and labels are significant only locally at the node, therefore, before retransmitting a packet, the current node must assure that the stored label in the outgoing packets matches the one that the next node has associated for this LSP.

The GMPLS standards allow packets to carry a set of labels in their header, conforming a stack of labels. Even though a packet may contain more than one label, LSRs must only read the first (or top) label in the stack in order to take forwarding decisions. This helps to reduce both the number of labels that need to be maintained on the core LSRs and the complexity of managing data forwarding across the backbone.

Stacking labels and label processing, in general, are standardized by the following set of operations that an LSR can perform over a given stack of labels:

SWAP: replace the label at the top by a new one,

PUSH: replace the label at the top by a new one and then push one or more onto the stack, and

POP: remove the label at the top of the label stack.

This diversity of operations allows the network operator to associate flows of packets differently at different points in the network. The way in which this association is made affects the number of labels stored in the node. As discussed previously, traditionally an LSR stores in its forwarding table an unique label for each LSP, which could induce some scalability and performance problems. Two schemes for reducing the number of labels stored in nodes have been studied so far for GMPLS: *label merging* [7, 18, 20] (not discussed here) and *label stacking* [19, 22]. We proceed to explain label stacking.

When two or more LSPs follow the same set of links, it is possible to push at the top of the stack a common label for this set of LSPs. In this way, subsequent nodes will need to store a single label to forward traffic from different LSPs. We can see this as if they were routed together "inside" a higher-level LSP, which we will denote as a *tunnel*. The tunnel ends once the pushed label is pop, allowing inner LSPs to recover their original label.

Figure 1 represents the general operations needed to configure a tunnel with the use of label stacking. At the entrance of the tunnel, λ PUSH are performed in order to route the λ flows or units of traffic through the tunnel. Then, only one operation (either a SWAP or a POP at the end of the tunnel) is performed in all the nodes along the tunnel, regardless of the inner label. In this figure, a stack of size 2 is used to route the λ units of traffic in one tunnel from node A to node E. The top label l, pushed at the entrance of the tunnel, is swapped and replaced at each hop: by l_1 at node B, by l_2 at node C, and is finally popped at node D. The λ units of traffic, at the exit of the tunnel at node E can end or follow different paths according to their bottom label k_i, for all $i \in \{1, 2, ..., \lambda\}$ in the stack.

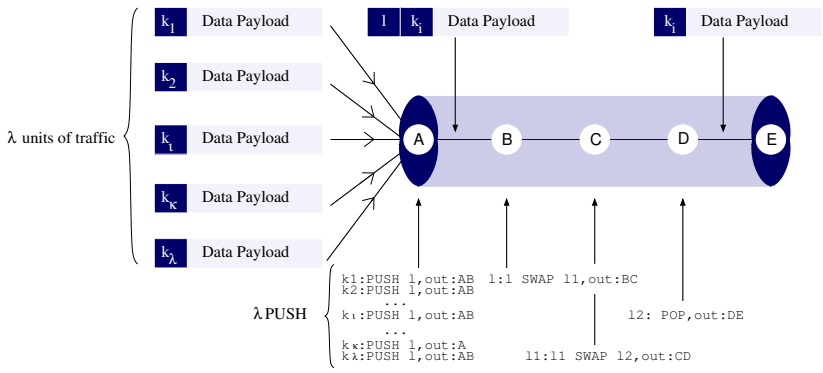

Figure 1: GMPLS operations performed at the entrance and at the exit of a tunnel.

Since a node can only check the top label of a packet, it is not possible for a node to differentiate LSPs within a tunnel. For instance, in the previous example node B cannot distinguish packets from different inner LSPs, since they are all marked with label l in the top of the header. This means that a node must perform the same forwarding and operation in the stack to packets from different LSPs in a tunnel. As a consequence, if a node pops the top label, all inner LSPs "exit" the tunnel. This leads to an important observation: LSPs can exit a tunnel in only one point - the last node of the tunnel. However, an LSP can join a tunnel at any point. For example, an LSP with label δ can join the tunnel in node B by adding the following entry in B: $\delta : PUSH l_1, out : BC$.

GMPLS forwarding implementation in AOLS is made directly at the optical domain. In AOLS, an optical packet switch separates the top label from the packet's header. In order to identify the content of the label (for packet forwarding), the label's optical pulses are matched pulse-by-pulse against a sequence already stored in the switch. The pulse-by-pulse matching is performed by an optical correlator device, which outputs a single-pulse upon a successful matching. As a consequence, an AOLS switch must be able to generate optical pulses for identifying each of the stored labels. Since fast reconfigurable optical correlators do not exist nowadays, an AOLS switch needs to employ several (non-reconfigurable) optical correlators and label generators in parallel for this purpose. This means that an AOLS switch needs one optical correlator and pulse generator for each stored label. Therefore, it is of major importance to reduce the number of employed optical correlators in every node, implying a reduction in the number of labels (as referred in the rest of the paper).

Since the number of labels used for GMPLS forwarding affects the cost of the AOLS architecture, in this paper we mainly focus on the minimization of the number of labels used. In our previous example, the total cost $c(t)$ of the tunnel t from node A to node E in terms of number of labels is $c(t) = \lambda + \ell(t) - 1$, where λ is the number of units of traffic forwarded through this tunnel and $\ell(t)$ is its length in terms of number of hops (which is 4 on this example). We will formally define the cost function of the problem in Section 2.

Previous work and our contribution. The label minimization problem in GMPLS networks has been widely studied in the literature during the last few years [7, 18, 19, 20, 21,

22]. All these articles focus mainly on proposing and analyzing heuristics to the problem, but there is a lack of theoretical results, like computational complexity or bounds on the approximation ratio of the proposed algorithms.

Gupta, Kumar, and Rastogi study the trade-off between the stack depth and the label size in tag switching protocols (see [12, 13, 14]). They calculate lower bounds for different problem instances. Amongst these problems, they propose solutions using at most $2 \cdot n^{1/2}$ labels when the network is a path and $2\Delta \cdot n^{1/2}$ when it is a tree, where n is the number of nodes and Δ the maximum degree of the topology. However, the proposed solutions lay on the assumption of non-practical label distribution protocols for MPLS.

In this article we provide the first theoretical framework for the label minimization problem in general GMPLS networks considering the constraints imposed by real distribution protocol, e.g., RSVP-TE. We translate the problem into finding a set of dipaths in a directed hypergraph. With this new formulation, it turns out that the problem is very similar to classical Virtual Path Layout (VPL) problems originating from ATM networks, where one imposes a constraint on the logical structure and then wishes to minimize either the maximum distance [5] or the average distance [10] traveled by the traffic. Nevertheless, there are two crucial differences between the GMPLS problem that we study and the classical VPL version of ATM networks. The first one is that we can enter a tunnel at any point. For example, on the path $[1, n]$, when all the requests have as destination node n, the optimal solution is simply the tunnel from node i to node n where i is the leftmost source. In that case the problem differs considerably from the classical VPL problems. However for one source the problem is similar to the VPL's ones. The second one is that the cost function we consider takes into account the *sum* of the length and the hop count costs, whereas usually in VPL problems the aim is to minimize the *maximum* value of either the length or the hop count in the network.

However, the approximation algorithms we give and the dynamic approach we use for path topology strongly rely on the already known algorithms for VPL problems.

Organization of the paper. In Section 2 we formally state the problem in terms of hypergraph layouts and fix the notation to be used throughout the article. In Section 3 we prove that for general directed networks it is NP-hard to find a $C \log n$-approximation, where C is a a positive constant and n is the number of nodes of the network. For symmetric directed networks, we prove that the problem is APX-hard[1], and therefore it does not accept a PTAS unless P=NP. In Section 4 we provide approximation algorithms to the problem for both general and symmetric networks, and discuss the gap with the hardness results. In Section 5 we focus on the directed path topology and present a dynamic programming approach solving the problem in polynomial time when the number of sources is fixed. Finally, Section 6 is devoted to conclusions and further research.

Some parts of this paper have been presented in conferences [2, 3].

[1]PTAS denotes the class of problems admitting a polynomial-time approximation scheme that is guaranteed to find a solution whose cost is within a $1 + \varepsilon$ factor of the optimum cost, for any $\varepsilon > 0$. When the solution is only guaranteed within a constant factor of the optimum cost, then the problem belongs to APX. An APX-hard problem does not accept a PTAS, unless P=NP (see for instance [24]).

2. GMPLS Logical Network Design as a Hypergraph Layout Problem

The logical network design problem that we address can be roughly described as follows: we are given a digraph (directed graph) G together with a set of weighted traffic demands (or requests) between couples of vertices in G, and we must find a set of tunnels of minimum cost allowing a routing upon this set of tunnels for all the traffic demands. Note that usually communication networks are symmetric digraphs (i.e. when operators set a link in one direction, they also set the opposite link). So it is interesting to study the symmetric case, which turns out to be computationally easier than the general directed case. Let us now precise each one of the above terms.

A *tunnel* is simply a directed path (or dipath) in G, and due to the technological constraints discussed in Section 1, traffic can enter anywhere in the tunnel but must leave only at the end of the tunnel.

To define the problem formally we need the following notation:

- $G = (V, E)$ is the underlying digraph (which can be symmetric or not) with $|V| = n$.
- $r_{i,j}$ is the request from node $i \in V$ to node $j \in V$, with multiplicity $m_{i,j}$. R is the set of all requests.
- $P(G)$ is the set of all simple dipaths in G.
- t stands for a tunnel, and T is the set of tunnels, that is, $t \in T \subseteq P(G)$.
- ℓ is a length function on the arcs, that is, $\ell : E \to \mathbb{N}^+$.
- A tunnel t has length $\ell(t) = \sum_{e \in t} \ell(e)$ and carries $w(t)$ flows, or as referred in the rest of the paper, $w(t)$ units of traffic.

Note that, a priori, $w(t)$ depends on the routing policy. The cost $c(t)$ of a tunnel t is then $c(t) = w(t) + (\ell(t) - 1)$, and the cost of a set of tunnels T is

$$c(T) = \sum_{t \in T} (w(t) + \ell(t) - 1) . \tag{1}$$

Each tunnel can be modeled as a directed hyperarc on the vertex set of G. This observation naturally leads to the definition of a hypergraph layout.

Definition 1 (Hypergraph layout)**.** *Given a graph G and a set $T \subseteq P(G)$, $H(T)$ is the directed hypergraph with $V(H(T)) = V(G)$, and where for each tunnel $t \in T \subseteq P(G)$ there is a directed hyperarc in $H(T)$ connecting any vertex of t to the end of t. $H(T)$ is called a* hypergraph layout*.*

Note that a hypergraph $H(T)$ defines a virtual topology on G. A hypergraph layout $H(T)$ is said to be *feasible* if for each request $r_{i,j} \in R$ there exists a dipath in $H(T)$ from i to j. The problem can then be simply expressed as finding a feasible hypergraph layout of minimum cost. Let us now rewrite the cost function of Equation (1).

Given a hypergraph layout $H(T)$, let $L(r_{i,j})$ be the number of hyperarcs that request $r_{i,j}$ uses, and let $d_H(i, j)$ be the distance from vertex i to vertex j in $H(T)$. Then the term $\sum_{t \in T} w(t)$ of Equation (1) can be rewritten as $\sum_{r_{i,j} \in R} L(r_{i,j}) \cdot m_{i,j}$ and, since $L(r_{i,j}) \geq d_H(i, j)$, we conclude that in an optimal solution the routing necessarily uses shortest

dipaths in the hypergraph layout. It follows that the cost function of Equation (1) can be rewritten w.l.o.g. as

$$c(T) = \sum_{t \in T} (\ell(t) - 1) + \sum_{r_{i,j} \in R} d_H(i,j) m_{i,j}. \tag{2}$$

The cost of a solution is of bicriteria nature. The first part is the cost of the hypergraph structure; we call it the *total length* of the layout. The second part is the total distance that the traffic travels in the hypergraph; we call it the *total hop count*. Both parts are conflicting. On the one hand, to minimize the hop count, it is enough to take a tunnel connecting any source to any destination. On the other hand, to minimize the total length of the layout, it is enough to consider a minimum arc-weighted hypergraph H such that for each request $r_{i,j} \in R$, vertices i and j lie on the same connected component of H.

Summarizing, the problem can be stated as follows.

> Minimum Cost Hypergraph Layout: Given a digraph G with a length function and a set R of traffic requests, find a feasible hypergraph layout of minimum cost, where the cost of a hypergraph layout is defined as in Equation (2).

If G is a symmetric digraph, the problem is denoted Minimum Cost Symmetric Hypergraph Layout.

Computation of a solution for the example of Figure 2. Consider the path $[s, 2]$ with one source s, $m_{s,1}$ units of traffic destined to node 1 at distance ℓ_1 from s ($\ell_1 - 1$ nodes between s and 1) and $m_{s,2}$ units of traffic destined to node 2 at distance $\ell_1 + \ell_2$ from s. See Figure 2 for an illustration. The optimal solution depends on the values ℓ_i and $m_{s,i}$. Indeed, two solutions have to be examined. In the first solution, a specific tunnel (s, i) is configured for each destination i, giving two tunnels $(s, 1)$ and $(s, 2)$ with a total cost: $(m_{s,1} + \ell_1 - 1) + (m_{s,2} + \ell_1 + \ell_2 - 1) = m_{s,1} + m_{s,2} + 2\ell_1 + \ell_2 - 2$. The second solution is composed of the two tunnels $(s, 1)$ and $(1, 2)$. The requests destined to 2 will first use the tunnel $(s, 1)$ and then the tunnel $(1, 2)$. The traffic carried by $(s, 1)$ is $m_{s,1} + m_{s,2}$ and the traffic carried by $(1, 2)$ is $m_{s,2}$. Therefore, the total cost is $(m_{s,1} + m_{s,2} + \ell_1 - 1) + (m_{s,2} + \ell_2 - 1) = m_{s,1} + 2m_{s,2} + \ell_1 + \ell_2 - 2$. The optimal solution is either the first one if $\ell_1 \leq m_{s,2}$ or the second one if $\ell_1 \geq m_{s,2}$.

We state now a lemma to be exhaustively used in the sequel.

Lemma 1. *In any network, there exists an optimal solution to the* Minimum Cost Hypergraph Layout *problem such that all the traffic units of each request are routed via a unique dipath.*

Proof. Suppose the traffic from node i to node j is routed via different shortest paths. Let P_1 be one of these dipaths. We can reroute the traffic of any other dipath via P_1. The second part of the equation is unchanged and the cost is either unchanged or decreases if some tunnels of length more than 1 used in the other path become empty. □

We note that the results of this paper can be easily generalized if we use in Equation (2) a cost function, where instead of $\ell(t) - 1$ we have a function $p : P(G) \to \mathbb{R}^+$.

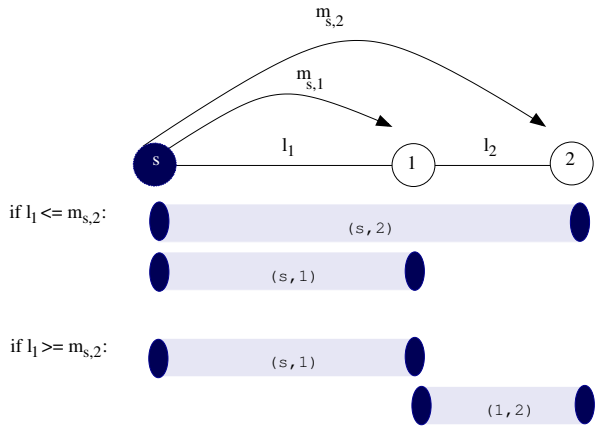

Figure 2: Depending on the values ℓ_1 and $m_{s,2}$, the optimal solution may be composed either of tunnels $(s, 1)$ and $(s, 2)$, or of tunnels $(s, 1)$ and $(1, 2)$.

3. Hardness Results

In this section we give hardness results for the Minimum Cost Hypergraph Layout problem. We distinguish two cases according to whether the underlying network is symmetric or not. We focus on these cases in Sections 3.1 and 3.2.

3.1. General case

Theorem 1. *The* Minimum Cost Hypergraph Layout *problem cannot be approximated within a factor $C \cdot \log n$ for some constant $C > 0$, even if the instance is a partial broadcast*[2]*, unless* $\mathrm{P} = \mathrm{NP}$.

Proof. The reduction is from the Minimum Set Cover problem: given a finite set $\mathcal{S}$ with p elements a_j, $1 \leq j \leq p$, and a collection $\mathcal{C}$ of subsets of $\mathcal{S}$, the aim is to find a subcollection $\mathcal{C}'$ of $\mathcal{C}$ of minimum cardinality that covers all the elements of $\mathcal{S}$.

To a Set Cover instance with sets $S_1, S_2, \ldots, S_k$, with $S_i \subseteq \{a_1, a_2, \ldots, a_p\}$, we associate the following digraph:

- We start with a distinguished node s.
- With each set S_i, $1 \leq i \leq k$, we associate a node v_i and a directed path of length $L + 1$ (L is a parameter to be specified later) from s to v_i.
- With each element a_j, $1 \leq j \leq p$, we associate a vertex u_j and we add the arcs (v_i, u_j) if $a_j \in S_i$.
- The request set is a partial broadcast from s to u_j, $1 \leq j \leq p$, each request with multiplicity 1.

[2]Recall that a *partial broadcast* is a request scheme with all the requests from a vertex s to some of the other vertices.

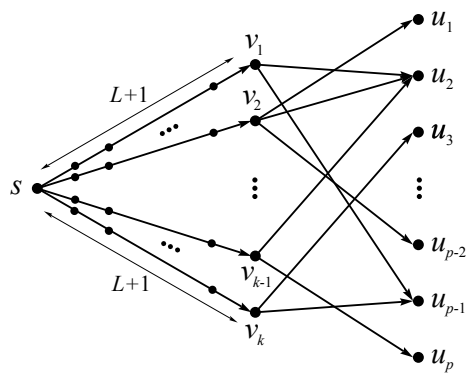

Figure 3: Reduction in the proof of Theorem 1.

This construction is illustrated in Figure 3. Observe that the transformation is done in polynomial time. Let OPT be the optimal cost of the Minimum Cost Hypergraph Layout instance, and let OPT_{SC} be the optimal cost of the Minimum Set Cover instance.

Note that any cover defined by $I \subseteq \{1, 2, \dots k\}$ induces an Hypergraph Layout obtained as follows: we consider a tunnel of length $L+1$ connecting node s to each node $v_i, i \in I$ corresponding to a set taken in the cover. Then for each u_j we set a tunnel from some v_i to u_j, for $i \in I$ such that $a_j \in S_i$. Such a solution has cost $L \cdot |I| + 2p$. In particular, considering an optimal solution to the Minimum Set Cover instance we get

$$OPT \;\leq\; L \cdot OPT_{SC} + 2p. \tag{3}$$

Conversely, consider a feasible optimal layout H of cost S. By Lemma 1, for each u_j, the dipath from s to u_j contains some v_i joined to u_j. Let I be the set of indices i of the v_i obtained in such a way. Then the sets S_i, for $i \in I$, cover $a_1, \dots, a_p$. For each $i \in I$, consider a particular u_j joined by v_i. The cost of the tunnels to route the traffic from s to u_j is exactly $L+2$. Indeed, if this routing uses h tunnels, the total length is $L+2-h$ and the total hop count is h as $m_{s,u_j} = 1$.

To reach the $p - |I|$ vertices u_j not already considered, we may reuse some tunnels. This increases for each node the total cost by at least 2 (one more tunnel with cost $L+1$ or one more tunnel with cost 0 and usage of 2 tunnels). So altogether the cost of the solution is at least $S \geq |I| \cdot (L+2) + 2(p - |I|) = L \cdot |I| + 2p$. Therefore, we have a solution to the Minimum Set Cover with cost $S_{SC} \leq \frac{S-2p}{L} \leq \frac{S}{L}$.

Suppose that $C_1 > 0$ is a constant such that we can approximate in polynomial time the Minimum Cost Hypergraph Layout problem within a factor $C_1 \cdot \log n$. That is, we can find a solution such that $S \leq C_1 \cdot \log n \cdot OPT$. By the above discussion, we can then find in polynomial time a solution to the Minimum Set Cover instance with cost S_{SC} such that

$$S_{SC} \;<\; \frac{S}{L} \;\leq\; \frac{C_1 \cdot \log n \cdot OPT}{L}. \tag{4}$$

Using Equation (3) in Equation (4) we obtain

$$S_{SC} \;<\; \frac{C_1 \cdot \log n \cdot (L \cdot OPT_{SC} + 2p)}{L} \;=\; C_1 \cdot \log n \cdot OPT_{SC} + \frac{2C_1 \cdot p \cdot \log n}{L}. \tag{5}$$

Let now $L = p$. As, w.l.o.g., $OPT_{SC} \geq 2$,

$$S_{SC} \;\leq\; C_1 \cdot \log n \cdot (OPT_{SC} + 2) \leq\; 2C_1 \cdot \log n \cdot OPT_{SC} \tag{6}$$

Therefore, we have obtained a polynomial-time $(2C_1 \cdot \log n)$-approximation algorithm for Minimum Set Cover. On the other hand, in [9, 17] it is proved that Minimum Set Cover is not approximable within a factor $C_3 \cdot \log n$, for some constant $C_3 > 0$, unless P = NP. So for $2C_1 = C_3$, approximating Minimum Cost Hypergraph Layout within a factor $C_1 \cdot \log n$ is also NP-hard. □

3.2. Symmetric case

When the input graph G is symmetric, we can consider G as an undirected graph where the edge $\{i, j\}$ corresponds to the two arcs (i, j) and (j, i).

Theorem 2. *The* Minimum Cost Symmetric Hypergraph Layout *problem is* APX*-hard even if the instance is a partial broadcast. Therefore, it does not accept a* PTAS *unless* P=NP.

Proof. The reduction is from Minimum Steiner Tree problem: given an edge-weighted graph $G = (V, E)$ and a subset $X \subseteq V$, find a connected subgraph Γ with minimum edge-weight containing all the vertices in X. This problem is known to be APX-hard even if the edge-weights are 1 or 2 [6], hence it does not accept a PTAS unless P = NP. We can assume, by subdividing the edges of weight 2, that all edge-weights are equal to 1, and then the objective is to minimize the number of edges.

Given an instance $(G = (V, E), X \subseteq V)$ of Minimum Steiner Tree on n vertices, we build an instance of Minimum Cost Symmetric Hypergraph Layout as follows. We take as underlying digraph G and as request set a partial broadcast from some vertex s in X to all the other vertices in X, all with multiplicity 1. We set all the edge lengths to $L + 1 > 0$, L being a parameter to be specified later.

Let OPT be the optimal cost to the Minimum Cost Symmetric Hypergraph Layout instance, and let OPT_{ST} be the optimal cost to the Minimum Steiner Tree instance, realized by a subgraph Γ. Let $d_\Gamma(s, x)$ denotes the distance from s to x in the graph Γ. Since Γ connects s to all the other vertices in X, it follows that

$$OPT \;\leq\; L \cdot OPT_{ST} + \sum_{x \in X \setminus \{s\}} d_\Gamma(s, x) \;\leq\; L \cdot OPT_{ST} + n^2. \tag{7}$$

Conversely, given any solution to the Minimum Cost Symmetric Hypergraph Layout instance with cost S and realized with a graph Γ_X, we can find a solution to the Minimum Steiner Tree instance (just by taking the edges used by some tunnel) with cost

$$S_{ST} \;\leq\; \frac{S - \sum_{x \in X \setminus \{s\}} d_{\Gamma_X}(s, x)}{L} \;\leq\; \frac{S}{L}.$$

Suppose for the sake of contradiction that there exists a PTAS for the Minimum Cost Symmetric Hypergraph Layout problem. Then, for each $\varepsilon > 0$ we can find in

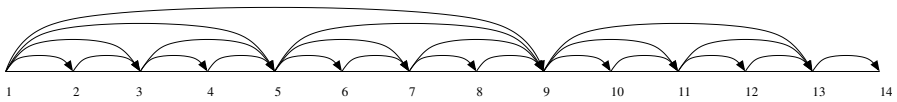

Figure 4: The binary layout depicted above gives a $\log n$-approximation algorithm for Minimum Cost Hypergraph Layout on the path. If the path is undirected, we add the symmetric tunnels.

polynomial time a solution S such that $S \leq (1+\varepsilon) \cdot OPT$. Then, we can find a solution to the Minimum Steiner Tree instance such that

$$S_{ST} \;\leq\; \frac{S}{L} \;\leq\; \frac{(1+\varepsilon) \cdot OPT}{L} \;\leq\; \frac{(1+\varepsilon) \cdot \big(L \cdot OPT_{ST} + n^2\big)}{L}, \tag{8}$$

where we have used Equation (7) in the last inequality. Let now $L = n^3$. Equation (8) becomes

$$S_{ST} \;\leq\; (1+\varepsilon) \cdot OPT_{ST} + \frac{1+\varepsilon}{n}.$$

That is, the existence of a PTAS for Minimum Cost Symmetric Hypergraph Layout would yield a PTAS for Minimum Steiner Tree, which is impossible by [6] unless P = NP. □

4. Approximation Algorithms

In this section we provide approximation algorithms for the Minimum Cost Hypergraph Layout problem. For the sake of presentation, we describe our algorithms when the network is a path, a tree, and a general graph in Sections 4.1, 4.2, and 4.3, respectively. Although the approximation factors are the same for paths and trees, we describe in Section 4.1 a simple and intuitive layout for the path, which differs from the approach of Section 4.2.

4.1. Case of the path

First assume that the path is directed. If $n-1$ is a power of two, we define the following *binary layout*: we connect node 1 to node $(n+1)/2$, node $(n+1)/2$ to node n, and we consider recursively the binary layout for $(n+1)/2$ on the subdipaths $[1,(n+1)/2]$ and $[(n+1)/2, n]$. If $n-1$ is not a power of two, we consider the smallest value $n' > n$ such that $n'-1$ is a power of two. Note that $n' \leq 2n$. We construct the binary layout for the dipath with n' vertices and then delete the last $n'-n$ vertices and their adjacent arcs. The construction is illustrated in Figure 4 for $n = 14$, where we did the binary layout for $n' = 17$ and deleted the vertices 15, 16, 17. It is clear that any request can be routed in this layout with at most $\log n'$ hops, and that the total length of this layout is bounded above by $\log n \cdot \ell([1,n])$, where $\ell([1,n])$ denotes the length of the dipath going from node 1 to node n. Therefore, as $n' \leq 2n$, the cost of this layout is at most $\log 2n \cdot \sum_{r_{i,j}} m_{i,j} + \log 2n \cdot \ell([1,n])$. We now distinguish two cases.

Consider first the case where the set of requests covers all the arcs of the path (an arc $(u, u+1)$ is covered by a request $r_{i,j}$ if $i \leq u < u+1 \leq j$). The total cost of a tunnel is at least $(\ell(t) - 1 + 1)$, as each tunnel carries at least one request. As the set of requests – and so the set of tunnels – covers all the arcs of the path, a lower bound

for the minimum cost is $\sum_e \ell(e) = \ell([1,n])$. Another trivial lower bound is $\sum_{r_{i,j}} m_{i,j}$. Therefore, $\frac{1}{2}\left(\ell([1,n]) + \sum_{r_{i,j}} m_{i,j}\right)$ is also a lower bound, and so using the binary layout in the whole path yields a $2\log(2n)$-approximation.

If the set of requests does not cover all the arcs, we consider the *span* of an instance as the minimum (in terms of length) set of disjoint intervals of the path such that any request can be routed using only one of these intervals. Each arc of these intervals is covered by at least one request included in the interval, so we can apply the binary layout described above for any interval. We obtain for an interval of length n_i a $2\log(2n_i)$-approximation, and thus a $2\log(\max_i(n_i)) < 2\log(2n)$ approximation for our problem.

If the path is undirected (or equivalently, a symmetric directed path), we add to the binary layout (defined analogously in the span of the instance) all the symmetric tunnels, hence multiplying the total length by two and keeping the total hop count constant. Summarizing,

Proposition 1. *When the network is a path, there exists a polynomial-time approximation algorithm for the* Minimum Cost Hypergraph Layout *problem with an approximation ratio* $\mathcal{O}(\log n)$.

4.2. Case of the tree

In [5] the authors studied the design of virtual topologies in ATM networks. Their model deals with point-to-point connections in the virtual graph, whereas in the Minimum Cost Hypergraph Layout problem, a tunnel can carry more than one request. Nevertheless, we can use the results of [5] to obtain good approximation algorithms. Namely, we are interested in the following result which establishes the trade-off between the maximum load c and the diameter of a virtual topology allowing to route an "all-to-all" (in the sense that each node sends traffic to all the nodes reachable from it) traffic in a general tree.

Theorem 3 (Bermond *et al.* [5])**.** *In a directed tree on n nodes such that each node sends traffic to all the nodes reachable from it, for each value of $c \geq 1$ there exists a virtual topology allowing to route all traffic with diameter at most $10c \cdot n^{\frac{1}{2c-1}}$ and load at most c. In addition, such a virtual topology can be constructed in polynomial time.*

In particular, if we set $c = \frac{\log n+1}{2}$, Theorem 3 implies that we can find in polynomial time a virtual topology with load $\mathcal{O}(\log n)$ and diameter at most $(5\log n + 5) \cdot n^{\frac{1}{\log n}} = 10\log n + 10 = \mathcal{O}(\log n)$.

Consider a general directed tree and suppose first that the instance of the Minimum Cost Hypergraph Layout problem is such that each node sends traffic to all the nodes reachable from it. Each arc must be used by some tunnel, and so like for paths, one lower bound is $\frac{1}{2}\left(\sum_{e\in E} \ell(e) + \sum_{r_{i,j}} m_{i,j}\right)$.

In the layout described above, each arc is used at most $\frac{\log n+1}{2}$ times, and therefore the total length of this layout is $\mathcal{O}(\log n \cdot \sum_{e\in E} \ell(e))$. Since the diameter is also $\mathcal{O}(\log n)$, the total hop count is $\mathcal{O}(\log n \cdot \sum_{r_{i,j}\in R} m_{i,j})$. So altogether, we have an $\mathcal{O}(\log n)$-approximation.

Suppose now that the traffic instance is a general one. Similarly to Section 4.1, we define the *span* of an instance as a minimum set of subtrees such that any request can

be routed within one of these trees. Then, we apply the layout of Theorem 3 to each connected component of the span, obtaining the same approximation ratio. Finally, for symmetric trees, we just multiply the length of the layout by 2 by adding the symmetric tunnels, as we did in Section 4.1. Summarizing,

Proposition 2. *When the network is a tree, there exists a polynomial-time approximation algorithm for the* Minimum Cost Hypergraph Layout *problem with an approximation ratio* $\mathcal{O}(\log n)$.

4.3. General network

Whereas the approximation algorithms described in Sections 4.1 and 4.2 have the same approximation ratios in general and symmetric paths or trees (which is not surprising, as in a tree the only path from i to j is also the only path from j to i if it exists), we shall see in this section that it is not the case in a general network. Namely, the problem seems much easier to approximate in symmetric networks.

Let us introduce the following problem, that will be used in the approximation algorithms presented in this section: in the Minimum Generalized Steiner Network problem, we are given a graph $G = (V, E)$, a weight function $w : E \to \mathbb{N}$, a capacity function $c : E \to \mathbb{N}$. Recall that $r_{i,j}$ is the request from node $i \in V$ to node $j \in V$, with multiplicity $m_{i,j}$. The objective is to find a *Steiner network* over G that satisfies all the requirements and obeys all the capacities, i.e., a function $f : E \to \mathbb{N}$ such that, for each edge $e \in E$, $f(e) \leq c(e)$ and, for any pair of nodes i and j, the number of edge-disjoint paths between i and j is at least $m_{i,j}$, where for each edge e, $f(e)$ copies of e are available. The aim is to minimize the cost of the network, i.e., $\sum_{e \in E} w(e) \cdot f(e)$. If the input graph G is undirected, the problem is approximable within $\mathcal{O}(\log r_{\max})$, where $r_{\max}$ is the maximum requirement [11], and within a constant factor 2 when all the requirements are equal [15]. The directed version of the problem is approximable within $\mathcal{O}(n^{2/3} \log^{1/3} n)$ [8].

Symmetric network. Suppose first that the network is symmetric. Given an instance (G, ℓ, R) of Minimum Cost Hypergraph Layout in a general symmetric network, we build an instance of the associated Minimum Generalized Steiner Network problem as follows. We take as underlying graph G itself, and we take as weight function the length function of G, that is, $w(e) = \ell(e)$ for all $e \in E(G)$. For $i, j \in V(G)$, we set $r_{i,j} = 1$ whenever $m_{i,j} > 0$, and $r_{i,j} = 0$ otherwise. Finally, we set $c(e) = +\infty$ for all $e \in E(G)$.

Let F be an optimal solution to this Minimum Generalized Steiner Network instance (note that F may be disconnected), and let $\ell(F) = \sum_{e \in E(F)} \ell(e)$. The following easy observation will be useful: since F is the minimum (in terms of total edge-length) subgraph of G such that any couple source-destination lies on the same connected component, the total length of any solution to the Minimum Cost Hypergraph Layout problem is at least $\ell(F)$. Using the algorithm of [15], we can find in polynomial time a Steiner network F' with $\ell(F') \leq 2 \cdot \ell(F)$. Since the edge capacities are set to $+\infty$, we can assume that such a Steiner network F' is a forest. The layout is then obtained by applying the algorithm described in Section 4.2 to each connected component of F'.

It is clear that the hop count of this layout is at most $\mathcal{O}(\log n)$ times the lower bound $\sum_{r_{i,j} \in R} m_{i,j}$. On the other hand, the total length of this layout is $\mathcal{O}(\log n \cdot \ell(F')) =$

$\mathcal{O}(\log n \cdot \ell(F))$. Since the total cost of any layout is lower-bounded by $\frac{1}{2}\left(\ell(F) + \sum_{r_{i,j}} m_{i,j}\right)$, the $\mathcal{O}(\log n)$-approximation follows. Summarizing,

Theorem 4. *In a general symmetric network, there exists a polynomial-time approximation algorithm for the* Minimum Cost Symmetric Hypergraph Layout *problem with an approximation ratio* $\mathcal{O}(\log n)$.

General directed network. In a not necessarily symmetric network, if we follow the same approach as in the symmetric case, the assumption that the graph F' (the solution to the Minimum Generalized Steiner Network) is a directed forest does not hold anymore, and therefore we cannot apply the layout of Section 4.2 directly to F'.

To overcome this problem, we proceed as follows: suppose that F' is connected, otherwise we proceed independently in each connected component. We partition F' into strongly connected components $F_1, \ldots, F_l$. Note that this partition can be found in linear time [23]. Then, if we shrink each F_k to a single vertex, for $1 \leq k \leq l$, we obtain a directed acyclic graph (a DAG for short). We remove arcs from this DAG until we obtain a directed tree T such that all the requests can be routed using only edges from T and F_k, $1 \leq k \leq l$.

For $k = 1, \ldots, l$, we select in component F_k an arbitrary distinguished node v_k, and let T_k^{out} and T_k^{in} be two directed spanning trees of F_k such that T_k^{out} is routed at v_k and such that v_k is reachable in T_k^{in} from any vertex in F_k. Note that T_k^{out} and T_k^{in} exist and can be efficiently computed since F_k is strongly connected [23].

The routing within each of the $2l+1$ trees T and $T_k^{\text{out}}, T_k^{\text{in}}$, $1 \leq k \leq l$, is carried out according to the layout described in Section 4.2, whose diameter (in each tree) is at most $\log n$.

Then our routing strategy is the following. If $m_{i,j} > 0$ and i, j lie on the same component F_k, we send the request from i to v_k using the arcs of T_k^{in}, and then from v_k to j using the arcs of T_k^{out}. Otherwise, if i lies on a component T_k and j lies on a component $T_{k'}$ with $k \neq k'$, we send the request from i to v_k using the arcs of T_k^{in}, then from v_k to $v_{k'}$ using the arcs of T, and finally from $v_{k'}$ to j using the arcs of $T_{k'}^{\text{out}}$.

Using the above routing scheme, each request is routed either with at most $2\log n$ hops (if source and destination lie on the same connected component of T) or at most $3\log n$ (otherwise).

Recall that the best approximation ratio to the Minimum Generalized Steiner Network problem is $\mathcal{O}(n^{2/3}\log^{1/3} n)$ [8], and therefore the total length of the obtained layout is at most $\mathcal{O}(n^{2/3}\log^{1/3} n)$ times the total length of an optimal one. The layout used in each tree introduces just a multiplicative term to the total length bounded by $\mathcal{O}(\log n)$ (see Section 4.2). On the other hand, by the arguments above the total hop count of this layout is at most $\mathcal{O}(\log n)$ times the total hop count of an optimal layout. Summarizing, we obtain an $\mathcal{O}(n^{2/3}\log^{4/3} n)$-approximation.

Theorem 5. *In a general network, there exists a polynomial-time approximation algorithm for the* Minimum Cost Symmetric Hypergraph Layout *problem with an approximation ratio* $\mathcal{O}(n^{2/3}\log^{4/3} n)$.

5. The Hypergraph Layout Problem on the Path

In this section we focus on the case when the underlying digraph is a directed path. Our approach consists in a dynamic programming algorithm that computes partial solutions induced on subdipaths of the original dipath. We provide the details for one and two sources in Sections 5.1 and 5.2, respectively, and then we present the general ideas in Section 5.3 to generalize the problem to a fixed number k of sources on the path.

5.1. Case of a single source

We present in this section the algorithm for a single source (a similar approach has been used in [10] with a different objective).

First, let us introduce some notations that will be useful in the sequel.

- Nodes are numbered from left to right $1, \ldots, n$. We denote by $[i, j]$ the subdipath from node i to node j (with $i < j$) and since the path is directed, we assume w.l.o.g. that the source is located in the leftmost node of the path (node 1).
- We denote by $OPT[i, j]$ the cost of an optimal solution for the dipath $[i, j]$ with a unique source located at i and sending to a node u, $i < u \leq j$ a request of multiplicity $m_{i,u} = m_{1,u}$.
- For a given node i and an interval $[u, v]$, let $\alpha(i)$ be the rightmost endvertex in $[u, v]$ of a tunnel starting in i (said otherwise, $(i, \alpha(i))$ is the longest tunnel issued from i and ending in $[u, v]$).

The first crucial observation is that the structure of the tunnels in an optimal solution is *non-crossing*, i.e., two tunnels can only intersect in an optimal solution if one is strictly inside the other, as stated in the following lemma.

Lemma 2 (Gerstel *et al.* [10])**.** *Let the network be a directed path, with a unique source at node i and with requests to nodes in $[i, j]$. The set of tunnels T of an optimal solution for* Minimum Cost Hypergraph Layout *is such that, if $(i, \alpha(i))$ is the longest tunnel from i to $[i, j]$ ($\alpha(i) \leq j$), then there is no tunnel (k, l) in T with $i \leq k < \alpha(i) < l \leq j$.*

Proof. Suppose there exists such a tunnel (k, l) (see Figure 5). As $\alpha(i)$ is the rightmost node, then $k \neq i$, otherwise (i, l) would have been longer than $(i, \alpha(i))$. Therefore, the number of consecutive tunnels from i to l, namely $h(i, l)$, satisfies $h(i, l) \geq 2$. Consider the set of tunnels T' obtained from T by deleting the tunnel (k, l) and adding, if it does not exist, the tunnel $(\alpha(i), l)$.

Any request from node i to some node u in $[l, j]$ which was routed via the tunnel (k, l) is now routed till l through two consecutive tunnels $(i, \alpha(i))$ and $(\alpha(i), l)$. It is an admissible solution whose cost satisfies:

$$c(T') \leq c(T) - \lambda_l h(i, l) - (l([k, l]) - 1) + 2\lambda_l + l([\alpha(i), l]) - 1,$$

where λ_l is the number of requests arriving at l or transiting via l. As $h(i, l) \geq 2$ and $\ell([\alpha(i), l]) < \ell([k, l])$, $c(T') < c(T)$. □

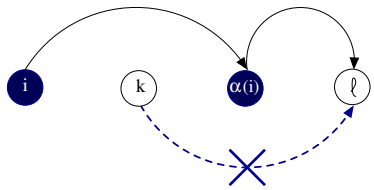

Figure 5: Case of a single source and the non-crossing property.

Lemma 2 leads to the following approach: consider the rightmost tunnel originating from the source, node 1 and assume it ends at node $\alpha(1)$. As there is no tunnel crossing $\alpha(1)$, all the requests for nodes in $[\alpha(1)+1, n]$ have to be routed first by tunnel $(1, \alpha(1))$ and so can be considered as emitted by a source at node $\alpha(1)$. Therefore, we can split the problem into two subproblems: find an optimal solution for the requests to $[1, \alpha(1)-1]$ and an optimal solution for the dipath $[\alpha(1), n]$ with source at $\alpha(1)$. This approach allows us to compute the optimal solution for a path with n vertices recursively.

Proposition 3. *The cost of an optimal solution $OPT[i,j]$ for problem* Minimum Cost Hypergraph Layout *on the dipath $[i,j]$ with source i may be expressed as follows:*

$$OPT[i,j] = \min_{i<\alpha(i)\leq j} C_{\alpha(i)}[i,j] \tag{9}$$

with

$$C_{\alpha(i)}[i,j] = \left(\sum_{k=\alpha(i)}^{j} m_{1,k} + \sum_{e\in E([i,\alpha(i)])} \ell(e) - 1 \right) + OPT[i,\alpha(i)-1] + OPT[\alpha(i),j].$$

Proof. By Lemma 2, let $\alpha(i)$ be the rightmost node in $[i,j]$ from i in an optimal solution. Then, the cost of the solution is the sum of the cost of the tunnel $(i,\alpha(i))$ equals $\sum_{k=\alpha(i)}^{j} m_{1,k} + \sum_{e\in E([i,\alpha(i)])} \ell(e) - 1$ plus the cost of an optimal solution on the subpath $[i,\alpha(i)-1]$ and the cost of an optimal solution on the subpath $[\alpha(i),j]$ with source in $\alpha(i)$, that is, $C_{\alpha(i)}[i,j]$. Then, $OPT[i,j]$ takes the $\alpha(i)$ in $[i,j]$ that minimizes the value $C_{\alpha(i)}[i,j]$. □

Theorem 6. *Let the network be a directed path $[1,n]$ with a unique source at node 1, then an optimal solution of the* Minimum Cost Hypergraph Layout *problem can be computed in $\mathcal{O}(n^3)$ time.*

Proof. The algorithm proceeds as follows.

- First, it computes optimal solutions for dipaths of length 1, namely $OPT[i,i+1] = m_{1,i+1} + \ell([i,i+1]) - 1$.
- Then, it computes solutions for dipaths of length 2, of the form $[i,i+2]$, using the values already computed as $OPT[i,i+2] = \min\{C_{i+1}, C_{i+2}\}$ where $C_{i+1} = m_{1,i+1} + m_{1,i+2} + \ell([i,i+1]) - 1 + OPT[i+1,i+2]$ and $C_{i+2} = m_{1,i+2} + \ell([i,i+2]) - 1 + OPT[i,i+1]$.

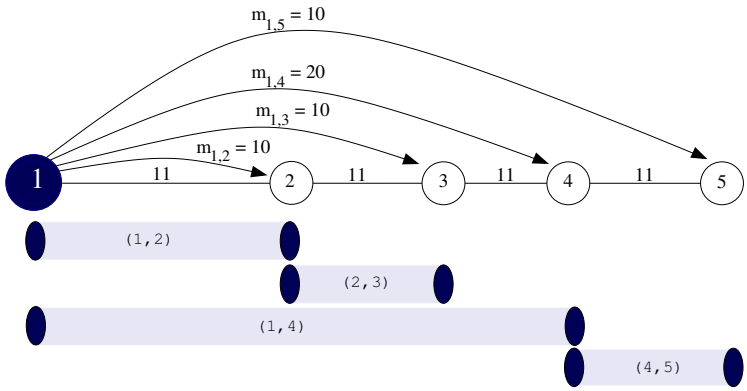

Figure 6: An example with its optimal solution.

- Then, it computes solutions for dipaths of length $3, 4, \ldots$ until dipath $[1, n-1]$ (see example of Figure 6).

Altogether we have to compute $\frac{n(n-1)}{2}$ values and each computation needs $\mathcal{O}(n)$ operations. Indeed, note that to compute the $\sum_{k=\alpha(i)}^{j} m_{1,k}$, we use a table of size n containing partial sum of the weights $M[j] = \sum_{i=2}^{j} m_{1,i}$, and so $\sum_{k=\alpha}^{\beta} m_{1,k} = M[\beta] - M[\alpha - 1]$.

So, we can compute the optimal solution using dynamic programming, with time complexity $\mathcal{O}(n^3)$ and space complexity $\mathcal{O}(n^2)$. □

Illustration on an example. Let us show how the algorithm works on the instance depicted on Figure 6. All the lengths are equal to 11, $m_{1,2} = m_{1,3} = m_{1,5} = 10$, and $m_{1,4} = 20$.

First, $OPT[1,2]$, $OPT[2,3]$, $OPT[3,4]$, and $OPT[4,5]$ are computed using the following formula: $OPT[i, i+1] = m_{1,i+1} + \ell([i, i+1]) - 1$.

Then, to compute $OPT[1,3]$, $OPT[2,4]$, and $OPT[3,5]$, we need to consider only two values of $\alpha(i)$. For example, for $OPT[1,3]$, the two values $\alpha(1) \in \{2,3\}$ are considered. The optimal solution is obtained with $\alpha(1) = 2$ with a cost of $m_{1,2} + m_{1,3} + \ell([1,2]) - 1 + OPT[1,1] + OPT[2,3] = 50$. Indeed, the solution with $\alpha(1) = 3$ implies a greater cost $m_{1,3} + \ell([1,2]) - 1 + OPT[1,2] + OPT[3,3] = 51$.

The algorithm uses the already computed values $OPT[1,1]$, $OPT[2,3]$, $OPT[1,2]$, $OPT[3,3]$.

Then, we compute $OPT[1,4]$ and $OPT[2,5]$ while considering three values of $\alpha(i)$.

Finally for the computation of the optimal solution on the whole path $[1,5]$, four values of $\alpha(1)$ must be considered:

1. for $\alpha(1) = 2$, $m_{1,2} + m_{1,3} + m_{1,4} + m_{1,5} + 10 + OPT[2,5] = 151$;
2. for $\alpha(1) = 3$, $m_{1,3} + m_{1,4} + m_{1,5} + 21 + OPT[1,2] + OPT[3,5] = 141$;
3. for $\alpha(1) = 4$, $m_{1,4} + m_{1,5} + 32 + OPT[1,3] + OPT[4,5] = 132$;
4. for $\alpha(1) = 5$, $m_{1,5} + 43 + OPT[1,4] = 154$;

	$s=1$	2	3	4	5
$s=1$	0	20	50	101	132
			$\alpha(1)=2$	$\alpha(1)=3$	$\alpha(1)=4$
2	-	0	20	61	91
				$\alpha(2)=4$	$\alpha(2)=4$
3	-	-	0	30	60
					$\alpha(3)=4$
4	-	-	-	0	20
5	-	-	-	-	0

Table 1: Computation of the tables OPT for the optimal solution of the instance on Figure 6.

and so the minimum is 132 obtained with $\alpha(1) = 4$.

From the table OPT showing the optimal costs for all the subpaths (Table 1), the set of tunnels can be found. Indeed, the optimal solution for the whole path $[1, 5]$ has cost 132 for $\alpha(1) = 4$. Thus, the optimal solution is composed of a tunnel $(1, 4)$ and of optimal solutions for the subpaths $[1, 3]$ and $[4, 5]$. The first subsolution has a minimum cost for $\alpha(1) = 2$. This gives tunnels $(1, 2)$, $(2, 3)$. The optimal solution for the subpath $[4, 5]$ is obtained for $\alpha(4) = 5$ implying the tunnel $(4, 5)$.

Finally, the optimal solution is composed of tunnels $(1, 2)$, $(2, 3)$, $(1, 4)$, and $(4, 5)$.

Closed formula when the requests and the lengths are uniform. In the special case where both the multiplicities of the requests and the lengths of the arcs are all 1, we give a closed formula of the cost of an optimal solution, as stated in the following proposition. The proof can be found in [4].

Proposition 4. *Let the network be a path $[1, n]$ with $n = 2^q + r$, where $0 \leq r < 2^q$, such that $\ell([i, i + 1]) = 1$ for all $i \in [1, n - 1]$, and with a unitary distribution, that is, for all $\forall i \in [2, n], m_{1,i} = 1$. Then the cost of an optimal solution is $2^q(q - 1) + 1 + (q + 1)r$.*

5.2. Case of two sources

We use a dynamic program similar to the one used for the single source case in Section 5.1, but slightly more complicated.

Following a referee's suggestion, we give only the idea of the proof. The complete proof with all details can be found in [4]. We then show how dynamic programming can be used to get an $\mathcal{O}(n^4)$ algorithm.

5.2.1. Key idea of the algorithm

Let s_1 and s_2 be the two sources with $s_1 < s_2$ and let $OPT(s_1, s_2; [s_1, n])$ the cost of an optimal solution with the 2 sources s_1 and s_2 on the dipath $[s_1, n]$. Like for one source, we will use dynamic programming and decompose the problem into two subproblems on smaller instances or on problems with only one source. For this decomposition, let us denote as i and j $(i \leq j)$ the two sources, one carrying the traffic of s_1, and the other, the traffic of s_2. There are three kinds of subproblems:

- one identical to the original problem, on a smaller subpath starting at i and whose optimal solution is denoted as $OPT(i, j; [i, u])$,

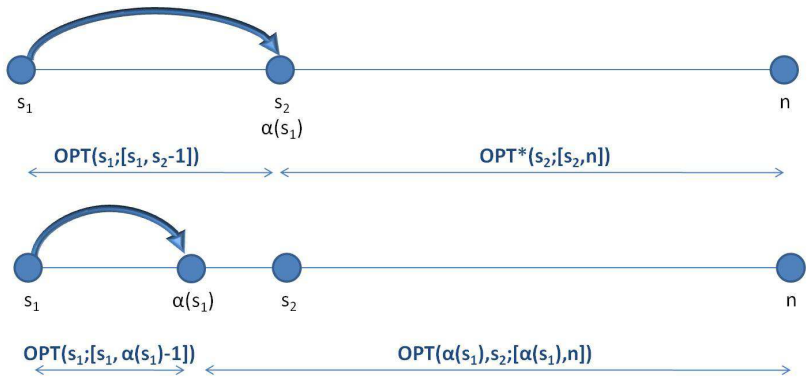

Figure 7: Decomposition of the problem with two sources with $\alpha(s_1) \leq s_2$.

- and another called *modified* problem, where the traffic is destined only for nodes after the second source j and whose optimal solution is denoted as $OPT'(i, j; [j, u])$.
- If $i = j$, we have a subproblem with only one source carrying the traffic of both s_1 and s_2. The optimal solution is denoted as $OPT^*(i; [i, u])$.

The key ideas to solve the *modified* problem are as follows: in an optimal solution, if $\alpha(i) > j$, then the traffic destined to the nodes after $\alpha(i)$ is routed via the tunnel $(i, \alpha(i))$[3] (see Lemma 3 after). If $\alpha(i) < j$ and $j \neq s_2$, the problem differs from the classical VPL problem as we can inject the traffic from i destined to nodes after j in the tunnel ending in j. Then for the nodes after j, j will act as a unique source carrying both traffic.

Let us explicit the first decompositions of the problem when $i = s_1$ and $j = s_2$.

If $\alpha(s_1) < s_2$, the problem is decomposed in two subproblems: (1) with one source s_1 on the dipath $[s_1, \alpha(s_1)-1]$, i.e., $OPT(s_1; [s_1, \alpha(s_1)-1])$ and (2) with sources $\alpha(s_1)$ (carrying the traffic of s_1) and s_2 on a smaller dipath $[\alpha(s_1), n]$, i.e., $OPT(\alpha(s_1), s_2; [\alpha(s_1), n])$. The first problem with one source has been already treated previously in the Section 5.1, and the second one, is the same as the original problem but on a smaller instance.

If $\alpha(s_1) = s_2$, the two simple subproblems with one source are: (1) the problem with s_1 on the dipath $[s_1, s_2-1]$, i.e., $OPT(s_1; [s_1, \alpha(s_1)-1])$ and (2) the problem with s_2 carrying the traffic from both sources s_1 and s_2 on the dipath $[s_2, n]$, i.e., $OPT^*(s_2; [s_2, n])$. See Figure 7 for an illustration of these two first cases.

Therefore, we consider in the following without loss of generality that $\alpha(s_1) > s_2$. When $\alpha(s_1) > s_2$, the problem is decomposed in two problems with two sources: (1) s_1 and s_2 on dipath $[s_1, \alpha(s_1) - 1]$, i.e., $OPT(s_1, s_2; [s_1, \alpha(s_1) - 1])$ and (2) s_2 and $\alpha(s_1)$ on dipath $[\alpha(s_1), n]$, i.e., $OPT'(s_2, \alpha(s_1); [\alpha(s_1), n])$. The first problem is the same as the original problem on a smaller instance. The second problem is the *modified* problem (as called previously) where we need to consider the two following cases for the traffic from the source s_2 to nodes after $\alpha(s_1)$:

[3]Recall that $(i, \alpha(i))$ is the longest tunnel starting from i in an optimal solution.

1. $\alpha(s_2) < \alpha(s_1)$. We inject the traffic of s_2 into the tunnel $(s_1, \alpha(s_1))$ and now we have a problem with one source $\alpha(s_1)$ with both traffic, i.e., $OPT^*(\alpha(s_1); [\alpha(s_1), n])$. Remark that $\alpha(s_2) \neq \alpha(s_1)$ in an optimal solution, indeed, for traffic towards $\alpha(s_1)$, s_2 inserts directly the traffic in $(s_1, \alpha(s_1))$, and there is no need of tunnel $(s_2, \alpha(s_1))$. More generally, in an optimal solution, there is at most one tunnel ending in one node of the path. Summarizing, this case leads to a known problem with one source.
2. $\alpha(s_2) > \alpha(s_1)$. By Lemma 3, the traffic from s_2 to nodes after $\alpha(s_2)$ is routed via the tunnel $(s_2, \alpha(s_2))$. Therefore, the problem $OPT'(s_2, \alpha(s_1); [\alpha(s_1), n])$ is decomposed into two *modified* subproblems: $OPT'(s_2, \alpha(s_1); [\alpha(s_1), \alpha(s_2) - 1])$ and $OPT'(\alpha(s_1), \alpha(s_2); [\alpha(s_2), n]))$.
   We repeat the decompositions for these two subproblems. For example, for the second subproblem $OPT'(\alpha(s_1), \alpha(s_2); [\alpha(s_2), n]))$, we have to distinguish two cases depending on the position of $\alpha^2(s_1)$.
   - If $\alpha^2(s_1) < \alpha(s_2)$ we inject the traffic of $\alpha(s_1)$ in the tunnel $(s_2, \alpha(s_2))$ and we have a problem with one source $\alpha(s_2)$, i.e., $OPT^*(\alpha(s_2); [\alpha(s_2), n])$.
   - Otherwise, if $\alpha^2(s_1) > \alpha(s_2)$, the traffic is routed via the tunnel $(\alpha(s_1), \alpha^2(s_1))$, and we have two subproblems with two sources: $OPT'(\alpha(s_1), \alpha(s_2); [\alpha(s_2), \alpha^2(s_1) - 1])$ and $OPT'(\alpha(s_2), \alpha^2(s_1); [\alpha^2(s_1), n])$, and so on. See Figure 8 for an illustration.

In summary, the traffic of s_1 is routed to nodes "far away" via tunnels $(s_1, \alpha(s_1))$, $(\alpha(s_1), \alpha^2(s_1))$, $(\alpha^2(s_1), \alpha^3(s_1))$, ... and simultaneously for s_2 via tunnels $(s_2, \alpha(s_2))$, $(\alpha(s_2), \alpha^2(s_2))$, ... till the end or till a node $\alpha^h(s_1) < \alpha^{h-1}(s_2)$ (or $\alpha^{h'}(s_2) < \alpha^{h'}(s_1)$) and the problem becomes a problem with one source in $\alpha^{h-1}(s_2)$ (resp. $\alpha^{h'}(s_1)$).

Let us state and prove formally this property in the following lemma:

Lemma 3. *Suppose we have two nodes i and j, with $i < j$, i carrying the traffic of one source and j the traffic of the other source. Suppose also that the traffic is destined to nodes in $[j, u]$. Let $(i, \alpha(i))$ denotes the longest tunnel from i (with $\alpha(i) \leq u$). If $\alpha(i) > j$, then there exists an optimal solution where the traffic of i destined for the nodes in $[\alpha(i), u]$ uses as first tunnel $(i, \alpha(i))$.*

Proof. Let us suppose that the lemma is not true and let i be the first value for which i is the source and some traffic to $[\alpha(i), u]$ is not carried by $(i, \alpha(i))$.

Note that i is of form $\alpha^h(s_\delta)$ with $\delta = 1$ or 2. So we consider an optimal solution satisfying the lemma for all $\alpha^{h'}(s_\delta)$ for $h' < h$ and if $\delta = 2$, $\alpha^h(s_\delta)$. As the lemma is not true, this solution uses a tunnel (k, l) with $k < \alpha(i) < l < u$ to bring the traffic of i to some node x, with $\alpha(i) < x < u$. By the minimality of i, $k \geq i$. Otherwise, k will carry the traffic of some source and $k = \alpha^{h'}(s_\delta)$. But then $\alpha(k) \leq j < l$ brings a contradiction as the tunnel (k, l) is longer than $(k, \alpha(k))$. Furthermore $k \neq i$, otherwise $(i, \alpha(i))$ would not be the longest tunnel from i. Therefore, $k > i$ and the solution uses $t \geq 1$ tunnels to route the traffic from i to k.

We now reroute the traffic from i to x by using first the tunnel $(i, \alpha(i))$ and then injecting the traffic arrived in $\alpha(i)$ into the tunnel (k, l), and then follow the same route as x in the optimal solution. Doing so, we have increased the cost by $m_{i,x}$ by using $(i, \alpha(i))$ but decreased the cost by at least $tm_{i,x}$ (perhaps more if some tunnel becomes

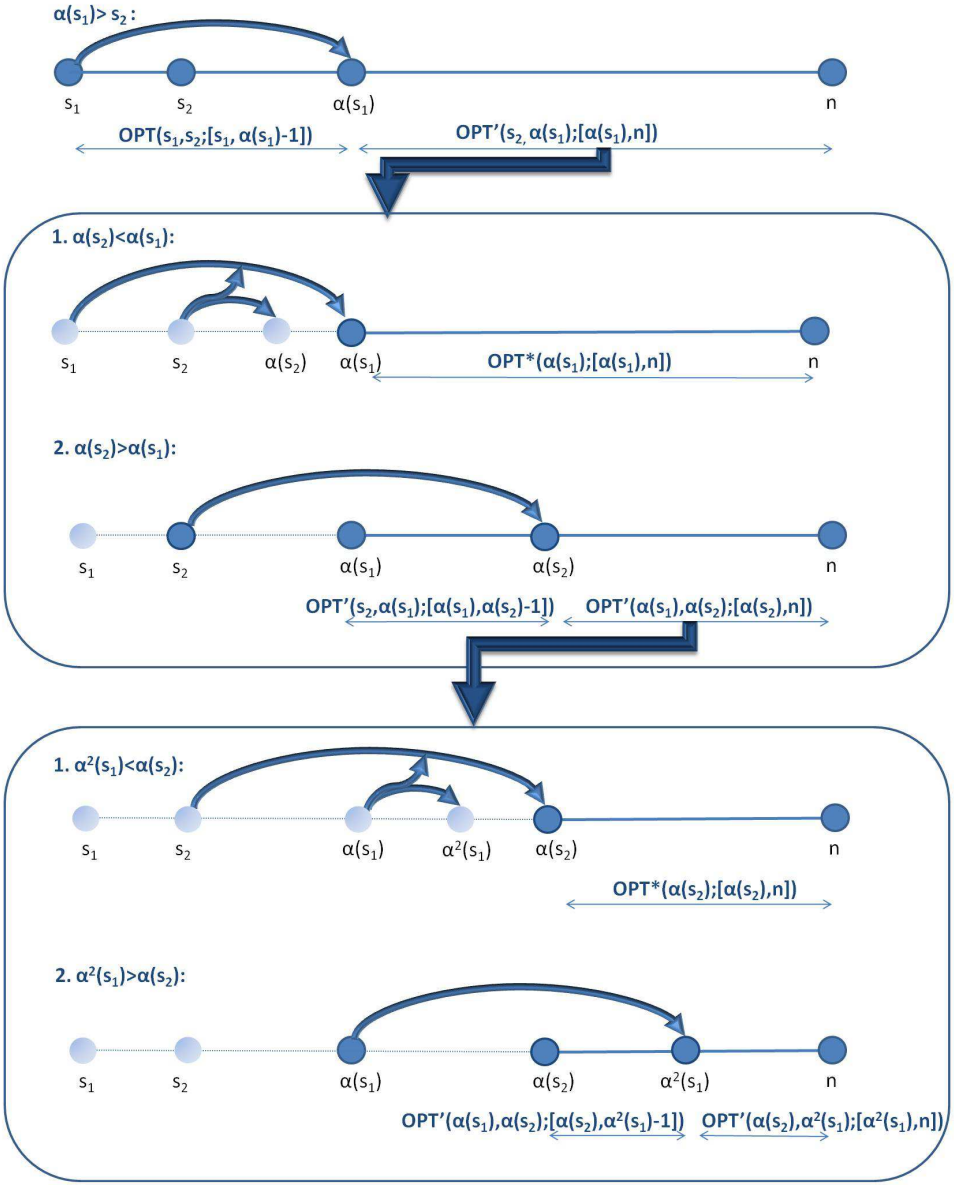

Figure 8: Decomposition of the problem with two sources when $\alpha(s_1) > s_2$.

empty). So the cost of the obtained solution is less than or equal to that of an optimal one, and therefore it is optimal too. □

5.2.2. The algorithm using dynamic programming

The full algorithm to compute $OPT(s_1, s_2; [s_1, n])$ is given in [4], where we show precisely that we have to compute only the two types of values defined above, and where we give the formulae to compute them:

1. $OPT(i, s_2; [i, u])$, the cost of an optimal solution on subdipath $[i, u]$ with sources i and s_2 where $s_1 \leq i \leq s_2$, $i < u \leq n$, and i carries the traffic of s_1. That is, $m_{i,x} = m_{s_1,x}$ for $i < x \leq u$.
2. $OPT'(i, j; [j, u])$, the cost of an optimal solution on subdipath $[j, u]$ with sources i and j, each carrying the traffic of one of the sources and where $s_2 \leq i < j < u \leq n$. Recall in that case there exists a tunnel $(\beta(j), j)$ with $\beta(j) < i$. Two cases appear: Either $\alpha(i) < j$, then the computation is reduced to $OPT^*(j; [j, u])$ or $\alpha(i) > j$, then we have to compute two solutions $OPT'(i, j; [j, \alpha(i) - 1])$ and $OPT'(j, \alpha(i); [\alpha(i), u])$.

In fact we can use dynamic programming and store only $\mathcal{O}(n^3)$ values. More precisely, we need to store the values $OPT(i, s_2; [i, u])$ for $1 \leq i < s_1 - 1$ and $u > i$, the values $OPT(s_1; [s_1, u])$ with $u \leq s_2$ (problems with one source already computed in Section 5.1), the values $OPT^*(j; [j, u])$ for $j \geq s_2$ and with both traffics of s_1 and s_2, and $OPT'(i, j; [j, u])$ where $s_2 \leq i < j \leq u$.

We can fill up the tables by increasing u. Indeed, suppose we have filled the table until $u - 1$. Then we fill up the values of $OPT'(i, j; [j, u])$ starting at $j = u$. Then we fill up the values for $j = u - 1$ and so on until $j = s_2 + 1$. Note that for a given j we need values with either an interval ending in $\alpha(i) - 1 < u$ or with a $j' > j$. Then we fill up $OPT(i, s_2; [i, u])$ by starting at $i = s_2 - 1$ and then decreasing i, the last value to be computed being $OPT(s_1, s_2; [s_1, u])$.

At each step, we need at most $\mathcal{O}(n)$ operations, so the overall complexity is $\mathcal{O}(n^4)$.

5.3. Generalization to an arbitrary number k of sources

In the case of k sources $s_1, s_2, \ldots, s_k$, the computation is similar to the case of two sources except that now we need to compute k different types of values. For $1 \leq h \leq k$, we need to compute $OPT_\pi(i_1, i_2, \ldots, i_h, s_{h+1}, \ldots, s_k; [i_h, u])$, where $i_1 < i_2 < \ldots < i_h < s_{h+1} < \ldots < s_k$, $s_2 < i_1$, $s_3 < i_2$, …, $s_h < i_{h-1}$, and where there exists a tunnel $(\beta(i_h), i_h)$ with $\beta(i_h) \leq i_1$. The node i_j acts as a source and carries the traffic of the source $s_{\pi(j)}$ with π, a permutation of $\{1, 2 \ldots, k\}$. In fact, for $j \geq h + 1$, $i_j = s_j$; so the source i_j carries the traffic of s_j and therefore for $j \geq h + 1$, $\pi(j) = j$. Details are given in [4].

Here again we use dynamic programming filling up the table by increasing u. For a given u, we fill up successively the table with $h = k$, then $h = k - 1$, and so on. For a given h, we start with the greatest i_h.

We need to store $\mathcal{O}(n^{k+1})$ values and we have to compare $\mathcal{O}(n)$ values. So we have an overall complexity of $O(n^{k+2})$ for a path with n nodes and k sources.

6. Conclusions and Further Research

In this paper we modeled a question raised by label minimization in AOLS and GMPLS networks as a hypergraph layout problem. In the unitary case ($m_{i,j} \in \{0, 1\}$) we showed the problem to be closely related to well studied VPL problems. However, the optimization criteria (average hop count and average load) that appear in our problem are among the less studied ones. We provided hardness results for the general directed case and for the symmetric case, and proposed approximation algorithms. More specifically, we gave a $\mathcal{O}(\log n)$-approximation on paths and trees, and observed that in a general network the hardness of our problem is essentially equivalent to the hardness of finding generalized Steiner networks. This is the reason why closing the approximability gap of our problem is challenging.

In the multi-sources case, we presented a dynamic program on the path that is polynomial when the number of sources is fixed. Namely, our algorithm runs in time $\mathcal{O}(n^{k+2})$ on a path with n nodes and k sources. In view of this running time, it is unlikely that the problem is NP-hard on the path, so finding a polynomial algorithm for an arbitrary number of sources on the path remains open. Likely, extensions of the dynamic program to the case of trees and bounded treewidth networks remain also to be done. The complexity of the problem when the routing is part of the input of the problem (that is, there is a dipath associated with each request) remains open. We want to investigate also the possibility of a constant factor approximation for a general graph when there is a single source. We suspect that the problem may become polynomial-time solvable when there is a single source that sends traffic to all the nodes of the network (note that the reductions of Section 3 do not apply to this case), but we have not been able to prove it. Last, we believe that more general approximation results can be given for low dimension Euclidean metric graphs using the classical Arora paradigm [1].

Acknowledgments. This work has been partly funded by the ANR JCJC DIMA-GREEN and the project "Optimization Models for NGI Core Network" (Polish Ministry of Science and Higher Education, grant N517 397334).

We thank the anonymous referees for their very helpful comments.

References

[1] S. Arora. Nearly Linear Time Approximation Schemes for Euclidean TSP and other Geometric Problems. In *Proc. of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 554–563, 1997.

[2] J.-C. Bermond, D. Coudert, J. Moulierac, S. Pérennes, H. Rivano, I. Sau, and F. Solano Donado. MPLS label stacking on the line network. In *Proc. of IFIP Networking*, volume 5550 of *LNCS*, pages 809–820, 2009.

[3] J.-C. Bermond, D. Coudert, J. Moulierac, S. Pérennes, I. Sau, and F. Solano Donado. Designing Hypergraph Layouts to GMPLS Routing Strategies. In *Proc. of the 16th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, volume 5869 of *LNCS*, pages 57–71, 2009.

[4] J.-C. Bermond, D. Coudert, J. Moulierac, S. Pérennes, I. Sau, and F. Solano Donado. GMPLS Label Space Minimization through Hypergraph Layouts. Research Report RR-7071, INRIA, oct 2009.

[5] J.-C. Bermond, N. Marlin, D. Peleg, and S. Pérennes. Directed virtual path layouts in ATM networks. *Theoretical Computer Science*, 291(1):3–28, 2003.

[6] M. Bern and P. Plassmann. The Steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32(4):171–176, 1989.

[7] S. Bhatnagar, S. Ganguly, and B. Nath. Creating Multipoint-to-Point LSPs for traffic engineering. *IEEE Communications Magazine*, 43(1):95–100, 2005.

[8] M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation algorithms for directed Steiner problems. In *Proc. of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 192–200, 1998.

[9] U. Feige. A Threshold of $\ln n$ for Approximating Set Cover. *Journal of the ACM*, 45(4):634–652, 1998.

[10] O. Gerstel, A. Wool, and S. Zaks. Optimal layouts on a chain ATM network. *Discrete Applied Mathematics*, 83:157–178, 1998.

[11] M. X. Goemans, A. V. Goldberg, S. Plotkin, D. B. Shmoys, E. Tardos, and D. P. Williamson. Improved approximation algorithms for network design problems. In *Proc. of the 5th annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 223–232, 1994.

[12] A. Gupta, A. Kumar, and R. Rastogi. Exploring the trade-off between label size and stack depth in MPLS routing. In *Proc. of IEEE INFOCOM*, volume 1, pages 544–554, 2003.

[13] A. Gupta, A. Kumar, and R. Rastogi. Traveling with a Pez Dispenser (or, Routing Issues in MPLS). *SIAM Journal on Computing*, 34(2):453–474, 2005.

[14] A. Gupta, A. Kumar, and M. Thorup. Tree based MPLS routing. In *Proc. of the ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 193–199, 2003.

[15] S. Khuller and U. Vishkin. Biconnectivity approximations and graph carvings. *Journal of the ACM*, 41:214–235, 1994.

[16] F. Ramos et al. IST-LASAGNE: Towards all-optical label swapping employing optical logic gates and optical flip-flops. *IEEE J. Sel. Areas Commun.*, 23(10):2993–3011, 2005.

[17] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proc. of the 29th annual ACM Symposium on Theory of Computing (STOC)*, pages 475–484, 1997.

[18] H. Saito, Y. Miyao, and M. Yoshida. Traffic engineering using multiple MultiPoint-to-Point LSPs. In *Proc. of IEEE INFOCOM*, volume 2, pages 894–901, 2000.

[19] F. Solano Donado, R. V. Caenegem, D. Colle, J. L. Marzo, M. Pickavet, R. Fabregat, and P. Demeester. All-optical label stacking: Easing the trade-offs between routing and architecture cost in all-optical packet switching. In *Proc. of IEEE INFOCOM*, pages 655–663, 2008.

[20] F. Solano Donado, R. Fabregat, and J. Marzo. On optimal computation of MPLS label binding for MultiPoint-to-Point connections. *IEEE/ACM Trans. Comm.*, 56(7):1056–1059, 2007.

[21] F. Solano Donado and J. Moulierac. Routing in all-optical label switched-based networks with small label spaces. In *13th Conference on Optical Network Design and Modeling (ONDM)*, page 6p, Braunschweig, Germany, Feb. 2009. IFIP/IEEE.

[22] F. Solano Donado, T. Stidsen, R. Fabregat, and J. Marzo. Label Space Reduction in MPLS Networks: How Much Can a Single Stacked Label Do? *IEEE/ACM Trans. Netw.*, 16(6):1308–1320, 2008.

[23] R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[24] V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2003.