GMPLS Label Space Minimization through Hypergraph Layouts

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

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

GMPLS Label Space Minimization through Hypergraph Layouts

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

N° 7071

Octobre 2009

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--7071--FR+ENG

GMPLS Label Space Minimization through Hypergraph Layouts

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

Thème COM — Systèmes communicants
Projet Mascotte

Rapport de recherche n° 7071 — Octobre 2009 — 35 pages

Abstract: All-Optical Label Switching (AOLS) is a new technology that performs packet forwarding without any optical-electrical-optical conversions. In this report, 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.

Mail: firstname.name@sophia.inria.fr or fs@tele.pw.edu.pl

This work has been partly funded by the European project FET AEOLUS, the ANR JCJC DIMA-GREEN and the project "Optimization Models for NGI Core Network".

* MASCOTTE, INRIA, I3S, CNRS, University Nice Sophia-Antipolis, France
† Graph Theory and Combinatorics Group at Applied Mathematics IV Dept. of UPC, Barcelona, Spain
‡ Institute of Telecommunications, Warsaw University of Technology, Poland

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Stratégies de routage dans les réseaux GMPLS basées sur les représentations par hypergraphes

Résumé : La commutation tout-optique de labels (*All-Optical Label Switching*, AOLS) est une nouvelle technologie permettant le traitement des paquets sans conversion optique-électronique-optique. Dans ce rapport nous étudions le problème de router un ensemble de requêtes dans un réseau AOLS avec l'objectif de minimiser le nombre d'états de routage dans les routeurs tout en utilisant la technologie GMPLS. Nous utilisons la technique de l'empilement de labels permettant la configuration de tunnels GMPLS. Tout d'abord, nous modélisons notre problème en associant à chaque stratégie de routage un hypergraphe dans lequel les hyperarcs représentent les tunnels GMPLS. La fonction de coût associée à l'hypergraphe dépend de la *somme des longueurs* des hyperarcs et du *nombre total de sauts*. Nous prouvons des résultats de complexité pour notre problème; notamment pour les graphes orientés, nous prouvons qu'il est NP-Difficile de trouver une $C \log n$- approximation, où C est une constante positive et n le nombre de sommets du graphe. Pour les graphes orientés symétriques, nous prouvons que le problème est APX-Difficile. Ces résultats de difficulté sont aussi valables dans le cas d'un *broadcast* partiel. Nous donnons de plus des algorithmes d'approximation, en particulier, une $\mathcal{O}(\log n)$-approximation pour les graphes orientés symétriques. Finalement, nous nous intéressons au cas particulier du chemin orienté et présentons un algorithme polynomial en $\mathcal{O}(n^{k+2})$ utilisant la programmation dynamique pour un nombre fixé k de sources.

Mots-clés : GMPLS, réseaux optiques, empilement de labels, hypergraphes, algorithmes d'approximation, programmation dynamique.

1 Introduction

All-Optical Label Switching (AOLS) [15] 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 problems. Indeed, since the forwarding functions are implemented directly at the optical domain, a specific correlator (device) is needed for each optical label processed in the node. Therefore, it is of major importance to reduce the number of employed correlators in every node, implying a reduction in the number of labels (as referred in the rest of the report). 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 protocol to be applied in AOLS-driven networks.

In GMPLS, traffic is forwarded through logical connections called Label Switched Paths (LSPs). When GMPLS is used with packet-based network, packets are associated to 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 distinguish and forward packets.

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 top in the label stack.

The labels stored in the forwarding table are significant only locally at the node and swapped all along the LSP (see Figure 1). Solutions deployed by GMPLS for reducing the number of labels are *label merging* [6, 17, 19] (not discussed here) and *label stacking* [18, 21]. With label stacking, when two or more LSPs follow the same set of links, they can be routed together "inside" a higher-level LSP, henceforth a *tunnel*. In order to setup a tunnel, multiple labels are placed in the packet's header.

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 λ. 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 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

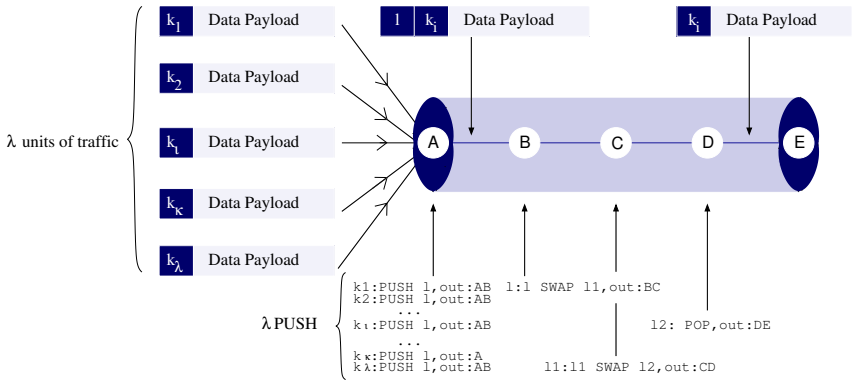

Figure 1: GMPLS operations performed at the entrance and at the exit of a tunnel.

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.

A consequence of the way in which the GMPLS operations can be configured at LSRs is the following: traffic can enter in any node of a tunnel but can exit in only one point, the last node of the tunnel. In other words, when some traffic is carried by a tunnel, it follows the tunnel until its end.

Since the number of labels used for GMPLS forwarding affects the cost of the AOLS architecture, in this report 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 [6, 17, 18, 19, 20, 21]. 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 [11, 12, 13]). 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. We provide hardness results and approximation algorithms for the problem in paths, trees, and general graphs. The approximation algorithms strongly rely on the already known algorithms for VPL problems. Finally, we focus on the path topology, providing a dynamic programming approach that runs in polynomial time for any bounded number of sources.

Organization of the report 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 report 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 [23]).

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 traffic demands (or requests) between couples of vertices in G, and we must find a set of tunnels of minimum cost allowing and a routing upon this set of tunnels for all the traffic requests. 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 $m_{i,j}$, (the traffic being differentiated, $m_{i,j}$ represents the number of different flows from i to j and j is not necessarily the final destination of the request $r_{i,j}$). 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 report, $w(t)$ 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 cost function parts are very much 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. We note that the cost function of Equation (2) can be naturally generalized to

$$\alpha \cdot \sum_{t\in T} p(t) + \beta \cdot \sum_{r_{i,j}\in R} d_H(i,j)m_{i,j} \ , \tag{3}$$

where α and β are positive constants and $p(t)$ is a general cost function $p : P(G) \to \mathbb{R}^+$. The cost function of Equation (2) corresponds to $p(t) = c(t) = \ell(t) - 1$.

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 $(m_{s,1} + m_{s,2} + \ell_1 - 1) + (m_{s,2} + \ell_2 - 1) =$

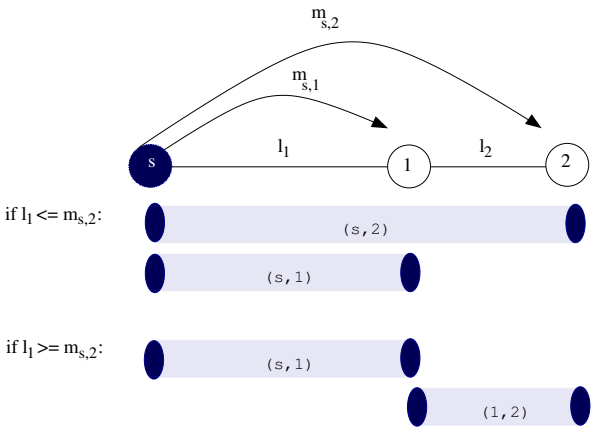

Figure 2: Depending on the values ℓ_1 and $m_{s,2}$, the optimal solution may be composed either of tunnels $(s,1)$ and $(s,2)$, or of tunnels $(s,1)$ and $(1,2)$.

$m_{s,1} + 2m_{s,2} + \ell_1 + \ell_2 - 2$. 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, asserting that there always exists an optimal solution such that all the traffic units of each request are routed via a unique set of consecutive tunnels, or dipath.

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. Let $r_{i,j}$ be a request from node i to node j, and let H be an optimal solution. Suppose that the units of traffic arriving at j from i are routed in H via $p > 1$ different dipaths $P_1, \ldots, P_p$. Let λ_m, $1 \leq m \leq p$, be the number of traffic units forwarded by P_m. Let h_m (h standing for *hops*), $1 \leq m \leq p$, be the number of consecutive tunnels in the dipath P_m. Let the order of the dipaths be such that P_1 is a dipath with the minimum number of consecutive tunnels h_1.

Then, for any other dipath P_m, with $m > 1$, reroute the λ_m requests routed via P_m via P_1. We obtain a new feasible solution H' whose cost is

$$c(H') \leq c(H) + \sum_{2 \leq m \leq p} (\lambda_m h_1 - \lambda_m h_m).$$

Indeed, the cost of each tunnel used in P_m is decreased by λ_m, plus possibly, if some tunnel t of P_m becomes empty, by $\ell(t) - 1 \geq 0$. On the other hand, the cost of each tunnel of P_1 is increased only by λ_m as the tunnel already exists. Therefore, as $h_1 \leq h_m$, we get

$c(H') \leq c(H)$, with strict inequality if $h_1 < h_m$ (if the dipath P_m is strictly longer than P_1) or if, in the rerouting, some tunnels of length more than 1 become empty. So H' is also an optimal solution. □

Relation with VPL problems The layout design problem defined above is quite similar to well studied VPL problems in ATM networks, where one imposes a constraint on the logical structure and then wishes to minimize either the maximum distance [4] or the average distance [9] traveled by the traffic. Concerning hardness and approximation, we shall see in the sequel of the article that the problem we study inherits most of the characteristics of the classical VPL problems studied since the 80s. It is not surprising that, even if new technologies like GMPLS are proposed to cope with the increasing bandwidth of communication networks, the computational complexity of the problems associated to these technologies remains essentially the same.

Nevertheless, there are two crucial differences between the GMPLS problem that we study and the classical VPL version of ATM networks. Indeed, we have seen that the GMPLS logical network design problem can be translated into finding a set of dipaths in a *directed hypergraph*, whereas the existing models for VPL problems deal with *digraphs* without multiple arcs. This feature will be exploited in the dynamic programming approach for the path presented in Section 5. The second difference 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. Finally, it is important to note that, if there is a single source in the GMPLS version (or, more generally, if the traffic instance is such that in an optimal solution each hyperarc has exactly 2 vertices), then the problem is equivalent to a VPL problem.

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.

Recall that a *broadcast* is a request scheme with all the requests from a vertex u_i to all vertices u_j, for $j = 1, \ldots, n$, $j \neq i$. A request scheme is a *partial broadcast* if some requests from u_i to u_j are missing.

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, unless* P = 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.
- To 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.
- To 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.

This construction is illustrated in Figure 3. Observe that the number of vertices of the MINIMUM COST HYPERGRAPH LAYOUT instance is $n = k \cdot (L+1) + p + 1$. Let OPT be the optimal cost to the MINIMUM COST HYPERGRAPH LAYOUT instance, and let OPT_{SC} be the optimal cost to the MINIMUM SET COVER instance.

Note that any cover defined by $I \subseteq \{1, 2, \ldots k\}$ 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

$$OPT \;\leq\; L \cdot OPT_{SC} + 2p. \tag{4}$$

Conversely, consider a feasible layout H of cost S. For each u_j, the dipaths from s to u_j contain some v_i joined to u_j. Let I be the set of indices i of the v_i obtained in such a way.

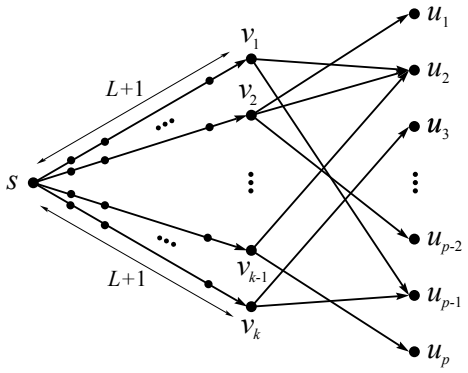

Figure 3: Reduction in the proof of Theorem 1.

Then the sets S_i, for $i \in I$, cover $a_1, \ldots, a_p$. 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 at least 2 (one more tunnel and one more request). So altogether the cost of the solution is at least $S \geq |I| \cdot (L+2) + 2(p-|I|) = k \cdot L \cdot |I| + 2p$. 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

$$S_{SC} \;<\; \frac{S}{L} \;\leq\; \frac{C_1 \cdot \log n \cdot OPT}{L}. \tag{5}$$

Using Equation (4) in Equation (5) we obtain

$$S_{SC} \;<\; \frac{C_1 \cdot \log n \cdot (L \cdot OPT_{SC} + 2p)}{L} \;=\; C_1 \cdot \log n \cdot OPT_{SC} + \frac{2C_1 \cdot p \cdot \log n}{L}. \tag{6}$$

Let now $L = p^2$. We can assume that the number of sets of the MINIMUM SET COVER instance is bounded by a polynomial on the number of elements [8]. Therefore, $n = k \cdot (p^2+1) + p + 1 \leq p^{C_2}$ for some fixed constant $C_2 > 0$. In other words, $\log n \leq C_2 \cdot \log p$. Using the latter inequality and the fact that $L = p^2$ in Equation (6) we get

$$S_{SC} \;\leq\; C_1 \cdot C_2 \cdot \log p \cdot OPT_{SC} + \frac{2C_1 \cdot C_2 \cdot \log p}{p}. \tag{7}$$

As the rightmost term of Equation (7) tends to 0 as p tends to ∞, we have obtained a polynomial time $(C_1 \cdot C_2 \cdot \log p)$-approximation algorithm for MINIMUM SET COVER. On the other hand, Raz and Safra [16] proved that MINIMUM SET COVER is not approximable within a factor $C_3 \cdot \log p$, for some constant $C_3 > 0$, unless P = NP. So for $C_1 = \frac{C_3}{C_2} > 0$, 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 $S \subseteq V$, find a connected subgraph with minimum edge-weight containing all the vertices in S. We can assume, by subdividing edges, that all edge-weights are equal to 1. This problem is known to be APX-hard [5], hence it does not accept a PTAS unless P = NP.

Given an instance $(G = (V, E), S \subseteq V)$ 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 S to all the other vertices in S, all with multiplicity 1. We set all the edge lengths to $L > 0$, 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 H. Since H connects s to all the other vertices in S, it follows that

$$OPT \;\leq\; L \cdot OPT_{ST} + \sum_{x \in S \setminus \{s\}} d_H(s, x) \;\leq\; L \cdot OPT_{ST} + n^2. \tag{8}$$

Conversely, given any solution to the MINIMUM COST SYMMETRIC HYPERGRAPH LAYOUT instance with cost S and realized with a graph H_S, we can find a solution to the MINIMUM STEINER TREE instance (just by taking the edges used by some tunnel) with cost

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

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 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

$$S_{ST} \;\leq\; \frac{S}{L} \;\leq\; \frac{(1 + \varepsilon) \cdot OPT}{L} \;\leq\; \frac{(1 + \varepsilon) \cdot \left(L \cdot OPT_{ST} + n^2\right)}{L}, \tag{9}$$

where we have used Equation (8) in the last inequality. Let now $L = n^3$. Equation (9) 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 [5] unless P = NP. □

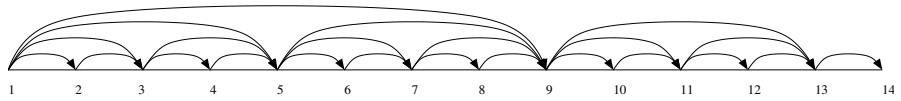

Figure 4: The binary layout depicted above gives a $\log n$-approximation algorithm for MINIMUM COST HYPERGRAPH LAYOUT on the path. If the path is undirected, we add the symmetric tunnels.

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 Proposition 1 is a particular case of Proposition 2, 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. We can suppose w.l.o.g. that $n-1$ is a power of two, otherwise, just add dummy vertices.

We define the following *binary layout*, such as illustrated in Figure 4: 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 $[(n+1)/2,n]$. 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 the cost of this layout is $\log n \cdot \sum_{r_{i,j}} m_{i,j} + \log n \cdot \ell([1,n])$. 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 $2\log(n)$-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 $2\log(n_i)$-approximation, and thus a $2\log(\max_i(n_i)) < 2\log(n)$ 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 [4] 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 [4] 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.* [4])**.** *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, 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 $c : E \to \mathbb{N}$, and a requirement function $r : V \times V \to \mathbb{N}$. 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 $r(i, j)$, 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 [10], and within a constant factor 2 when all the requirements are equal [14]. The directed version of the problem is approximable within $\mathcal{O}(n^{2/3} \log^{1/3} n)$ [7].

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 [14], 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 [22].) 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 [22].)

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 $\mathcal{O}(n^{2/3} \log^{1/3} n)$ [7], 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 5.2 and 5.3, respectively, and then we show in Section 5.4 how to generalize the previous ideas to deal with any fixed number k of sources on the path.

5.1 Some notations

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 by $OPT(s_1, s_2, \ldots, s_k; [s_1, n])$ the cost of an optimal solution with k sources $s_1 < s_2 < \ldots < s_k$ on the dipath $[s_1, n]$.

Notation $\alpha(i)$. 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 $[u, v]$).

Note that on the path $[1, n]$, when all the requests are destined to node n, the optimal solution is simply the tunnel from node i to node n where i is the leftmost source.

5.2 Case of a single source

We present in this section the algorithm for a single source (Gerstel *et al.* used a similar approach in [9]). We also give a closed formula of the optimal cost when the requests and the lengths of the arcs are uniform (see Proposition 4).

For one source, we denote $OPT(i; [i, j])$ by simply $OPT[i, j]$, that is, 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 $m_{i,u} = m_{1,u}$.

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). 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. *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 $(\alpha(i), l)$.

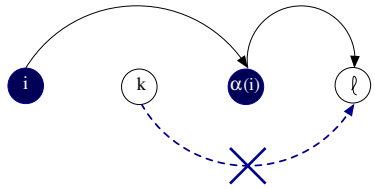

Figure 5: Case of a single source and the non-crossing property.

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 $\ell([\alpha(i),l]) < \ell([k,l])$, $c(T') < c(T)$. □

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:*

$$OPT[i,j] = \min_{i < \alpha(i) \leq j} C_{\alpha(i)}[i,j] \tag{10}$$

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, $C_{\alpha(i)}[i,j]$. □

Algorithm 1: Polynomial-time algorithm computing an optimal solution for MINIMUM COST HYPERGRAPH LAYOUT on a directed path with one source.

Input: Path $[1, n]$ from 1 to n, where 1 is the source and a set of requests $r_{1,i}$ of multiplicity $m_{1,i}$, for $2 \leq i \leq n$

Output: $OPT[1, i]$ and the set of tunnels enabling to route the requests $r_{1,i}$, for $2 \leq i \leq n$

begin

OPT is a table of size $n \times n$ indicating the costs all the subsolutions;
S is a table of size $n \times n$ indicating the $\alpha(i)$ associated to the optimal subsolutions;
M is a table of size n storing partial sums of weigths, $M[1] = 0$, $M[j] = \sum_{i=2}^{j} m_{1,i} = M[j-1] + m_{1,j}$
for $i \in [1, n]$ **do**
  $OPT[i, i] = 0$;
for $i \in [1, n-1]$ **do**
  $OPT[i, i+1] = m_{1,i+1} + \ell([i, i+1]) - 1$;*//Cost of the tunnel* $(i, i+1)$ *carrying traffic towards* $i+1$
  $S[i, i+1] = i+1$;
for $k \in [2, n]$ **do**
  for $\forall i \in [1, n-k]$ **do**
    $\min = +\infty$;
    //We consider all requests from 1 *to some node in the interval* $[i+1, i+k]$
    for $\forall \alpha(i) \in [i+1, i+k]$ **do**
      //value is the cost of the solution if $\alpha(i)$ *is the splitting point for subdipath* $[i, i+k]$
      $value = (M[i+k] - M[\alpha(i)-1]) + \ell([i, \alpha(i)]) - 1 + OPT[i, \alpha(i)-1] + OPT[\alpha(i), i+k]$;
      if $value < \min$ **then**
        $\min = value$;
        $OPT[i, i+k] = value$;
        $S[i, i+k] = \alpha(i)$;
Compute the optimal set of tunnels from the table S;

end

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 by Algorithm 1.*

Proof. Algorithm 1 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 $OPT[i, i+2] =$

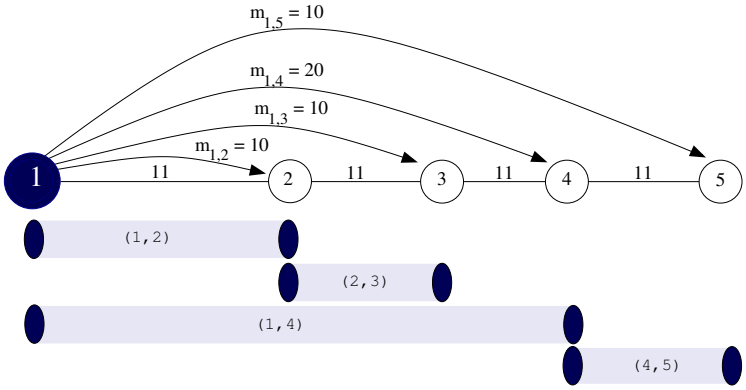

Figure 6: An example with its optimal solution.

$\min\{C_{i+1}, C_{i+2}\}$ where $C_{i+1} = m_{1,i+1} + m_{1,i+2} + \ell([i, i+1]) - 1 + OPT[i+1, i+2]$ and $C_{i+2} = m_{1,i+2} + \ell([i, i+2]) - 1 + OPT[i, i+1]$. Then, it computes solutions for dipaths of length 3, 4,... 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 (Algorithm 1), with time complexity $\mathcal{O}(n^3)$ and space complexity $\mathcal{O}(n^2)$. □

The optimal algorithm in the example of Figure 6 Let us compute an optimal solution to the example in Figure 6 using Algorithm 1 where all the lengths are equal to 11 and $m_{1,2} = m_{1,3} = m_{1,5} = 10$ and $m_{1,4} = 20$. We first have to compute the table OPT containing the costs of the suboptimal solutions for each subpath.

First, $OPT[1,2]$, $OPT[2,3]$, $OPT[3,4]$, and $OPT[4,5]$ are directly computed with $OPT[i, i+1] = m_{1,i+1} + \ell([i, i+1]) - 1$.

Then, to compute $OPT[1,3]$, $OPT[2,4]$, and $OPT[3,5]$, two splitting points are considered by the algorithm. For example, for $[1,3]$, the optimal solution implies a splitting point $\alpha(1) = 2$ with cost $m_{1,2} + m_{1,3} + \ell([1,2]) - 1 + OPT[1,1] + OPT[2,3] = 50$ (the splitting point 3 implying 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 $OPT[2,5]$ and finally for the computation of the optimal solution on the whole path $[1,5]$, four splitting points, $2, 3, 4$, and 5 must be considered:

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

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

Table 1: Computation of the tables OPT and S for the optimal solution of the example on Figure 6, the nodes in brackets representing the splitting points of table S.

- for $\alpha(1)=5$, $m_{1,5}+43+OPT[1,4]=154$;

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

When the table OPT showing the optimal costs for all the subpaths has been computed as presented in Table 1, the set of tunnels composing the optimal solution can be deduced from the splitting points. The optimal solution for the subpath $[1,5]$ has cost 132 and a splitting point $\alpha(1)=4$. 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 splitting point $\alpha(1)=2$ which gives tunnels $(1,2)$, $(2,3)$. The optimal solution for the subpath $[4,5]$ is obviously 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.

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 $2^q(q-1)+1+(q+1)r$.*

Proof. Let $c(k)=OPT[1,k]=OPT[j+1,j+k]$ as all the lengths and multiplicities are equal to 1. By proposition 3, we have by letting $\alpha(1)=i+1$, $c(n)=\min_i[(i)+c(n-i)+n-1]$.

Let us consider only the solutions satisfying the non-crossing lemma, that we call *normal solutions*. A normal solution for $[1,n]$ is obtained by taking for $\alpha(1)=i+1$, a normal solution S_1 for $[1,i]$ and a normal solution S_2 for $[i+1,n]$, with the source at $i+1$. Therefore, the cost is: $c(S_1)+c(S_2)+n-1$. Associate to a normal solution S, a binary tree $T(S)$ on $n-1$ vertices obtained as follows: $T(S)$ consists of a root joined to the root of the left subtree $T(S_1)$ associated to S_1 and to the root of the right subtree $T(S_2)$ associated to S_2.

Let $d(T(S))$ be the sum of the distances of the nodes to the root node in $T(S)$. Then, we prove by induction that the cost of a solution S is $c(S)=d(T(S))+|T(S)|$. That is true for $n=2$ as $T(S)$ consists of the tree reduced to one vertex and the cost of the solution is 1.

n	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
c^*	1	3	5	8	11	14	17	21	25	29	33	37	41	45	49	54	59	64	69

Table 2: Optimal cost for the path with a uniform distribution.

Now $c(S) = c(S_1) + c(S_2) + n - 1 = d(T(S_1)) + |T(S_1)| + d(T(S_2)) + |T(S_2)| + |T(S)|$. But the distance in $T(S)$ of a node of $T(S_1)$ (resp. $T(S_2)$) is exactly one more than in its respective subtree. So $d(T(S_1)) + |T(S_1)| + d(T(S_2)) + |T(S_2)| = d(T(S))$.

Therefore, minimizing the cost for n is equivalent to minimizing the distance of the nodes to the root in a binary tree with n nodes. The optimal tree is very simple: it is just a complete binary tree with an extra non-complete level. Since a complete binary tree with depth $q-1$ has $2^q - 1$ vertices, we get when $n = 2^q + r$, $0 \leq r < 2^q$ that the sum of the distances of the nodes to the root is equal to $\sum_{i \in [1,q-1]} i2^i + qr + n - 1 = 2^q(q-1) + 1 + (q+1)r$. □

Table 2 gives the values up to 20 nodes on the path with uniform distribution, each node at distance one from its neighbours.

5.3 Case of two sources

We use a dynamic program similar to the one used for the single source case in Section 5.2, but slightly more complicated.

Let s_1 and s_2 be the two sources with $s_1 < s_2$. First, we give the idea of the algorithm, then the recursive formulae used to compute $OPT(s_1, s_2; [s_1, n])$, and finally prove their validity, obtaining an $\mathcal{O}(n^4)$ algorithm.

5.3.1 Key idea of the algorithm

The key idea of the algorithm is that we can find an optimal solution satisfying the following property analogous to Lemma 2. If i acts as a source (representing s_1 or s_2) and $(i, \alpha(i))$ denotes the longest tunnel from i, then the traffic of i, destined to nodes after $\alpha(i)$ is routed via the tunnel $(i, \alpha(i))$. So starting at s_1, $\alpha(s_1)$ acts as a source for the nodes in $[\alpha(s_1), n]$. Three cases may happen:

1. if $\alpha(s_1) < s_2$, then we consider the same problem with sources $\alpha(s_1)$ (carrying the traffic of s_1) and s_2 on a smaller dipath $[\alpha(s_1), n]$.

2. if $\alpha(s_1) = s_2$, then we consider the problem with only one source s_2 carrying the traffic from both sources s_1 and s_2.

3. if $\alpha(s_1) > s_2$, then we consider the problems with two sources s_1 and s_2 on dipath $[s_1, \alpha(s_1) - 1]$ and the two sources s_2 and $\alpha(s_1)$ on dipath $[\alpha(s_1), n]$. We will consider two cases for the traffic from the source s_2 to nodes after $\alpha(s_1)$.

   - **Case 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. 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.
   - **Case 2:** $\alpha(s_2) > \alpha(s_1)$. The traffic from s_2 to nodes after $\alpha(s_2)$ is routed via the tunnel $(s_2, \alpha(s_2))$. For the interval $[\alpha(s_2), n]$, we have to deal with a problem with two sources $\alpha(s_1)$ and $\alpha(s_2)$.

     We repeat the argument with $\alpha^2(s_1)$ for the traffic towards nodes after $\alpha(s_2)$.
     - 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 $\alpha(s_2)$.
     - Else 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 a problem with two sources $\alpha(s_2)$ and $\alpha^2(s_1)$, and so on.

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. $\alpha^{h'}(s_1)$).

5.3.2 Definition and computation of the formulae $OPT(i, s_2; [i, u])$ and $OPT'_\delta(i, j; [j, u])$

Now let us give the algorithm to compute $OPT(s_1, s_2; [s_1, n])$ and show that it is correct. For that, we will have to compute only two types of values defined as follows:

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. $OPT'_\delta(i, j; [j, u])$, with $\delta = 1$ or 2 the cost of an optimal solution on subdipath $[j, u]$ with sources i and j, where $s_2 \leq i < j < u \leq n$ and where there exists a tunnel $(\beta(j), j)$ with $\beta(j) < i$. For $OPT'_1(i, j; [j, u])$, i carries the traffic of s_1 ($m_{i,x} = m_{s_1,x}$ for $x \in [j, u]$) and j the traffic of s_2. For $OPT'_2(i, j; [j, u])$, i carries the traffic of s_2 and j the traffic of s_1. Note that in OPT', the left source i is outside the interval of destination nodes, but we have the possibility to use the tunnel $(\beta(j), j)$ where $\beta(j) = \alpha^{-1}(j)$ with a cost reduced to its weight (so we use a notation OPT').

Note also that in both cases if $i = s_2$ or $i = j$ the problem is reduced to the case of one source i carrying the traffic of both sources s_1 and s_2, that is, $m_{i,x} = m_{s_1,x} + m_{s_2,x}$ for $i < x \leq u$. We denote this special source by i^*.

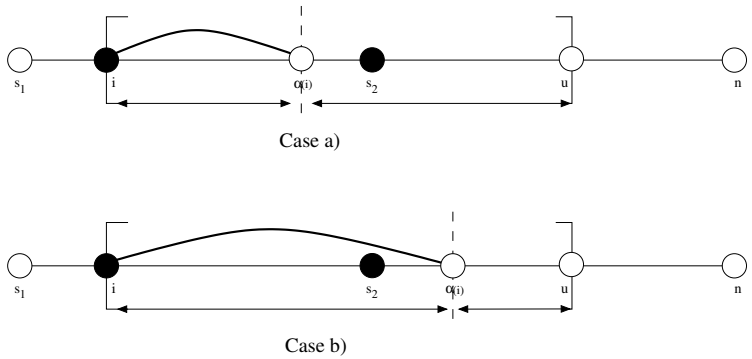

Figure 7: To compute $OPT(i, s_2; [i, u])$, we need for case a) $OPT(i; [i, \alpha(i) - 1])$ and $OPT(\alpha(i), s_2; [\alpha(i), u])$ and for case b) $OPT(i, s_2; [i, \alpha(i) - 1])$ and $OPT'_2(s_2, \alpha(i); [\alpha(i), u])$.

Let us now show how to compute the formulae. See Figure 7 for an illustration and Subsection 5.3.4 describing the computation of a solution on the Figure 9.

1. Computation of $OPT(i, s_2; [i, u])$, with $s_1 \leq i \leq s_2$ and $i < u \leq n$.

$$OPT(i, s_2; [i, u]) = \min_{\alpha(i)} C_{\alpha(i)}[i, u], \text{ where}$$

a) if $\alpha(i) \leq s_2$, $C_{\alpha(i)}[i, u] =$

$$OPT(i; [i, \alpha(i) - 1]) + OPT(\alpha(i), s_2; [\alpha(i), u]) + \sum_{x \in [\alpha(i), u]} m_{s_1, x} + \ell(i, \alpha(i)) - 1,$$

where $OPT(i; [i, \alpha(i) - 1])$ is the optimal cost of a problem with a single source i carrying the traffic from s_1. Note that if $\alpha(i) = s_2$, the problem is reduced to $OPT(s_2^*; [\alpha(i), u])$ where s_2^* carries the traffic from both sources.

b) if $\alpha(i) > s_2$, $C_{\alpha(i)}[i, u] =$

$$OPT(i, s_2; [i, \alpha(i) - 1]) + OPT'_2(s_2, \alpha(i); [\alpha(i), u]) + \sum_{x \in [\alpha(i), u]} m_{s_1, x} + \ell(i, \alpha(i)) - 1.$$

In both cases $\alpha(i)$ carries the traffic of i, that is of s_1.

2. Computation of $OPT'_\delta(i, j; [j, u])$, with $s_2 \leq i < j \leq u$, $\delta = 1$ or 2 and where i carries the traffic of source δ, j that of the other source $3 - \delta$, and there exists a tunnel $(\beta(j), j)$ (in fact, $\beta(j) = \alpha^{-1}(j)$).

Figure 8 is an illustration of the two cases a) and b) for the computation of $OPT'_\delta(i, j; [j, u])$.

$$OPT'_\delta(i, j; [j, u]) = \min[C_{\alpha(i)}[j, u], \text{ for } j < \alpha(i) \leq u \ ; \ C_{j^*}[j, u]], \text{ where}$$

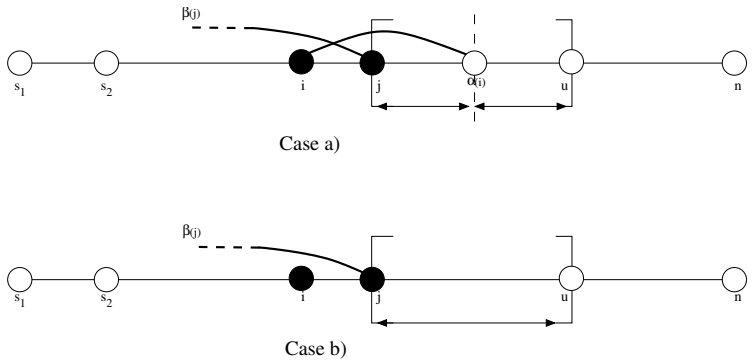

Figure 8: To compute $OPT'_\delta(i,j;[j,u])$, we need for case a) $OPT'_\delta(i,j;[j,\alpha(i)-1])$ and $OPT'_{3-\delta}(j,\alpha(i);[\alpha(i),u])$ and for case b) $OPT(j^*;[j,u])$.

a) if $j < \alpha(i) \leq u$, $C_{\alpha(i)}[j,u] =$

$$OPT'_\delta(i,j;[j,\alpha(i)-1]) + OPT'_{3-\delta}(j,\alpha(i);[\alpha(i),u]) + \sum_{x\in[\alpha(i),u]} m_{i,x} + \ell(i,\alpha(i)) - 1,$$

where $\alpha(i)$ carries the traffic of i (that is, of the source it represents).

b) if there is no tunnel $(i,\alpha(i))$ with $j < \alpha(i) \leq u$, then the value is

$$C_{j^*}[j,u] = OPT(j^*;[j,u]) + \sum_{x\in[j,u]} m_{i,x},$$

where OPT corresponds to a problem with a unique source j^* carrying the traffic of both sources i and j (that is, $m_{s_1,x} + m_{s_2,x}$ for $j < x \leq u$). Note that in this case the traffic from i uses the tunnel $[\beta(j),j]$ and so there is no cost corresponding to the length.

5.3.3 Proof of the computation of the formulae $OPT(i,s_2;[i,u])$ and $OPT'_\delta(i,j;[j,u])$

First, let us prove the following lemma, which is valid for any pair of sources i,j, and that will be needed to prove the formulae.

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, and $j < \alpha(i) \leq u$. There exists an optimal solution where the traffic of i destined for the nodes in $[\alpha(i),u]$ uses as first tunnel $(i,\alpha(i))$.*

Note that in particular the optimal solution for $OPT'_\delta(i,j;[\alpha(i),u])$ is independent of the tunnels used to bring traffic to nodes $x < \alpha(i)$.

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 $\delta = 2$, $\alpha^h(s_)$. As the lemma is not true, this solution uses to bring the traffic of i to some node x, with $\alpha(i) < x < u$, a tunnel (k, l) with $k < \alpha(i) < l < u$. 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. 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 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. □

Remark. We can see the analogy between Lemmas 2 and 3, as we have no tunnel (k, l) crossing $\alpha(i)$ with $k < \alpha(i) < l$ except perhaps for $k = j$. In some cases the cost of using directly a tunnel (j, l), that is $m_{j,l} + \ell(j, l) - 1$, might be less than that of using the tunnel $(i, \alpha(i))$ and a tunnel $(\alpha(i), l)$, of value $2m_{j,l} + \ell(\alpha(i), l) - 1$.

Let us now prove that the computation of the above formulae is valid, first for the formula to compute $OPT(i, s_2; [i, u])$ and then for the formula to compute $OPT'_\delta(i, j; [j, u])$.

1. Computation of $OPT(i, s_2; [i, u])$.

For case a), a proof analog to that of Lemma 2 shows that in an optimal solution there does not exist a tunnel (k, l) with $i < k < \alpha(i) < l \leq u$. Indeed, deleting the tunnel (k, l) and adding, if it does not exist, the tunnel $(\alpha(i), l)$ we obtain a better solution. So in case a) we can separate the traffic into that destined for the nodes in $[i, \alpha(i) - 1]$, which is only that of i (that is, of s_1) as the source s_2 is outside the interval, plus the traffic destined for the nodes in $[\alpha(i), u]$ which uses as first tunnel $(i, \alpha(i))$. We have to add the cost of the tunnel $(i, \alpha(i))$, that is $\sum_{x \in [\alpha(i), u]} m_{s_1, x} + \ell(i, \alpha(i)) - 1$.

In case b), by Lemma 3, there exists an optimal solution in which the traffic destined for the nodes in $[\alpha(i), u]$ uses first the tunnel $(i, \alpha(i))$. So the cost of an optimal solution consists of the optimal cost for the traffic to the nodes in $[i, \alpha(i) - 1]$, that is $OPT(i, s_2; [i, \alpha(i) - 1])$, plus the cost of the tunnel $(i, \alpha(i))$ and the cost of an optimal solution for the traffic to $[\alpha(i), u]$ with the two sources s_2 and $\alpha(i)$, where $\alpha(i)$ carries the traffic from i, that is of s_1, that is $OPT'_2(s_2, \alpha(i); [\alpha(i), u])$.

Note that s_2 is on the left of the interval $[\alpha(i), u]$ but can use the tunnel $(i, \alpha(i))$.

2. Computation of $OPT'_\delta(i, j; [j, u])$.

For case a), by Lemma 3, there exists an optimal solution such that the traffic to $[\alpha(i), u]$ uses as first tunnel $(i, \alpha(i))$. So we can separate the cost into the cost of an optimal solution to send the traffic of i and j to the nodes in $[j, \alpha(i) - 1]$, that is $OPT'_\delta(i, j; [j, \alpha(i) - 1])$, and the cost of an optimal solution with the two sources j and $\alpha(i)$ to the nodes in $[\alpha(i), u]$,

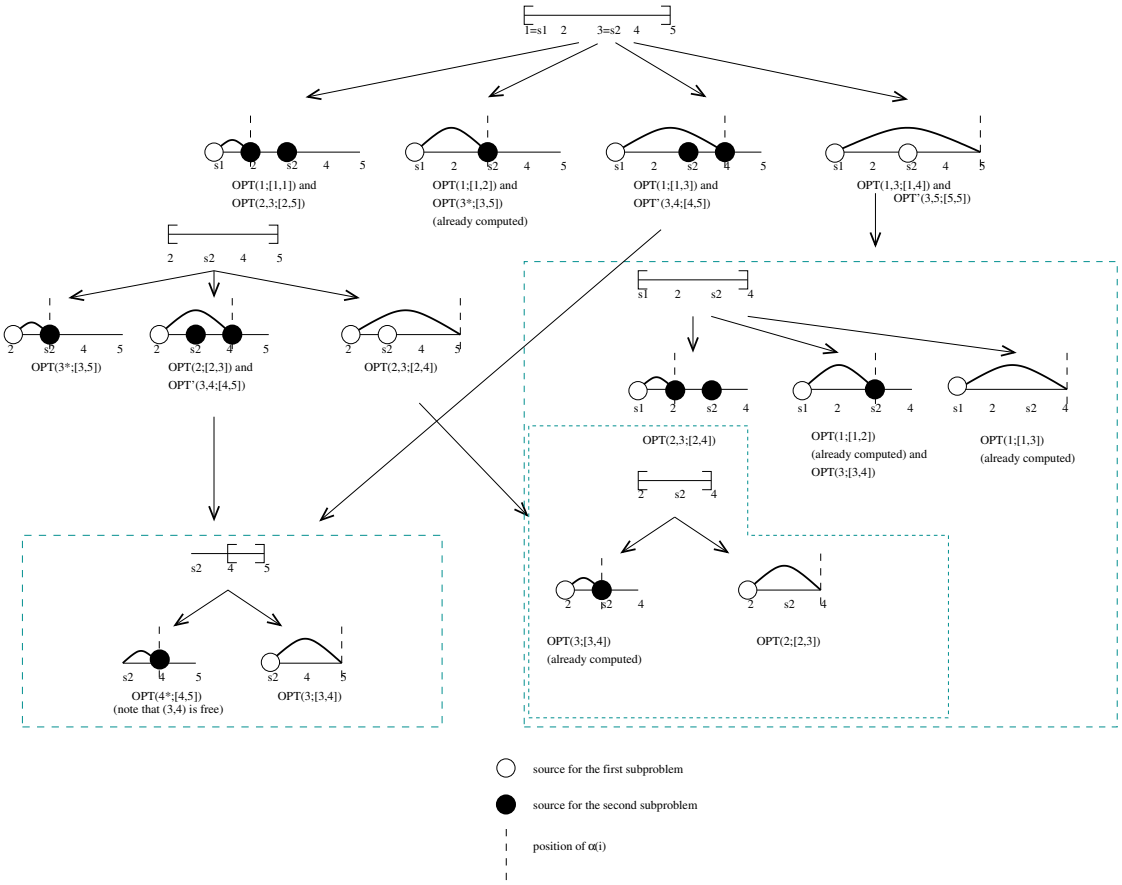

Figure 9: Dynamic programming on the path $[1, 5]$ with 2 sources s_1 and s_2.

where $\alpha(i)$ now carries the traffic of i to x such that $\alpha(i) < x \leq u$, plus the cost of the tunnel $(i, \alpha(i))$.

For case b) we can inject the traffic of i into the existing tunnel $(\beta(j), j)$ for a minimum cost of $\sum_{x \in [j,u]} m_{i,x}$. Then we have the traffic from both sources to x, such that $j < x \leq u$, concentrated in node j, so we use an optimal solution for one source.

5.3.4 The optimal algorithm in the example of Figure 9

Figure 9 shows an example on a path with 5 vertices $1, 2, 3, 4, 5$ and two sources $s_1 = 1$ and $s_2 = 3$. We have to compute $OPT(1, 3; [1, 5])$, and to find the node $\alpha(1)$ for which this value is minimum:

- If $\alpha(1) = 2$, we need $OPT(1; [1, 1]) = 0$ and $OPT(2, 3; [2, 5])$.
- If $\alpha(1) = 3$, we need $OPT(1; [1, 2]) = m_{s_1,2} + \ell(1, 2) - 1$ and $OPT(3^*; [3, 5])$, where 3^* carries both traffics of s_1 and s_2.
- If $\alpha(1) = 4$, we need $OPT(1, 3; [1, 3])$ and $OPT'_2(3, 4; [4, 5])$.
- If $\alpha(1) = 5$, we need $OPT(1, 3; [1, 4])$ and $OPT'_2(3, 5; [5, 5]) = m_{3,5}$

Recursively, we compute $OPT(1, 3; [1, 3])$, $OPT(1, 3; [1, 4])$, $OPT(2, 3; [2, 5])$, and also $OPT'_2(3, 4; [4, 5])$. Note that during the computations of $OPT(1, 3; [1, 4])$, we will need $OPT(2, 3; [2, 4])$ (for $\alpha(1) = 2$), but $OPT(2, 3; [2, 5])$ will also need $OPT(2, 3; [2, 4])$ (for $\alpha(2) = 5$).

In fact we can use dynamic programming and store only $\mathcal{O}(n^3)$ values. 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 5.2), the values $OPT(j^*; [j, u])$ for $j \geq s_2$ and with both traffics of s_1 and s_2, and $OPT'_\delta(i, j; [j, u])$ where $s_2 \leq i < j \leq u$ where i carries the traffic of s_δ and j that of $s_{3-\delta}$.

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 $j = u$, in which case $OPT'_\delta(i, u; [u, u]) = m_{s_\delta,u}$. 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 $\mathcal{O}(n^4)$.

5.4 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 $i_1 < i_2 < \ldots < i_h < s_{h+1} < \ldots < s_k$, $s_2 < i_1$, $s_3 < i_2$, ..., $s_h < i_{h-1}$, 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 $j \geq h+1$, $\pi(j) = j$.

Using a lemma analogous to Lemma 3 and letting for a node i and an interval $[u, v]$, $\alpha(i)$ be such that $(i, \alpha(i))$ is the longest tunnel from i and ending in $[u, v]$. We get

$$OPT_\pi(i_1, i_2, \ldots, i_h, s_{h+1}, \ldots, s_k; [i_h, u]) \ = \min[C_{\alpha(i_1)}, \text{for } i_h < \alpha(i_1) \leq u; C_{i_h^*}]$$

where

a) if $\alpha(i_1) > i_h, C_{\alpha(i_1)} =$

$$\begin{aligned} & OPT_\pi(i_1, \ldots, i_h, s_{h+1}, \ldots, s_k; [i_h, \alpha(i_1) - 1]) \\ + \ & OPT_{\pi'}(i_2, \ldots, i_h, s_{h+1}, \ldots, s_{h+m}, \alpha(i_1), s_{h+m+1}, \ldots, s_k; [\alpha(i_1), u]) \\ + \ & \sum_{x \in [\alpha(i_1), u]} m_{i_1, x} + \ell(i_1, \alpha(i_1)) - 1. \end{aligned}$$

The nodes carry the same traffic as before, except $\alpha(i_1)$ which carries the traffic of i_1, that is of $s_{\pi(1)}$.

The permutation π' is such that:

- for $1 \leq j < h$, as $i'_j = i_{j+1}$, $\pi'(j) = \pi(j+1)$,
- if $\alpha(i_1) < s_{h+1}$, then $\pi'(h) = \pi(1)$,
- if $s_{h+m} \leq \alpha(i_1) < s_{h+m+1}$, then $\pi'(h+m) = \pi(1)$.
- Otherwise, $\pi'(j) = \pi(j)$.

If $\alpha(i_1) = s_{h+m}$ for some m, then the node s_{h+m} will carry the traffic of both sources s_{h+m} and $s_{\pi(1)}$, and we apply the algorithm with $k-1$ sources.

b) there is no tunnel from i_1 ending in $[i_h, u]$. Then the traffic of i_1 can be directly sent to i_h via the tunnel $(\beta(i_h), i_h)$ and then we have to compute the optimum for $k-1$ sources.

$$C_{i_h^*} \ = \ OPT_{\pi'}(i_2, \ldots, i_h^* = i_1, s_{h+1}, \ldots, s_k; [i_h, u]) + \sum_{x \in [i_h, u]} m_{i_1, x}.$$

where $i_h = i_1$ carries the traffic of both sources $s_{\pi(1)}$ and $s_{\pi(h)}$, the other nodes carrying their previous traffic.

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 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 report we modeled a question raised by label minimization in GMPLS networks as a hypergraph layout problem. In the single commodity case 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 $\log n$-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 looks like a challenging problem.

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 $\mathcal{O}(n^{k+2})$ time 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 to 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].

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. Perennes, 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. Perennes, 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*, 2009. To appear.

[4] 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.

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

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

[7] 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.

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

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

[10] 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.

[11] 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*, 2003.

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

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

[18] 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.

[19] 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.

[20] F. Solano Donado and J. Moulierac. Routing in All-Optical Label Switched-based Networks with Small Label Spaces. In *Proc. of the 13th Conference on Optical Network Design and Modeling (ONDM)*, 2009. To appear.

[21] 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.

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

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

Contents

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399