MASTER'S DEGREE THESIS

Interuniversity Master in Statistics and Operations Research

Title: Column generation algorithm for flexgrid optical network problems

Author: David Rebolo Pérez

Advisor: Luis Velasco Esteban

Co-Advisor: Marc Ruiz Ramírez

Facultat de Matemàtiques i Estadística
UNIVERSITAT POLITÈCNICA DE CATALUNYA

UNIVERSITAT DE BARCELONA

Acknowledgements

I would like to express my deep gratitude to my advisors, Luis Velasco and Marc Ruiz, for his valuable and constructive suggestions during the planning and development of this research work and their willingness to give his time so generously.

My special thanks to all the researchers of the Optical Communications Group of the UPC for their help and advice for my initiation in the field of optical networks. I also would like to thank them for providing me a space and a computer for my project.

I wish to thank Carlos Garrigós and Eider Luzarraga, whose enthusiasm, interest and support in this project have given me the motivation to realize this achievement.

Finally, I wish to thank my family for their support and encouragement throughout my entire career; this dissertation would be simply impossible without them.

Index

List of Figures

List of Tables

Chapter 1.

Introduction

1.1 Motivation and objectives

The unstoppable explosion of Internet traffic due to the increase of multimedia services such as video conferencing, HDTV, or IP telephony is rapidly consuming capacity resources in optical transport networks. Nowadays operators are looking for technology solutions for near future to deal with such traffic increase in a cost-effective way. Recently proposed *flexgrid optical networks* are presented as the most promising solution to deal with the expected high volumes of data traffic. In flexgrid optical networks, *optical connections* (i.e. *lightpaths*) use as frequency spectrum as they need, tightly fitting the allocated capacity with the bandwidth requirements. In contrast, current *fixed grid networks* allocate the same amount of spectrum to each lightpath, thus leading to a notorious waste of usable capacity.

In flexgrid optical networks, the problem of finding unoccupied spectrum resources so that to establish a lightpath is called the *Routing and Spectrum Allocation* (RSA) problem. RSA concerns assigning a contiguous fraction of frequency spectrum to a connection request subject to the constraint of no frequency overlapping in network links. Moreover, it is commonly assumed that the same piece of spectrum is used in all links traversed by a lightpath (i.e. spectrum continuity). To solve the RSA optimization problem, which is proven to be NP-hard, efficient integer linear formulations have been proposed. However, when facing real instances involving thousands or even millions of integer variables, solving the RSA problem by means of such formulations becomes unaffordable.

To deal with those instances with a large set of variables, decomposition methods can be derived to improve their tractability. *Column generation* is one of these optimization decomposition methods which allows reducing the amount of variables (referred to as columns) in linear formulations. This technique, based on iteratively solving a master problem that grows at each iteration and a pricing problem in charge of finding good columns to feed the master problem, is also

referred as *path generation* in the context of network flow problems (since variables to find are paths over a network).

In this work, we present the novel *lightpath generation algorithm* (called LIGERO algorithm) for RSA-based formulations. This algorithm derives from path generation theoretical basis but adapted to optical paths with a specific and unique assigned piece of spectrum. The presented method uses the notation and primal formulation proposed in [Ve12]. To the best of our knowledge, this is one of the first works concerning the application of column generation methods for solving RSA-based problems in flexgrid optical networks. It is worth mentioning that this large scale optimization technique has been deeply studied in the context of fixed grid optical networks, for example, in [Ja09]. However, the formulations and algorithms proposed in [Ja09] are not applicable to our case since the adaptable spectrum allocation in flexgrid optical networks differs from rigid spectrum assignment in fixed grid networks.

1.2 Report organization

The remainder of this document is organized as follows:

Chapter 2 introduces some background on optical networks, network flows, and column generation methodology, needed for a better understanding of the contributions in this work. In Chapter 3 the LIGERO approach is presented for solving a generic RSA problem, detailing the different algorithm versions designed and implemented. The performance evaluation of LIGERO, putting special attention on the selection of the best version, is done in Chapter 4 by means of medium-size and large-size instances over a real operator optical network. To illustrate the utility and applicability of our method, Chapter 5 presents a use case where the proposed RSA problem applies and where LIGERO could be applied to obtain good-quality solutions. Finally, Chapter 6 concludes the report and opens new branches to extend the contributions here presented.

Chapter 2.

Background

In this chapter, the necessary concepts of optical networks and operations research are introduced in order to simplify the understanding of the contents of this project.

Firstly, some concepts on optical networks are introduced, including their components and their formal representation. After this, high-capacity networks and specially flexgrid networks are described and the main problem of the thesis, the Routing and Spectrum Assignment (RSA) problem is defined. Secondly and regarding network flows, the link-path formulation is detailed, as well as the shortest path algorithms that will be derived and used in LIGERO. Finally, the general column generation procedure for linear programming problems and its application to network flows, called path generation, are illustrated in details. Moreover, two techniques to obtain integer solutions after applying the column generation method are described.

2.1 Optical Networks

2.1.1 Basic Concepts

An optical network can be defined as a network topology with its representative equipment based on a certain optical technology. In general, it is represented by an undirected graph where the edges are fiber optic links and the vertices are nodes capable of routing traffic, establishing, and deleting optical connections between source and destination nodes. The optical technology is limited within a range of frequencies of the total frequency spectrum, the so called Optical Spectrum (OS). The OS defines a certain capacity within the fiber optic link and it is measured in Gigahertz (GHz). In addition, this capacity depends on other factors like the equipment utilized in the nodes or the spectral efficiency of established connections.

Basically, a *demand* is a petition of bandwidth (or bitrate) to be transported between the source node (s_d) and the termination node (t_d), usually expressed in Megabits per second (Mbps or Mb/s) or Gigabits per second (Gbps or Gb/s). Hereafter, we consider that every demand requests for bitrate between a minimum and a maximum value. If the minimum cannot be ensured, then the demand will become *blocked* (i.e. not served at all), whereas if the minimum bandwidth can be served but there is not enough resources to serve the maximum one, then the demand will be accepted but *partially un-served*. When a demand is accepted, an optical connection in the network is established between the source and the termination nodes; these optical connections are called *lightpaths* since they allow the data transmission as a light wave. Moreover, a fiber optic can transport more than one lightpath at the same time, since each of them is allocated in different parts of the available OS.

2.1.2 High Capacity Networks

In order to establish an optical connection to serve a demand, it is necessary to solve the problem of finding a route with available free spectrum to support the lightpath. In *Dense Wavelength Division Multiplex* (DWDM) networks (i.e. fixed grid optical networks) [G694], this problem is known as *routing and wavelength assignment* (RWA) problem. These networks divide the OS in wavelengths and each one of them can transport at most one lightpath at the same time. This OS multiplexing allows very high capacity for transporting data in the network and represents the most accepted current solution to carry with high volumes of traffic. Moreover, DWDM networks are a sort of rigid optical networks, since the occupied spectrum width for all optical connections is the same. Contrarily to this definition, a flexible optical network can adapt the OS width assigned to each optical connection to the required bandwidth of the demands. This improves the network performance and the usage of the resources due to their adaptability in relation with the spectrum width and the spectrum allocation.

2.1.3 Flex-Grid Networks

In [G964] standardization, it has been included the definition of a *flexible grid* (*flexgrid*) (previously introduced in [Li11]). In a flexgrid optical network, the OS is divided into *slots*, which are portions of the OS with a fixed width of few GHz (e.g. 6.25 GHz). The central frequency (CF) defines where the assigned spectrum is centred and thus it allows positioning the slots within the whole OS. Moreover, a subset of contiguous (adjacent) slots is called *channel* and it is characterized by its CF and the number of slots that contains. In order to illustrate the concepts introduced above, Figure 2-1 represents the spectrum of a fiber optical link within an elastic optical network using the flexgrid technology.

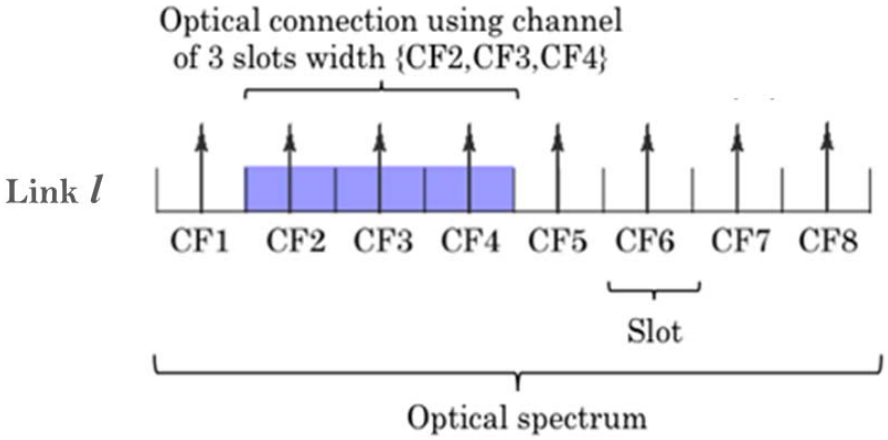


Figure 2-1 Logical representation of a fiber optic link

Similarly to the RWA problem, the *Routing and Spectrum Allocation* (RSA) problem is solved in flexgrid networks. The objective of the RSA problem is to find a route with enough free spectrum width to serve the required bandwidth for traffic demands. The spectrum allocation (SA) of an optical connection consists of finding a certain channel which must accomplish the contiguity and continuity constraints; that is, all slots in a lightpath must be one next to the other (contiguity constraint) as well as the assigned channel must be placed in the same part of the OS (i.e. using the same CF) for all links conforming that optical connection (continuity constraint).

As an example of the RSA problem and for illustrating the continuity and contiguity constraints, Figure 2-2 shows the routing and spectrum allocation for serving a demand with a required bitrate equivalent to 2 slots from the source node B to the destination node D. In a first approach, illustrated in Figure 2-2a, it seems that the route B-A-D would be the one to choose as it is the shortest one. But when looking in detail, it can be seen that the links from B-A and A-D do not have two contiguous slots in the same portion of the OS and therefore the continuity and contiguity constraints are not satisfied if the route B-A-D is chosen. Because of this, another route must be selected, that is the shortest route satisfying the contiguity and continuity constraints. This is the case illustrated in Figure 2-2b, where the selected route is B-A-C-D and the assigned channel uses the slots {S5, S6} for this connection.

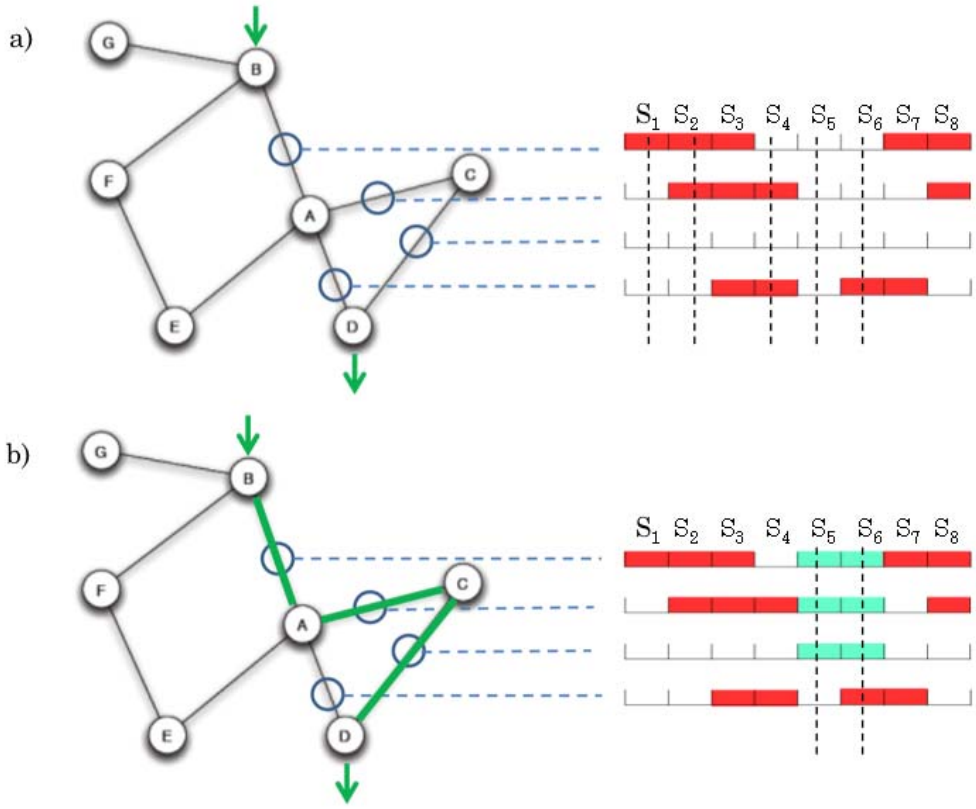


Figure 2-2 RSA, continuity and contiguity constraints

2.2 Operational Research

2.2.1 Formulations of Network Problems

The problem we are dealing with is a multi-commodity flow problem: multiple unitary flow demands between different source and destination nodes must be routed. It can be formulated using either the *node-link* or the *link-path* formulations. The node-link formulation considers every link as a choice for every demand flow and keeps the continuity of the flows by means of degree and sub-tour elimination constraints. On the other hand the link-path formulation uses a set of pre-computed routes between every pair of nodes origin-destination corresponding to a demand [Ah93].

In order to explain with more details the link-path formulation, which is used to formalize the RSA problem, we are going to introduce a multi-commodity flow problem involving continuous variables and express it using a link-path based formulation. Let us consider an undirected graph $G = (E,V)$ and a set of demands D to be satisfied. For every d in D we have to assign a flow demand h_d throughout a pre-computed set of paths between the source and destination nodes P_d. Suppose

that there exist link capacities c_e for every link e in E and we want to minimize the total un-served flow. Let us consider the set of non-negative real variables z_d for every d in D, which represents the un-served bandwidth for demand d. Consider also the set of non-negative variables x_{dp} for every d in D and p in P_d, whose values are the amount of flow that path p serves for demand d. Then, the problem can be formalized as follows:

$$(MP) \quad Minimize \qquad \sum_{d \in D} z_d \tag{2.1}$$

Subject to:

$$[\lambda_d] \qquad \sum_{p \in P_d} x_{dp} + z_d = h_d \qquad d \in D \tag{2.2}$$

$$[\pi_e \geq 0] \quad \sum_{d \in D} \sum_{p \in P_d : e \in p} x_{dp} \leq c_e \qquad e \in E \tag{2.3}$$

$$x_{dp} \geq 0 \qquad d \in D, p \in P_d \tag{2.4}$$

$$z_d \geq 0 \qquad d \in D \tag{2.5}$$

Constraint (2.2) jointly with objective function (2.1) ensures that the un-served demand is minimized by serving demands with paths, while constraint (2.3) does not permit to excess link capacities. Finally constraints (2.4) and (2.5) are non-negativity constraints.

It is important to notice that in the link-path formulation the amount of routes increases exponentially. Hence it is necessary to consider only a subset of routes in order to reduce the computational complexity. Unfortunately, this might result that the reached solution is not optimal. However, this set of routes can be carefully chosen to guarantee that the optimal solution (or almost optimal) is reached, for instance using an algorithm to find the *k-shortest routes* between two nodes such as the Yen algorithm [Ye70].

Finally, remember that the RSA involves integer (i.e. binary) variables. In the literature several alternative Mixed Integer Programming (MIP) formulations of RSA problems can be found (e.g. see [Kl11], [Ch11], [Wa12], and [Ve12]). Among them, the link-path MIP formulation proposed in [Ve12] appears to be the most effective since both continuity and contiguity constraints are removed from the MIP by using both pre-computed paths and channels.

2.2.2 Shortest Path Algorithms

As we will see in section 2.2.3, we need to find the shortest path between several pairs of nodes in the pricing problem. In this section we are going to present two shortest-path methods: the Dijkstra and the Floyd-Warshall algorithms. The Dijkstra algorithm finds the shortest route between a source and every other node

in the network, whereas the Floyd-Warshall algorithm computes the cost of the shortest route between every pair of nodes but does not return any route explicitly.

The Dijkstra algorithm [Di59] starts setting to infinity the distance of every node except the origin which is initialized as 0. A list *V'* contains the nodes whose neighbours have not been explored (in the first iteration *V'* is equal to *V*). While at least one node has not been visited (i.e. *V'* is not empty), the node with the minimum distance is taken (in the first iteration this is the source node) and the distances of all its neighbours are updated if the current routes from the source is shorter than the preceding ones. Every time the distance of a node is updated the previous node in the new route is saved. When all the neighbours of a node are visited the node is erased from the list *V'*. Table 2-1 shows a pseudo-code of the Dijkstra Algorithm.

Observe that for a connected undirected graph all nodes are visited before the algorithm is finished. Nevertheless a directed graph may have some nodes which are not accessible from the source and then the distance remains as infinity along the algorithm. In that case, when *V'* only contains the non-reachable nodes, lines 8 and 9 of Table 2-1 stops the algorithm ensuring the shortest path from source to any other accessible node have been found.

Table 2-1 Dijkstra Algorithm Pseudo-code

```
INPUT: Network G(V,E), source
OUTPUT: dist[], previous[]
1:    for each node v in V do
2:       dist[v] ← infinity
3:    dist[source] ← 0
4:    V' ← V (set of all nodes)
5:    while V' is not empty do
6:       u := argmin{dist[v] : v in V' }
7:       V'=V'\{u}
8:       if dist[u] = infinity
9:          break
10:      for each neighbour v of u in V' do
11:         aux ← dist[u] + weight(u, v)
12:         if aux < dist[v] then
13:            dist[v] ← aux
14:            previous[v] ← u
```

The Floyd-Warshall algorithm [Fl62], shown in Table 2-2, computes the distance between every two nodes of a graph. It starts setting the distance between each pair of nodes as following. The distance of a node with itself is 0. If an edge between two different nodes exists, the distance is the weight of this edge. If there does not exist any edge between two different nodes, the distance is infinity. Then, chosen an intermediate node *w*, the algorithm checks for every pair of nodes whether using intermediate node *w* reduces the current route distance. If the distance is lower when the route visits *w* then the distance between these nodes is updated.

Table 2-2 Floyd-Warshall Algorithm Pseudo-code

```
INPUT: Network G(V,E)
OUTPUT: dist
1:    for each pair of nodes u and v do
2:       dist(u,v) ← infinity
3:       dist(v,u) ← infinity
4:    for each node v
5:       dist(v,v) ← 0
6:    for each edge (u,v)
7:       dist(u,v) ← weight(u,v)
8:    for each w in V do                  //half-way node
9:       for each s in V-{w} do           //source node
10:         for each t in V-{s,w} do      //target node
11:            if dist(s,t) > dist(s,w)+dist(w,t) then
12:               dist(s,t) ← dist(s,w)+dist(w,t)
```

2.2.3 Column Generation Procedure

Column generation (CG) is a decomposition method for solving large-scale linear programming problems and it can be regarded as an application of the Dantzig-Wolfe decomposition [Ba98]. This technique solves iteratively two simpler problems in order to obtain a solution of the original one, which is usually called *master problem* (MP). The *restricted master problem* (RMP) is exactly as the MP, besides it contains only a subset of variables (columns), which makes possible to solve it in practice time using, for instance, the simplex algorithm or interior point methods. If the current solution obtained by the RMP is not optimal (since it does not contain all the variables), the *pricing problem* (PP) finds new columns to provide a better solution. We will see later how it works and why this technique provides the optimal solution for linear programming problems. An illustration of the interaction between the RMP and the PP is shown in Figure 2-3.

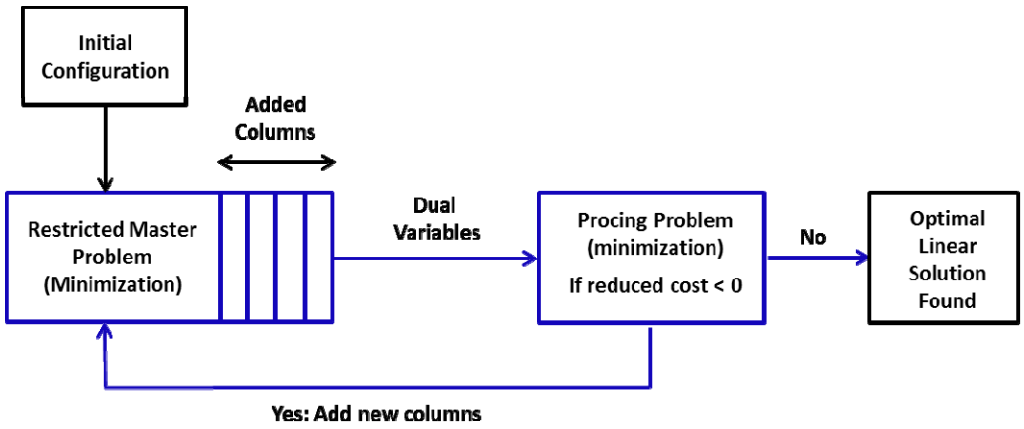


Figure 2-3: Column Generation Diagram

For the sake of clarity, let us consider a general formulation of a linear programming problem and the derivation of its pricing problem. We call the master problem of this linear programming problem (*L-MP*).

$$(L-MP) \quad Minimize \quad c^T x \tag{2.6}$$

Subject to:

$$[\pi] \quad Ax \geq b \tag{2.7}$$

$$x \geq 0, \; x \in \Re^n \tag{2.8}$$

where $\pi \in \Re^m$ are the dual variables of restriction set (2.7).

As we said before, the restricted master problem (L-RMP) has the same structure as the L-MP problem but only considering an initial subset of r variables x_R indexed by $R \subset \{1,2,...,n\}$ and their corresponding cost and constraint coefficients c_R and A_R. The L-RMP is shown below.

$$(L-RMP) \quad Minimize \quad {c_R}^T x_R \tag{2.9}$$

Subject to:

$$[\pi] \quad A_R x_R \geq b \tag{2.10}$$

$$x_R \geq 0, \; x_R \in \Re^r \tag{2.11}$$

Thus, the dual problem (L-DP) of the restricted master problem is the following:

$$(L-DP) \quad Maximize \quad b^T \pi \tag{2.12}$$

Subject to:

$$[x_R] \quad A_R^T \pi \leq c_R \tag{2.13}$$

$$\pi \geq 0, \; \pi \in \Re^m \tag{2.14}$$

If there exists a $k \in R^C$ such that the variables x_k and the corresponding constraint makes the current solution of the dual problem to become not feasible, then the corresponding primal solution is not optimal for the L-MP. Hence, in order to find the necessary variables to reach the optimum, we must be able to solve the pricing problem (L-PP) which consists of finding the minimum value of the reduced costs for each potential variable in R^C. If this value is negative, then the corresponding constraint is violated and thus, the current optimal solution for the L-RMP is not optimal for the L-MP. Contrarily if this value is positive, then the current optimal solution for the L-RMP is also the optimal solution for the L-MP.

$$(L-PP) \quad \underset{k \in R^C}{Minimize} \{c_k - \pi a_k : a_k \; k-th \, column \; of \; matrix \; A\} \tag{2.15}$$

This pricing problem is *NP*-hard, especially when the number of non-considered variables is large, which is the scenario that we are considering to apply the column generation algorithm. For this reason this problem is usually solved by means of a heuristic approach.

In Table 2-3 we show a pseudo-code of the general column generation algorithm:

Table 2-3: General Column Generation Algorithm

```
INPUT: R
OUTPUT: x*
1:  (x*,π*) ← Solve the RMP with columns R
2:  z* ← Solve the PP
3:  while z* < 0 (new columns are found) do
4:     k ← Argument of PP: z* = c_k - π* a_k
5:     R ← R ∪ {k}
6:     (x*,π*) ← Solve the RMP with columns R
7:     z* ← Solve the PP
```

2.2.4 Path Generation Procedure

When column generation method is applied in a network flow problem it is called *path generation* [Pi04]. We are going to see how to apply the column generation method for the problem formulated in section 2.2.1, considering only a subset of columns (paths) *P'*.

In order to state the pricing problem and get a better understanding of the path generation method, the dual problem of the RMP is formulated below.

$$(DRMP) \quad Minimize \qquad \sum_{d \in D} h_d \lambda_d - \sum_{e \in E} c_e \pi_e \tag{2.16}$$

Subject to:

$$[x_{dp} \geq 0] \qquad \lambda_d \leq \sum_{e \in p} \pi_e \qquad d \in D \;\; p \in P_d' \tag{2.17}$$

$$[z_d \geq 0] \qquad \lambda_d \geq 1 \qquad d \in D \tag{2.18}$$

$$\pi_e \geq 0 \qquad e \in E \tag{2.19}$$

Once the RMP is solved, we have an optimal solution for the primal (x^*,z^*) and the dual (λ^*,π^*) problems. Even though, it might result that this solution is not the optimal solution for the MP since we did not consider the full set of routes. Thus we would like to know whether the current solution is optimal and otherwise how to find out which of the non-considered routes are necessary to reach the optimal solution.

Observe that the set of constraints (2.17) force the value λ_d^* (where $(\lambda_d^*)_d = \lambda^*$) to be the shortest path of P_d' with respect to link metrics induced by π^*. Now suppose

that there exists a route p_d for a demand d of D outside the list P_d' that is sharply shorter with respect to metrics π^* than λ_d^*. Adding this route to the list P_d' and the corresponding constraint (2.20), we can see that the current optimal dual solution becomes non-feasible as a consequence of adding this new constraint, opening the possibility to decrease the optimal dual objective and at the same time decreasing the optimal primal objective.

$$\lambda_d \leq \sum_{e \in p_d} \pi_e \tag{2.20}$$

Hence, we could search for the shortest route between source and target nodes of the demand d and check whether the new possible constraint (2.20) would be violated in the current solution. If it is violated, in other words, if equation (2.21) is satisfied, then the current solution is not optimal and p_d is one of the missing routes for the demand d. Moreover, if for every demand d in D the new constraint is not violated means that the solution is optimal for the master problem.

$$Q(\lambda_d, \pi) := \lambda_d - \sum_{e \in p_d} \pi_e > 0 \tag{2.21}$$

Notice that we can try to find few new route for every demand d in D and add as many columns (routes) as desired in each iteration of the procedure. In the next path-generation algorithm one route for each demand is added as long as new useful routes are found.

Table 2-4: Path Generation Algorithm

```
INPUT: P'
OUTPUT: x*
1:   (x*,z*,λ*,π*) ← Solve the RMP with columns P'
2:   for each d in D do
3:      p_d ← Shortest path with metric π*
4:   while Q(λ_d*,π*)> 0 (new routes are found) do
5:      for each d in D do
6:         P_d' ← P_d' U { p_d }
7:      (x*,z*,λ*,π*) ← Solve the RMP with columns P'
```

2.2.5 Column Generation for MIP's

The column generation procedure is a method for linear programming problems. Despite this, the problems we are to solve evolve integer variables; for this reason we present some techniques that can be used after solving the relaxed integer problem by means of column generation in order to derive a heuristic MIP solution as similar as possible to the relaxed solution.

One technique is to apply *branch-and-bound* (*B&B*) to the restricted master problem with the restricted set of variables including the added columns. This technique is often called *Price and Branch* (*P&B*) and differs from *Branch and*

Price (*B&P*) on when the columns are added. Specifically, the P&B procedure applies the pricing to add new columns only in the root node of the B&B method, whereas the B&P technique applies the pricing in each node of the branching tree. The benefit of proceeding this way instead of applying *B&B* directly to the original problem is that probably the original problem could not be solved by means of *B&B* method or it can take too long. However, after applying column generation the number of variables should be much smaller than the original one and hence possible to solve (or solve it faster) by *B&B*.

Another technique that can be applied is the so called *rounding method,* which, at each iteration, sets few variables rounding their values to integers and solves the restricted master problem using (iteratively) the column generation procedure until an optimal solution is reached. Since few variables are set as integer at each iteration of the rounding method, it finishes providing an integer solution.

2.3 Summary

In this section the main concepts of optical networks and operation research were presented. First, the necessary definitions such as node, link, optical spectrum and demand were introduced. Then the concept of lightpath and the RWA were presented. The highlight was focused on flex-grid optical networks defining what slices, slots and channels are. Moreover the RSA problem and the continuity and contiguity constraints were stated.

In the operations research background, network problems and the way they can be formulated, the node-link and the link-path formulations were introduced, Then, a general procedure of the column generation method for linear programming problems was explained. The path generation procedure was described with more details stating the restricted master, the dual and the pricing problems as well as a detailed pseudo-code of the path-generation algorithm. Since a shortest-path algorithm is needed in order to solve the pricing problem, Dijkstra and Floyd-Warshall algorithms and pseudo-codes were explained in detail. Finally, two techniques to obtain integer solutions after applying the column generation method were described.

In the following chapter, the details of the LIGERO algorithm, using some of the concepts described in this chapter, will be exposed.

Chapter 3.

Lightpath Generation Algorithm

This section is devoted to the *LIghtpath GEneRation algOrithm* (LIGERO). To this aim, the problem statement of the considered RSA problem is presented in the context previously defined. A mixed integer formulation of the problem is proposed and the derivation of the dual and pricing problems are introduced for a complete understanding of the proposed methodology. Then, two path-generation based algorithms are proposed. Finally, three strategies to start our approach with an initial set of lightpaths are defined.

3.1 Problem statement and formulation

3.1.1 Problem Statement

The problem we aim to solve is briefly defined as follows: for each demand, we want to find the route over the flexgrid optical network and the spectrum allocation to minimize the number of rejected demands (primary objective) and the amount of un-served bitrate (secondary objective). The served bitrate of each demand is a value between the minimum and the maximum bitrate respectively. The detailed problem statement is the following:

Given:

- A network topology represented by a graph *G(V,E)*, being *V* the set of nodes and *E* the set of bidirectional fiber links connecting two nodes.
- A set *S* of available slots of a given spectral width for each link in *E*.
- A set of allowable channels C_d for each demand *d* of *D*.
- A set *D* of demands to be served. Each demand *d* is defined by its source node (s_d), its target node (t_d), its minimum (h_d) and maximum (H_d) bitrates.

- An initial set of allowable lightpaths P_d for every demand d. The full set of lightpaths is denoted by P.

Output:

- The lightpaths used to serve the demands.
- The set of blocked demands
- The amount of un-served bandwidth.

Objective:

- Primary objective: minimize the number of blocked demands.
- Secondary objective: minimize the total amount of un-served bandwidth.

Constraints:

- *Spectrum contiguity*: the subset of slots, that joined with a route, perform a lightpath have to be contiguous (adjacent). Channels are defined and used with this purpose.
- *Spectrum continuity*: the spectrum frequency should be the same along all the links on the route. In other words, the same channel must be used throughout every link on the route.
- *Channel capacity*: a channel cannot serve a bitrate higher than its capacity, which is proportional to the number of slots that compose it.
- *Slot capacity*: a slot in a link can be allocated, at most, to one demand.

3.1.2 Notation

In this section a list with the sets and the parameters is given in order to summarize all the input information necessary to formulate the mathematical programming problem. Later the decision variables used in the model are listed.

Parameters:

D	Set of demands
h_d	Minimum volume of demand $d \in D$ (in Gbps)
H_d	Maximum volume of demand $d \in D$ (in Gbps)
E	Set of network links
S	Set of spectrum slots at a link
B	Bandwidth carried by one slot
$n_d = \lceil h_d/B \rceil$	Number of slots required to carry h_d, $d \in D$
$N_d = \lceil H_d/B \rceil$	Number of slots required to carry H_d, $d \in D$

C	Set of channels
m_c	Number of slots used by channel $c \in C$
$S(c)$	Set of (contiguous) slots composing channel c
$C_d \subset C$	Channels allowable for demand $d \in D$
	$c \in C_d \leftrightarrow n_d \le m_c \le N_d$
P_d	Set of allowable lightpaths for demand $d \in D$
$P = \cup_{d \in D} P_d$	Set of all allowable lightpaths
$Q_{des} \subseteq P_d$	Set of lightpaths of d using slot $s \in S$ on link $e \in E$
$E(p)$	Set of links traversed by lightpath $p \in P$
g_p	Bandwidth carried on lightpath $p \in P$, computed as follows: $g_p = \begin{cases} H_{d(p)} & \text{if } m_{c(p)} = N_{d(p)} \\ B \cdot m_{c(p)} & \text{if } n_{d(p)} \le m_{c(p)} < N_{d(p)} \end{cases}$
A	Weight for objective function

Decision Variables:

$X_d,\ d \in D$	binary, $X_d = 1$ when d is blocked
$x_{dp},\ d \in D, p \in P_d$	binary, $x_{dp} = 1$ when route p carries its demand d
$Y_d,\ d \in D$	continuous, un-served bandwidth of demand d

3.1.3 Primal ILP formulation

The primal formulation of the above-stated problem (RSA-P) is as follows:

$$(\text{RSA}-\text{P}) \quad \textit{Minimize} \quad F = A\sum_{d \in D} h_d X_d + \sum_{d \in D} Y_d \tag{3.1}$$

Subject to:

$$X_d + \sum_{p \in P_d} x_{dp} = 1 \quad d \in D \tag{3.2}$$

$$\sum_{d \in D} \sum_{p \in Q_{des}} x_{dp} \le 1 \quad e \in E, s \in S \tag{3.3}$$

$$Y_d + \sum_{p \in P_d} g_p x_{dp} = H_d \quad d \in D \tag{3.4}$$

$$\begin{aligned} & X_d \in \{0,1\}, Y_d \in \Re \\ & x_{dp} \in \{0,1\} \end{aligned} \quad d \in D, p \in P_d \tag{3.5}$$

The objective function (3.1) minimizes the number of rejected demands, using a weight factor *A*, and the amount of un-served bitrate. Each constraint form set (3.2) either assigns a lightpath or blocks the demand. Moreover, it guarantees that at most one lightpath will be assigned per demand. Constraints (3.3) make sure that capacity of slots is not violated, by ensuring that the number of used routes sharing one specific slot is at most equal to 1. Constraints (3.4) set Y_d as the un-served bitrate. Finally constraints (3.5) define the type of decision variables and limit their range.

3.2 The LIGERO algorithm

In this section, the LIGERO algorithm is presented in the context of the problem previously defined. Before that, the derivation of the pricing problem is introduced for a complete understanding of the proposed methodology. Two approaches for solving LIGERO are presented and few improving techniques are explained afterwards. Finally, three different strategies to generate an initial set of lightpaths are defined.

3.2.1 Master Problem

The *Master Problem* (RSA-M) formulation is the same as RSA-P besides that integrity of variables has been relaxed by continuity in the real domain, becoming thus a LP formulation. Equations (3.6) to (3.10) show the RSA-M with the associated dual variables for each constraint (in square brackets at left side of constraints). A second linear problem with the same algebraic form can be defined, the *Restricted Master Problem* (RSA-RM), that contains a subset of those variables present in the RSA-M problem. More specifically, we consider that RSA-RM contains the whole set of *X* and *Y* variables and a subset of all admissible *x* variables (i.e. routes). Since there are no appreciable differences between RSA-M and RSA-RM algebraic formulations, we specify the following for both:

$$(\text{RSA}-[\text{R}]\text{M}) \quad \textit{Minimize} \quad F = A\sum_{d\in D} h_d X_d + \sum_{d\in D} Y_d \tag{3.6}$$

Subject to:

$$[\lambda_d] \quad X_d + \sum_{p\in P_d} x_{dp} = 1 \quad d \in D \tag{3.7}$$

$$[\pi_{es} \geq 0] \quad \sum_{d \in D} \sum_{p \in Q_{des}} x_{dp} \leq 1 \quad e \in E, s \in S \tag{3.8}$$

$$[\gamma_d] \quad Y_d + \sum_{p \in P_d} g_p x_{dp} = H_d \quad d \in D \tag{3.9}$$

$$\begin{matrix} X_d \geq 0, Y_d \text{ unconstrained in sign} \\ x_{dp} \geq 0 \end{matrix} \qquad d \in D, p \in P_d \tag{3.10}$$

From the dual variables specified in the above-defined RSA-M formulation and applying a common technique based on Lagrangean Relaxation (see [Pi04] for more details), we obtain the dual formulation (RSA-D). Specifically, we first obtain the Lagrangean function L of the RSA-M problem, which combines primal and dual variables as denoted in equation (3.11).

$$\begin{aligned} L(x, X, Y; \lambda, \pi, \gamma) = & \sum_{d \in D} \gamma_d H_d + \sum_{d \in D} \lambda_d - \sum_{e \in E} \sum_{s \in S} \pi_{es} \\ & + \sum_{d \in D} (Ah_d - \lambda_d) X_d + \\ & + \sum_{d \in D} (1 - \gamma_d) Y_d + \\ & + \sum_{d \in D} \sum_{p \in P_d} \left(-\lambda_d - g_p + \sum_{e \in E(p)} \sum_{s \in S(p)} \pi_{es} \right) x_{dp} \end{aligned} \tag{3.11}$$

Then, from equation (3.11) the derivation of the RSA-D formulation is straightforward:

$$(\text{RSA} - \text{D}) \quad \textit{Maximize} \quad W = \sum_{d \in D} \gamma_d H_d + \sum_{d \in D} \lambda_d - \sum_{e \in E} \sum_{s \in S} \pi_{es} \tag{3.12}$$

Subject to:

$$\gamma_d = 1 \quad d \in D \tag{3.13}$$

$$\lambda_d \leq AH_d \quad d \in D \tag{3.14}$$

$$\lambda_d + g_p \leq \sum_{e \in E(p)} \sum_{s \in S(p)} \pi_{es} \quad d \in D, p \in P_d \tag{3.15}$$

$$\pi_{es} \geq 0 \quad e \in E, s \in S \tag{3.16}$$

3.2.2 Pricing Problem

At this point, we recall that the reduced cost of a variable x_{dp} derived from equation (3.15) must be non-positive at the optimal solution. The reduced cost of variable x_{dp} is defined as z_{dp} in equation (3.17).

$$z_{dp} = \lambda_d + g_p - \sum_{e \in E(p)} \sum_{s \in S(p)} \pi_{es} \quad d \in D, p \in P_d \tag{3.17}$$

When the RSA-RM is solved, if we want to improve the reached optimal solution, new variables with positive reduced costs must be found. Thus, given a demand d and a new lightpath p^* not included in current P_d set, the lightpath p^* could be useful to improve the current solution if and only if its reduced cost z_{dp^*} is greater than 0. Equation (3.18) shows which condition concerning dual variables must be met to include the new lightpath.

$$z_{dp^*} > 0 \quad \Leftrightarrow \sum_{e \in E(p^*)} \sum_{s \in S(p^*)} \pi_{es} < \lambda_d + g_p \tag{3.18}$$

In fact, in order to obtain the lightpath p that most improves the current solution; we want to find the reduced cost with the maximum value. This is, indeed, the pricing problem for our LIGERO procedure, whose model is shown in equation (3.19). Note that the pricing problem is independent for each demand d in D, so we can find a best improving p not included in P_d for each d in D.

$$\textit{(RSA-PP)} \quad \textit{Maximize} \{ z_{dp} = \lambda_d + g_p - \sum_{e \in E(p)} \sum_{s \in S(p)} \pi_{es} : d \in D, p \in P_d \} \tag{3.19}$$

Note that the pricing problem has three distinguishable components. The dual variables λ_d only depend on the chosen demand; the parameter g_p takes the same values for lightpaths with a same number of slots; whereas the factor evolving π variables depends on the route and the specific slots. Hence, for a given demand and a given channel size (between n_d and N_d) the lightpath with the maximum reduced cost is the one with the shortest route in terms of the metric induced by π. This metric assigns to the link e the cost $f_e(c)$ given by equation (3.20).

$$f_e(c) = \sum_{s \in C} \pi_{es} \quad e \in E \tag{3.20}$$

3.2.3 Main Algorithm

Before presenting the LIGERO algorithm in more detail, we define the main procedure that iteratively executes our algorithm and which is common for any path generation method. This main algorithm (detailed in Table 3-1) starts generating an initial set of lightpaths and solves the RSA-RM linear formulation. Then the LIGERO algorithm is iteratively executed as long as it finds new lightpaths to be added to the RSA-RM. Every time a subset of new routes is found, it is added to the set of existing routes and the problem is re-solved. When no more routes are found, we can ensure that the last solution is optimal not only for the RSA-RM but also for the RSA-M.

Table 3-1 Main Algorithm Pseudo-code

```
INPUT: Network G(V,E), Demand set D
OUTPUT: Solution
1:  Define an initial set P_d of lightpaths, d ∈ D
2:  P ← U_{d∈D} P_d
3:  P'← P
4:  while P' ≠ ∅ do
5:      Solution ← Solve RSA-MR with lightpaths in P
6:      P'← Solve LIGERO
7:      P ← P ∪ P'
```

In the following section two versions of LIGERO are presented: the *Dijkstra-based* (D-B) version and the *Floyd-Warshall-based* (FW-B) version. The first one, as it names suggests, applies several Dijkstra algorithm computations for each demand (one for each possible channel size and allocation in the spectrum) in order to find the most suitable lightpath to be added to the RSA-RM. With the aim to provide a less complex version, the FW-B version computes Floyd-Warshall to obtain the cost of shortest paths between all node pairs for each possible channel allocation. With this action, the detection of those demands needing new variables (with the reduced cost associated to them) should be obtained faster than using an exhaustive search like the proposed in D-B version. Then, only for those demands for which a suitable new lightpath is found, the Dijkstra algorithm is afterwards applied to obtain the explicit route.

3.2.4 Dijkstra-Based Version

Table 3-2 shows the pseudo-code of the D-B version. This algorithm finds, for each demand, the lightpath with the highest positive reduced cost, which is the one that would provide the highest improvement on the quality of the solution. To this aim, the shortest route over the network with metric links depending on π dual variables is found. Since π variables are related with single slots in links, the metric of a given link (f_e) depends on the selected channel and is computed as defined in equation (3.20). Since link metrics depend on the channel selected, it is necessary to compute several shortest routes (one for all possible channels whose number of slots is between n_d and N_d) in order to find the best lightpath to enter. However, this search can be early stopped when some conditions are satisfied ensuring that the best route has been found (if it exists). These conditions to improve the algorithm are explained later.

Given a certain channel size, we know before route computation whether it is necessary to explore channels of such size. Thus, we define z_{max} as the largest value that z can reach for a given demand d and any channel of n slots. This z_{max} value is computed from those elements that do not depend on the route and assuming that a route with the minimum cost over the current network is found. Although we can assume that this minimum cost for a route can be equal to 0, we can slightly

improve z_{max} by subtracting the value of the lowest π variable. If this z_{max} is lower than 0, we can conclude that no better routes than existing ones can be found for this size and, therefore, we avoid computing a significant number of routes. In case of obtaining a positive z_{max} bound, this size should be explored until either all possible channels have been studied or a route with $z^* = z_{max}$ is found.

Table 3-2 D-B LIGERO Algorithm Pseudo-code

```
INPUT: G(V,E),D, Solution
OUTPUT: P'
1:    P'= Ø
2:    λ, π, γ ← get duals from Solution
3:    π_min ← min(π_es), e ∈ E, s ∈ S
4:    for each d in D do
5:       z* ← -inf
6:       newP_d=Ø
7:       for (n=N_d; n≥n_d; n--) do
8:           g'← carried bandwidth in a n-slot channel for d
9:            z_max ← λ_d + g' - π_min
10:           if z_max≤0 ||z*≥z_max then
11:               break
12:           Create C with all channels of n slots
13:           for each channel c ∈ C do
14:               Update link metrics f using equation (3.20)
15:               r_d ← Shortest route of demand d over G (Dijkstra)
16:               z ← λ_d - g' - dist(r)
17:               if z □ 0 and z > z* then
18:                   z* ← z
19:                   Add p_d = (r_d,c) (r with cost z) to newP_d
20:                   if z*≥ α z_max then
21:                       break
22:       if newP_d≠Ø then
23:           p_d = {p ∈ newP_d | maximum z}
24:           P'=P' U p_d
```

In addition to the previous condition, we can also ensure that when z_{max} is either worse than the incumbent z^* or non-positive for a given channel size, then we can skip searching remaining channels sizes. For the sake of clarity, let *g(n)* and *g(m)* be the served bitrate in a channel of n and m slots respectively. Assuming that $n<m$, then by definition of parameter g, we can conclude that g(n)≤ g(m). Thus, the value of z_{max} for channels of size n will be lower than those of size m. This condition allows stopping lightpath search when $z_{max} \leq 0$ or $z_{max} \leq z^*$ for a given channel size. Note that this property is useful if and only if channel sizes are explored from highest to lowest size.

Aiming at reducing even more the complexity of route search, we could assume that if a new lightpath is found with z^* accomplishing that $z^* \geq \alpha\ z_{max}$, where α belongs to the interval [0,1], then that channel size will be explored no more. In our studies, however, we consider that $\alpha = 1$.

Finally and in order to compute the complexity of this LIGERO version, let us consider the case when all demands have the same size and, therefore, the set of

possible channels to serve a demand (i.e. C) is equal for all demands. Additionally, we consider that the complexity of Dijkstra algorithm can be assumed as $O(|V|^2)$, where $|V|$ is the cardinality of the node set (i.e. the number of nodes). Then, the complexity of D-B algorithm is $|D| \cdot |C| \cdot O(|V|^2)$.

3.2.5 Floyd-Warshall-Based Version

The FW-B version of LIGERO, shown in Table 3-3, computes, for each channel size and for every channel of the considered size, the cost of the shortest routes over the network with metric given by equation (3.20) for every demand using Floyd-Warshall algorithm. Recall that Floyd-Warshall algorithm computes the costs of the shortest route but does not return explicitly the route. Then for each demand it computes the reduced cost z following the equation (3.17); if z is as good as the incumbent value z^* then the corresponding demand and channel are added into the list T. It is important to emphasize that if a better solution is found, then all the elements in T are removed and only the new pair demand-channel is kept. When z has been computed for all channels of every size, we iterate over all the demands in order to randomly choose a channel from $T_d = \{[d',c] \in T : d' = d\}$. This channel is used to update the metric of the network according to equation (3.20) and the shortest route of the considered demand is computed using the Dijkstra algorithm. This pair channel-route performs the lightpath that LIGERO algorithm provides to the RMP for the considered demand.

Table 3-3 FW-B Algorithm Pseudo-code

```
INPUT: G(V,E),D,Solution
OUTPUT: P'
1:    P'= Ø
2:    λ, π, γ ← get duals from Solution
3:    for each d in D do
4:       z_d* ← -inf
5:    n_min ← min{n_d : d in D}
6:    N_max ← max{N_d : d in D}
7:    for (n=n_min; n≤N_max; n++) do
8:       for each channel c ∈ C_n do
9:          Update link metrics f using equation (3.20)
10:         Solve Floyd-Warshall Algorithm
11:         for each d in D do
12:            C_d ← Cost of shortest route for demand d (F-W)
13:            b_c ← min(nB, H_d)
14:            z ← λ_d - μ_d + b_c γ_d - C_d
15:            if z≥0 || z≥ z_d* then
16:               z* ← z
17:               T = Ø
18:            if z= z_d* then
19:               T ← T U [d,c]
20:   for each d in D do
21:      c ← channel randomly selected from T_d
22:      Update link metrics f using equation (3.20)
23:      r_d ← Shortest route of demand d over network (Dijkstra)
24:      p_d ← [r_d,c]
25:      P'=P' U p_d
```

For analyzing the complexity of this algorithm, we take the same assumption than for D-B and also assume that the complexity of Floyd-Warshall is $O(|V|^3)$. Thus, the complexity is $|C| \cdot O(|V|^3) + |D| \cdot O(|V|^2)$. Since in real networks we can state that $|V| << |D|$ (we will see the examples in next chapter), then the following comparison can be done:

$$O(FW\text{-}B) = |C| \cdot O(|V|^3) + |D| \cdot O(|V|^2) < |D| \cdot |C| \cdot O(|V|^2) = O(D\text{-}B)$$

Nevertheless, we will see the performance of both versions when solving real instances before taking conclusions about the complexity of the algorithms.

3.2.6 Initial Sets of Lightpaths

For evaluating the impact of the initial set of lightpaths P on the LIGERO performance, we have used three initialization procedures, referred to as *MaxLeft*, *GreedyRSA,* and *MinAll*. For the sake of clarity, a detailed example over the network depicted in Figure 3-1 is performed to give a better understanding about how these three methods proceed. This network has to supply two demands whose routes concur in the fiber link A-D, so that the corresponding channels have to share resources appropriately. Let us assume that the slot width and the spectral efficiency are such that the optical spectrum is divided in 10 slots. Suppose that the minimum and maximum bandwidth for the first demand corresponds to 2 and 5 slots respectively; whereas the minimum and maximum bitrate for the second demand need an amount of 1 and 4 slots.

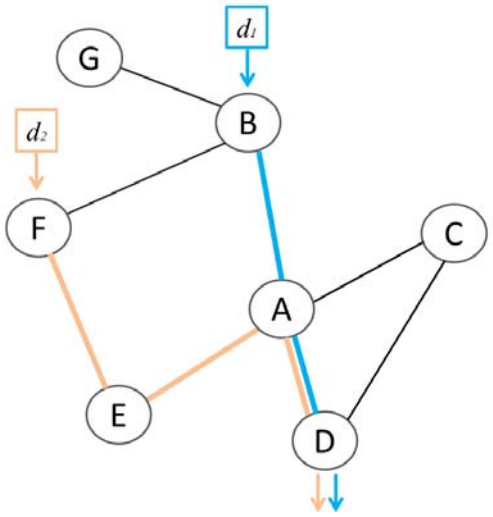


Figure 3-1 Example of a network with two demands

The *MaxLeft* procedure assigns the channel with the first N_d slots to each demand. Table 3-4 shows the algorithm used in order to get this initial solution. Note that this initial solution is the same as what LIGERO would do if the initial set of variables was empty. However it is necessary an initial set of dual variables for the

first iteration of the algorithm so that it is necessary a first solution of the relaxed problem.

Table 3-4 MaxLeft procedure

```
INPUT: G(V,E),Demands
OUTPUT: P
1:    for each d in Demands
2:       N_d ← Necessary number of slots to satisfy H_d
3:       sr_d ← Shortest route using Dijkstra algorithm
4:       c_left ← First channel of size N_d
5:       p_d ← (sr_d ,c_left)
6:       P  ← P U {p_d}
```

For the example explained above, the *MaxLeft* initial procedure offers for each demand the most left channel. Thus, since they share a fiber link and they cannot use the same slots, for the solution at the first iteration one of them has to be chosen. The channel for demand 1 is preferred since it provides a higher bandwidth (more slots). Figure 3-2 depicts the explained example.

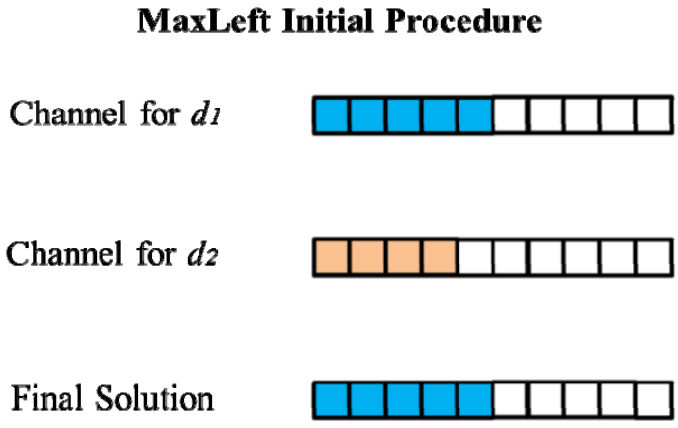


Figure 3-2 *MaxLeft* illustrative example

The second method, called *GreedyRSA*, is similar to *MaxLeft* except because the assigned lightpath is not the leftmost one. Instead the demands are previously sort in the decreasing order of $H(d)$ and processed sequentially in this order. Then a simple first-fit is applied in order to find (in the shortest route) enough capacity to serve H_d. The first demand is assigned to its shortest route $sr(d_1)$ and the leftmost channel c_{left} as in *MaxLeft* (since all the slots are still free) and the slots used by this channel are permanently deleted from the links of $sr(d_1)$. Next, for d_2 we find the shortest route and checks whether a channel of size N_d is available in all the links of the route. If there is a free channel of size N_d in the shortest route then they are assigned to d_2, otherwise it searches a free channel of the same size in the second shortest route. It searches for a free channel in at most the K shortest routes; if any of them could not be used then it assigns the demand to the shortest

route with the left channel of size N_d. The algorithm to perform this procedure is shown in Table 3-5.

Table 3-5: GreedyRSA procedure

```
INPUT: G(V,E),Demands
OUTPUT: P
Demands set is sorted in the decreasing order of H(d)
for each d in Demands
    N_d ← Necessary number of slots to satisfy H_d
    ksr_d ← K-shortest routes using Dijkstra algorithm.
    for each route in ksr_d (from shortest to longest)
        for each channel c of size N_d
            if c is not used by any link of route
                p_d ← (route,c)
                P ← P U {p_d}
                break
        if p_d ≠ ø
            break
if p_d = ø
    c_first ← First channel of size N_d
    p_d ← (sr_d ,c_first)
    P ← P U {p_d}
```

In our illustrative example, the *GreedyRSA* initial procedure, has enough number of slots to supply both requested demands from the beginning in the shortest route (the ones we are considering). Then it provides channels that do not share any slot, so the final solution can serve both demands and no further iterations would be needed. This is depicted in Figure 3-3.

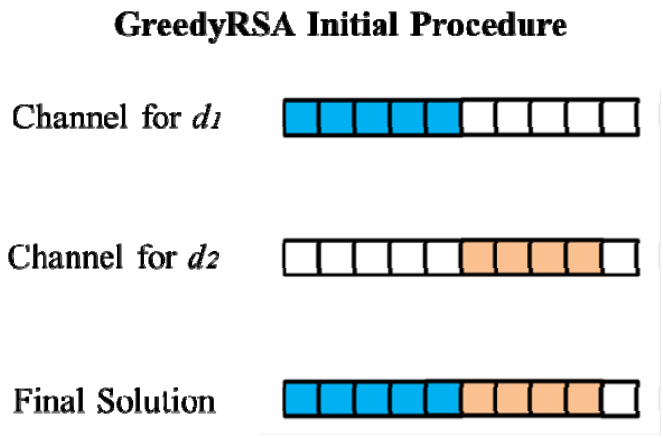


Figure 3-3 *GreedyRSA* illustrative example

The two above described procedures assign the maximum number of slots to the selected demand lightpaths. Hence, using lightpaths P produced by *MaxLeft* or *GreedyRSA* will result in a solution with two groups of demands: those with maximum bandwidth allocation and those not realized at all (blocked). Since the objective function strongly penalizes (through A big enough) the presence of demands of the latter group, we consider the third initialization procedure, *MinAll*,

that aims at alleviating this penalization. This initialization procedure provides, for each demand, all the lightpaths which are a combination of the shortest route and a channel of size n_d. Namely, the set of lightpaths for each demand is given by the equation (3.21)

$$P_d = \{(sr_d, c) \ : \ c \in C_d, \ m_c = n_d\} \tag{3.21}$$

Note that the number of initial variables that are obtained with this method is much larger than the obtained with the other two initial solutions. Table 3-6 shows the pseudo-code of the initial procedure explained above.

Table 3-6 MinAll procedure

```
INPUT: G(V,E),Demands
OUTPUT: P
1:    for each d in Demands
2:       n_d ← Necessary number of slots to satisfy h_d
3:       sr_d ← Shortest route using Dijkstra algorithm
4:       for each channel c with size n_d do
5:          p_d  ← (sr_d,c)
6:          P  ← P U {p_d}
```

The example for the *MinAll* procedure shows that, for each of the demands, it provides all the possible channels of minimum size; it is 2 and 1 slots respectively. This is a total amount of 9 channels of 2 slots for the first demand and 10 channels of 1 slot for the second demand. The final solution provides any combination that supplies both demands with their minimum bandwidth, since the model contains all the lightpaths with these channels. In order to illustrate a solution, demand 1 uses its second channel and demand 2 choses channel number 8. From this point, each iteration applying LIGERO will add lightpaths with more capacity than the minimum in order to decrease the un-served bandwidth.

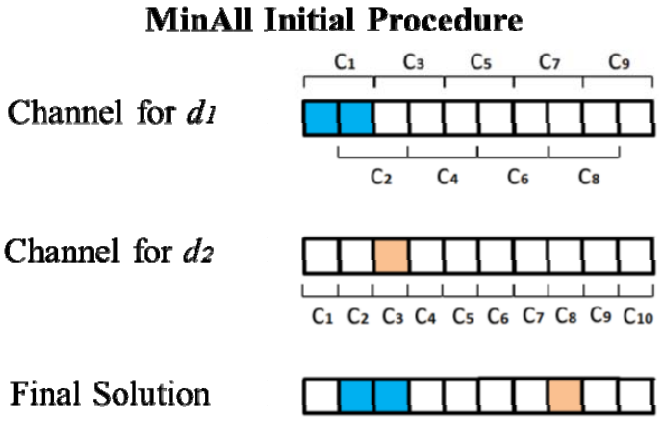


Figure 3-4 *MinAll* illustrative example

3.3 Summary

In this section we have studied the problem statement and a proposed mixed integer linear programming problem formulation. Then, the derivation of the dual and pricing problems have been given and linked with the proposed lightpath generation methodology. After that, two lightpath generation algorithms have been shown highlighting the difference between them. Finally, three strategies to start the LIGERO approach with an initial set of lightpaths have been proposed with emphasis in the different objectives they have been geared toward.

The numerical evaluation of the algorithm here presented will be done in the following chapter.

Chapter 4.

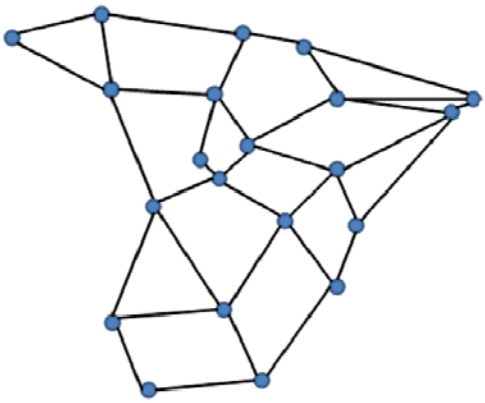
Numerical Results

In this chapter we study and compare the two versions of the LIGERO method and the three techniques used to generate initial set of lightpaths. In order to choose the best version of LIGERO and the most suitable initial procedure, an analysis with different loads is performed. Hereafter a large-scale evaluation of the LIGERO algorithm is made for a large number of demands and channels. For the sake of comparing the quality of the generated variables by the LIGERO procedure, a stochastic approach to generate variables is eventually performed with the aim to produce the same number of lightpaths than LIGERO.

4.1 Reference Scenario

For numerical evaluation we consider the 21-node Spanish Telefónica optical network (detailed in Figure 4-1) [Ca12]. For this network we created instances consisting in sets of randomly generated demands. For each one of these demands, first a random source/destination pair is selected, being each possible pair chosen with the same probability. Then, a class of the traffic profile defined in Figure 4-1 is assigned with a certain probability. Each class is defined by its specific values of minimum and maximum bitrate and required number of slots. For computing the number of required slots, we assumed a slot width of 12.5 GHz and a spectral efficiency of 2bit/s/Hz, so that the bitrate that a single slot can carry is up to 25 Gbps. To refer us to the size of an instance we will use equally the number of demands or the total load of bitrate to serve (in Tbps).

We implemented the LIGERO algorithm in Matlab [MLB] and making use of the linear solver engine in CPLEX 12.2 [CPLX]. We run all experiments on a 2:4GHz Quad-core machine with 8GB RAM running Linux.



Telefónica (TEL) network

# of nodes ($\|V\|$)	21
# of links ($\|E\|$)	37
# of slots ($\|S\|$)	40

Traffic Profile (TP)

h (Gbps)	H (Gbps)	n (# slots)	N (# slots)	%
10	40	1	2	25%
40	100	2	4	25%
100	400	4	16	50%

Figure 4-1 Evaluated Network and Traffic Characteristics

Aiming at providing a reference for the later performance evaluation of our LIGERO approach, we generated medium instances which were solved to primal optimality by applying CPLEX. To solve ILP formulations we generated sets of pre-computed lightpaths as proposed in [Ve12]. Specifically, for each demand we computed the K shortest routes (with K=5) and, for each one of them, we created one lightpath for any possible channel with capacity ranging from n_d to N_d slots. In this work, we will call this method *K-PreComputed Lightpaths* (K-PCL). We set a 10 hours limit to CPLEX in order to find the optimal integer solution; otherwise the best integer solution and the optimality gap were returned. Although this time could seem short for a network planning problem, in the next section we will show (with a real application) that the execution time is a limiting factor in real optical network problems. Table 4-1 show the obtained average values of 5 randomly generated independent runs per traffic load.

Table 4-1 ILP solutions for small/medium instances

Total load (in Tbps)	# demands ($\|D\|$)	# routes ($\|P\|$)	linear relax.	integer upper bound	Time to solve(in sec.)	optimality MIP gap
2.5	40	27230	0	0	59.20	0
3	48	32666	0	0	2,280	0%
3.5	56	38122	0	6.67	8,502	20%(1)
4	64	43568	220	334	36,000(2)	42%

(1) – Only one instance exceeded 10 hour time limit, with gap = 100%

(2) All instances exceeded 10 hour limit (36,000 seconds)

For loads 2.5 and 3 Tbps, both the relaxed and the integer solutions reach 0 as optimal objective value. This indicates that these requested loads are low enough to be fully served. Contrarily, loads between 3.5 and 4 Tbps show positive integer objective functions, which implies that some instances had few demands which have been blocked or partial un-served. Although it is not visible in the

summarized table, the amount of rejected demands and/or un-served bandwidth in these higher loads represent common and accepted values in real in-operation working networks [Kl13]. Note that 4 Tbps instances exhausted the 10 hour limit without reaching the optimal. Hereafter, we will put special attention to these latter instances since, finding high-quality solutions requires much effort than instances with lower loads.

4.2 Analysis of proposed algorithms

For the instances described in the previous section, we applied our two LIGERO versions with all initial lightpath set generation procedures. Table 4-2 and Table 4-3 show the number of initial lightpaths and final (i.e. initial + generated) ones; the number of LIGERO iterations; the value of the relaxed objective function, and the time to reach such optimal relaxation. The comparison of the detailed experiment leads to the main following observation: the value of the optimal relaxation is met for all initial strategies and both LIGERO versions (compare with relaxation values in Table 4-1). This allows validating our LIGERO algorithms since the optimal relaxation of the original problem is always met and, consequently, this convergence is dependent neither on the version nor on the initial set of lightpaths.

Table 4-2 LIGERO Dijkstra Performance (medium instances)

	MaxLeft					***MinAll***					***GreedyRSA***				
Total load Tbps	**init. var**	**final var**	**iter.**	**linear relax.**	**time sec.**	**init. Var**	**final var**	**iter.**	**linear relax.**	**time sec.**	**init. var**	**final var**	**iter.**	**linear relax.**	**time sec.**
2.5	40	1025	27	0	33.9	1530	2527	27	0	35.9	40	374	8	0	6.3
3.0	48	1512	32	0	66.1	1836	3255	31	0	68.4	48	963	20	0	38.8
3.5	56	1639	28	0	76.1	2142	3787	30	0	85.5	56	1009	17	0	34.6
4.0	64	2639	46	220	361.3	2448	4314	30	220	146.1	64	2375	42	220	440.2

Table 4-3 LIGERO Floyd-Warshall Performance (medium instances)

	MaxLeft					***MinAll***					***GreedyRSA***				
Total load Tbps	**init. var**	**final var**	**iter.**	**linear relax.**	**time sec.**	**init. Var**	**final var**	**iter.**	**linear relax.**	**time sec.**	**init. var**	**final var**	**iter.**	**linear relax.**	**time sec.**
2.5	40	295	6	0	9.36	1530	1824	7	0	11.06	40	236	5	0	7.99
3.0	48	506	9	0	17.12	1836	2299	9	0	17.87	48	423	9	0	16.12
3.5	56	657	9	0	20.78	2142	2748	10	0	24.29	56	651	10	0	20.71
4.0	64	1653	24	220	66.25	2448	3381	13	220	37.07	64	1177	23	220	62.82

Focusing on the differences between LIGERO versions, we can see that the FW-B version converges in less iterations and faster than the D-B one. Specifically, the reductions in terms of execution time and number of iterations of the FW-B version in comparison with the D-B are in the range of [50%, 75%].

Going into a detailed analysis of the differences among initial procedures, it can be seen that both the total execution time and the number of LIGERO applications are strongly dependent on the set of previously existing lightpaths. For loads between 2.5 and 3.5 Tbps both LIGERO versions show lower number of iterations and execution times with the *GreedyRSA* procedure whereas for the 4.0 Tbps instances the *MinAll* strategy is the one that reaches lower values in the same terms. Even though the FW-B version does not show such significant differences, in relative terms the execution times and the number of iterations of this version has the same behaviour than the D-B one.

To complement the previous analysis, Figure 4-2 shows the execution time of LIGERO as a function of the load. In view of the figure, different behaviours are recognizable. While the time increases linearly with the load for the *MinAll* approach in both LIGERO versions, the performance of *MaxLeft* and *GreedyRSA* strategies is clearly worse since the time sharply increases with the highest load. Thus, the *MinAll* algorithm provides an initial set of variables that affects positively to the scalability of LIGERO when the size of the instances increases.

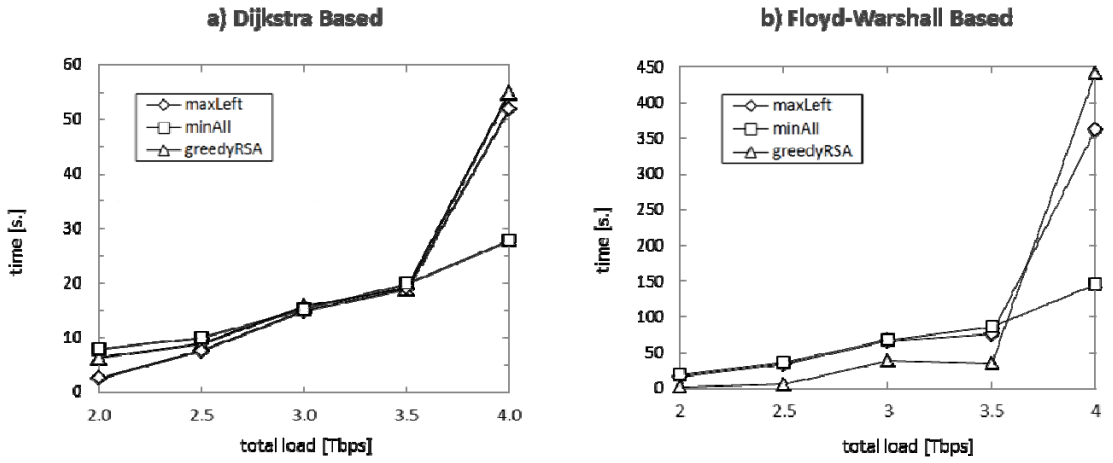


Figure 4-2 Effect of the initial set over LIGERO performance: time vs load for a) D-B, b) FW-B

Let us focalize now on load 4 Tbps, where similar values for *MaxLeft* and *GreedyRSA* strategies are observed, whereas *MinAll* provides significantly lower values (reductions of almost 50% in time and number of iterations). These differences are detailed in Figure 4-3 where the value of the relaxed optimal solution is depicted as a function of time for a representative single run of 4 Tbps. In view of the curves, we can see the high disparity of solutions before applying the LIGERO algorithm (the first marker of each curve) and how this initial solution is

improved along consecutive LIGERO iterations. As explained in previous results, *MinAll* provides the fastest solution in both LIGERO versions. More precisely, the convergence of *MinAll* approach to the optimal relaxation when the incumbent solution is near-optimal is faster than in the other cases. To support this conclusion, Table 4-4 shows, using the same example, the time needed to reach a near-optimal solution (whose objective function is 300) and the time needed for reaching the optimal (objective function equal to 200), as well as the time elapsed between both solutions (in seconds and in percentage with respect to the total execution time). As can be observed, the time to converge to the optimal when the incumbent is approaching is the shortest when *MinAll* is used. In relative terms, *MinAll* consumes around 35% of time in improving the near-optimal solution, whereas with the other strategies, this time raises up to 84%. Therefore, we conclude that the *MinAll* approach provides the best LIGERO performance independently of the version used, although we can also see that running times for FW-B version are remarkably smaller than times for D-B version.

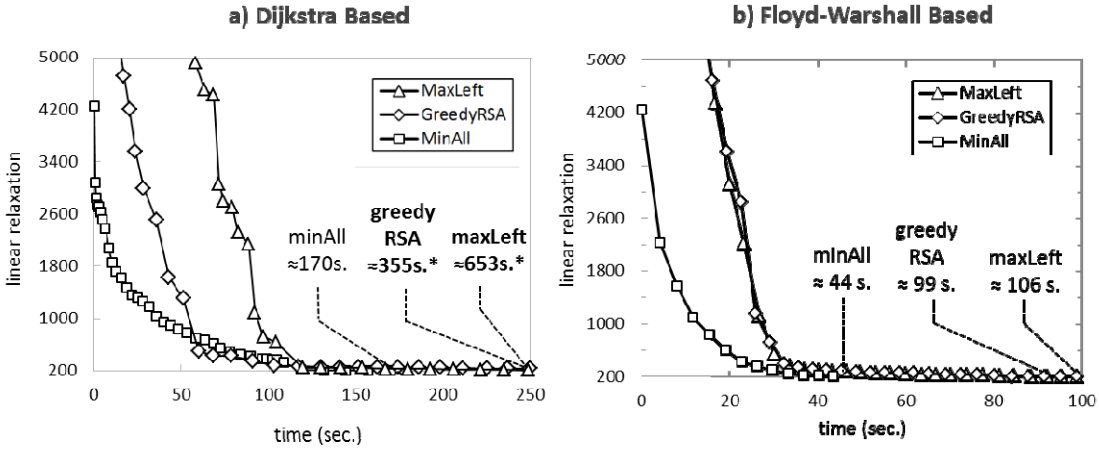


Figure 4-3: Effect of the initial lightpaths over LIGERO performance: o.f. vs time
* The time needed is out of the range of the figure

Table 4-4 Convergence from near-optimal to optimal

	D-B			***FW-B***		
Initial Strategy	***MaxLeft***	***MinAll***	***greedy RSA***	***MaxLeft***	***MinAll***	***greedy RSA***
Time to Near-Optimal Solution (=300) (sec.)	119.74	109.20	103.45	40.08	29.71	42.02
Time to Optimal Solution (=200) (sec.)	355.19	170.17	653.12	98.92	43.66	106.24
Elapsed Time (sec.)	235.45	60.97	549.67	*58.84*	*13.95*	64.22
Elapsed Time vs Total Time (%)	66.3%	35.8%	84.2%	59.5%	32.0%	60.4%

4.3 Quality of integer solutions

The previous results and conclusions concern to the quality of the proposed algorithms for obtaining optimal solutions for the linear relaxation of the problem. Aiming at evaluating the quality of the integer solutions obtained by applying the B&B algorithm after solving LIGERO, we have performed two different experiments.

Firstly, we have obtained the best integer solution after 10 hours of the 4 Tbps instances for all the possible combinations among LIGERO algorithm versions and initial procedures. The average results are shown in Table 4-5. In light of the results we can conclude that *MinAll* procedure provides the best values in terms of quality of the integer solution (see the accumulated values in the last row of Table 4-5). However, although the D-B LIGERO version presents a worse convergence and higher execution times and number of generated variables than FW-B one (shown in previous results), the quality of the integer solutions provided by D-B is better (as summarized in the last column of Table 4-5). This could be mainly due to the total amount of variables that this version contains after being applied, which is significantly higher than in FW-B version. Therefore, after comparing the performance of the LIGERO algorithm and the quality of the best integer solutions, we cannot conclude which version is better from this set of medium-size instances.

Table 4-5 Quality of Integer Solutions for medium instances

	MaxLeft	***MinAll***	***GreedyRSA***	***MaxLeft + MinAll + GreedyRSA***
D-B	717.00	454.00	872.00	**2043.00**
FW-B	672.00	570.00	1029.00	2271.00
D-B + FW-B	1389.00	**1024.00**	1901.00	

Secondly, we compared our LIGERO versions initialized using the *MinAll* procedure against other solutions with also a reduced number of variables. In this regard, the most direct way to reduce the number of lightpaths of the problem to solve consists of considering a lower number of pre-computed lightpaths for the RSA-P. This can be done by reducing the value of *K* for the K-PCL when computing different routes for each demand. Figure 4-4 depicts, for a given instance with 3 Tbps load, the objective function of the integer solution and the number of lightpaths obtained with both LIGERO methods and the K-PCL algorithm with K from 1 to 5. Note that we selected such load to ensure that the K-PCL method find the optimal integer solution in a reasonable time (not possible with higher loads).

In a first look to these results, we can also see how for the K-PCL method the number of variables increases drastically for larger values of K. Specifically, we stated that to obtain a solution as good as the one obtained by our methods, it is necessary to pre-compute all lightpaths from the shortest 4 routes. This represents a number of variables close to 27000, which is over an order of magnitude higher than the number of variables generated by our LIGERO approaches (lower than 2000 lightpaths). In view of this, we propose to use one of our methods to generate good sets of lightpaths instead of the commonly used K-PCL method.

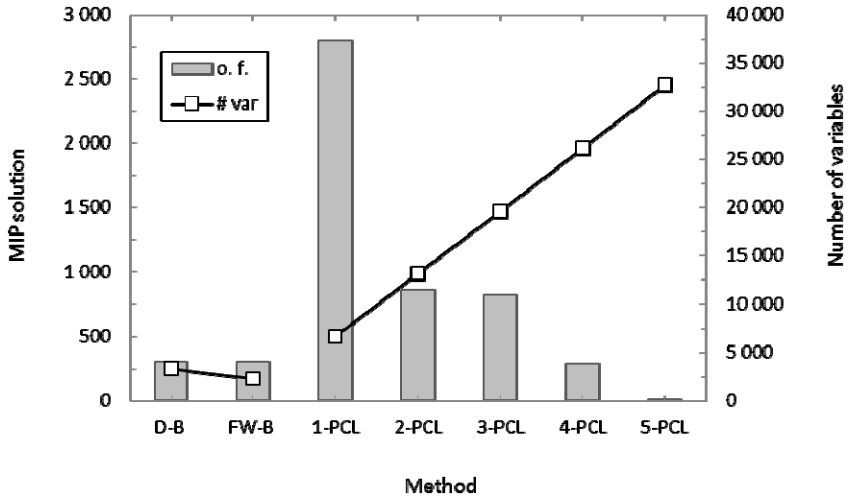


Figure 4-4: Quality of generated paths for LIGERO with *MinAll* and K-PCL

After this numerical evaluation, the selection of *MinAll* procedure as the best initial algorithm is clearly supported by the results (LIGERO performance and quality of integer solutions). Regarding the LIGERO version, the convergence of LIGERO is faster and better scalable with respect to the instance size for the FW-B version than for the D-B one. However, the quality of the obtained integer solutions (at a given B&B execution time) is better for the latter. In order to definitely decide which the best algorithm version is, we will illustrate the performance of LIGERO when solving large-scale instances in the next section. Recall that our goal is to provide an efficient method to solve instances involving thousands or millions of variables.

4.4 Performance Evaluation with Large Instances

To illustrate the applicability of this proposal, we generated large instances based on the same network topology but with large number of demands and spectrum slots. By also considering a large K value (equal to K=10, to practically ensure that optimal solutions will be reached), the number of variables rises to un-tractable values. Table 4-6 shows meaningful results for these large instances. In both cases

(K-PCL and LIGERO), the time limit set to CPLEX to return the best integer solution was 10 hours. Since FW-B adds some randomness in the final set of selected variables and this can affect to the quality of the integer solution, we repeat 5 times each load, showing in the table the average values. As can be observed, our methods provide, with up to 2 orders of magnitude less number of variables, better integer solutions in the same running time that simply applying CPLEX with pre-computed lightpaths. Moreover, in such cases when the number of variables is too large to allow generating the problem by CPLEX (*out-of-memory* messages appear), our LIGERO approaches provide an affordable way to obtain feasible and good-quality solutions. Furthermore, even though the integer upper bounds for both LIGERO versions are similar, the FW-B version clearly outperforms the D-B version in terms of number of lightpaths and integer objective values. For this reasons, we definitely chose the FW-B version of LIGERO as the best of our methods to solve the RSA problem with large instances.

Table 4-6 LIGERO Performance (Large instances)

			Pre-Computed Lightpaths			D-B LIGERO			FW-B LIGERO		
Total BW	**|D|**	**|S|**	**|P|**	**linear relax.**	**integer upper bound**	**|P|**	**linear relax.**	**integer upper bound**	**|P|**	**linear relax.**	**integer upper bound**
4	64	40	132400	520	1895	4176	520	670	3287	520	775
7	112	68	269640	0	7420	11017	0	835	9338	0	785
10	180	96	8675190	*out-of-memory*		22750	0	1020	17203	0	857

Finally, we compare the FW-B version of LIGERO against a random selection of lightpaths. Specifically, we performed a stochastic approach of the K-PCL (with *K* from 1 to 4) algorithm. This approach consists of a random selection of the lightpaths generated by the K-PCL algorithm, selecting exactly the same number of lightpaths than the LIGERO procedure generates. Table 4-7 exposes the average, the minimum and the maximum values of 5 repetitions of the same instance. It shows the number of lightpaths obtained with the FW-B version of LIGERO algorithm and the objective values for the relaxed and the incumbent solutions found running the LIGERO and the K-PCL (*K=1, 2, 3* and *4*) algorithms with a limit of 10 hours.

Table 4-7 LIGERO vs Stochastic Approach of K-PCL (7 Tbps)

	LIGERO			K=1		K=2		K=3		K=4	
#	**|P|**	**linear relax.**	**integer upper bound**	**linear relax.**	**integer upper bound**	**linear relax.**	**integer upper bound**	**linear relax.**	**integer upper bound**	**linear relax.**	**integer upper bound**
avg.	9338	0	785	1913.64	2110	1186.35	1790	1181.26	1850	1337.54	2025
min	9300	0	755	1893.75	2100	1180.44	1770	1151.40	1760	1254.56	1920
max	9409	0	820	1949.48	2115	1195.61	1820	1233.96	1935	1416.07	2180

As can be observed, the average of the relaxed objective function is 0 for the FW-B LIGERO algorithm, which means that for all the repetitions the optimal relaxed solution is reached. Contrarily for the relaxed K-PCL, the minimum value obtained is 1151.4, and thus none of the repetitions reached the optimal relaxed solution by means of this method. Furthermore, the integer solutions achieved with the FW-B version of LIGERO are, in all the cases, better than the obtained by the K-PCL. Actually, in the worst case, the FW-B LIGERO objective value decreases in a 53% with respect to the obtained solution with the K-PCL algorithm.

4.5 Summary

In this chapter a real network was chosen in order to study and compare the two versions of the LIGERO method and three initial set of lightpath generators. First the network performance and capacity was studied solving the K-PCL method (with *K=5*). Then an analysis with different loads for both versions and the three initial procedures was performed. This analysis leads to the conclusion that the FW-B version converges in a lower number of iterations, and acquires better execution times than the D-B version. Namely, D-B needs 80% more iterations and 70% more time to converge to the linear optimal than FW-B. The *MinAll* initial procedure was chosen as the most consistent, since the execution times obtained applying this initial method before the LIGERO algorithm increase smoothly with the load than using other initial procedures. Moreover for medium loads (4 Tbps) *GreedyRSA* needs 3 times the number of iterations and 3.6 times the execution time that *MinAll* procedure does. Hereafter a large-scale evaluation was made comparing the LIGERO and the *K-PCL* (with *K=10*) algorithms for larger number of demands (high loads) and larger number of channels. The number of variables with LIGERO was dramatically lower and the upper bound of the integer objective function that the LIGERO algorithm reached in 10 hours of execution time was almost an order of magnitude (9.5x) lower than the reached by the K-PCL procedure. Finally an examination of the LIGERO algorithm was made in order to compare the quality of the generated variables. With this goal, a stochastic approach of the K-PCL (with *K=5*) algorithm was performed so that it generates the same number of lightpaths than the LIGERO procedure. The results show that CGA outperforms the stochastic approach by returning a set of lightpaths that provides results a 40% better on average than the results obtained using the set of lightpaths returned by the stochastic approach. Table 4-8 summarizes the main results.

Table 4-8 Summary of obtained results

Compared methods	**Comparison**	**Compared methods**	**Comparison**
D-B vs FW-B: iterations (2-4 Tbps)	80%	D-B vs FW-B: exec. time (2-4 Tbps)	70%
GreedyRSA vs MinAll: iterations (4 Tbps)	3x	*GreedyRSA vs MinAll:* exe. time (4 Tbps)	3.6x
10-PCL vs LIGERO: integer upper bound (7 Tbps)	9.5x	Random (avg) vs LIGERO: integer upper bound (7 Tbps)	2.6x

Chapter 5.

Illustrative Use Case

In this section, a specific real scenario of application of our LIGERO algorithm is presented, highlighting the integration of the method in the architecture of an operational real optical network.

5.1 Periodical Network Re-optimization

While in static traffic environments the optimal RSA solution can be computed beforehand during the planning phase, the optimal use of spectrum resources is a challenging problem when dynamic traffic scenarios are considered. Subsequently to the network design, some strategies of resources re-optimization should be applied to periodically adapt the network to traffic fluctuations. *Lightpath Rerouting* consists of rerouting an existing lightpath from its original route to a different one, changing neither the source nor the destination (e.g. see [Ch07]). The rerouting procedure, designed for improving the performance of the network by, for example, reducing the amount of un-served traffic, follows a scheme like this:

1) The network *operator* triggers the network re-optimization, by launching an ad-hoc implemented procedure in the *planning tool*. Without entering into details, the planning tool is a network element that contains the hardware and software needed for solving all the optimization problems related with the planning, configuration, and re-optimization of the network. A component of the software implemented is the LIGERO algorithm explained in this project.
2) The planning tool gets the information of the current state of the network from the *Network Management System* (NMS). The NMS, in charge of managing the core network and implementing fault, configuration, administration, performance and security (FCAPS) functions, returns the information of the current set of routed lightpaths used to serve the set of

demands, as well as the needs (minimum and maximum bitrates) of such demands.

3) The planning tool solves the re-optimization problem with the desired configuration (e.g. total execution time). Finally, returns the solution to the operator.

4) The operator decides if the obtained solution substantially improves the current RSA. If it does, then the solution is provided to the NMS and its distribution to the *engineering department* is done. The engineering department is in charge of doing the manual operations to perform the changes provided by the planning tool. When the manual operations start, the traffic is partially/totally interrupted.

5) Once all changes have been performed and tested, the traffic is restored again.

Figure 5-1 shows the considered network architecture with all the elements described above.

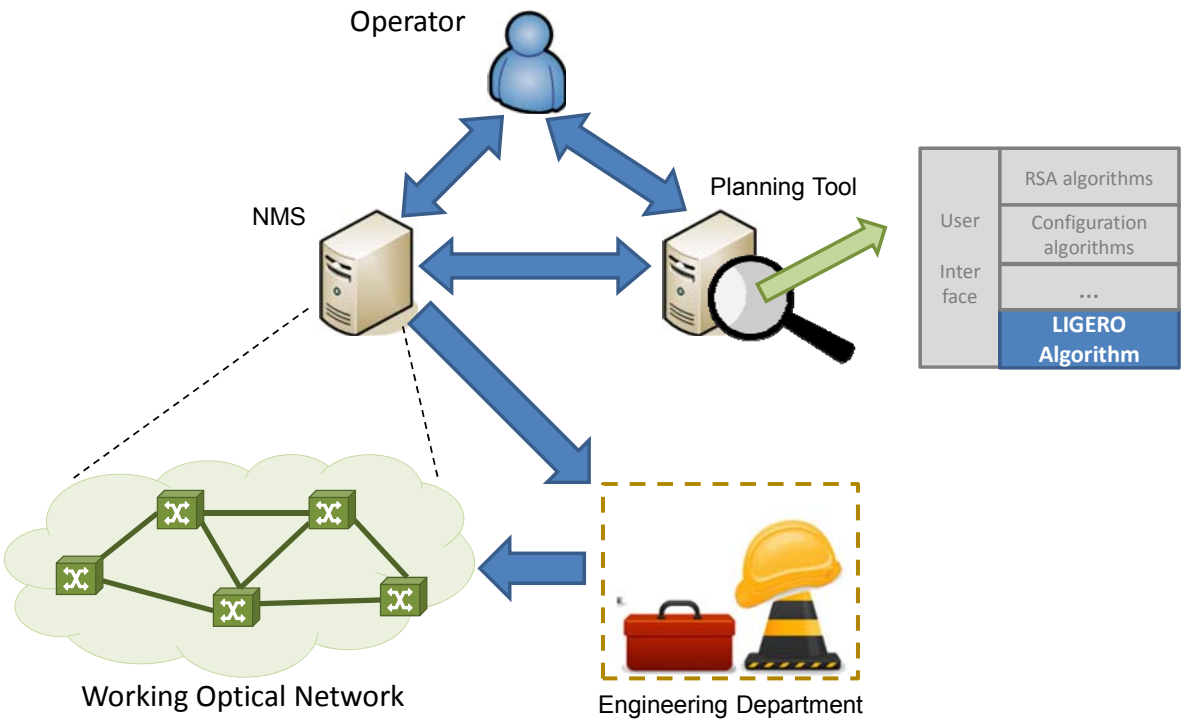


Figure 5-1 Considered Network Architecture

As previously anticipated, rerouting entails the temporary interruption of current optical connections and this represents a high cost for the provider. However, it is well known that during night hours the traffic carried is much lower than during the day. To illustrate this, Figure 5-2 shows the network traffic evolution along the

day (based on [UA09]), which can be also assumed equal for the effective bitrate fluctuations on a single lightpath. Thus, during night time, the percentage of lightpath bitrate that is used to transport data decreases remarkably. Therefore, this fact opens the possibility to perform the traffic re-optimization during such night period to avoid high revenues penalization.

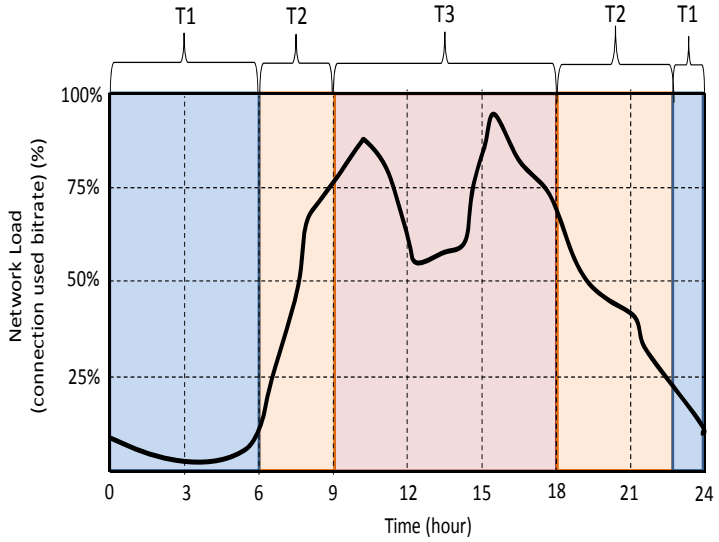


Figure 5-2 Daily evolution of traffic load

In a dynamic scenario like the one introduced here, new lightpaths could appear to serve new clients or extend the service to already served ones. From time to time, the operator receives new connections requests which will be established if the network contains enough free resources. To compute the best RSA for a single lightpath over a network with existing traffic, a kind of online RSA algorithm can be solved to obtain the solution (e.g. see [Ca12]), which will be also implemented in the planning tool. However, the operations for establishing and testing a new optical connection require from a manual action and, without loss of generality, we can assume that new lightpaths are created only during central working hours. At this point, we can consider that a single day could be split in three time frames (as illustrated in Figure 5-2):

- Low-Activity Period (T1): characterized by low effective loads carried in lightpaths and the absence of set-up of new optical connections.
- Medium-Activity Period (T2): consists in that period when, although new connection setup is not allowed, the volumes of traffic dissuade from perform traffic interruptions (in order to avoid high penalization costs).
- High-Activity Period (T3): characterized by high volumes of traffic in lightpaths and the possibility of establishing new connections or releasing active ones (i.e. changes in the set of demands).

Therefore, the best moment to start with the re-optimization is at the beginning of T2, since we can assume that the pool of established connections will remain static for a long time period (T2+T3). Moreover, the solution of the re-optimization problem must be obtained and sent to the engineering department during (the latest) T1, to guarantee that manual actions will be done in the most appropriate time period (i.e. affecting as less as possible to the current service). Therefore, from the values of Figure 5-2, the best moment to start the re-optimization is at hour 18, while the application of the changes must be done before hour 6. This represents a time frame of 12 hours for computing and implementing the re-optimization. Considering that manual actions require some time (e.g. 2 hours), the algorithm computation time should not exceed 10 hours.

Finally, note that in a dynamic scenario the continuous changes in the set of routed demands make that, if the optimal solution of a re-optimization problem is found, this becomes suboptimal after few incomes/outcomes of connections. For this reason, our LIGERO algorithm becomes a good method for solving this problem since the need of obtaining optimal solutions is not a necessary condition. This is in line with works like [Ca12] where heuristics for even simpler re-optimization problems are presented.

Chapter 6.

Concluding Remarks

6.1 Contributions and work impact

In this work we have developed a lightpath generation algorithm (named LIGERO) to solve a RSA problem for flexgrid optical networks. The designed procedure is divided in two phases: first an algorithm in used to find a reasonable set of lightpaths to be used as starting point of the second phase, consisting in a specifically designed column generation algorithm. The method aims at improving current approaches consisting of using large sets of pre-computed lightpaths that lead to excessively many optimization variables in the MIP formulations when applied to realistic network instances.

To consider different alternatives, we have developed two different versions of the LIGERO algorithm, based on different known routing algorithms which allow an efficient search of variables suitable to improve the solution of the RSA problem. Moreover, different criteria to produce initial set of variables have been designed and implemented.

Our numerical results for small, medium, and large network instances show that the proposed LIGERO algorithm remarkably reduces the number of variables in the MIP formulation of RSA (up to 2 orders of magnitude) and maintains (or even improves) the solution quality as compared with other approaches. It is worth mentioning that although our approach is heuristic, it performs better than other reasonable ways of approaching the considered problem.

Part of the work here presented has been included in the journal paper titled "Column Generation Algorithm for RSA Problems in Flexgrid Optical Networks" which has been recently accepted for publication in *Photonic Network Communications* journal (indexed in the *Journal Citation Reports*).

Finally, this work has been partially supported by the FP7 project IDEALIST (grant agreement no. 317999) and by the Spanish Ministry of Science through the TEC2011-27310 ELASTIC Project.

6.2 Personal Evaluation

For the achievement of this project, I was introduced to the optical networks field by means of reading specific papers and the valuable help of my advisors. I also studied the column generation method to deeply understand the idea and how it can be applied. Particularly I brought into focus the path generation algorithm and how it can be put into practise for optical networks. Finally I learnt how to implement a mathematical programming problem in Matlab making use of the linear solver engine in CPLEX 12.2.

The knowledge that I had to use for the contents of the project, are absolutely related with the courses I attended in the MIEIO master degree. Along the project I had to model a mixed integer mathematical programming problem. I also had to build its dual problem by means of its Lagrangean relaxation. Since the column generation algorithm is a decomposition method, I used my knowledge in large-scale optimisation where many techniques divide the original problem in few simpler problems. It is remarkably necessary to be familiar with networks since the RSA problem is a multi-commodity flow problem. It was especially important to be able to interpret the relaxed solutions and how the B&B method works. Finally, since the LIGERO algorithm does not provide optimal solutions, being able to discern the similitudes and the differences with other heuristic methods was notably important.

Personally, working with a group of professional and competitive researchers was a great experience for my career. Moreover, I was part of a research project from the development of the idea. I worked on it during its process, its implementations as well as the study of the results and the possible improvements. Finally I contributed on the writing of the paper which helped me to realize how a research report must be structured and explained.

I also attended to an international meeting of the IDEALIST European project. There, besides living a great personal experience, I learnt what international research groups and companies meet for. For instance I could see how the different groups put their ideas and objectives in common and take the relevant decisions together.

Last but not least, I would like to highlight the constant and significant dedication that this project requires and the enormous effort that it meant as my first step in research.

6.3 Future Work

To continue the work already done, some extensions have been considered:

- Enhancement of the proposed procedure (LIGERO + Branch & Bound) by strengthening linear formulations with problem-specific valid inequalities.
- Development of a Branch & Price Algorithm for solving the RSA problem to optimality. When the optimal solution is really necessary (offline RSA planning), this method could improve the common used Branch & Bound algorithm with a large set of pre-computed lightpaths.
- Development of specific and more complex use cases that are attracting research interest in the context of flexgrid optical networks. Two of them are:
    - *Multi-Hour RSA*: In this problem, demands vary the required bandwidth on time and, although changes of spectrum allocations for lightpaths are allowed between consecutive time periods, the route must remain invariant [Kl13].
    - *After Failure Repair Network Re-optimization*: in this problem, some new resources (i.e. fiber links) appear after a previous failure has been repaired. Then, the traffic can be re-optimized to use new capacity resources but all the re-routings are forced to use part of the new capacity.

Apendix A.

Implemented Code

This appendix shows a part of the code implemented in this work, specifically the main Matlab function for executing the initial procedures and the chosen version of the LIGERO algorithm. Then both LIGERO versions codes and the three initial procedures are included.

Main Algorithm

```
function [nvar,nvarIni,nconst, fvalrelCG, iteCG, CGTime_alg,CGTime_cplex,
fvalILP,unsDemandsILP,unsBandILP, fvalHEU, unsDemandsHEU,unsBandHEU,ILPTime,
HEUTime, mipgap] = CG_MAIN_FW(network, OD, parameters, outSol, inisol,FW)

%Solution = RSA_MAIN(network, sw, ss, OD)
%  network <- Matrix with [idLinks, Node, Node, Distance]
%  OD <- [origin destination bandwidth_min bandwidth_max]
%  paramters <- Struct with all the parameters
%  outSol <- binary. 1=print the solution;   0= no print
%  inisol <- 1=smallest channel; other=channels for maximum demand

% Initial parameters

demands=size(OD,1);
T=[0 0];

% Initial set of variables

if inisol==1        %Demanda mÃ-nima, 1 channel heurÃ-stica
    [Data]=RSA_inisol_minmax1(network, OD, parameters, outSol, 0);
elseif inisol==2    %Demanda maxima, 1 channel heurÃ-stica
    [Data]=RSA_inisol_minmax1(network, OD, parameters, outSol,1);
elseif inisol==3    %Demanda mÃ-nima, 1 channel Izquierda
    [Data]=RSA_inisol_minmaxleft(network, OD, parameters, outSol, 0);
elseif inisol==4    %Demanda mÃ¡xima, 1 channel Izquierda
    [Data]=RSA_inisol_minmaxleft(network, OD, parameters, outSol, 1);
elseif inisol==5    %Demanda mÃ-nima, Todos channels Izquierda
    [Data]=RSA_inisol_minmaxall(network, OD, parameters, outSol, 0);
elseif inisol==6    %Demanda mÃ¡xima, Todos channels Derecha
```

```
    [Data]=RSA_inisol_minmaxall(network, OD, parameters, outSol, 1);
elseif inisol==7    %Demanda mÃ-nima, 1 channels, HEURÃ?STICA_PRO
    [Data]=RSA_inisol_minHeuristica(network, OD, parameters, outSol, 0);
elseif inisol==8    %Demanda mÃ¡xima, 1 channels, HEURÃ?STICA_PRO
    [Data]=RSA_inisol_minHeuristica(network, OD, parameters, outSol, 1);
end

nvarIni=Data.('nvar');

CGclock=clock;
CGTime_alg=0;
CGTime_cplex=0;

[cplex]=RSA_ROW(Data, parameters);
T(1,2)=etime(clock,CGclock);


%Initialization of parameters
file=fopen('Solution.txt','w');
STOP=0;
ite=0;
status=1;
checkMax=0;
checkMin=0;
checkSum=0;
tries=0;
maxtries=10;

cplexTimeRem=0;
algTimeRem=0;
lastIteTime=0;
newvar=0;
% Add columns while necessary
while STOP~=1
    ite=ite+1;
    % Solving restricted linear relaxation
    if status==1
        clockIni=clock;
        [status,fvalPrel,xPrel,duals,fvalD,
redCosts,intsol,usedlpaths]=RSA_SOLVER_RELAX(Data,cplex,outSol);

        cplex_etime=etime(clock,clockIni);
        CGTime_cplex=CGTime_cplex+cplex_etime;
        T(ite+1, 1)=etime(clock,CGclock);
        maxIte=max(usedlpaths);
        minIte=min(usedlpaths);
        sumIte=sum(usedlpaths);
    end

    if (checkMin==minIte && checkMax==maxIte && checkSum==sumIte)

        tries=tries+1;
        cplexTimeRem=cplexTimeRem+cplex_etime;
        algTimeRem=algTimeRem+lastIteTime;
        nvarRem=nvarRem+newvar;
        if tries==maxtries
            STOP=1;
            continue
        end

    else
        checkMin=minIte;
```

```
        checkMax=maxIte;
        checkSum=sumIte;
        tries=0;
        cplexTimeRem=0;
        algTimeRem=0;
        nvarRem=0;
    end

        if  fvalPrel>0
            fprintf(file,'Iteration %i:\n',ite);

            % Computing new weights for network links and adding a new column
            if outSol==1
                fprintf('\n COLUMN GENERATION (iteration: %i) \n\n', ite)
            end

            clockIni=clock;
            if FW==1
                [STOP, Data]=RSA_CG_vFW(duals, network, parameters, OD, Data);
            else
                [STOP, Data]=RSA_CG_v2(duals, network, parameters, OD, Data);
            end
            CGTime_alg=CGTime_alg+etime(clock,clockIni);
            lastIteTime=etime(clock,clockIni);

            nvar=GETFIELD(Data,'nvar');
            newvar=GETFIELD(Data,'newvar');
            fprintf('Iteration: %i Var: %i (+%i)\n',ite,nvar-newvar,newvar);

            if newvar==0 %If we did not add any column
                STOP=1;
            else %If we added columns
                clockIni=clock;
                [cplex]=RSA_COLUMN(Data,cplex);
                CGTime_cplex=CGTime_cplex+etime(clock,clockIni);
                T(ite+1,2)=etime(clock,CGclock);
            end
        else
            STOP=1;
        end
%    end
end

fvalrelCG=fvalPrel;

CGTime=etime(clock,CGclock);
fclose(file);


% Storing solution
nvar=length(xPrel);
nvar=nvar-nvarRem;
fileaux=fopen('nvarLimit.txt','w');
fprintf(fileaux,'%i\n',nvar);
fclose(fileaux);
nconst=length(duals);

CGTime_cplex=CGTime_cplex-cplexTimeRem;
CGTime_alg=CGTime_alg-algTimeRem;

iteCG=ite-tries;
```

```
intsol=0;
ILPclock=clock;

% Solving ILP
fvalPilp=-1;
if (status==1 && intsol==0)

    fprintf('\n\n SOLVING ILP \n\n')
    cplex.Model.ctype = char(ones(1,length(xPrel))*'I');
    [status,fvalPilp,xPilp, egap]=RSA_SOLVER_ILP(Data,cplex,outSol);
else if status==1 && intsol==1
        xPilp=xPrel;
        egap=0;
    end
end

fvalILP=fvalPilp;
ILPTime=etime(clock,ILPclock);

% Storing solution
nconst=length(duals);
unsDemandsILP=sum(xPilp(1:demands));
unsBandILP=sum(xPilp(demands+1:2*demands))/sum(OD(:,4));
mipgap=egap;

HEUclock=clock;

[ SolHeu , MSS] = HeurPostCG_Marc_v1( Data, parameters, xPrel, OD );
HEUTime=etime(clock,HEUclock);
% % Storing solution
A=GETFIELD(parameters,'A');

fvalHEU=A*(SolHeu(:,1)'*OD(:,3))+sum(SolHeu(:,2));
unsDemandsHEU=sum(SolHeu(:,1));
unsBandHEU=sum(SolHeu(:,2))/sum(OD(:,4));

clear cplex
```

LIGERO Dijkstra-Based Version

```
function [stopCG, Data]=RSA_CG_v2(duals,network,parameters,OD, Data)

% Add OS parameters to the struct parameters
sw=GETFIELD(parameters,'sw');
ss=GETFIELD(parameters,'ss');
mf=GETFIELD(parameters,'mf');
prec=GETFIELD(parameters,'prec');
alfa=GETFIELD(parameters,'alfa');
LDold=GETFIELD(Data,'LD');
LEold=GETFIELD(Data,'LE');
LSold=GETFIELD(Data,'LS');

% Take parameters and dual variables
demands=size(OD,1);
slots=floor(ss/sw);
nodes=max(max(network(:,2:3)));
links=size(network,1);
cardduals=length(duals);
```

```
duals_mu=-duals(1:demands);
duals_lambda=duals(demands+1:2*demands);
duals_pi=-duals(2*demands+1:2*demands+links*slots);
duals_gamma=duals(cardduals-demands+1:cardduals);

for i=1:length(duals_mu)
    if abs(duals_mu(i))<=prec
        duals_mu(i)=0;
    end
end

for i=1:length(duals_lambda)
    if abs(duals_lambda(i))<=prec
        duals_lambda(i)=0;
    end
end

for i=1:length(duals_pi)
    if abs(duals_pi(i))<=prec
        duals_pi(i)=0;
    end
end

minpi=min(duals_pi);

for i=1:length(duals_gamma)
    if abs(duals_gamma(i))<=prec
        duals_gamma(i)=0;
    end
end

% Initialize the matrix to add new variables (columns)
LD=zeros(1,demands);
LE=zeros(1,links);
LS=zeros(1,slots);
betaL=0;
newvar=0;
Newvars=zeros(1,demands);
stopCG=1;

% For every demand
for d=1:demands
    zetaInc=-inf;
    % Compute the number of slots of size maximum and minimum
    csMax=ceil(OD(d,4)/(mf*sw)); % Formula to compute number of slots
    csMin=ceil(OD(d,3)/(mf*sw));
    costDemand=duals_lambda(d)-duals_mu(d); % Constant part (for the same
demand)of dual costs
    % For eacch possible channel size (from max to min)
    for cs=csMax:-1:csMin
        % Compute dual variables for this size of channel
        bc=min(OD(d,4),cs*mf*sw);  %bandwidth of the channel
        costChannel=bc*duals_gamma(d);
        zetaMax=costDemand+costChannel-minpi;
        %%%%%%%%
        if zetaMax<=0 || zetaInc>=zetaMax
            break;
        end
        %%%%%%%%
        % For each channel of this size
        for cp=1:slots-cs+1
            channel=zeros(1,slots);
```

```
            channel(cp:cp+cs-1)=1;
            for e=1:links
                % Set the link metrics using pi dual variables
                network(e,4)=channel*duals_pi((e-1)*slots+1:e*slots);
            end

            [MATRIX, ID,DIST]= networkMatricesInf(network);
            [RUTASAUX, costRoute] = dijkstra(DIST, OD(d,1),OD(d,2));
            RUTAS=zeros(1,3+nodes);
            RUTAS(4:3+length(RUTASAUX))=RUTASAUX;


            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %Compute  zeta
            zeta=costDemand+costChannel-costRoute;

            % If zeta >0 and better than the previous one
            if zeta>0 && zetaInc<zeta % Candidate new variable

                % Save data of the candidate variable
                zetaInc=zeta;
                csInc=cs;
                bcInc=bc;
                channelInc=channel;
                RUTASinc=RUTAS(1,:);
                IDinc=ID;
                if zetaInc>=alfa*zetaMax
                    break;
                end
            end
        end
    end

    if zetaInc>prec
       % New variable found for demand d, save data.
       stopCG=0;
       newvar=newvar+1;
       Newvars(d)=1;

       LD(newvar,:)=zeros(1,demands);
       LD(newvar,d)=1;

       LE(newvar,:)=zeros(1,links);

       for n=4:length(RUTASinc)
           if RUTASinc(1,n+1)==0
               break;
           end
           linkID=IDinc(RUTASinc(n),RUTASinc(n+1));
           LE(newvar,linkID)=1;
       end

       LS(newvar,:)=zeros(1,slots);
       LS(newvar,:)=channelInc;

       betaL(newvar)=bcInc;
    end

end

% Merge old variables we the new ones
LE=[LEold; LE];
```

```
LD=[LDold; LD];
LS=[LSold; LS];
lightpaths=size(LE,1);

% Add new variables to Data
Data=SETFIELD(Data,'LD',LD);
Data=SETFIELD(Data,'LE',LE);
Data=SETFIELD(Data,'LS',LS);
Data=SETFIELD(Data,'lightpaths',lightpaths);
Data=SETFIELD(Data,'nvar',lightpaths);
Data=SETFIELD(Data,'betaL',betaL);
Data=SETFIELD(Data,'newvar',newvar);
Data=SETFIELD(Data,'Newvars',Newvars);
end
```

LIGERO Floyd-Warshall-Based Version

```
function [stopCG, Data]=RSA_CG_vFW(duals,network,parameters,OD, Data)

lexicographical=1;

if lexicographical~=1
    multLex=-inf;
else
    multLex=1;
end

% Add OS parameters to the struct parameters
sw=GETFIELD(parameters,'sw');
ss=GETFIELD(parameters,'ss');
mf=GETFIELD(parameters,'mf');
prec=GETFIELD(parameters,'prec');
LDold=GETFIELD(Data,'LD');
LEold=GETFIELD(Data,'LE');
LSold=GETFIELD(Data,'LS');

% Take parameters and dual variables
demands=size(OD,1);
slots=floor(ss/sw);
nodes=max(max(network(:,2:3)));
links=size(network,1);
cardduals=length(duals);
duals_mu=-duals(1:demands);
duals_lambda=duals(demands+1:2*demands);
duals_pi=-duals(2*demands+1:2*demands+links*slots);
duals_gamma=duals(cardduals-demands+1:cardduals);

for i=1:length(duals_mu)
    if abs(duals_mu(i))<=prec
        duals_mu(i)=0;
    end

end

for i=1:length(duals_lambda)
    if abs(duals_lambda(i))<=prec
        duals_lambda(i)=0;
    end
```

```
end

for i=1:length(duals_pi)
    if abs(duals_pi(i))<=prec
        duals_pi(i)=0;
    end

end


for i=1:length(duals_gamma)
    if abs(duals_gamma(i))<=prec
        duals_gamma(i)=0;
    end

end

% Initialize the matrix to add new variables (columns)
LD=zeros(1,demands);
LE=zeros(1,links);
LS=zeros(1,slots);
betaL=0;
newvar=0;
Newvars=zeros(1,demands);
stopCG=1;

csMax=ceil(max(OD(:,4))/(mf*sw)); % Slots formula
csMin=ceil(min(OD(:,3))/(mf*sw));


%a)
newlpaths=struct;
for d=1:demands
    newlpaths.(['D' int2str(d)]).('width')=inf;
    newlpaths.(['D' int2str(d)]).('hoplength')=inf;
    newlpaths.(['D' int2str(d)]).('redcost')=-inf;
    newlpaths.(['D' int2str(d)]).('channels').('number')=0;
end

for cs=csMax:-1:csMin
    for cp=1:slots-cs+1
            channel=zeros(1,slots);
            channel(cp:cp+cs-1)=1;
            I=ones(nodes).*inf;
            for e=1:links
                network(e,4)=channel*duals_pi((e-1)*slots+1:e*slots);
                I(network(e,2),network(e,3))=network(e,4);
                I(network(e,3),network(e,2))=network(e,4);

            end
%c)
            [I,Ihop] = FastFloyd(I);


            for d=1:demands
                csMind=ceil(OD(d,3)/(mf*sw));
                csMaxd=ceil(OD(d,4)/(mf*sw));
                bc=min(cs*mf*sw,OD(d,4));
                if csMind>cs || cs>csMaxd
                    continue;
```

```
                end

                costRoute=I(OD(d,1),OD(d,2));
                costHops=Ihop(OD(d,1),OD(d,2));
                costDemand=duals_lambda(d)-duals_mu(d);
                costChannel=bc*duals_gamma(d);

                zeta=costDemand+costChannel-costRoute;
                redcost=newlpaths.(['D' int2str(d)]).('redcost');
                width=newlpaths.(['D' int2str(d)]).('width');
                hoplength=newlpaths.(['D' int2str(d)]).('hoplength');

                if ((zeta>0 && zeta>redcost) || (zeta>0 && zeta==redcost &&
cs<width) || (zeta>0 && zeta==redcost &&  cs==width &&
costHops<hoplength*multLex))

                    newlpaths.(['D' int2str(d)]).('width')=cs;
                    newlpaths.(['D' int2str(d)]).('hoplength')=costHops;
                    newlpaths.(['D' int2str(d)]).('bandch')=bc;
                    newlpaths.(['D' int2str(d)]).('redcost')=zeta;

                    newlpaths.(['D' int2str(d)]).('channels')=[];
                    newlpaths.(['D' int2str(d)]).('channels').('number')=0;

                end

                if (zeta==newlpaths.(['D' int2str(d)]).('redcost') &&
cs==newlpaths.(['D' int2str(d)]).('width'))
                    nch=newlpaths.(['D'
int2str(d)]).('channels').('number')+1;
                    newlpaths.(['D' int2str(d)]).('channels').('number')=nch;
                    newlpaths.(['D' int2str(d)]).('channels').(['C'
int2str(nch)])=channel;
                end
            end
    end
end


for d=1:demands
    if newlpaths.(['D' int2str(d)]).('channels').('number')>0
       nch=newlpaths.(['D' int2str(d)]).('channels').('number');
       choice=ceil(rand*nch);
       channel=newlpaths.(['D' int2str(d)]).('channels').(['C'
int2str(choice)]);

       for e=1:links
           network(e,4)=channel*duals_pi((e-1)*slots+1:e*slots);
       end
       [MATRIX, ID,DIST]= networkMatricesInf(network);
       [RUTASAUX, costRoute] = dijkstra(DIST, OD(d,1),OD(d,2));
       RUTAS=zeros(1,3+nodes);
       RUTAS(4:3+length(RUTASAUX))=RUTASAUX;

       newvar=newvar+1;
       Newvars(d)=1;
       LD(newvar,:)=zeros(1,demands);
       LD(newvar,d)=1;
       LE(newvar,:)=zeros(1,links);
       for n=4:length(RUTAS)
           if RUTAS(1,n+1)==0
               break;
```

```
            end
            linkID=ID(RUTAS(n),RUTAS(n+1));
            LE(newvar,linkID)=1;
        end

        LS(newvar,:)=zeros(1,slots);
        LS(newvar,:)=channel;

        betaL(newvar)=newlpaths.(['D' int2str(d)]).('bandch');

        stopCG=0;
    end

end

LE=[LEold; LE];
LD=[LDold; LD];
LS=[LSold; LS];
lightpaths=size(LE,1);

%Add new variables to struct Data
Data=SETFIELD(Data,'LD',LD);
Data=SETFIELD(Data,'LE',LE);
Data=SETFIELD(Data,'LS',LS);
Data=SETFIELD(Data,'lightpaths',lightpaths);
Data=SETFIELD(Data,'nvar',lightpaths);
Data=SETFIELD(Data,'betaL',betaL);
Data=SETFIELD(Data,'newvar',newvar);
Data=SETFIELD(Data,'Newvars',Newvars);
```

MaxLeft Initial Procedure

```
function [Data]=RSA_inisol_minmaxleft(networkLinks, OD, parameters, outSol,
minmax)

sw=GETFIELD(parameters,'sw');
ss=GETFIELD(parameters,'ss');
mf=GETFIELD(parameters,'mf');

varID=[0 0 0];
nvar=0;
slots=ceil(ss/sw);
demands=size(OD,1);
Data=struct();

[MATRIX,ID,DIST]= networkMatrices(networkLinks);
links=size(networkLinks,1);

lightpaths=0;
LE=zeros(1,links);    % matrix lightpath-links
LD=zeros(1,demands); % matrix lightpath-demands
LS=zeros(1,slots);    % matrix lightpath-slots
ES=zeros(links, slots); %Matrix slots-links

%For each demand, we add a path.
for d=1:demands
    [RUTAS]=SP(MATRIX,MATRIX,OD(d,1),OD(d,2)); %RUTAS only contains a route
    for r=1:size(RUTAS,1) %This loop just do an iteration.
```

```
        if RUTAS(r,1)>0
            if minmax==0
                ns=ceil(OD(d,3)/(mf*sw)); %Number of slots to cover the
minimum demand.
            else
                ns=ceil(OD(d,4)/(mf*sw)); %Number of slots to cover the
maximum demand.
            end
            [ES, LE, LS] = lightpath_left(RUTAS(4:length(RUTAS(1,:))), ns, ES,
LE, LS, ID);
            lightpaths=lightpaths+1;
            LD(lightpaths,d)=1;
        end
    end
end


%Create the set of variables x_l
for d=1:demands
    for l=1:lightpaths
        if LD(l,d)==1
            nvar=nvar+1;
            varID(nvar,:)=[nvar d l];
        end
    end
end

%Print Initial Solution (it might not be feasible)
if (outSol==1)
    fprintf('*** Initial Solution ***\n\n')
    for v=1:nvar
        d=varID(v,2);
        l=varID(v,3);
        fprintf('D %i -> LP %i (Slots: %i) \n\n',d,l,sum(LS(l,:)));
        fprintf('\t \t Links')
        disp(find(LE(l,:)))
        fprintf('\t \t Slots')
        disp(find(LS(l,:)))
    end
    fprintf('\n')
end

Data=setfield(Data,'links',links);
Data=setfield(Data,'slots',slots);
Data=setfield(Data,'demands',demands);
Data=setfield(Data,'lightpaths',lightpaths);
Data=setfield(Data,'minbw',OD(1:demands,3));
Data=setfield(Data,'maxbw',OD(1:demands,4));
Data=setfield(Data,'LE',LE);
Data=setfield(Data,'LD',LD);
Data=setfield(Data,'LS',LS);
Data=setfield(Data,'ES',ES);
Data=setfield(Data,'nvar',nvar);
Data=setfield(Data,'varID',varID);

channelSizes=zeros(demands,slots);
channelSizes(:,1)=1;
Data=setfield(Data,'channelSizes',channelSizes);
end
```

```
function [ES, LE, LS] = lightpath_left(ruta, ns, ES, LE, LS, ID)
% [ES, LE, LS] = lightpath_left(ruta, ns, ES, LE, LS, ID)
%
%INPUT
%   ruta <- ruta que debe ser validada
%   ns <- numero de slots que se tienen que añadir
%   LS <- matriz lightpath-slot que hay que modificar
%   LE <- Matriz lightpath-Link
%   lightpaths <- numero de lightpaths (sin contar el que se genera en esta
fciï¿½n)
%   Matrix node node with the identity of the link instead of 1's
%
%OUTPUT
%   ES <- Matriz link-slot
%   LE <- Matriz Lightpath-Link (con el nuevo lightpath)
%   LS <- matriz lightpath-slot modificada


slots=size(ES',1);
links=size(LE',1);
if norm(LE(1,:))==0
    lightpaths=0;
else
    lightpaths=size(LE,1);
end

%Convertimos la ruta en vector path con 1's en los links del path
path=zeros(1,links);
vls=zeros(1,slots);
for i=1:length(find(ruta))
    if ruta(i+1)~=0
        e=ID(ruta(i),ruta(i+1));
        path(e)=1;
        ES(e,1:ns)=ones(1,ns);
        vls(1:ns)=ones(1,ns);
    end
end

%Construimos matrices LE y LS
LE(lightpaths+1,:)=path;
LS(lightpaths+1,:)=vls;

end
```

MinAll Initial Procedure

```
function [Data]=RSA_inisol_minmaxall(networkLinks, OD, parameters, outSol,
minmax)

sw=GETFIELD(parameters,'sw');
ss=GETFIELD(parameters,'ss');
mf=GETFIELD(parameters,'mf');

slots=ceil(ss/sw);
demands=size(OD,1);
Data=struct();

[MATRIX,ID,DIST]= networkMatrices(networkLinks);
```

```
links=size(networkLinks,1);

LE=zeros(1,links);   % matriz lightpath-links
LD=zeros(1,demands); % matriz lightpath-demands
LS=zeros(1,slots);   % matriz lightpath-slots
ES=zeros(links, slots); %Matriz slots-links (Creo que son los channels en
vertical)

%Para each demand we add a path.
for d=1:demands
    [RUTAS]=SP(MATRIX,MATRIX,OD(d,1),OD(d,2)); %RUTAS solo contiene una ruta
    if RUTAS(1,1)>0
        ruta=RUTAS(1,:);
        ruta(1:3)=[];
        [ LD, LE, LS ] = allchannels_minmax(d, ruta, LD, LE, LS, OD, ID,
parameters, minmax);
    end
end

links=size(LE',1);
slots=size(LS',1);
lightpaths=size(LE,1);

Data=setfield(Data,'links',links);
Data=setfield(Data,'slots',slots);
Data=setfield(Data,'demands',demands);
Data=setfield(Data,'lightpaths',lightpaths);
Data=setfield(Data,'nvar',lightpaths);
Data=setfield(Data,'minbw',OD(1:demands,3));
Data=setfield(Data,'maxbw',OD(1:demands,4));
Data=setfield(Data,'LE',LE);
Data=setfield(Data,'LD',LD);
Data=setfield(Data,'LS',LS);
Data=setfield(Data,'ES',ES);

channelSizes=zeros(demands,slots);
channelSizes(:,1)=1;
Data=setfield(Data,'channelSizes',channelSizes);
end


function [ LD, LE, LS ] = allchannels_minmax(d, ruta,LD, LE, LS, OD, ID,
parameters, minmax)

sw=GETFIELD(parameters,'sw');
mf=GETFIELD(parameters,'mf');

 %numero de slots necesarios para cubrir la demanda maxima.
if minmax==0
    ns=ceil(OD(d,3)/(mf*sw));
else
    ns=ceil(OD(d,4)/(mf*sw));
end

slots=size(LS',1);
links=size(LE',1);

if norm(LE(1,:))==0
    lightpaths=0;
else
    lightpaths=size(LE,1);
end
```

```
%Convertimos la ruta (cadena de nodos) en vector path (cadena de links)
path=zeros(1,links);
for i=1:length(find(ruta))
    if ruta(i+1)~=0
        path(ID(ruta(i),ruta(i+1)))=1;
    end
end
%Creamos un lightpath para cada posible channel con maxns slots
for i=1:slots-ns+1
    lightpaths=lightpaths+1;
    LE(lightpaths,:)=path;
    LS(lightpaths,i:i+ns-1)=ones(1,ns);
    LD(lightpaths,d)=1;
end

end
```

GreedyRSA Initial Procedure

```
function [Data]=RSA_inisol_minHeuristica(networkLinks, OD, parameters, outSol,
minmax)

sw=GETFIELD(parameters,'sw');
ss=GETFIELD(parameters,'ss');
mf=GETFIELD(parameters,'mf');

varID=[0 0 0];
nvar=0;
slots=ceil(ss/sw);
demands=size(OD,1);
Data=struct();

[MATRIX,ID,DIST]= networkMatrices(networkLinks);

links=size(networkLinks,1);

lightpaths=0;
LE=zeros(1,links);    % matriz lightpath-links
LD=zeros(1,demands); % matriz lightpath-demands
LS=zeros(1,slots);    % matriz lightpath-slots
ES=zeros(links, slots); %Matriz slots-links (Creo que son los channels en
vertical)

%Para cada demanda aÃ±adimos un path. A la vez construimos las matrices
%path-link y path-demands.
for d=1:demands
    [RUTAS, nrutas]=KSP(MATRIX,MATRIX,OD(d,1),OD(d,2), 5);
    ruta1=RUTAS(1,:);
    nrutas=min(nrutas,5);
    for r=1:nrutas
        if RUTAS(r,1)>0
            if minmax==0
                ns=ceil(OD(d,3)/(mf*sw)); %numero de slots necesarios para
cubrir la demanda minima.
            else
                ns=ceil(OD(d,4)/(mf*sw));
            end
```

```
            [ES, LE, LS, exito] = valid_rute_v3(RUTAS(r,4:length(RUTAS(r,:))),
ns, ES, LE, LS, ID, ruta1(4:length(ruta1)), r, nrutas);
            if exito==1
                lightpaths=lightpaths+1;
                LD(lightpaths,d)=1;
                break
            end
        end
    end
end

%Creamos el conjunto de variables x_l
%i.e. Cada variable cual es su demanda(id) y que lightpath usa.
for d=1:demands
    for l=1:lightpaths
        if LD(l,d)==1
            nvar=nvar+1;
            varID(nvar,:)=[nvar d l];
        end
    end
end

if (outSol==1)
    fprintf('*************** SoluciÃ³n Inicial ***************\n\n')
    for v=1:nvar
        d=varID(v,2);
        l=varID(v,3);
        fprintf('D %i -> LP %i (Slots: %i) \n\n',d,l,sum(LS(l,:)));
        fprintf('\t \t Links')
        disp(find(LE(l,:)))
        fprintf('\t \t Slots')
        disp(find(LS(l,:)))
    end
    fprintf('\n')
end

Data=setfield(Data,'links',links);
Data=setfield(Data,'slots',slots);
Data=setfield(Data,'demands',demands);
Data=setfield(Data,'lightpaths',lightpaths);
Data=setfield(Data,'minbw',OD(1:demands,3));
Data=setfield(Data,'maxbw',OD(1:demands,4));
Data=setfield(Data,'LE',LE);
Data=setfield(Data,'LD',LD);
Data=setfield(Data,'LS',LS);
Data=setfield(Data,'ES',ES);
Data=setfield(Data,'nvar',nvar);
Data=setfield(Data,'varID',varID);

channelSizes=zeros(demands,slots);
channelSizes(:,1)=1;
Data=setfield(Data,'channelSizes',channelSizes);
end


function [ES, LE, LS, exito] = valid_rute_v3(ruta, ns, ES, LE, LS, ID, ruta1,
intentos, nrutas)
% [ valid, capac ] = valid_rute(ruta, nslots, capac)
%
%INPUT
%   ruta <- ruta que debe ser validada
%   ns <- numero de slots que se tienen que aï¿½adi
```

```
%   LS <- matriz lightpath-slot que hay que modificar
%   LE <- Matriz lightpath-Link
%   lightpaths <- numero de lightpaths (sin contar el que se genera en esta
fciï¿½n)
%   Matrix node node with the identity of the link instead of 1's
%
%OUTPUT
%   ES <- Matriz link-slot
%   LE <- Matriz Lightpath-Link (con el nuevo lightpath)
%   LS <- matriz lightpath-slot modificada

exito=0;
slots=size(ES',1);
links=size(LE',1);
if max(LE(1,:))<=0.001
    lightpaths=0;
else
    lightpaths=size(LE,1);
end

%Convertimos la ruta en vector path con 1's en los links del path
path=zeros(1,links);
for i=1:length(find(ruta))
    if ruta(i+1)~=0
        path(ID(ruta(i),ruta(i+1)))=1;
    end
end

%Vemos que slots estï¿½n disponibles a lo largo de toda la ruta
nodisp=zeros(1,slots); %slot no disponibles en algun link de la ruta = 1
for e=1:links
    if path(e)==1
        nodisp=nodisp+ES(e,:);
    end
end

%Recorremos nodisp para ver si hay algun espacio que nos valga.
%Lo pondremos lo mï¿½s a la izquierda posible.
%Si no cabe en ningun sitio lo pondremos lo mï¿½s a la derecha posible.
vls=zeros(1,slots);
for i=1:(slots-ns+1)
    if max(nodisp(i:(i+ns-1)))<=0.001 %Si es cero
        exito=1;
        for e=1:links
            if path(e)==1
                ES(e,i:(i+ns-1))=ones(1,ns);
            end
        end
        vls(i:(i+ns-1))=ones(1,ns);
        break;
    elseif intentos==nrutas
        exito=1;
        path1=zeros(1,links);
        for j=1:length(find(ruta1))
            if ruta1(j+1)~=0
                path(ID(ruta1(j),ruta1(j+1)))=1;
            end
        end
        for e=1:links
            if path1(e)==1
                ES(e,slots-ns+1:slots)=ones(1,ns);
            end
```

```
        end
        vls(slots-ns+1:slots)=ones(1,ns);
        break;
    end
end

%Construimos matriz PE
if exito==1
    LE(lightpaths+1,:)=path;
    LS(lightpaths+1,:)=vls;
end

end
```

References

[Ah93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Network Flows: Theory, Algorithms and Applications, Prentice Hall, 1993.

[Ba98] C. Barnhart, E. Johnson, G. Nemhauser, G. Savelsbergh, and P. Vance, “Branch-and-Price: column generation for solving huge integer programs,” Operat. Res., vol. 46, no. 3, pp. 316–329, 1998.

[Ca12] A. Castro, L. Velasco, M. Ruiz, M. Klinkowski, J. P. Fernández-Palacios, and D. Careglio, "Dynamic Routing and Spectrum (Re)Allocation in Future Flexgrid Optical Networks," Elsevier Computers Networks, 56, 2869-2883, 2012.

[Ch07] X. Chu, T. Bu, and X. Li, “A study of lightpath rerouting schemes in wavelength-routed WDM networks,” in Proc. IEEE Int. Conf. Communications, Glasgow, UK, 2007.

[Ch11] K. Christodoulopoulos, I. Tomkos, and E. Varvarigos, “Elastic bandwidth allocation in flexible OFDM based optical networks,” IEEE J. Lightw. Technol., vol. 29, no. 9, pp. 1354–1366, 2011.

[CPLX] IBM ILOG CPLEX, http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/

[Di59] Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271, 1959.

[Fl62] Floyd, R.W.: Algorithm 97: Shortest path. Commun. ACM 5(6), 345. DOI 10.1145/367766.368168, 1962.

[G694] Spectral grids for WDM applications: DWDM frequency grid. ITU-T G.694.1 (ed. 2.0) (2012)

[Ja09] B. Jaumard, C. Meyerb, and B. Thiongane, “On column generation formulations for the RWA problem,” Discrete. Appl. Math., vol. 157, pp. 1291–1308, 2009.

[Kl11] M. Klinkowski and K. Walkowiak, “Routing and spectrum assignment in spectrum sliced elastic optical path network,” IEEE

Commun. Lett.,vol. 15, no. 8, pp. 884–886, 2011.

[Kl13] M. Klinkowski, M. Ruiz, L. Velasco, D. Careglio, V. Lopez, and J. Comellas, "Elastic Spectrum Allocation for Time-Varying Traffic in FlexGrid Optical Networks," IEEE Journal on Selected Areas in Communications (JSAC) 31, 26-38, 2013.

[Li11] Y. Li, F. Zhang, and R. Casellas, "Flexible grid label format in wavelength switched optical network," IETF RFC Draft, Jul. 2011.

[MLB] Matlab®, Mathworks® http://www.mathworks.es/products/matlab/

[Pi04] M. Pióro and D. Medhi, Routing, Flow, and Capacity Design in Communication and Computer Networks. Morgan Kaufmann, 2004.

[UA09] Memoria Universidad de Alicante 2008-09

http://web.ua.es/es/memoria08-09/vr-tecnologia/informatica.html

[Ve12] L. Velasco, M. Klinkowski, M. Ruiz, and J. Comellas, "Modeling the routing and spectrum allocation problem for flexgrid optical networks," Phot. Netw. Commun., vol. 24, no. 3, pp. 177–186, 2012.

[Wa12] Y. Wang, X. Cao, Q. Hu, and Y. Pan, "Towards elastic and fine-granular bandwidth allocation in spectrum-sliced optical networks," IEEE/OSA J.of Opt. Commun. and Netw., vol. 4, no. 11, pp. 906–917, 2012.

[Ye70] Yen, Jin Y, "An algorithm for finding shortest routes from all source nodes to a given destination in general networks", Quarterly of Applied Mathematics, 27, 526–530, 1970.