BACHELOR'S DEGREE THESIS

Degree in Mathematics

Title: Incremental capacity planning in flexgrid optical networks

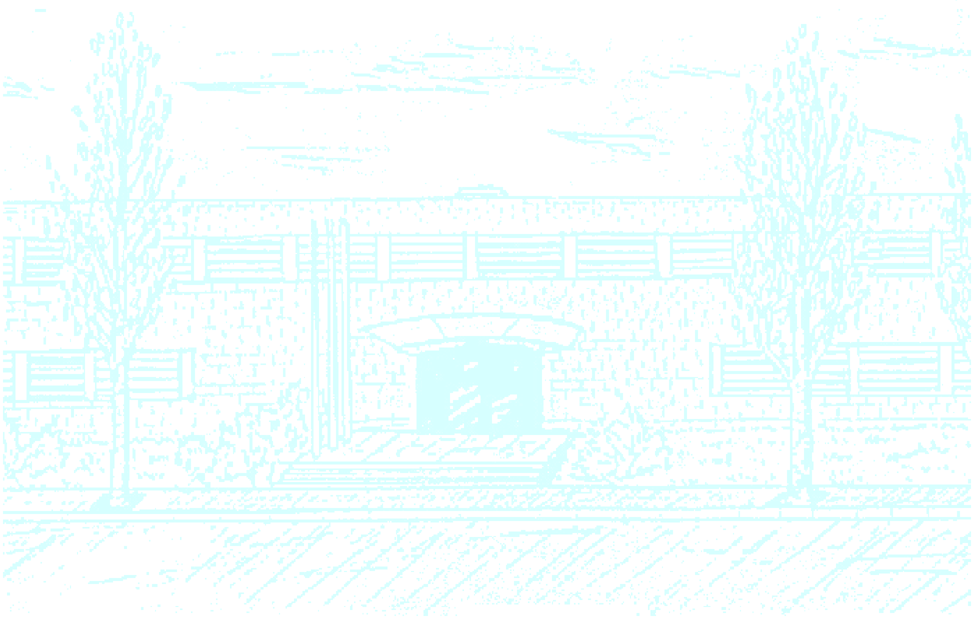
Author: Fernando Morales Alcaide

Advisor: Luis Velasco Esteban

Co-Advisor: Marc Ruiz Ramírez

Department: Computers Architecture

Academic year: 2014-2015



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

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.

Finally, I wish to thank my family and friends 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 increase of global Internet users, the fast proliferation of smart city technologies and the advent of Internet of Things are imposing unprecedented challenges to telecommunication network operators. Specifically, by 2018, there will be nearly four billion global Internet users (2.5 billion in 2013), which represents about 52% of the world's population [CISCO]. Flexgrid technology [Ji09] has been recently presented as the most promising option for upgrading the currently operating fixed grid optical networks and extending their capacity to be able to deal with massive traffic volumes forecast for the next decade.

Due to the constant increase and evolution of Internet traffic, underlying optical transport networks must be periodically re-designed (in response to predicted or monitored traffic changes) in order to keep committed service requirements for the next period [Ru14.1]. Incremental network capacity planning requires the placement of new network resources (i.e. fiber links) to be deployed over the current legacy infrastructure. This network upgrade must be done to satisfy some requirements such as expected traffic demands and network failure scenarios. To that end, a planning tool can be used to decide the hardware to be installed at minimum cost [Gi14]. Note that this is just a use case of holistic network planning, where the planning tool has access to both, an inventory database with the available equipment and the current state of the network stored in the traffic engineering databases.

One of the key requirements commonly considered in optical network design is survivability [Li09]. In an in-operation dynamic network, an on-line recovery mechanism can be implemented and applied each time a failure impacts the network to restore the affected traffic. In this regard, fast and efficient dynamic restoration based on bulk path computation algorithms has been proven to restore optical connections affected by a link failure [Ca14]. It is worth mentioning that significant traffic changes in terms of volume or distribution could lead to saturate part of the network capacity resources affecting the effectiveness of the restoration algorithm to find alternative routes in case of failure. Roughly speaking, traffic evolution could lead to an increase of network vulnerability.

In this work, we face the incremental capacity planning problem for minimizing network vulnerability in flexgrid optical networks (hereafter referred as *Vulnerability-aware Incremental Capacity Planning problem VINCI*). This optimization problem, based on the Routing and Spectrum Allocation (RSA) problem (that belongs to the class of the so-called NP-hard problems), aims at finding the set of new links in the inventory with the minimum cost that allow reducing network vulnerability to above a desired threshold. This network vulnerability is measured in terms of traffic restorability in the event of single link failures. In fact, the restoration algorithm in [Ca14] can be used to find which current links violate the restorability threshold and, if any, trigger the incremental capacity planning.

This work contributes to solve the aforementioned problem by presenting two alternative methods. Firstly, an integer linear programming formulation is presented to provide optimal solutions. This formulation is based on a node-link formulation of a basic topology design problem involving RSA constraints. However, due to the complexity of this exact method, we propose a heuristic approach based on the GRASP meta-heuristic framework [Fe95]. Both methods are compared and evaluated through numerical results obtained from solving realistic instances. It is important to remark that the C++ heuristic implementation has been integrated within the framework of an OMNET++-based network simulator [As13], using the standardized architecture and protocols of a real flexgrid optical network.

1.2 Report organization

The rest of the document is organized as follows. Chapter 2 approaches the necessary background to clearly understand the contributions of this work. Basically, the main concepts of optical networks, holistic planning, and operational research used in this work are briefly presented. In Chapter 3, the concept of incremental capacity planning for minimizing network vulnerability is introduced, as well as the VINCI problem is stated. In Chapter 4, the mathematical formulation is described, detailing the notation and equations, as well as precise explanations and details about complexity. Chapter 5 is devoted to the heuristic algorithm, focusing on the specifications of the GRASP-based procedure. Chapter 6 presents the numerical results of the project, consisting in a comparison of both exact and heuristic methods, an exhaustive performance analysis of the heuristic algorithm, and an analysis of the problem solutions leading to major conclusions of this work. Finally, Chapter 7 concludes the report with the main contributions and conclusions of the project.

The document includes few appendices containing part of the implemented code and extended tables containing the raw results used to compose tables and figures in the document.

Chapter 2.

Background

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

2.1 Dynamic optical networks

2.1.1 Basic concepts

An optical network can be defined as a graph 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 optical nodes, named as Optical Cross Connects (OXC), capable of establishing and deleting optical connections. The optical technology uses a range of frequencies of the total Optical Spectrum (OS), and it is measured in Gigahertz (GHz). The capacity of an optical link depends on the OS width and other factors like the spectral efficiency of established connections.

Basically, a traffic *demand* is a request 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 (Mb/s) or Gigabits per second (Gb/s). If the demand cannot be ensured due to the lack of capacity resources, then the demand will become *blocked* (i.e. not served). When a demand is accepted, an optical connection in the network is established between source and termination nodes; these optical connections are called as *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, each of them allocated in different parts of the available OS. In the signaling process of optical connections for real operator networks, connections are also referred to as *label switched paths* (LSP).

2.1.2 Flexgrid optical networks

In ITU-T Recommendation G.964 [G964], it has been included the definition of a *flexible grid* (*flexgrid*) (previously introduced in [Li11]). Flexgrid optical networks require specific components such as Bandwidth-Variable Wavelength Selective Switches to build Bandwidth-Variable Optical Cross Connects. The OS is divided into *slices*, 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 centered and thus it allows positioning the slices within the whole OS. Moreover, a subset of contiguous (adjacent) slices is called *slot* and it is characterized by its CF and the number of slices that contains. In order to illustrate the concepts introduced above, Figure 2-1 represents the spectrum of a fiber optical link using the flexgrid technology.

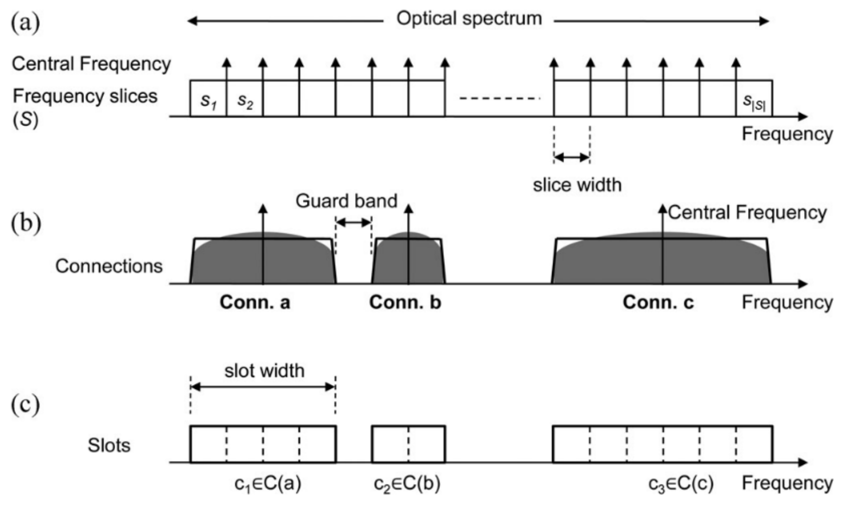


Figure 2-1: Logical representation of fiber link

In order to find the route and spectrum allocation for lightpaths in flexgrid optical networks, the *Routing and Spectrum Allocation* (RSA) problem must to be solved. 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 in finding a certain slot which must accomplish the contiguity and continuity constraints. This means that all slices in a lightpath must be one next to the other (spectrum contiguity constraint) as well as the assigned slot must be placed in the same part of the OS (i.e. using the same CF) for all links conforming that optical connection (spectrum 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 slices 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 slices 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 slot uses the slices {S5, S6} for this connection.

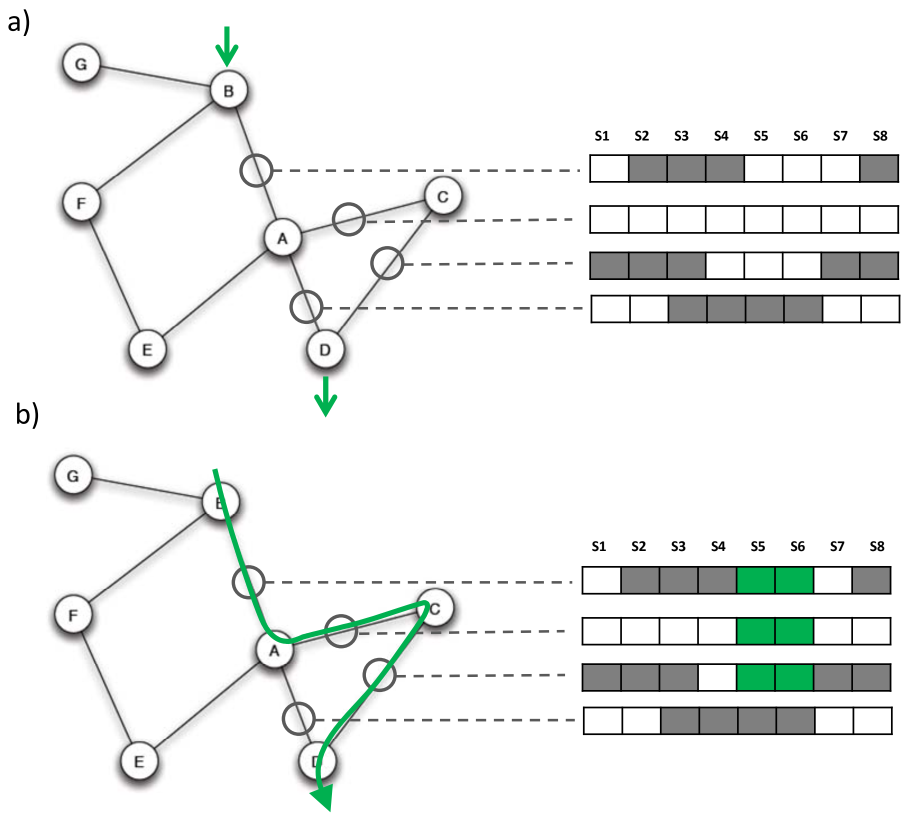


Figure 2-2: RSA, continuity and contiguity constraints.

2.1.3 Static and dynamic traffic

Two different approaches can be considered for planning and operating communications networks: static and dynamic traffic scenarios [Gu04]. In static traffic scenarios, no changes are considered in the connections established during the working period of the network. Thus, the information about client demands to be served, i.e. the source and destination nodes and the required bandwidth, is known in advance. Since the routing of those demands can be computed before the network begins to operate and no changes are allowed during the working time, the optimality of the network planning is always kept. When the network resources are limited and some demands cannot be established, we can define the blocking rate as the proportion of those refused demands over the total.

On the contrary, in dynamic traffic scenarios demands are not known in advance and connections are continuously set up and torn down. We assume that client requests arrive to the network following a certain probability distribution function.

Moreover, connections remain active during a certain period of time, i.e. the service time, which can be also modelled by another probability function. The most common model for dynamic traffic is the Erlang model [ITU05], where arrivals are modelled following a Poisson probability function identified by the mean time between two consecutive arrivals, namely inter-arrival time (*iat*).

When Poisson arrivals are assumed, the service time follows an exponential probability function identified by the mean holding time (*ht*). The inverses of *iat* and *ht* are called inter-arrival rate (λ) and service rate (μ), respectively. The traffic intensity (or offered load) can be computed as *ht/iat* or, alternatively, λ/μ and its unit is the *erlang*. The traffic intensity represents the mean number of established connections in a network at a random time. The source and destination of the demands are also random variables and they can follow several models, e.g. uniformly distributed, proportional to the distance between nodes, etc.

As anticipated before, when the network does not contain enough free resources to establish a connection request, that connection is blocked. Then, we can define the *blocking probability* as the probability to refuse (block) a connection request at a random time. To compute the blocking probability of a network during a period of time, the amount of blocked connections is divided into the amount of connections requested. The blocking probability of a network is used to define and quantify the Grade of Service (GoS) of the network.

2.1.4 Network recovery

It is worth mentioning that a working optical network is subject to failures during its operation. In general, a recovery mechanism consists of a set of actions to return a network to its normal condition after a failure occurs, e.g.: a fiber cut. To quantify the quality of a recovery mechanism, two parameters can be defined and computed: resilience and availability. Resilience is generally defined as the capability of the network to continue into operation even when a failure occurs. On the other hand, availability is strictly defined as the probability that a network will be found in the operating state at a random time in the future [Gr03]. Note that a value equal to one represents that the network is always available.

Two types of recovery mechanisms can be implemented to increase the availability of the network: restoration and protection. When restoration is implemented, the data flow affected by a failure is re-routed after the failure is detected, using network spare capacity. Since some affected paths could not be re-routed due to the lack of capacity, the availability could be less than total.

Aiming at increasing the availability, protection can be implemented; it consists in replacing the failed working connection with a pre-assigned backup path. If each backup path is dedicated to protect only one working connection, the scheme is called Dedicated-Path Protection (DPP). On the contrary, when a spectrum slot can be used to protect more than one working connection, the scheme is called Shared-Path Protection (SPP). Note that the availability of DPP networks is equal to 1 for single fiber link failures.

2.1.5 Basic concepts on restorability

From now on, we will consider a set Q of *failure scenarios*. Each failure scenario $q \in Q$ considers only one single link failure. Thus, an element q consists of a tuple containing the affected link and the set of *affected demands D(q)* in use of that link.

When re-routing the set of affected demands of failure scenario q, there will be a particular set of network resources available for use, the *network resources at failure time* of failure scenario q. These are the available network resources before the link failure plus the set of resources in use by all affected demands (without including, obviously, the resources of the failed link that are useless during the failure). The set of affected demands shall be re-routed during the recovery time, which should be in the order of hundreds of milliseconds for real network operators [Ca14].

We say that an affected demand is *restorable* if it can be re-routed using the available network resources at failure time, without re-routing any other demand from $D(q)$. In case that a restoration lightpath could not be found, we will say that the demand is *not restorable.* The set of all lightpaths available for one demand $d \in D(q)$ using the available resources at failure time at scenario q, without re-routing any other element of $D(q)$ is called the *set of restored paths* of d at q, denoted as $LSP(q,d)$.

When trying to restore demands in a failure scenario, a combinatorial problem arises: each affected demand $d \in D(q)$ may have multiple restored paths $LSP(q,d)$. In addition, restoring multiple affected demands will require that none of these restored paths conflicts with each other. Of course, we would like to restore as many affected demands as possible. Thus, we define *the set of restorations of a failure scenario q* as $Z(q) \subseteq 2^{D(q)}$, whose elements are subsets of $D(q)$ – affected demands – which can be restored all at the same time using restoration paths that do not conflict with each other. Since every element $z \in Z(q)$ is finite – since $D(q)$ always it is – we can endow $Z(q)$ with a total order relation by means of the cardinality of their elements. Thus, given $y, z \in Z(q)$, we can define the following order: $y \leq z \Leftrightarrow |y| \leq |z|$.

From the total order and the finiteness of $Z(q)$, we deduce the existence of a maximal element when $Z(q)$ is non-void and define $R^q \in [0,1]$ as the *restorability coefficient* (or, simply, *restorability*) of a failure scenario *q*, computed as follows:

$$R^q = \frac{|z_{\max}(q)|}{|D(q)|} \tag{2.1}$$

where $z_{max}(q)$ is the maximum element in the ordered set $Z(q)$. Note that if $Z(q)$ is void we define it as $R^q = 0$.

There exists a weighted version of the previous coefficient that uses the bitrate of demands. Given the set of bitrates $b_1, \ldots, b_{|D(q)|}$ requested from each affected demand of $D(q)$, we can rewrite the previous coefficient as:

$$R^q = \frac{\sum_{i \in z_{\max}(q)} b_i}{\sum_{j \in D(q)} b_j} \tag{2.2}$$

which expresses the proportion of restored bitrate with respect to the total bitrate affected by a failure, when $Z(q)$ is non-void and zero otherwise. Note that if the bitrate of all demands is the same, then equation (2.2) is equivalent to (2.1). In this project we will work with this definition.

In some situations, it would be enough to restore not all affected demands ($R^q = 1$), but at least some. This can expressed by the choice of a *restorability threshold* denoted as $R^{threshold} \in [0,1]$. Fixing a restorability threshold, we say that *a link is vulnerable* when the following condition is attained:

$$R^q < R^{threshold} \tag{2.3}$$

In this paper, we will consider the same restorability threshold for any failure scenario.

2.1.6 Elements and protocols in flexgrid optical networks

To operate a flexgrid optical network, a control plane based on a Path Computation Element (PCE) [Fa06] is deployed. The Path Computation Element Protocol (PCEP) [Va09] is used for communicating Path Computation Client (PCC) and the PCE. PCE computes routes in response to path computation requests (PC Req) sent by a PCC. To that end, the PCE queries the Traffic Engineering Database (TED) and runs algorithms implemented within PCE to solve the RSA problem. In addition to solve the RSA problem for a given LSP request, some other algorithms are able to compute different type of requests, for example to perform elastic operations to increase or decrease the amount of spectrum allocated to the given LSP. The response (PC Rep) received by the PCC contains an Explicit Route Object (ERO) for the requested LSP. That ERO is then used by the Resource Reservation Protocol with Traffic Engineering extensions (RSVP-TE) [Aw01] during the signaling process.

In addition, to be able to re-optimize the usage of network resources, a Label Switched Path Data Base (LSP-DB) and LSP delegation are also necessary [Cr13]. Therefore, the PCE is in fact an *active stateful PCE* with a Global Concurrent Optimization (GCO) module [Le09]. The GCO module provides functionalities for obtaining better network-wide solutions by computing paths for a set of queries grouped together. In that way, near-optimal solutions for path requests can be obtained. As an example, authors in [Ca14] use the GCO module for restoration purposes. Figure 2-3 summarizes the aforementioned network architecture.

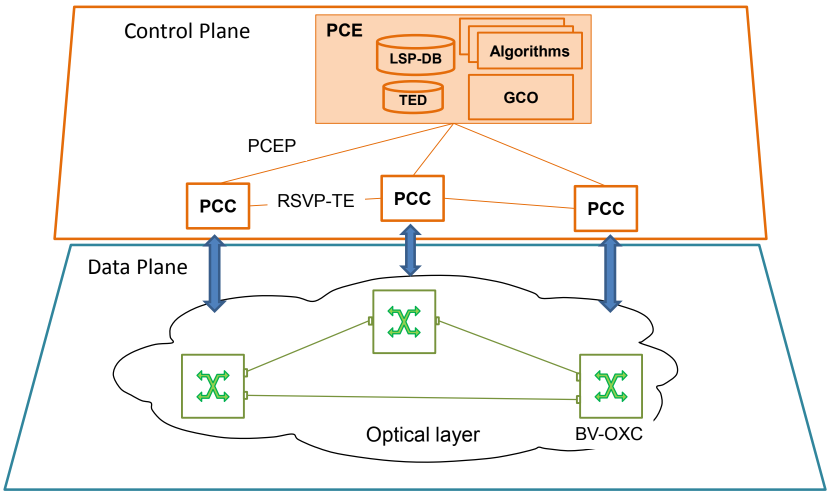


Figure 2-3: Considered flexgrid network architecture

2.1.7 OMNet++-based network simulator

In this work, some of the defined optimization methods will be implemented and embedded in an OMNeT++-based simulator [As13] that emulates a real performance of a flexgrid optical network using the architecture described in previous section. This simulator is developed in OMNeT++ 4.5 using C++11 with BGL 1.53.0 and Xerces-C++ 3.1.1 libraries. The simulator is organized in a number of modules representing either logical or physical elements in flexgrid optical networks: BV-OXC, PCE, and TE links, among others. Additionally, a Network Configuration module is included in the simulator in order to centralize generic configuration parameters, such as the frequency slice width or the modulation efficiency. In order to cover a wide range of scenarios, those modules can be configured for each specific scenario.

Each component includes its own set of methods simulating realistic functionalities. These modules and the interaction between them emulate the architecture and the protocols described before. By combining and configuring nodes and connections among them, specific flexgrid optical network topologies can be created in a .ned file; experiments can be afterwards run over that topology. A *Basic.ned* file is already defined with a default configuration for each module.

Each node with traffic generation capability generates new requests messages according to a Poisson process independently, i.e. without prior knowledge of the traffic generated by other nodes. The holding time of the connection requests is exponentially distributed with a configurable mean value. Connection's destination is randomly chosen with equal probability (uniform distribution) among the rest of nodes with traffic generation capabilities. Different values of the offered network load are created by changing the inter arrival time while keeping the mean holding time constant. Furthermore, the bandwidth demand of each connection request is randomly selected according to a traffic profile.

PCEP and RSVP-TE implementations are based on OMNeT++ messages which are exchanged between PCCs and PCE. Physical characteristics of control links can be emulated by configuring a number of parameters, such as transmission delay.

To illustrate the interaction among the different modules Figure 2-4 represents the messages exchanged to establish a new LSP on the network. The source PCC for the requested LSP sends a PCEP PC Req message (message 1 in Figure 2-4) to the PCE requiring a new path computation. Upon the reception, PCE processes the message, finds the proper algorithm for that request and calls it (2); in the example, the selected algorithm is that for solving the RSA problem. After the algorithm finds a feasible solution, it builds the ERO with the route and spectrum allocation and returns it to PCE, which builds a PCEP PC Rep message, includes the ERO and sends it back to the originating PCC. Upon receiving the response message, the originating PCC starts the signaling process by sending a RSVP-TE PATH message to the next node in the route of the LSP. To that end, it uses the information in the received ERO. Once the destination PCC receives the PATH message, it allocates physical resources and sends a RSVP-TE RESV message back through the same path so that intermediate PCC allocate also physical resources. When the RESV message reaches the source PCC and the physical resources are allocated in the node, requested LSP is established.

Apart from the modules and functionalities described above, a set of auxiliary classes for different purposes is considered, such as those for statistics and data input and output. Although OMNeT++ provides statistics functionalities, additional output files with the desired format can be created by implementing ad-hoc classes.

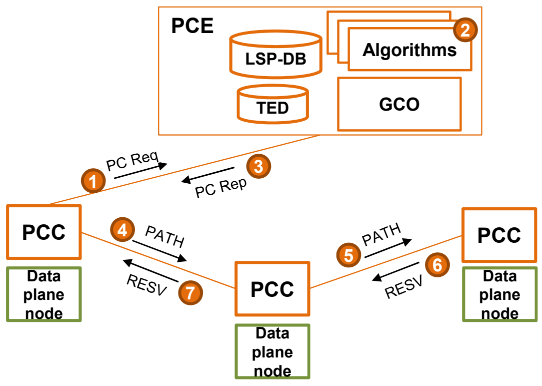


Figure 2-4: Messages exchanged to establish a new LSP (from [As13]).

Finally, it is worth noting that the proposed architecture can be extended with new modules and functionalities. Specifically, new algorithms can be implemented and tested easily by adding new C++ classes and making them accessible by the PCE. This will be done in this project to integrate the algorithm developed to solve the VINCI problem.

2.2 Network planning

2.2.1 Network planning cycle

The design of a telecom network as a set of gradual consecutive upgrading steps cannot be planned at the starting time because of several aspects, such as traffic uncertainty, which make it impossible to compute precise solutions for the future. As a result, solving each design step taking as input precise data for the next period seems to be the most practical way to deal with the planning problem.

Figure 2-5 shows the planning flow chart considered in this work [Ru14.1], where the following list of inputs involved in the process is assumed:

- The Network Management System (NMS) managing the core network, implementing fault, configuration, administration, performance, and security (FCAPS) functions.
- A Planning Department administrating the planning process, i.e., analyzing the network performance and finding bottlenecks, receiving potential clients' needs, evaluating network extensions and new architecture, etc.
- An inventory database containing all equipment already installed in the network, regardless they are in operation or not.
- An Engineering Department, performing actions related to equipment installation and set-up.

- A planning tool in charge of computing solutions for each planning step. Several sub-problems related to network reconfiguration, planning, and dimensioning, among others, need to be solved.

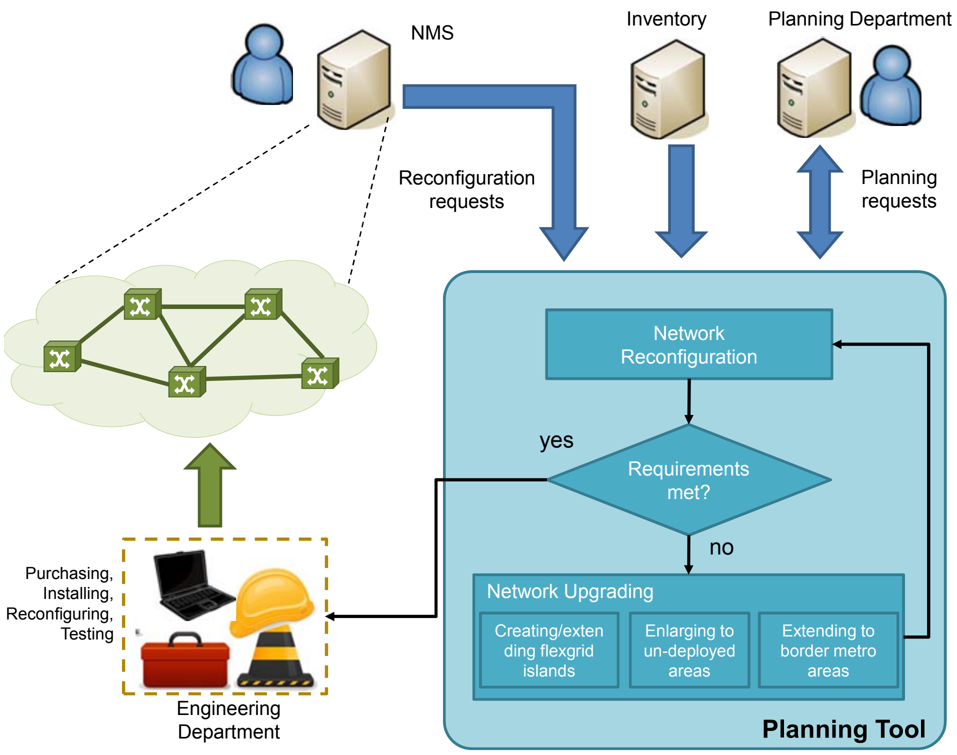


Figure 2-5: Network planning flow chart

We consider that a planning step begins when the planning tool receives a request that can be originated in different systems responding to different reasons:

- Operators analyzing data gathered by the NMS detect that a planning step can be attempted to improve the performance of the current network. E.g., bottlenecks have been detected in some parts of the network and its current configuration will not be able to allocate expected traffic, so *reconfiguration* can be attempted. Note that these triggers arise asynchronously (i.e., without a predefined schedule).

- Planners request network re-planning to serve new clients or cover new areas. Contrary to reconfigurations coming from NMS, *planning requests* can be better synchronized with other network departments, such as the engineering department.

The planning tool solves the design problem in two phases. Firstly, the *network reconfiguration* aims at reconfiguring the existing network resources and served traffic to meet target requirements. The solution of this problem consists in a set of actions that can be done in the network without purchasing and installing new flexgrid equipment. Therefore, the aim of this process is to exploit the possibilities of the currently available resources as much as possible before purchasing and installing new equipment.

Among others, some of the possible actions that could form a solution of this reconfiguration phase are, (i) modifying the physical intra-connectivity at central stations, (ii) moving physical devices, e.g., transponders, from one location to another in a different part of the network, and (iii) set-up and tear-down optical connections. Additionally, already purchased and installed network resources not yet activated can be put in operation in this phase. Thus, the reconfiguration process should process inventory data to decide whether some of these resources must be activated or not. It is worth highlighting that to implement such reconfigurations, the engineering department needs to perform manual actions that need to be scheduled and, therefore, not immediately processed. In fact, these manual interventions are usually performed during low activity periods since they might require temporally cutting some services, and therefore the whole reconfiguration process might last several weeks.

In the case when network reconfiguration is not sufficient to fulfill all the requirements, the *network upgrading* process, the second phase of the migration problem, is started. Network upgrading involves several network planning and dimensioning sub-problems, such as migrating selected regions to flexgrid, enlarging the network to cover new areas, extending the core towards the borders, etc. Obviously, the overall objective is to find solutions minimizing the total cost of the planning step, including purchasing, installing and configuring new equipment.

When a solution is found, the reconfiguration phase in invoked to guarantee that all the requirements can be met. If the solution is acceptable, it usually requires to be accepted by operators at the planning department, who then send it to the engineering department, which, in turn, organizes and schedules the set of processes that will physically implement the solution in the network. Although this whole process may take several weeks or even months to be completed, a new migration request can be started as soon as a subset of the new equipment is installed to partially reconfigure the network.

2.2.2 Migration towards in-operation network planning

As introduced before, the classical network planning life-cycle typically consists of several steps that are performed sequentially. The initial step receives inputs from the service layer and from the state of the resources in the already deployed network and configures the network to be capable of dealing with the forecast traffic, for a period of time. That period is not fixed and actual time length usually depends on many factors, which are operator and traffic type specific. Once the planning phase produces recommendations, the next step is to design, verify and manually implement the network changes. While in operation, the network capacity is continuously monitored and that data is used as input for the next planning cycle. In case of unexpected increases in demand or network changes, nonetheless, the planning process may be restarted.

As technologies are developed to allow the network to become more agile, it may be possible to provide response to traffic changes by reconfiguring the network near real-time. In fact, some operators have deployed Generalized Multi-Protocol Label Switching (GMPLS) control planes, mainly for service set-up automation and recovery purposes. However, those control only parts of the network and do not support holistic network reconfiguration. This functionality will require an in-operation planning tool that interacts directly with the data and control planes and operator polices via OSS platforms, including the NMS.

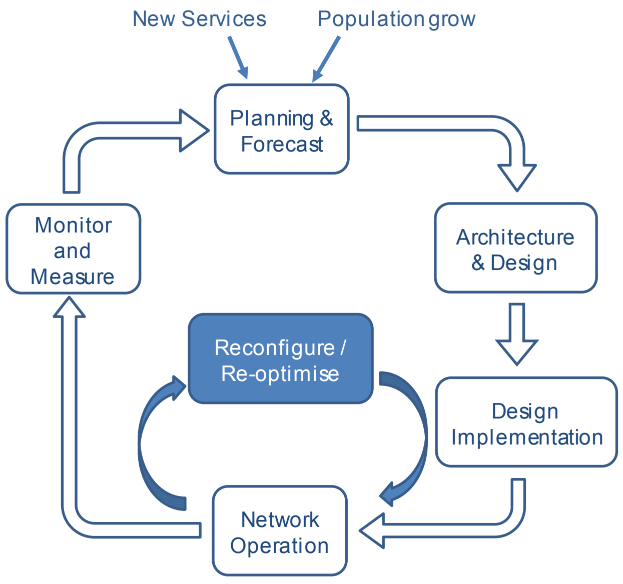


Figure 2-6: Networks life-cycle.

Assuming the benefits of operating the network in a dynamic way are proven, the classical network life cycle has to be augmented to include a new step focused on reconfiguring and re-optimising the network, as represented in Figure 2-6. We call that step *in-operation planning* and, in contrast to the traditional network planning, the results and recommendations can be immediately implemented on the network [Ve14.1].

To support dynamicity, however, the current network will need evolve to include a functional block between the service layer and the network elements to support multi-service provisioning in multi-vendor and multi-technology scenarios; two standard interfaces are required. Firstly, the *north bound* interface that, among other tasks, gives an abstracted view of the network, enabling a common entry point to provision multiple services and to provision the planned configuration for the network. Moreover, this interface allows coordinating network and service layer according to service requirements. Secondly, the *south bound interface* covering provisioning, monitoring, and information retrieval.

Finally, operators will typically require human-machine interaction, this is to ensure new configurations and network impact are reviewed and acknowledged, before being implemented in the network.

2.2.3 ABNO Architecture and required functionalities for in-operation planning

Standardisation bodies, especially the IETF, have been working to address all the above requirements, and as a result, the ABNO architecture is now being proposed as a candidate solution [Ki13]. The ABNO architecture consists of a number of standard components and interfaces (e.g. PCE) which, when combined together, provide a method for controlling and operating the network. A simplified view of the ABNO architecture is represented in Figure 2-7.

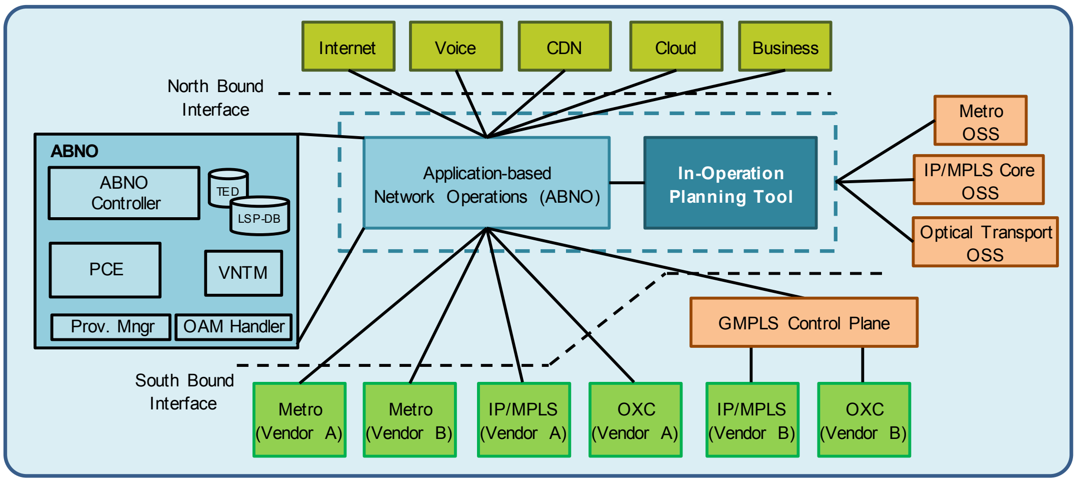


Figure 2-7: The ABNO architecture enabling in-operation planning

Directly connected to the ABNO architecture, the in-operation planning tool can be deployed as a dedicated back-end PCE for performance improvements and optimisations. The back-end PCE is accessible via the PCEP interface, so the ABNO components can forward requests to the planning tool.

Furthermore, in-operation network planning can only be achievable if planning tools are synchronised with the state of network resources, so new configurations can be computed with updated information, and those configurations can be easily deployed in the network. In the proposed architecture, the back-end PCE gathers network topology and current state of network resources, via the ABNO components, using protocols designed to convey link-state and traffic engineering information, such as BGP-LS.

2.2.4 Holistic network planning

In holistic network planning, the planning tool has access to both, an inventory database with the available equipment and the current state of the network stored in the traffic engineering databases. Thus, when solving an optimization problem in this context, not only the data from operational data bases will be required, but also a new data structure containing information about hardware components, such as spare linecards and inactive optic fibers, and their corresponding relationship within the corresponding network. This new data structure is known as the *inventory database*, or *Inventory DB* for the sake of simplicity.

Figure 2-8 presents such architecture. The central element is a planning tool that receives planning requests from the NMS. Each request must identify the specific planning algorithm to be executed, e.g. spare equipment placement. The algorithm can access both operation and inventory data to create an augmented graph of the networks to decide, e.g. which links need to be added.

To illustrate how data collected from different DBs is related, Figure 2-9 shows the relationship among elements in the TED, LSP-DB, and inventory. Note that TED and LSP-DB are already related since a LSP is defined as a pair of nodes and a hop list with nodes and interfaces, identified by their *id*.

The inventory DB contains, among other, equipment, linecards, and optical fibers that need to be related to nodes, interfaces, and links in the TED. Thus, ids of the elements in the TED are used in the inventory. Specifically, the *equipment* class includes the *ted-optical-node-ref* attribute linking to the *node-id* attribute of the *optical-node* class in the TED. In addition, an *interface* in the TED can be correlated with another in the inventory and an *optical-fiber* in the TED can be correlated with a *fiber-link* in the inventory.

With the proposed element linkage, planning algorithms can get data and perform some computations. For instance, from data about physical locations, a planning algorithm can compute the distances from a node that needs to be expanded to a set of warehouses where linecards are placed; that way, the linecard at a shortest distance can be selected.

The output of a planning request consists in a set of actions to be performed. For instance, to move a linecard from its current location to another warehouse in the spare equipment placement, or to install a linecard in a specific slot in an equipment.

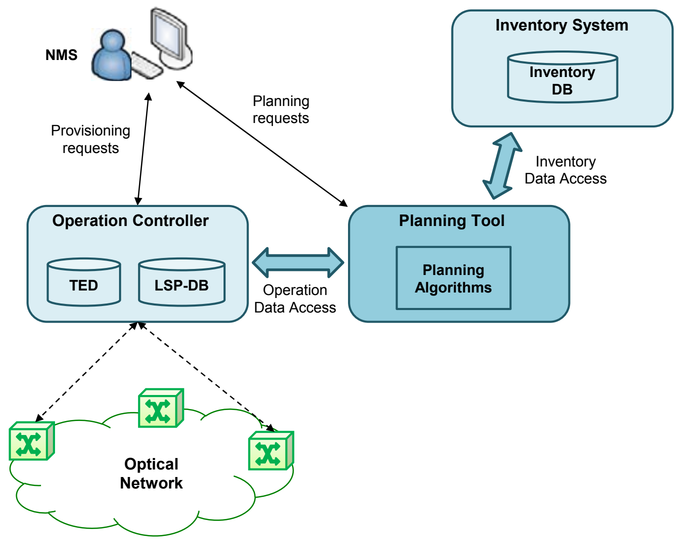


Figure 2-8: Holistic network planning architecture.

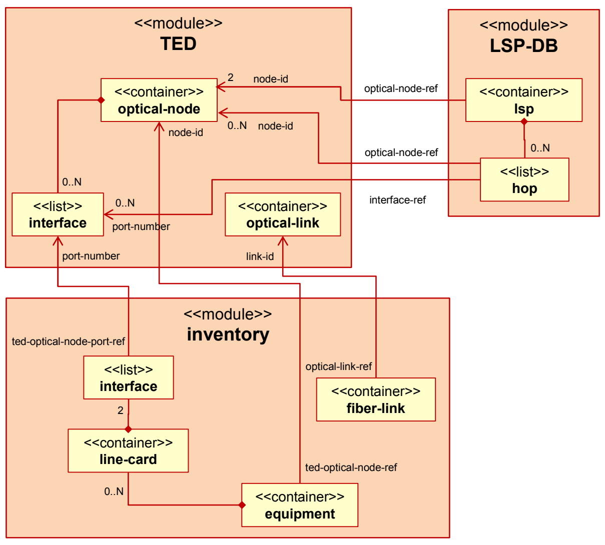


Figure 2-9: Operation and inventory databases relationship.

In the next section we present the proposed inventory model that consists of a set of related elements. In addition, the way to access some common data is also described.

2.2.5 The inventory model

The proposed inventory model is divided into three main blocks of classes: *i*) hardware-related; *ii*) fiber-related; and *iii*) building-related classes. A diagram is shown in Figure 2-10.

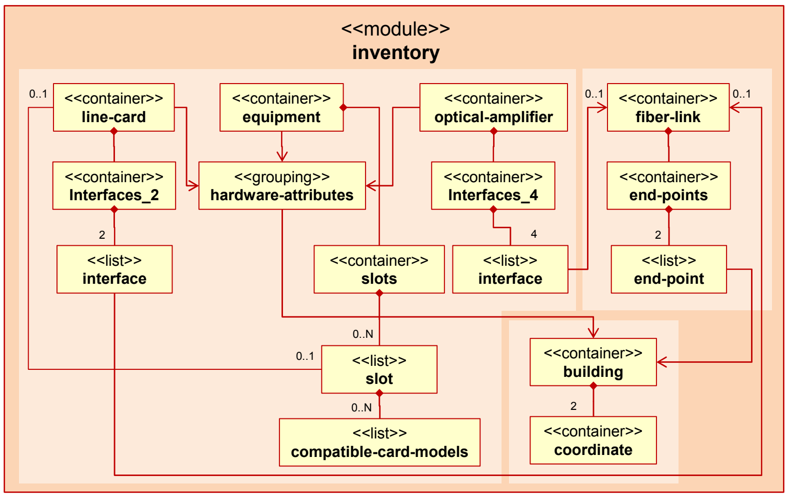


Figure 2-10: Class diagram summarizing the proposed inventory model.

The hardware-related block includes *equipment*, *linecard*, and *optical-amplifier* classes. The main attributes of these elements are grouped into the so called *hardware-attributes* class. An equipment is modelled as container of card *slots*. A card (in this version of the inventory model only linecards are considered) can be plugged into one single slot provided that the slot and the card are compatible. Hence, the list of compatible card models for each slot is also considered. Regarding linecards, we consider each one with two *interfaces* for optical transmission and reception, with an optical signal reach. Table 2-1 summarizes the main attributes of key hardware-related classes. The hardware-attributes class includes the *adm-id* attribute to link each piece of hardware to any other external DB, and the identifier of the building. Finally, note that the attribute *installed* can be used to identify spare cards.

The fiber-related block includes the *fiber-link* and the *end-point* classes; Table 2-2 summarizes its main attributes. A fiber link is modelled as a number of spans with already connected optical amplifiers and the total link length. The two end-points of a fiber link include a reference to the building, the exact location of the patch panel within the building and within the patch panel where the fiber is terminated.

Finally, the building class includes, in addition to the city and address of the building, its geographical coordinates for planning algorithms to compute distances between two buildings.

Table 2-1: Main attributes of fiber-related classes.

fiber-link	end-point
adm-id	building-ref
length	location
spans	num-in-patch-panel
active	
ted-optical-link-ref	

Table 2-2: Main attributes of the building class.

building
adm-id
short-name
city
address

2.3 Operations Research

In order to make the content of this project more understandable, we now introduce a well-known mathematical formulation to solve a basic planning problem in flexgrid optical networks, as well as a meta-heuristic intended to be used in following chapters.

2.3.1 Formulation of Network Problems

The problem we are dealing with is a kind of multi-commodity flow problem: multiple unitary flow demands between different source and destination nodes must be routed. It can be mathematically 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. 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.

Since the node-link formulation is the one selected along this project, we present an example for solving a single topology design problem based on [Ve14.2]. This problem aims at minimizing the amount of links needed to route a given set of demands. If a link is used by at least one demand, the link must be activated. Demands must be served by lightpaths accomplishing both spectrum continuity and contiguity constraints.

The following sets and parameters are defined:

N	set of nodes, index *n*.
E	set of all links, index *e*.
$E(n)$	set of links incident on node *n*, index *e*.
D	set of demands, each identified by an unique LSP, index *d*.
$C(d)$	set of all pre-computed slots for demand *d,* index *c*.
S	set of spectrum slices, index *s*.
q_{cs}	binary, equal to 1 if slice *s* is part of slot *c*; 0 otherwise.

o_d origin node of demand *d*.

t_d destination node of demand *d*.

The decision variables are:

x_{dec} binary, equal to 1 if demand *d* is routed through link *e* and slot *c*; 0 otherwise.

z_e binary, equal to 1 if link *e* is activated; 0 otherwise.

Thus, the formulation reads as follows:

$$\min \sum_{e \in E} z_e \tag{2.4}$$

Subject to

$$\sum_{e \in E(n)} \sum_{c \in C(d)} x_{dec} = 1, \quad \forall d \in D, \forall n \in \{o_d, t_d\}. \tag{2.5}$$

$$\sum_{e \in E(n)} \sum_{c \in C(d)} x_{dec} \le 2, \quad \forall d \in D, \forall n \notin \{o_d, t_d\}. \tag{2.6}$$

$$\sum_{\substack{e \in E(n) \\ e \neq e'}} x_{dec} \ge x_{de'c}, \quad \forall d \in D(q), c \in C(d), n \notin \{o_d, t_d\}, e' \in E(n). \tag{2.7}$$

$$\sum_{d \in D(q)} \sum_{c \in C(d)} q_{cs} x_{qdec} \le z_e, \quad \forall e \in E, \forall s \in S. \tag{2.8}$$

The objective function (2.4) minimizes the amount of links to be installed. Constraints (2.5) to (2.7) find a lightpath for every demand. Specifically, constraint (2.5) ensures that one lightpath for each demand is created with end nodes equal to the source and destination of demand. Constraint (2.6) guarantees that each lightpath is a connected set of links using the same slot along the route, whilst constraint (2.7) assures that the route does not contain any loop. Finally, constraint (2.8) prevents that any slice in any link is used by more than one demand, while installing the link when any slice is used.

Note that constraints (2.5) to (2.7) do not avoid the presence of cyclic paths. A cycle could be found in one specific case: where the source and destination nodes for one demand could be adjacent, using a single link to connect them and letting an arbitrary set of links to form a cyclic path using available resources, and with no use whatsoever. In this case, s simple post-processing removing such cycle could bring us the optimal solution.

2.3.2 GRASP meta-heuristic

The GRASP meta-heuristic is an iterative procedure consisting of a two-phase main algorithm which finds a good-quality solution at each iteration [Fe95]. Within the first phase of the algorithm (*constructive* phase) one feasible solution is built by means of an *ad-hoc* randomized greedy algorithm. The degree of randomness is determined by the parameter α. Next, the *local search* phase, designed to explore the neighborhood of the solution, is applied aiming at improving the current solution. The procedure finish when some criterion is met, e.g.: a number of iterations without improving the best solution or a maximum execution time. Table 2-3 shows the main algorithm of the GRASP meta-heuristic for minimization.

Table 2-3: GRASP Main Algorithm.

```
Procedure GRASP main Algorithm
begin
   x* = ∞.
   while stop criteria is not attained do
   x = constructivePhase(g(.), α);
   x = localSearch(f(.), x);
   if f(x)<f(x*) then
      x* = x;
   return x*
end
```

The *constructive phase* (Table 2-4) is characterized by a cost evaluation function (*g*) that allows ordering the elements to be included in the solution. At each *constructive* phase iteration, a candidate list (*CL*) containing all elements suitable to be included in the solution is created. Then, the restricted candidate list (*RCL*) is defined as a subset of *CL* containing the best elements given a certain *g*. The size of the *RCL* is determined by the α parameter. When $\alpha = 0$. *RCL* is equal to the best element, whereas when $\alpha = 1$, then *RCL*=*CL*.

Table 2-4: GRASP Constructive Phase.

```
Procedure constructive Phase(g(.),α)
begin
   x = Ø;
   Initialize CL;
   while CL ≠Ø do
     Build RCL(CL,g);
     Select s randomly from RCL;
     x = x ∪ {s};
     Update CL;
  return x
end
```

The following equation is used to create the *RCL*:

$$RCL(CL, g) = \{l \in CL : g(l) \le g^{\min} + \alpha \cdot (g^{\max} - g^{\min})\} \tag{2.9}$$

where:

$$g^{\min} = \min_{l \in CL} g(l) \tag{2.10}$$

$$g^{\max} = \max_{l \in CL} g(l) \tag{2.11}$$

Regarding the *local search* algorithm (Table 2-5), a neighborhood *H* of the solution *x* is built from a certain algorithm or function, for example, a simple exchange between one element in *x* and other not in *x*. Thus, depending on the improving strategy we can distinguish two cases: the *best-improving* and the *first-improving* strategies. While in the former all neighbors are investigated and the current solution is replaced by the best, in the latter the current solution moves to the first neighbor whose cost function value is smaller than that of the current solution.

Table 2-5: GRASP local search.

```
Procedure local Search(f(.),NB(.),x)
begin
  Compute H(NB,x)
  while H≠∅ do
    Select x from H.
    Compute H(NB,x)
  return x
end
```

2.4 Summary

In this section, we have reviewed the theoretical framework necessary to understand the aim of this project. We have seen how dynamic optical networks work, as well as the concept of restorability and holistic network planning. They will provide us with a proper structure to develop optimization methods to solve the network planning problem attempted in this project.

In the following chapter we introduce the concept of *incremental network planning problem* as well as the statement of the VINCI problem, which is the central point of this project.

Chapter 3.

Incremental capacity planning problem

This chapter introduces the optimization problem that is the focus of this project. After introducing the concept of incremental capacity planning for minimizing network vulnerability, the *Vulnerability-aware Incremental Capacity Planning problem,* hereafter referred to the VINCI problem, is stated.

3.1 Introduction

The incremental capacity planning problem consists in deciding the resources that need to be added to the network to reach the desired grade of service. Let us assume that the triggering event of incremental capacity planning is after a connection request could not be served. In such a case, the NMS might decide to request an *in-operation* network re-configuration, to reallocate the currently established LSPs, so as to make enough room for the new connection. In the case that the in-operation re-configuration could not find a feasible solution, it is clear that the only way to serve the connection request is by adding new resources to the network.

To illustrate incremental capacity planning, let us assume that a request for a 100Gb/s connection between X1-X6 in Figure 3-1a could not be served as a result of lack of resources in the network. The incremental capacity planning algorithm might decide that the optimal solution is adding a new optical link connecting X4 to X6, as shown in Figure 3-1b. To that end, the algorithm needs to know not only the current state of the network resources, represented by the TED, but also the capabilities of the physical equipment representing each optical node, the

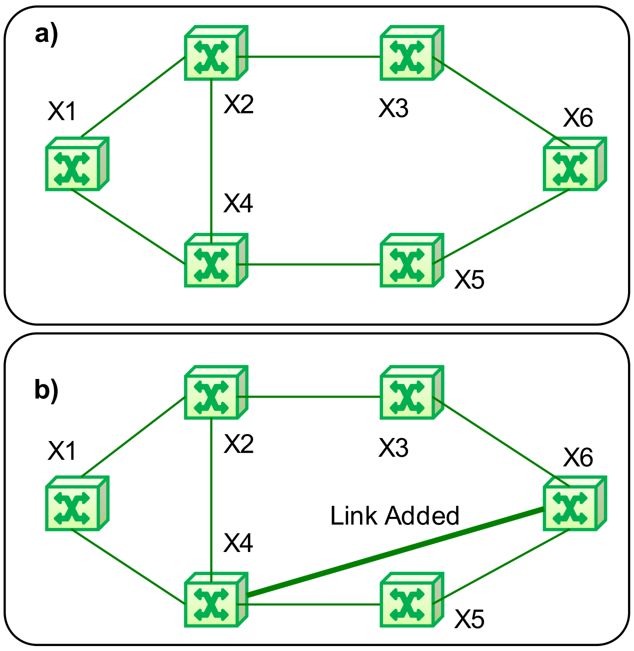


Figure 3-1: Example of incremental capacity planning.

availability of compatible linecards, available fiber links, etc., stored in an inventory database.

Table 3-1 presents the main pseudocode that summarizes the steps to follow in order to solve the incremental capacity problem. Firstly, the network topology graph is setup at line 1. From lines 2 to 5 we retrieve the inventory data and obtain the hardware equipment and the free fibers. At line 6 the topology is augmented by means of adding the set of free fibers obtained the line above.

Table 3-1: Incremental Capacity Planning Main Algorithm.

```
INPUT TED, LSP-DB, Inventory, $R^{threshold}$
OUTPUT LinksToAdd
1:    G(N, E) ← getTopology (TED)
2:    N.eq ← getEquipment (N, Inventory)
3:    C ← getLineCardsInWarehouse (N.eq, Inventory)
4:    NC ← computeCompatibility (N.eq, C)
5:    L ← getFreeFiberLinks (Inventory)
6:    G'(N, E∪L) ← augmentTopology(G(N, E), L, C, NC)
7:    S ← getFreeSpectrum(TED)
8:    P ← getLSPs(LSP-DB)
9:    B ← getBuildings (building(N) ∪ building(C), Inventory)
10:   $K_1$ ← computeInstallCosts (C, N)
11:   $K_2$ ← computeLinkCosts (E, L)
12:   solution ← solveMP (G', C, NC, B, P, S, $K_1$, $K_2$, $R^{threshold}$)
13:   if solution.infeasible then return INFEASIBLE
14:   return solution.L*
```

After this, we load the spectrum and lightpath data from the network at lines 7 and 8. From lines 9 to 12, we load the data related with transport and installation: building locations and installation costs between warehouses and network nodes.

Finally, at line 13 we solve the optimization problem behind the incremental capacity planning use case we are dealing with. The aim is to obtain a subset of L to be activated, thus improving some performance metric of the network. Nonetheless, several constraints shall be accomplished while solving this problem.

Next, we give a detailed definition of the vulnerability minimization use case behind the VINCI problem.

3.2 Vulnerability minimization use case

This use case of incremental capacity planning problem is related to low restorability of currently served traffic. The triggering event here is the detection of vulnerable links in our network. We define a link as vulnerable when the restorability of the demands crossing that link is lower than a required threshold. Without loss of generality, we assume that link restorability is computed by means of the equation (2.2).

When one or more vulnerable links are detected and assuming that re-routing affected demands does not change resources in use by not affected demands, the only way to ensure higher restorability in these links is by adding new resources to the network. Augmenting the topology we could solve the problem by restoring more demands than before, thus ensuring higher restorability and avoiding vulnerable links, but with extra costs.

Figure 3-2a illustrates this idea. The percentages close to links show the current restorability in case of such link fails. Note that we assume single link failures, that is, no more than one failure can be present in the network at the same time. Considering a restorability threshold of 95%, there are two vulnerable links (X2-X6 and X5 – X6). The topology can be therefore extended by adding a new link (X3- X6) that allows increasing the restorability of the vulnerable links above the required threshold. In fact, by adding new capacity resources, restorability can be either improved or kept invariant, but never can be decreased, as illustrated in Figure 3-2b.

At this point, we have all the necessary to state the VINCI problem in the following section.

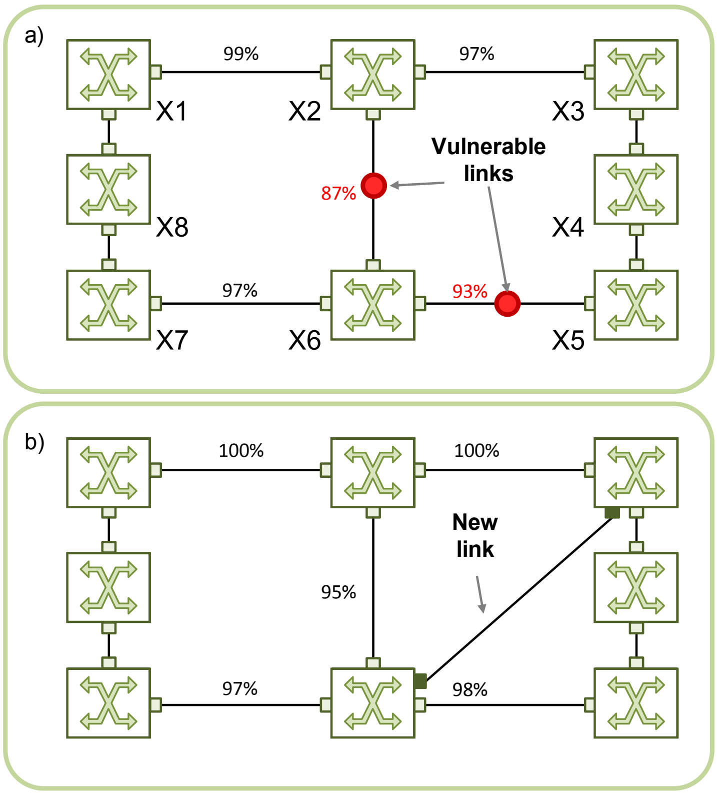


Figure 3-2: Increasing capacity to ensure higher restorability.

3.3 The VINCI problem

We can state the VINCI problem as follows:

Given:

- The augmented network topology.
- The set of failure scenarios, one for each vulnerable link.
- The available network resources for each scenario.
- Fiber links activation costs.
- Spare linecards installation costs.
- Nodal linecard compatibilities.
- A global restorability threshold.

Objective:

- Minimize the cost of extending the network topology, consisting in the cost of activating new links and installing new linecards.

Subject to the following constraints:

- Ensure restorability above the given threshold in all scenarios.
- Provide restoration lightpaths in all scenarios.
- Use the network available resources at failure time in every scenario.
- Installed linecards must be compatible with end nodes and have reach enough to support their respective link lengths.
- Satisfy nodal degree constraints when installing new linecards.
- Any extension of the topology will be available from any failure scenario.

In addition to the statement above, we assume the following constraints related to some optical network concepts introduced in Chapter 2:

- Each demand is satisfied using a single lightpath.
- Demands use the same spectrum assignment along the lightpath (i.e. no spectrum conversion is allowed)
- Demands use symmetrical lightpaths for inbound and outbound connections (same route and spectrum in both directions).
- We assume an initial scenario where all links are active and an arbitrary number of demands is satisfied.

3.4 Summary

In this section we have introduced the concept of incremental capacity planning for minimizing network vulnerability. After this, the statement of the VINCI problem, the optimization problem faced in this project, has been presented.

In the next chapter, the formulation of the VINCI problem as integer linear programming is detailed.

Chapter 4.

ILP formulation

As we introduced in the problem statement from Chapter 3, the problematic related with restorability and vulnerability involves several decision making. These decisions range from adding new links to assigning restoration paths and choosing specific inventory hardware. The large amount of binary decision variables makes this problem hard to solve optimally at first glance, thus creating the need of a mathematical model to provide optimal solutions.

In this Chapter, we introduce a mathematical formulation in terms of a binary integer linear programming for the VINCI problem. This formulation extends the basic RSA formulation presented in 2.3.1 by adding the specific features of VINCI:

- Considers several failure scenarios sharing the same topology.
- Measures the number of not re-routed paths in each scenario.
- Measures the restorability for each scenario, forcing this to be above a given threshold.
- Considers capacity and inventory resources not available for use.
- Adds a set of constraints related with inventory installation: linecard compatibility and reachability, and maximum nodal degree constraints due to slot capacity.

4.1 Notation

The following sets and parameters are necessary to define the problem:

N	set of nodes, index n.
$R(n)$	set of empty linecard slots at node n, index r.
T	set of spare linecards, index t.

E	set of links, index *e*.
$E(n)$	set of links incident on node *n*, index *e*.
$L \subseteq E$	set of inactive links, index *l*.
Q	set of failure scenarios, index *q*.
D	set of demands, each identified by an unique lightpath, index *d*.
$D(q)$	set of demands in scenario *q*, index *d*.
$C(d)$	set of all pre-computed slots for demand *d,* index *c*.
S	set of spectrum slices, index *s*.
f_e	binary, equal to 1 if link *e* is already in use; 0 otherwise.
g_{qec}	binary, equal to 1 if any affected demand from scenario *q* can use slot *c* of link *e*; 0 otherwise.
q_{cs}	binary, equal to 1 if slice *s* is part of slot *c*; 0 otherwise.
m_{rt}	binary, equal to 1 if linecard *t* is compatible with slot *r*; 0 otherwise.
c_{nt}	cost of installing linecard *t* in node *n*.
c_e	cost of activating link *e*.
km_e	length in kilometres of link *e*.
$reach_t$	reachable length in kilometres of linecard *t*.
o_d	origin node of demand *d*.
t_d	destination node of demand *d*.
b_d	bitrate in Gb/s of demand *d*
rth	restorability threshold.

The decision variables (all of them binary) are:

u_{er}	binary, equal to 1 if a new link *e* is connected to slot *r*; 0 otherwise.
v_r	binary, equal to 1 if a spare linecard is installed in slot *r*; 0 otherwise.
y_{rt}	binary, equal to 1 if linecard *t* is installed in slot r; 0 otherwise.
w_{dq}	binary, equal to 1 if demand *d* is not restored in the failure scenario *q*; 0 otherwise.
x_{qdec}	binary, equal to 1 if demand *d* is routed through link *e* and slot *c* in the failure scenario *q*; 0 otherwise.

z_e binary, equal to 1 if link *e* is activated; 0 otherwise.

4.2 Formulation

The formulation of the VINCI problem is as follows:

$$(\text{VINCI}) \quad \min \sum_{e \in L} z_e c_e + \sum_{n \in N} \sum_{r \in R(n)} \sum_{t \in T} m_{rt} y_{rt} c_{nt}. \tag{4.1}$$

Subject to

$$\sum_{e \in E(n)} \sum_{c \in C(d)} g_{qec} x_{qdec} + w_{dq} = 1, \quad \forall q \in Q, \forall d \in D(q), \forall n \in \{o_d, t_d\}. \tag{4.2}$$

$$\sum_{e \in E(n)} \sum_{c \in C(d)} g_{qec} x_{qdec} \le 2(1 - w_{dq}), \quad \forall q \in Q, \forall d \in D(q), \forall n \notin \{o_d, t_d\}. \tag{4.3}$$

$$\sum_{\substack{e \in E(n) \\ e \neq e'}} g_{qec} x_{qdec} \ge g_{qe'c} x_{qde'c}, \quad \forall q \in Q, \forall d \in D(q), \forall c \in C(d), \forall n \notin \{o_d, t_d\}, \forall e' \in E(n). \tag{4.4}$$

$$x_{qdec} \le g_{qec}, \quad \forall q \in Q, \forall d \in D(q), \forall c \in C(d), \forall e \in E. \tag{4.5}$$

$$\sum_{d \in D(q)} \sum_{c \in C(d)} q_{cs} x_{qdec} \le z_e, \quad \forall q \in Q, \forall e \in E, \forall s \in S. \tag{4.6}$$

$$1 - \frac{\sum_{d \in D(q)} b_d w_{dq}}{\sum_{d \in D(q)} b_d} \ge rth, \quad \forall q \in Q. \tag{4.7}$$

$$\sum_{r \in R(n)} u_{er} = z_e - f_e, \quad \forall n \in N, \forall e \in E(n) \cap L. \tag{4.8}$$

$$\sum_{e \in E(n)} u_{er} = v_r, \quad \forall n \in N, \forall r \in R(n). \tag{4.9}$$

$$\sum_{t \in T} m_{rt} y_{rt} = v_r, \quad \forall n \in N, \forall r \in R(n). \tag{4.10}$$

$$y_{rt} \le m_{rt}, \quad \forall n \in N, \forall r \in R(n), \forall t \in T. \tag{4.11}$$

$$\sum_{n \in N} \sum_{r \in R(n)} m_{rt} y_{rt} \le 1, \quad \forall t \in T. \tag{4.12}$$

$$\sum_{t \in T} reach_t y_{rt} \ge \sum_{e \in E(n)} km_e u_{er}, \quad \forall n \in N, \forall r \in R(n). \tag{4.13}$$

The objective function (4.1) minimizes the cost of expanding the original network topology. The first sum computes the cost of activating the set of new links, and the second one computes the installation cost of those network cards required to activate the set of new links.

For the sake of comprehension, we can split the set of constraints into two groups. The first one, related with routing and spectrum allocation of demands, and the second one, related with inventory assignment.

Constraints (4.2) to (4.6) find routes and allocate spectrum for each affected demand in every failure scenario. In more detail, constraints (4.2) and (4.3) ensure that each affected demand will either be satisfied or not, without any spectrum assignment in this last case. Coefficients w_{dq} are responsible for this behaviour: when set to one, all variables x_{qdec} must be zero, thus removing any possible light path for that demand. When set to zero, constraints (4.2) to (4.5) force to serve the corresponding demand. Note that w_{dq} will be used to compute the restorability of each scenario.

Constraint (4.4) guarantees that no light path will use more than one single channel along every link, from the source to its destination node (this is the so-called spectrum continuity constraint). In conjunction with constraint (4.3), they ensure a nodal degree of either zero or two for every channel, for every demand, in every failure scenario. That is, if the demand is served, it must follow a non-bifurcated path from source to destination. Note that constraint (4.5) bounds the routing binary variable to one if and only if there are free spectrum resources.

Constraint (4.6) ensures that a spectrum slice in a link would be part of at most one light path. Coefficients q_{cs} are responsible for using a contiguous set of slices from pre-computed channels, and coefficients g_{qec} prevent affected demands from using spectrum resources not available for use. This concludes the routing and spectrum assignment part of the problem.

Constraint (4.7) stores the key point of this formulation: ensures restorability for every failure scenario above the given threshold. It is worth mentioning that this constraint is based on equation (2.2) to compute the link restorability.

Constraints (4.8) to (4.13) deal with inventory assignment. Constraint (4.8) connects each new link to an available slot at each link end node. For each of these slots with a new connected link, an available linecard is installed by constraint (4.9), whereas constraints (4.10) and (4.11) ensure the compatibility of the installed card with the hosting slot. In addition, it is necessary to ensure that every linecard in the inventory can be used no more than once, which is guaranteed by constraint (4.12). Finally, constraint (4.13) avoids assigning linecards whose maximum reach is under the fiber link length.

4.3 Problem complexity and size

The basic topology design problem (based on basic RSA constraints) presented in Chapter 2 was proved to be NP-complete in [Ch11] and [Wa11]. We can deduce as well the high complexity of this problem, since it includes several embedded RSA sub-problems, in addition to the new set of constraints related with restorability and the inventory. Therefore, we can deduce that the VINCI problem is NP-complete.

To finish this chapter, we present the (approximated) dimension of this problem in terms of the number of binary variables and constraints as a function of the sizes of the sets involved. Table 4-1 shows those expressions, where |*| represents the cardinality of the set. To really appreciate such dimensions, a simple computation based on the instances presented in Chapter 6 is here provided. Thus, even moderated-size instances from a topology with 8 nodes, 12 active links, 20 potential links to active, 160 spectrum slices, 2 vulnerable links and 60 affected demands per link generates approximately ILP instances with 6e5 binary variables and 5e6 constraints. In view of these numerals, it is worth noting that solving the ILP will result in an unaffordable task even using powerful hardware and efficient solver implementations when facing real instances with hundreds of links and demands.

Table 4-1: VINCI problem size

# of binary variables	$\lvert Q\rvert\lvert D\rvert\lvert E\rvert\lvert C\rvert + \lvert N\rvert\lvert R\rvert\left(\lvert E\rvert + \lvert T\rvert\right)$
# of constraints	$\lvert Q\rvert\lvert E\rvert\left(\lvert N\rvert\lvert D\rvert\lvert C\rvert + \lvert S\rvert\right) + \lvert N\rvert\lvert R\rvert + \lvert T\rvert$

4.4 Summary

In this chapter, the ILP formulation of the VINCI problem has been presented. In light of the complexity and the size of the problem, we conclude the need of providing another method to solve this problem to be able to offer results in an affordable way. In next chapter, we introduce a heuristic based on the GRASP meta-heuristic framework intended to provide a good balance between solution quality and computation time.

Chapter 5.

Heuristic approach

As argued in the previous chapter, solving the VINCI problem with an ILP could be expensive in terms of time and computation resources. Hence, a heuristic approach is required in order to obtain a good balance between solution quality and computation time.

The heuristic presented consists in a main constructive algorithm that includes a GRASP meta-heuristic to be solved for each failure scenario. The proposed randomized constructive procedure and the consideration of different criteria to sort failure scenarios before processing them provide the needed diversification to effectively explore the problem solution space.

5.1 Main algorithm

The purpose of the main algorithm is to compute different solutions iteratively, according to some differentiator behavior in each iteration in order to increase the chances of finding a better solution.

Table 5-1 shows the main algorithm of the VINCI problem we are facing. After initializing some variables and augmenting the current topology with the potential links to be added from the inventory (lines 1 to 4), the main algorithm runs for a certain number of iterations. Each iteration solves sequentially each failure scenario and accumulates the set of resources to use from the inventory (links and linecards). The order in which the failure scenarios are processed can be either randomly or greedily chosen. Note that, while the former criteria just need one main iteration, the latter must be executed few times in order to ensure enough diversity in the solution construction.

We can classify the code executed in each iteration into 3 phases. The first one, the pre-processing phase (lines 5 to 10), resets all necessary variables to their original state, since they will carry changes from the last iteration, preparing this way the inventory and the network for the next phase.

The second phase, or processing phase, is responsible for computing the solution of that current iteration, including the execution of the GRASP meta-heuristic in each failure scenario. In this phase, the inventory as well as the set of paths from the failure scenario are modified in order to construct a solution. This phase starts at line 11 and ends at line 20 or 24.

In the last phase we post process the solution obtained after the processing phase. We compare the constructed solution against the best solution obtained at that point (incumbent solution), and perform the necessary changes.

Table 5-1: Main algorithm

```
INPUT: G(V,E), $Inv_0$, $Q_0$, sortMode, maxMainIte, maxGraspIte, $\alpha$, rth
OUTPUT: (incumbent,incumbentCost)

1:   L ← getNewLinks(G,$Inv_0$)
2:   G ← extendTopology(G,L)
3:   incumbent.cost ← +∞
4:   incumbent.feasible ← false
6:   for 1..maxMainIte do
5:        iterSol.feasible ← true
7:        setCost(E,1)
8:        setCost(L,100)
9:        Q ← sort(sortMode,$Q_0$)
10:       Inv ← $Inv_0$
11:       for each q ∈ Q do
12:            releaseWorkingPaths(G,D(q))
13:            disableFailureLink(q)
14:            graspSol ← doGRASP(Inv,G,D(q),maxGraspIte,$\alpha$,rth)
15:            releaseRestoredPaths(G,D(q))
16:            enableFailureLink(q)
17:            allocateWorkingPaths(G,D(q))
18:            if graspSol.feas = false then
19:                 iterSol.feas ← false
20:                 break
21:            else
22:                 updateCosts(sol.links ∩ L, 1)
23:                 iterSol.inv ← decreaseInventory(graspSol.inv)
24:       if iterSol.feas = true then
25:            iterSol.cost ← computeCost(iterFibers,iterLcards)
26:            if iterSol.cost < incumbent.cost then
27:                 incumbent.cost ← iterSol.cost
28:                 incumbent.feas ← true
29:                 incumbent.inv ← iterSol.inv
30:  if incumbent.feasible then
31:       return INFEASIBLE
32:  else
33:       return (incumbent)
```

5.1.1 Pre-processing phase

Each main iteration starts with an instance of the original inventory, and a fixed cost for each link of the topology. The solution for that iteration is also initialized empty at this point.

At line 9 of the pseudocode, the set of failure scenarios is sorted according to one of the following sorting criteria:

- *greedy-sorted*: failure scenarios are sorted in descendent order of number of affected demands. In other words, those scenarios with a higher number of affected demands are processed before those with a lesser number.
- *random-sorted*: the set of failure scenarios is randomly sorted. Note that this option adds significant diversity when several failure scenarios are processed.

5.1.2 Processing phase

Once the necessary data has been set up at the beginning of the iteration, each failure scenario is processed, respecting the order established by the sorting criterion explained before. The following steps will be followed in each scenario:

- Line 12: Remove from the network the demands affected by the failure scenario. This action releases the capacity used for those demands in its current route.
- Line 13: Disable the vulnerable link of failure scenario. Recall that this link cannot be used for restoring paths.
- Line 14: Apply the GRASP-based algorithm (*doGRASP* function) to restore the affected demands. This phase will update the current topology with the new resources necessary to ensure the restorability threshold for this failure scenario.
- Line 15 - Remove all restoration paths done by the GRASP algorithm. This is the first step needed to prepare the network for processing the next failure scenario.
- Line 16 - Enable again the vulnerable link.
- Line 17 - Restore the network capacity resources to its original state.
- Lines 18 to 23 - Check whether the obtained solution is feasible or not. Moreover, costs and inventory are updated according to the results provided by the *doGRASP* function. Specifically, costs of new links to install found in this iteration are set as they were really installed. In this way, they can be reused for next failure scenarios without adding additional cost to the solution. Note that inventory needs to be accordingly updated not only in

terms of the new links to install but also in terms of the linecards needed to support new links installation.

5.1.3 Post-processing phase

A main algorithm iteration ends when every failure scenario is successfully processed or when anyone of them returns infeasible at *doGRASP* – see line 20. In the former case, the current iteration solution is compared against the best solution found obtained so far. At line 24 of the pseudocode, we check if the solution from that iteration is feasible or not. In case of feasibility, all the inventory necessary for every failure scenario will be supplied to function *getCost* – at line 25. This function is responsible for computing the cost of the solution, consisting in the activation of new added fibers, installation of linecards and their transport from buildings (warehouses) to nodes. At line 26, the iteration solution updates the incumbent one if the cost of the former is lower than the cost of the latter. Eventually, the heuristic returns the incumbent solution if found.

Following, the *doGRASP* function is described in detail.

5.2 GRASP-based heuristic

The *doGRASP* algorithm is a customization of the GRASP meta-heuristic. As we will see, this customization is intended to convert this meta-heuristic to a vulnerability-specific heuristic. This adaptation does not implement the local search phase, only the constructive phase.

Table 5-2 shows the pseudocode of the *doGRASP* algorithm, which follows the basics of the GRASP meta-heuristic introduced in the background chapter. The specifications needed to understand the procedure behind the *doGRASP* algorithm are described in following subsections.

5.2.1 Input data and initialization

Each call to *doGRASP* accepts the following input: the affected demands of the failure scenario and the current state of the inventory, as well as the all necessary configuration parameters. Each GRASP iteration will start from this same input data. This is, the links cost and inventory state at the beginning of each iteration will be always equal to the supplied input data.

Lines 1-3 of Table 5-2 shows the initialization needed at the beginning of each *doGRASP* execution. Basically, it consists in initializing the incumbent solution. Then, the main body of the algorithm is run for a given number of iterations (*maxGraspIte*), each one starting with the initialization of the iteration solution and the creation of the CL (lines 4 to 8).

Table 5-2: doGRASP algorithm.

```
INPUT: Inv,G,D(q),maxGraspIte,α,rth
OUTPUT: graspSol

1:   graspSol ← ∅
2:   graspSol.cost ← ∞
3:   graspSol.feas ← false
4:   for 1..maxGraspIte do
5:       iteSol ← ∅
6:       iteSol.inv ← Inv
7:       Generate CL from D(q) and Inv
8:       iteRth ← 0
9:       while CL ≠∅ && iteRth<rth do
10:           Build RCL from CL, g, and α following eq (2.9)
11:           Select d randomly from RCL
12:           Update iteRth
13:           iteSol.paths ← iteSol.paths U {d.path}
14:           Update link costs of G
15:           Allocate(G,d.path)
16:           Update iteSol.inv
17:           Update CL
18:       if iteRth≥rth then
19:           if cost(iteSol)<graspSol.cost then
20:               graspSol ← iteSol
21:               graspSol.cost ← cost(iteSol)
22:               graspSol.feas ← true
23:       Release(G,iteSol.paths)
24:       return (graspSol)
```

5.2.2 Candidate elements

In our case, the GRASP heuristic will be applied to single failure scenarios, with no direct relation between them. The set of possible candidates is the set of affected demands of that scenario. The only relation between scenarios will be the modification of the inventory as well as the link costs, without affecting the structure of the GRASP heuristic in any case.

5.2.3 Evaluation cost of candidate elements

The following steps test whether an affected demand can be added to the *CL* or not (all included in lines 7 and 16):

- A restored path must exist. This path is computed with the RSA algorithm presented in [Ca12], which basically extends the Dijkstra algorithm for computing shortest paths [Di59] including spectrum allocation.
- All non-installed links in the restored path must be installable using the available inventory equipment, all of them at the same time.
- If all the above conditions are satisfied, the affected demand is inserted into the *CL*. Otherwise it is rejected.

As introduced in Chapter 2, the cost evaluation function *g* allows ordering the elements to be included in the solution. In our problem, this function is based on the inventory and the available network resources at failure time embedded into the extended topology. The associated cost of a candidate is the cost of its restored path. The cost of a restored path is the sum of the costs of its links. From the main pseudocode in Table 5-1, links from topology will have rather a cost of 1 or 100 at the beginning of each main iteration following the next rule: if a link existed prior to the topology extension, it will have a cost of 1; it will have a cost of 100 if it was added to the topology as a result of the extension. These costs can be modified during *doGRASP* algorithm execution. Note that we arbitrarily fix these costs to 1 and 100, being other values also valid. The rationale behind this set up is to promote the use of installed links and force using new links to install only when current resources are not enough to restore the affected demands. This explanation is extended in next section.

5.2.4 Candidate callback function

When a candidate is chosen and popped from the *RCL* (line 11), the iteration solution is updated (lines 12 and 13) and costs of links in the restoration path are changed to 1 (line 14). The reason behind this can be explained in terms of penalties: the first time a link of length 100 is used, it makes no sense to keep such a high cost, because it will have already been decided to activate it and other demands could take profit from it, instead of choosing among other high cost links. Note that the restoration path must be setup in order to allocate the capacity resources used by it (line 15).

The inventory required to activate such links is occupied and stored in the solution object corresponding to that GRASP iteration (line 16). The modified links before mentioned are stored as well. Before attempting to add a new element in the solution, the CL must be updated (line 17).

5.2.5 Stop condition

In the standard formulation, the GRASP meta-heuristic stops when it reaches an empty *RCL*. The remaining solution becomes feasible or not depending on some parameters – usually one will decide it depending on whether the *CL* is empty or not. In this particular version, however, we are not going to use any of these conditions.

Instead, we will use a stop condition based on the restorability coefficient of the failure scenario being processed (line 9). Recall that candidates are a subset of the set of affected demands. Then, if the *RCL* is not empty but the restorability coefficient is already above the given threshold, the solution will be feasible and the GRASP iteration will end successfully (line 18). On the other hand, if the restorability coefficient could not be increased above the threshold – even with a non-empty *RCL* – the solution will be inevitably infeasible and the GRASP iteration will prematurely stop as well.

If the iteration solution is feasible and it improves the incumbent one, the incumbent solution is properly updated (lines 19 to 22).

5.2.6 Output data

A *doGrasp* call will return the best solution among all GRASP iterations. It could return an infeasible solution as well. This solution will be used in the remaining steps of the processing phase in order to update the network and the inventory with the following information:

- The set of linecards and fibers necessary to perform demands restoration in that failure scenario – we will mark them as occupied from this point on.
- A subset of links from the topology extension which are being used by the restored demands. Recall that we will change the cost of these links to one from this point on.

Proceeding this way, next failure scenarios from the same main iteration will dispose of updated routing costs as well as the updated version of the inventory. Recall that each new main iteration will reset the inventory to its original state at the pre-processing phase.

5.3 Complexity analysis

Next, we analyze the complexity of the heuristic presented above.

Firstly, the topology costs are reset in order to restore possibly changed costs from the previous iteration. These steps have a complexity proportional to $E + L$, since every link from the topology must be revised. Next, the set of failure scenarios is sorted. In the worst case – the greedy sorting –this step will have a complexity of $Q \log Q$. The best case is given by the random sort, which shuffles all scenarios with a complexity of Q. We consider here the worst case complexity. Next, each scenario is processed making use of the GRASP heuristic, whose complexity is $k|D||N|\log|N| + |T|$. Assume that GRASP performs k iterations, and each of them has to compute a restored path for each affected demand – D in the worst case. Restored paths are computed using the Dijkstra algorithm, of complexity $N \log N$, which justifies the expression above. The term T refers to the number of linecards to be processed in the worst case to install the set of new links of that scenario. Therefore, grouping every term explained so far and simplifying, we end with an overall complexity of $|E| + |L| + |Q|\left(\log|Q| + k|D||N|\log|N| + |T|\right)$.

5.4 Summary

In this chapter, the heuristic designed to solve the VINCI problem has been presented. Consisting in a main algorithm that runs a GRASP-based heuristic inside, the heuristic explores a variety of feasible solutions in the aim of finding a good quality solution. Moreover, the overall complexity of the algorithm is low, since it basically consists in running iterations of sets of shortest path computations over a network.

In the following chapter, both ILP and heuristic methods will be evaluated through numerical results.

Chapter 6.

Numerical results

In previous chapters, we have defined the VINCI problem and offered two different approaches for solving it, namely, an ILP model and a GRASP-based heuristic. This chapter is devoted to present numerical results from solving the VINCI problem with the aforementioned methods. The chapter starts introducing the details of the reference scenario and illustrates a solution of the problem by means of snapshots of the visualization tool developed within the project. Then, comparison between the ILP and the heuristic is provided, as well as an exhaustive analysis on the heuristic performance. In view of all these results, conclusions are eventually presented.

6.1 Reference scenario

6.1.1 Network configuration and inventory

All network topologies presented in this chapter are based on two core networks (depicted in Figure 6-1). The TEST topology was specifically created for testing different solving methods and configurations, while the DT topology, which represents the optical core network of the operator Deustche Telekom in Germany, is provided with the aim to extend further conclusions for real optical networks.

For each of these two reference networks, we have created three topologies corresponding to different inventory configurations. Starting from a certain amount of non-active links (i.e. $|L|$), each topology increases the amount of active links by adding few of the links in the former set. In this way, we can evaluate how the VINCI problem would adapt to incremental networks in a realistic way. Moreover, we are increasing the amount of network scenarios to offer a wider set of numerical results. Thus, a total number of 6 different topologies are used in this numerical analysis, whose main attributes are summed up in Table 6-1. Note that

TEST_8_12 and DT_12_20 correspond to the topologies depicted in Figure 6-1 but adding the set $|L|$ of inactive links.

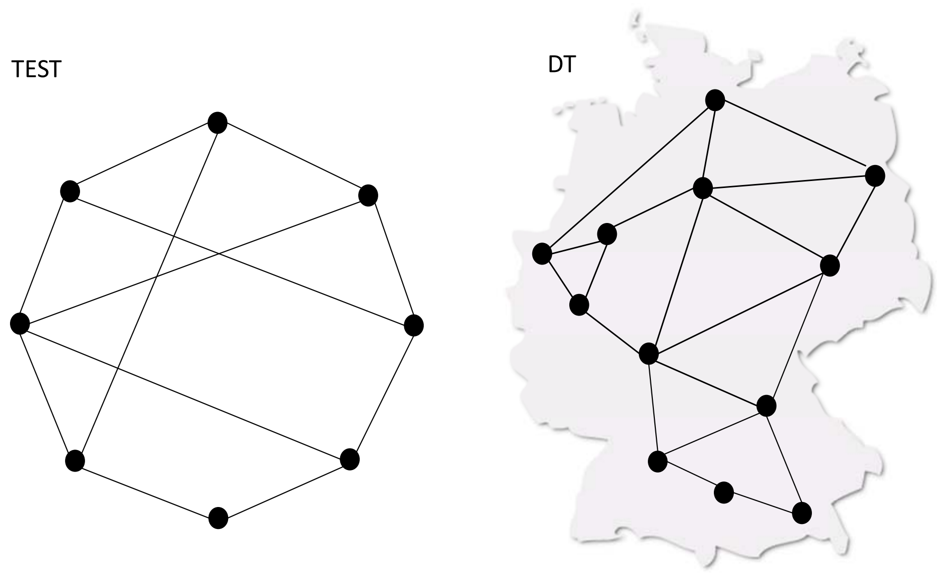


Figure 6-1: Reference networks

Table 6-1: Main characteristics of TEST-based and DT-based topologies

name	$\|N\|$	$\|E\|$	$\|L\|$
TEST_8_12	8	12	16
TEST_8_16	8	16	12
TEST_8_20	8	20	8
DT_12_20	12	20	46
DT_12_25	12	25	41
DT_12_30	12	30	36

Regarding the amount of linecards available in the inventory, we provided enough to ensure that every link in L could be installed if necessary, regardless of the capacity in terms of slots in network nodes. We considered two types of cards (short and long reach) with around unitary c_{mt} costs. On the contrary, link installation costs c_e were set in the order of 10^4 cost units. To restrict the amount of new installed links, we fix the amount of node slots able to hold a linecard to 8, so that the degree of a node can never exceed this value. For the sake of simplicity, we have considered a unique centralized warehouse to store all linecards.

Finally, we considered two different optical spectrum configurations: 1 THz and 4 THz, both fragmented into slices of 6.25GHz. The first configuration, used for ILP evaluation due to its lower size, generates 160 spectrum slices, whereas the second one, which is a more realistic configuration, involves 640 slices.

6.1.2 Instance generation

By means of the OMNeT++-based simulator presented in Chapter 2, instances for the VINCI problem using the already defined network configurations have been created. To obtain an instance, the following procedure was executed:

1. The simulator was run for a given offered load, so that a blocking probability in the steady-state is reached (e.g. 1%)
2. The bulk restoration algorithm described in [Ca14] was implemented in the PCE and run to detect which links had its restorability lower than a predefined threshold.
3. If vulnerable links were found, then the input data for both ILP and heuristic methods was collected and the execution of the VINCI problem triggered.

We fixed the restorability threshold to 95% for every instance. By properly tuning the offered load, we obtained instances with a wide range on the number of vulnerable links and affected number of demands per link. In all experiments we consider demands of 100 Gb/s, consuming each of them 6 spectrum slices.

6.1.1 Hardware and implementation details

All instances were solved in an Intel Core i7-4770 @ 3.40GHz processor with 4 cores and 16 GB of RAM running Ubuntu 14.04 64 bit.

We implemented the ILP in MATLAB R2014b using IBM CPLEX 12.5.1 as solver engine. Thus, each time a problem instance was generated, a XML containing all the necessary input data for the ILP was created. For checking purposes, an HTML 5 visualization tool, using KineticJS was used to graphically plot ILP solutions. On the other hand, the heuristic was integrated in the OMNeT++ 4.5 simulator, specifically as an independent ad-hoc class accessible from the GCO module.

6.2 Visualizing VINCI solutions

For illustration purposes, Figure 6-2 shows a small instance for the topology TEST_8_12 by means of the HTML5 visualization tool. To the left, the list of possible failure scenarios as well as the original network state is own. We can also observe the list of demands established in the network, being the working path of demand 6 depicted over the network in the center of the figure. Note that demand 6 (depicted in green, from node 3 to 8) crosses a vulnerable link. In fact, the restorability of this link is below the threshold because such demand 6 cannot be restored using the available network resources.

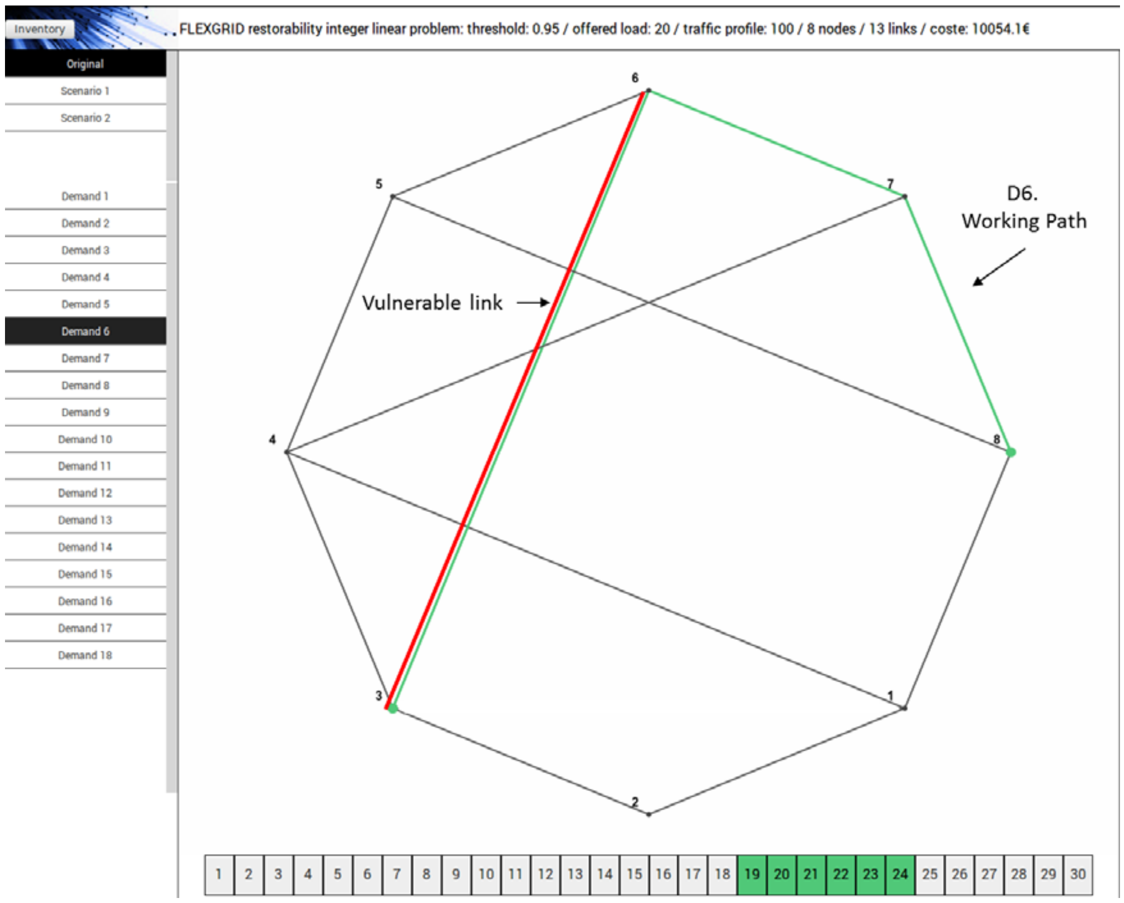

FLEXGRID restorability integer linear problem: threshold: 0.95 / offered load: 20 / traffic profile: 100 / 8 nodes / 13 links / coste: 10054.1€

Figure 6-2: Affected demand (green) and its corresponding failure link (red).

The solution provided by the ILP is shown in Figure 6-3. We can observe that a new link has been added between nodes 1 and 3 (dotted line). This link is necessary to restore demand 6. Indeed, it was added to the network in order to ensure a restoration path for that affected demand. Note that the restoration path (in blue) actually uses the new link and the free resources in previously existing links. Although alternative routes bypassing the vulnerable link could be found, the residual capacity (not shown in the figure) avoids finding a shorter route. In the upper-right corner of the figure, the restorability of this link is showed after the network is extended, showing a value of 100%. The activation cost is displayed as well.

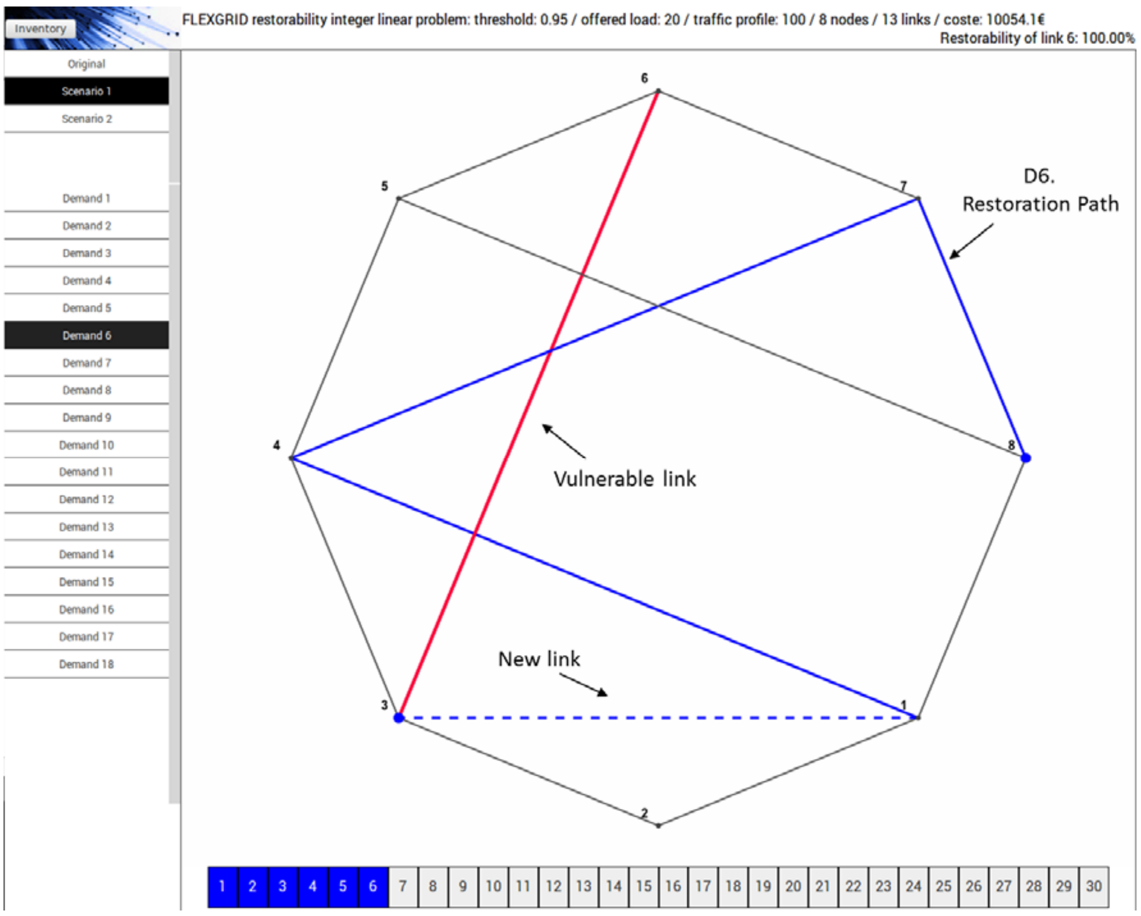


Figure 6-3: Restored demand (blue) using a new link (dashed).

6.3 ILP vs heuristic comparison

In this section, we compare the ILP and heuristic solutions for several instances in terms of objective function and execution time. Unfortunately and due to time and computational limitations, we got few ILP solutions since they need for several hours of computing each of them.

We launched instances for all TEST-based networks and for DT_12_20, each of them with a fixed spectrum of 1 THz (i.e. 160 slices). The number of vulnerable links of each instance ranged from 1 to 4. We limited the CPLEX execution time (again, to be able to obtain a set of instances during project execution) to 10 hours and the optimality gap to 0.01%.

From the experiment above, we got 9 instances where the ILP solution was obtained before reaching the time limit of 10 hours. For those instances (ranging between 500,000 and 3,500,000 binary variables), the heuristic with the best observed configuration reached the same objective function value than that from solving the ILP. However, execution times were enormously different, as can be seen in Figure 6-4. The figure compares the time to generate and solve the ILP and the time to solve the heuristic. A logarithmic scale was applied to the time axis in order to let a proper visualization. In summary, a difference of 2 orders of magnitude between the generation time and solving time in the ILP, as well as a difference of 7 orders of magnitude when comparing only the generation time of the ILP versus the heuristic solving time is observed.

From Figure 6-4, we can observe a couple of important points. On the one hand, the ILP generation time – this is, the time it takes to prepare and populate all constraints to the CPLEX model object – predominates over the solving time. Specifically, the order of magnitude of the generation time lies between 5 and 6 – thousands of seconds (i.e. several hours). On the other hand, despite the fact that the ILP takes few seconds to solve, there exists a difference of approximately three orders of magnitude in contrast with the solving time of the heuristic, since this last takes only a few milliseconds to solve the same instance.

Before taking conclusions of this comparison, let us analyze few instances where the ILP did not reach the optimal solution after 10 hours. Note that these instances needed several hours for generating the problem (as seen in Figure 6-4) and more than 10 hours to obtain the optimal solution. Table 6-2 provides relevant information about three suboptimal solutions. We can observe that the first two instances had a GAP near to 100% with an ILP objective function remarkably worse than the heuristic one. Therefore, we can easily deduce that the ILP could not reach the integer optimal solution in these two cases. The last instance, however, did not reached the established GAP of 0.01%, but kept close to it with a value of 0.04%. As we can see in this case, the solution provided by the heuristic is almost the same, since both solutions installed one single link with the same cost for the linecard installation. The only difference is that the activated link from the ILP was approximately 8 km shorter than the link provided by the heuristic, which represents a negligible difference between both solutions.

In light of the previous results and despite the fact that the comparison performed lacks from exhaustiveness due to the limitations of this project, we can conclude that the GRASP-based heuristic provides high-quality solutions (optimal or near-optimal in our study) in affordable running times several orders of magnitude lower than those of the ILP. Finally, note that solving the ILP for larger instances considering larger DT topologies and 4 THz of spectrum was not possible due to the impossibility to generate the problem, even when a large working memory of 3072GB and few TB of disk space were tuned. Therefore, we choose the heuristic approach to solve the VINCI problem for realistic instances.

In the next section, an exhaustive analysis of the heuristic is done in order to find the best configuration and its performance.

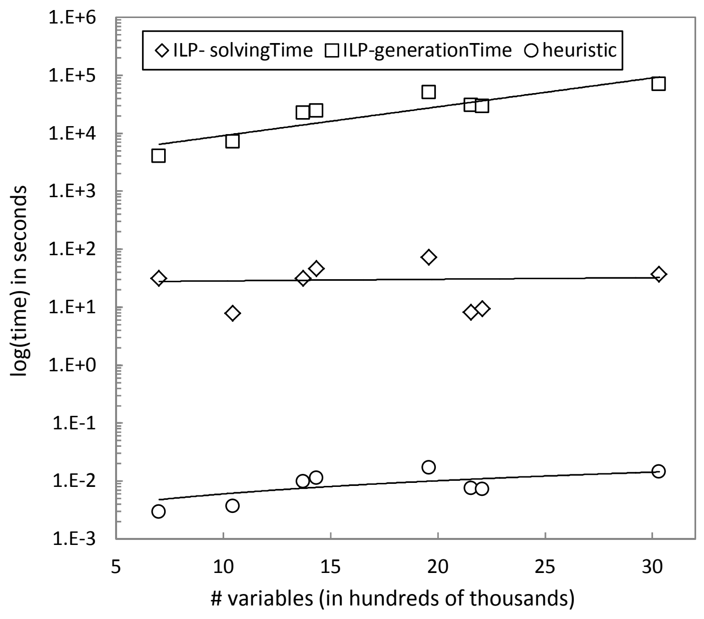


Figure 6-4: Execution time comparison: ILP vs. heuristic.

Table 6-2: Results of sub-optimal instances after 10 hours

# variables	# constraints	ILP			heuristic		
		optimality gap	**obj. func.**	**# new links**	**obj. func.**	**# new links**	**solving time (sec)**
793804	263408	98.56%	80296	8	10034	1	0.013
1319065	346105	97.43%	90325	9	10034	1	0.015
672256	211264	0.04%	10034	1	10042	1	0.009

6.4 Heuristic performance

In this last part of the chapter, we make a deeper study of the heuristic with the aim of finding the optimal configuration. For this study, we used a spectrum width of 4 THz – 640 slices – to fit the real spectrum size of flexgrid optical networks. The number of instances generated to perform this numerical analysis was 4250, as a result of the multiple combinations of the set of parameters that define each instance, namely, topology, network load, sorting criterion, number of main and GRASP iterations (*maxMainIter, maxGraspIter*) and α. A table with part of the raw results presented in this section is available in Appendix B.

First of all, let us analyze the best configuration for the *α* parameter of the GRASP-based heuristic. To this aim, we have fixed *maxGraspIter* to 100. Moreover, in case of using the random-sorted criterion for failure scenarios, *maxMainIter* was set to 10. Figure 6-5 shows the average value objective function as a function of the *α* parameter separated into the TEST-based and DT-based topologies. Furthermore, Table 6-3 shows the optimal value of α observed for each topology and overall. In addition, we can see three measures of the relative solution gain in each case. The first represents the gain of using the optimal value of *α* instead of *α*=0 for the greedy-sorted criterion. The same applies for the second row, except that the comparison is done for the random-sorted criterion. The third row shows the relative solution gain when using an optimal α and the random-sorted criterion, when compared against using the greedy-sorted one.

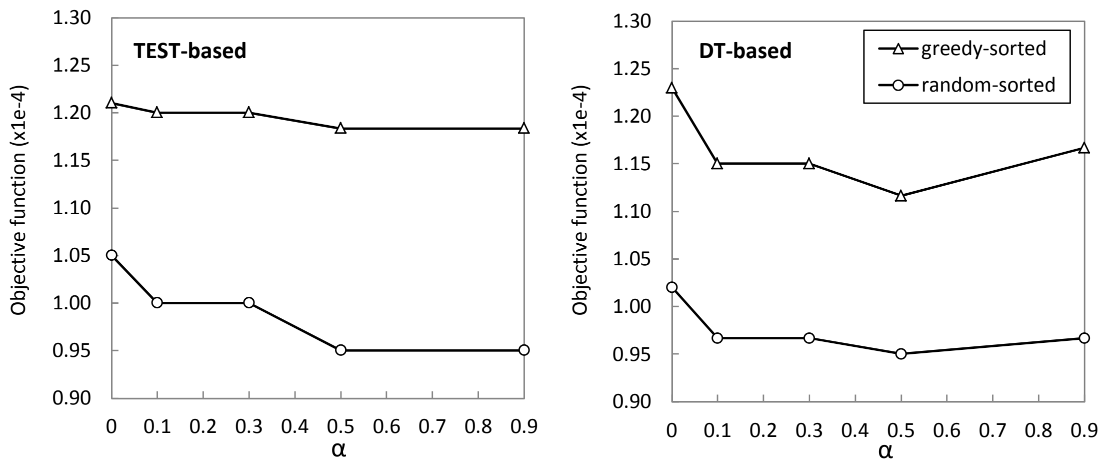


Figure 6-5: Objective function as a function of *α*

Table 6-3: α configuration and relative gains

	test	**DT**	**overall**
Optimum α (α^{opt})	>0.5	0.5	**0.5**
% gain (α^{opt}/ α=0) [greedy]	2.5%	8.9%	**4.9%**
% gain (α^{opt}/ α=0) [random]	9.5%	6.8%	**7.7%**
% gain (random/greedy) [α^{opt}]	19.5%	15.2%	**17.4%**

As can observed, the lowest objective function value is always reached at *α=0.5* for any network and sorting criterion. This characterizes strongly the best *α* parameter configuration for this heuristic. In fact, using this value improves the quality of the solution up to a 10% with respect to consider *α*=0. Note that the latter case will allow reducing the computational time since no GRASP iterations are necessary. Nevertheless and since we are dealing with a planning problem where execution time is not strongly restricted, it is preferable to use this optimal α and increase the total computational time in the hope of finding a significantly better solution.

Moreover, it is worth noting that the random-sorted criterion highly improves the greedy-sorted one (near to 20% of gain).

Once the best *α* parameter have been found, let us analyze how the heuristic converges to the best solution as a function of the execution time. Figure 6-6 shows the on-average incumbent objective function along the execution time when *α*=0.5 is selected. It is clear that the heuristic quickly converges to a solution close to the best obtained one. For example, for the DT-based topologies and random-sorted criterion, a solution only 5% far from the best found solution is obtained in only 0.25 seconds, which is approximately the 20% of the total execution time. Note that the random-sorted criterion always produces a better solution than the greedy-one for any given execution time. This result allows us finally selecting the random-sorted criterion as the best choice.

In view of Figure 6-6, it is clear that we can increase the number of GRASP iterations to increase the execution time and increase the probability of obtaining an even better solution. In this regard, since the heuristic took up to 3 seconds and considering that we could have up to few hours to solve the problem in real life, we could highly increase this couple of parameters to try to improve the obtained solution as much as possible.

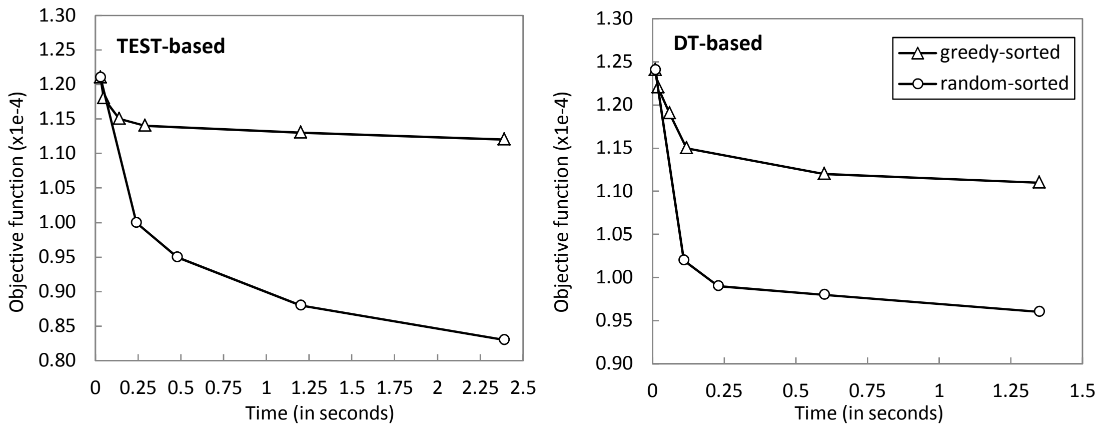


Figure 6-6: Objective function as a function of execution time

After selecting the configuration consisting in the random-sorted criterion with *α*=0.5, we analyze whether the selected value for *maxMainIter* (fixed to 10) was large enough. Figure 6-7 shows the average and maximum number of main algorithm iterations needed to found the incumbent solution for the DT-based instances. Obviously, as the amount of vulnerability scenarios increases, it requires more iterations of the main algorithm to find the incumbent. Although, in average, around 7 iterations are needed to solve instances with 6 vulnerable links, the maximum reaches 10 iterations. Therefore, for instances with higher or equal number of scenarios, a higher number of main iteration should be

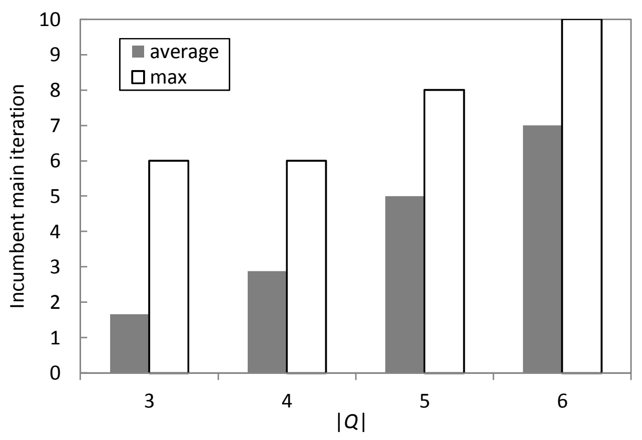


Figure 6-7: Incumbent main iteration analysis for DT-based instances

necessary. However, as argued before for the number of GRASP iterations, the number of main iterations could be extended to ensure reaching the incumbent solution in a main algorithm iteration farther from the fixed limit.

Finally and after a detailed analysis on the solved instances, we realized that some of the best obtained solutions return a topology adding no extra links. This is, the restoration algorithm used to detect vulnerabilities found instances with vulnerable links that, actually, could be successfully restored if the problem of restoring demands after a failure was better solved. Recall that a stringent time limitation is imposed to find a solution for such restoration algorithm and, consequently, better restoration paths could be found. This behavior is depicted in Figure 6-8, where the amount of problem instances with the need of installing, at least, one new link is depicted as a function of the time to solve the VINCI problem. The execution time is normalized to the time invested for the restoration algorithm to detect vulnerabilities, being 1 this mentioned time.

What can be concluded from Figure 6-8 is that when investing a similar (or moderately higher) execution time for solving VINCI than the time for vulnerabilities detection, the heuristic returns new links in the 100% of the tested instances. Nonetheless, when the computational effort to solve VINCI increases, not only better solutions are obtained (as seen in previous figures) but also instances that do not really need additional capacity resources rise. Concretely, when the computational effort increases 10 times, only the 66% of instances with one vulnerable link need of a capacity extension. For instances with 2 vulnerable links, it is necessary to invest 40 times more than the detection algorithm to decrease the number of instances with increased capacity to a 71%.

In light of this behavior, it is clear that a more complex heuristic for the planning process, although providing better solutions, would imply a larger difference with the restoration algorithm. This is the main reason we do not go further in adding intensification mechanisms (e.g. a local search phase) to the heuristic.

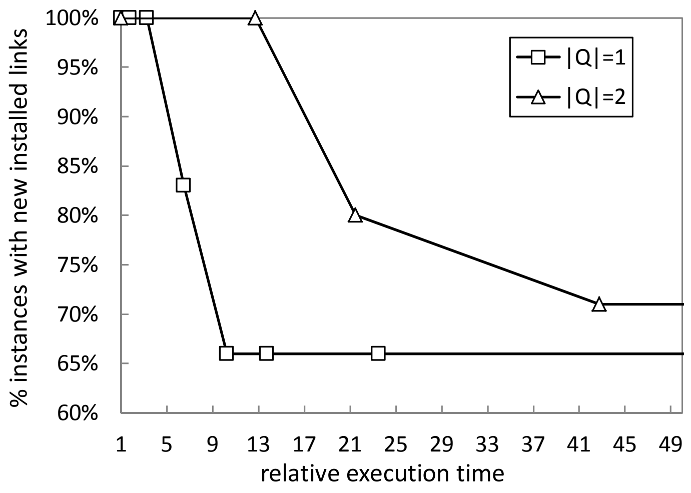


Figure 6-8: % of instances with new links vs time for DT-based topologies.

6.5 Summary

In this chapter we provided numerical evidence to evaluate the performance of both ILP and heuristic methods. A first comparative analysis allows highlighting differences in orders of magnitude between solving times. In fact, the heuristic approach reached the optimal solution in execution times up to 7 orders of magnitude lower than the total time needed to generate and solve the ILP.

Once the heuristic has been shown as the most affordable method to solve VINCI, a fine tuning has been provided by means of solving and analyzing a large set of heterogeneous instances. Results illustrated that the random-sorted criterion with α=0.5 and a large number of iterations is the best configuration. The time needed to solve the larger instances with this configuration needs no more than 5 seconds. Therefore, there is enough room to increase the computational effort since execution time could reach several hours (if necessary) in a real environment in the hope of finding even better solutions.

As a final conclusion, one can easily state that improving the restoration algorithm will lead to invoking the VINCI problem less number of times. This could be done in case of improving the restoration algorithm without surpassing the stringent time limitations of dynamic restoration. On the contrary, we may have to ensure solutions that the restoration algorithm could always detect above the restorability threshold by adding additional constraints or post-processing obtained solutions before returning them. This situation shall be studied in the future.

Chapter 7.

Concluding Remarks

7.1 Contributions and work impact

The main contribution of this work is the study of the problem of incremental capacity planning for vulnerability minimization. The so-called VINCI problem has been stated, formulated, and solved for the first time in this project. In order to solve the VINCI problem, an ILP formulation and a heuristic have been designed, implemented, and tested by means of a wide set of instances.

From the results, the heuristic method has been proven as a fast and efficient way to obtain near-optimal solutions for the VINCI problem. Note that the heuristic was implemented as a new module in a multi-purpose flexgrid optical network simulator implementing standard recommendations. In this way, we ensure the real applicability of this method for solving this new use case on incremental capacity planning for networks in operation.

This work has been developed as part of the research work of the Optical Communications Group (GCO) at the UPC. In addition to the extension of its network simulator by adding the aforementioned module for solving the VINCI problem (thus enriching the capabilities of such simulator), part of the methodology and results in this project will be shortly included in a conference paper and hopefully extended to an indexed journal paper.

7.2 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 studied the RSA algorithm as a first step to understand the idea behind the VINCI problem. After that, I studied the inventory structure and applied some mathematical modelling ideas present in some set covering and traffic problems learned in my career. I am thankful to my advisors for their enlightening tips that helped me understand the multi scenario structure present in the ILP formulation. This all helped me to properly formulate the VINCI problem introduced in this work.

After the ILP formulation work, I was introduced into OMNeT++ so as to implement the VINCI heuristic. I am glad to have had the inexhaustible source of answers from my advisors and people from the GCO, without those I could have never finished the heuristic implementation. I have improved my programming and modeling skills a lot since I started this work with them.

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.

I also attended an internal workshop, where I could devise the almost endless amount or possibilities that the optical networks field offer for operations research and programming. I was surprised to find such an interesting source of problems that suits so well for a mathematician. I would like to highlight as well the constant and significant dedication that this project requires and the enormous effort that it meant as my first step in research.

Last but not least, I cannot end without giving thanks to my family and friends. Their constant love and support are the main reasons behind my willingness to work; without them I could not have finished this work.

7.3 Future Work

As denoted in Chapter 6, the comparison between ILP and heuristic should be extended to really validate the latter method. In addition, larger instances generated from other reference topologies with higher number of nodes and links should be conducted.

The conflict between the detection algorithm and the VINCI heuristic mentioned in Chapter 6 should be also analyzed in detail. Thus, an improvement of the restoration algorithm or the adaption of the VINCI problem to deal with scenarios with no need of extending the network could be possible lines to follow. We should keep in mind the stringent execution time requirements of the restoration algorithm if an improvement of this was intended.

Finally, solving the VINCI problem in a traffic evolution scenario, where the network is periodically extended in response to small and continuous traffic changes is a must. In this way, a practical application of the VINCI problem will be done. In this regard, the VINCI problem could be compared against other alternatives on incremental capacity planning, e.g. extending the network when the blocking probability reaches a given threshold.

Apendix A.   Implemented Code

```
function REST_model(Data, CPLEX_Params)

timeMountIni = clock;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%                          Routing data                            %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

nN = Data.('nN'); %% Number of nodes.
nE = Data.('nE'); %% Number of links (active and inactive).
nD = Data.('nD'); %% Number of demands.
nC = Data.('nC'); %% Number of slots (links).
nS = Data.('nS'); %% Number of slices.
nQ = Data.('nQ'); %% Number of failure scenarios.

MatNE    = Data.('MatNE');   %% Matrix nN x nE: incidences between nodes-
links.
MatD     = Data.('MatD');    %% Matrix nD x  3: demands MatD[i] = [source,
destination, bitrate, xpos, ypos] .
MatDC    = Data.('MatDC');   %% Matrix nD x nC: slots tat demand d can use.
MatCS    = Data.('MatCS');   %% Matrix nC x nS: slices that are part of each
slot c.
MatQD    = Data.('MatQD');   %% Matrix nQ x nD: affected demands from each
scenario
MatF     = Data.('MatF');    %% Matrix  1 x nE: vector of inactive links.
MatQEC   = Data.('MatQEC');  %% Matrix nQ x nE x nC: available slots per link
for re-allocation in failure scenario q.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%                           Inventory data
%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

nT = Data.('nT'); %% Number of available linecards.
nR = Data.('nR'); %% Nuber of slots (linecard slots; active and inactive);

MatNR    = Data.('MatNR');   %% Matrix nN x nR: slots (linecards) de cada
nodo.
MatRT    = Data.('MatRT');   %% Matrix nR x nT: compatibilities/costs between
nodes and linecards (0 incompatible; cost otherwise).
MatCostE = Data.('MatCostE');%% Matrix  1 x nE: vector of link activation
costs.
MatA     = Data.('MatA');    %% Matrix  1 x nE: vector of link lengths.
MatB     = Data.('MatB');    %% Matrix  1 x nT: vector of linecard reaches.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%                               General data
%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Restorabilty threshold
R_th = Data.('R_th');
net_params(5) = R_th;

TILIM=CPLEX_Params.('TILIM');
GAP=CPLEX_Params.('GAP');

end

%*********************** Obj. function ************************%

fprintf('Adding objective function\n');
tic
% (1)
cplex = Cplex();
cplex.Model.sense = 'minimize';
objFunc = zeros(nvar,1);
%% Only inactive links
for e=1:nE
      if MatF(1,e) == 0
            objFunc(z(e)) = MatCostE(1,e);
      end
end
for n=1:nN
      for r=1:nR
            if MatNR(n,r) == 0
                  continue;
            end
            for t=1:nT
                  objFunc(y(n,r,t)) = MatRT(r,t);
            end
      end
end
toc

%*********************** Variables **************************%

fprintf('Adding variables\n');
tic
lb = [zeros(nvar,1)];
ub = [ones(nvar,1)];
ctypes = char(ones(1,nvar)*('B'));
cplex.addCols(objFunc, [], lb, ub, ctypes);
toc

%************************ Constraints **********************%

constraints = 0;

fprintf('Adding (2) and (3) \n');
tic
% (2) y (3)
for q=1:nQ
      for d=1:nD
            if MatQD(q,d) == 0
                  continue;
            end
            origin = MatD(d,1);
            target = MatD(d,2);
            for n=1:nN
                  row = zeros(1,nvar);
                  if n == origin || n == target
                        row(w(q,d)) = 1;
                  else
                        row(w(q,d)) = 2;
                  end
                  for e=1:nE
                        for c=1:nC
                              row(x(q,d,e,c)) =
MatNE(n,e)*MatDC(d,c)*MatQEC(q,e,c);
                        end
                  end
                  if n == origin || n == target
                        cplex.addRows(1, row, 1);
                  else
                        cplex.addRows(0, row, 2);
                  end
                  constraints = constraints + 1;
            end
      end
end
toc
fprintf('Adding (4)\n');
tic
% (4)
for q=1:nQ
      for d=1:nD
            if MatQD(q,d) == 0
                  continue;
            end
            for c=1:nC
                  if MatDC(d,c) == 0
                        continue;
                  end
                  origin = MatD(d,1);
                  target = MatD(d,2);
                  for n=1:nN
                        if n == origin || n == target
                              continue;
                        end
                        for e=1:nE
                              if MatNE(n,e) == 0
                                    continue;
                              end
                              row = zeros(1,nvar);
                              row(x(q,d,e,c)) = MatQEC(q,e,c);
                              lb = 0;
                              for e1=1:nE
                                    if e1 == e
                                          continue;
                                    end
                                    k = MatNE(n,e1)*MatQEC(q,e1,c);
                                    row(x(q,d,e1,c)) = -k;
                                    lb = lb - k;
                              end
                              cplex.addRows(lb, row, 0);
                              constraints = constraints + 1;
                        end
                  end
            end
```

```
        end
end
toc
fprintf('Adding (5)\n');
tic
% (5)
for q=1:nQ
        for e=1:nE
                for s=1:nS
                        row = zeros(1,nvar);
                        row(z(e)) = -1;
                        for d=1:nD
                                if MatQD(q,d) == 0
                                        continue;
                                end
                                for c=1:nC
                                        row(x(q,d,e,c)) = MatDC(d,c)*MatCS(c,s);
                                end
                        end
                        cplex.addRows(-1, row, 0);
                        constraints = constraints + 1;
                end
        end
end
toc
fprintf('Adding (6)\n');
tic
% (6)
for e=1:nE
        row = zeros(1,nvar);
        row(z(e)) = 1;
        cplex.addRows(MatF(1,e), row, 1);
        constraints = constraints + 1;
end
toc
fprintf('Adding (7)\n');
tic
% (7)
for q=1:nQ
        row = zeros(1,nvar);
        ub = 0;
        for d=1:nD
                if MatQD(q,d) == 0
                        continue;
                end
                row(w(q,d)) = 1;
        end
        cplex.addRows(0, row, (1-R_th)*sum(MatQD(q,:)));
        constraints = constraints + 1;
end
toc
fprintf('Adding (8)\n');
tic
% (8)
for n=1:nN
        for e=1:nE
                if MatNE(n,e) == 0 || MatF(1,e) == 1
                        continue;
                end
                row = zeros(1,nvar);
                row(z(e)) = -1;
                for r=1:nR
```

```
                    if MatNR(n,r) == 0
                          continue;
                    end
                    row(u(n,e,r)) = 1;
              end
              cplex.addRows(0, row, 0);
              constraints = constraints + 1;
       end
end
toc
fprintf('Adding (9), (10) and (12)\n');
tic
% (9), (10) y (12)
for n=1:nN
       for r=1:nR
              if MatNR(n,r) == 0
                    continue
              end
              % Constraint 9
              row = zeros(1,nvar);
              row(v(n,r)) = -1;
              for e=1:nE
                    if MatNE(n,e) == 0
                          continue
                    end
                    row(u(n,e,r)) = 1;
              end
              cplex.addRows(0, row, 0);
              % Constraint 10
              row = zeros(1,nvar);
              row(v(n,r)) = -1;
              for t=1:nT
                    if MatRT(r,t) ~= 0
                          row(y(n,r,t)) = 1;
                    end
              end
              cplex.addRows(0, row, 0);
              % Constraint 12
              row = zeros(1,nvar);
              lb = 0;
              for e=1:nE
                    if MatNE(n,e) == 0
                          continue;
                    end
                    row(u(n,e,r)) = MatA(1,e);
              end
              for t=1:nT
                    row(y(n,r,t)) = -MatB(1,t);
                    lb = lb - MatB(1,t);
              end
              cplex.addRows(lb, row, 0);
              % 3 more constraints
              constraints = constraints + 3;
       end
end
toc
fprintf('Adding (11)\n');
tic
% (11)
for t=1:nT
       row = zeros(1,nvar);
       for n=1:nN
```

```
            for r=1:nR
                  if MatNR(n,r) == 0 || MatRT(r,t) == 0
                        continue;
                  end
                  row(y(n,r,t)) = 1;
            end
      end
      cplex.addRows(0, row, 1);
      constraints = constraints + 1;
end
toc

%********************** Solve ****************************%

fprintf ('\nSolving problem...\n');

timeMount = etime(clock, timeMountIni);
timeSolveIni=clock;
solve(cplex);
timeSolve=etime(clock, timeSolveIni);
disp(cplex.Solution.statusstring);

try
    printSolution(cplex.Solution.objval,cplex.Solution.x,timeSolve,timeMount);
catch exception
      fprintf ('Exception found.\n');
end

function solve(cplex)
      % Solve
      fprintf ('Problem mounting:\n%i variables and %i
constraints.\n\n',nvar,constraints);
      size(cplex.Model.A)
      cplex.Param.mip.strategy.file.Cur=2;
      cplex.Param.workmem.Cur=3072;
      cplex.Param.threads.Cur=7;
      % Set InfoCallback
      cplex.InfoCallback.func = @stopCB;
      cplex.InfoCallback.data.tilim = 36000;
      cplex.InfoCallback.data.MipGap = 0.001;
      cplex.InfoCallback.data.timeSolveIni = clock;
      cplex.solve();
end

%******************** Solution ***********************%

function printSolution(objVal,X,timeSolve,timeMount)

      % Generate output data...

end

% Callback function
function stop = stopCB(info, data)
      if (~isempty (info.IncObj))
            currentGap = abs(info.MipGap);
            elapsedTime=etime(clock, data.timeSolveIni);
            stop = (currentGap < data.MipGap) || (elapsedTime > data.tilim);
      end
end

end
```

Apendix B.   Heuristic Results

This is an extract of 200 instances that belong to the set of 4250 instances used for the heuristic performance analysis. They correspond to the TEST_8_12 network using the random sorting criterion, and a limit of 10 and 100 GRASP iterations.

Legend:

- *network:* network topology used.
- *vul:* number of failure scenarios.
- *sort:* sorting criterion used – greedy or random.
- *alpha: GRASP alpha parameter.*
- *maxGraspIter: GRASP number of iterations.*
- *maxMainIter: main number of iterations.*
- *obj_total: objective function total cost.*
- *obj_link: partial obj. function cost – link activation.*
- *obj_cards: partial obj. function cost – linecard installation.*
- *inst: number of new links installed into the topology.*
- *total_time:* heuristic computation time.
- *inc_time:* time when the incumbent is found.
- *inc_iter:* iteration where the incumbent is found.

network	vul	sort	alpha	max Grasp Iter	max Main Iter	obj_total	obj link	obj cards	inst	total_time	inc_time	inc_iter
test_8_12	2	random	0	10	10	10034.1	10014.1	20	1	0.0441615	0.00442293	1
test_8_12	6	random	0	10	10	10040	10020	20	1	0.139862	0.0140392	1
test_8_12	4	random	0	10	10	10034.1	10014.1	20	1	0.0952769	0.00967251	1

test_8_12	7	random	0	10	10	20076.5	20036.5	40	2	0.136688	0.0410888	3
test_8_12	2	random	0	10	10	10034.1	10014.1	20	1	0.0355269	0.00357675	1
test_8_12	6	random	0	10	10	20068.2	20028.2	40	2	0.128279	0.0259998	2
test_8_12	2	random	0	10	10	10034.1	10014.1	20	1	0.040137	0.00403858	1
test_8_12	2	random	0	10	10	10034.1	10014.1	20	1	0.0361154	0.00723331	2
test_8_12	4	random	0	10	10	10034.1	10014.1	20	1	0.0979682	0.0196969	2
test_8_12	3	random	0	10	10	10034.1	10014.1	20	1	0.0554684	0.00561054	1
test_8_12	1	random	0	10	10	10034.1	10014.1	20	1	0.0165859	0.00165118	1
test_8_12	2	random	0	10	10	10034.1	10014.1	20	1	0.0435129	0.00440017	1
test_8_12	2	random	0	10	10	10034.1	10014.1	20	1	0.0286144	0.00285567	1
test_8_12	2	random	0	10	10	10034.1	10014.1	20	1	0.0274832	0.0109691	4
test_8_12	0	random	0	10	10	0	0	0	0	6.70758e-05	1.83371e-06	1
test_8_12	2	random	0	10	10	10034.1	10014.1	20	1	0.0314793	0.00314457	1
test_8_12	2	random	0	10	10	10034.1	10014.1	20	1	0.0293009	0.00586164	2
test_8_12	2	random	0	10	10	10034.1	10014.1	20	1	0.0363507	0.0109294	3
test_8_12	1	random	0	10	10	10034.1	10014.1	20	1	0.0126428	0.00126839	1
test_8_12	0	random	0	10	10	0	0	0	0	6.80506e-05	1.75104e-06	1
test_8_12	2	random	0.1	10	10	10034.1	10014.1	20	1	0.0441942	0.00443016	1
test_8_12	6	random	0.1	10	10	10040	10020	20	1	0.140137	0.0140439	1
test_8_12	4	random	0.1	10	10	10034.1	10014.1	20	1	0.0954828	0.00966109	1
test_8_12	7	random	0.1	10	10	20076.5	20036.5	40	2	0.137069	0.0410083	3
test_8_12	2	random	0.1	10	10	10034.1	10014.1	20	1	0.0352809	0.0035737	1
test_8_12	6	random	0.1	10	10	20068.2	20028.2	40	2	0.128615	0.0259671	2
test_8_12	2	random	0.1	10	10	10034.1	10014.1	20	1	0.0399914	0.00405182	1
test_8_12	2	random	0.1	10	10	10034.1	10014.1	20	1	0.0360609	0.00722367	2
test_8_12	4	random	0.1	10	10	10034.1	10014.1	20	1	0.0983963	0.0197396	2
test_8_12	3	random	0.1	10	10	10034.1	10014.1	20	1	0.0552594	0.00559116	1
test_8_12	1	random	0.1	10	10	10034.1	10014.1	20	1	0.0165941	0.00165325	1
test_8_12	2	random	0.1	10	10	10034.1	10014.1	20	1	0.0436468	0.00439975	1
test_8_12	2	random	0.1	10	10	10034.1	10014.1	20	1	0.0286457	0.0028576	1

test_8_12	2	random	0.1	10	10	10034.1	10014.1	20	1	0.0273236	0.0109445	4
test_8_12	0	random	0.1	10	10	0	0	0	0	6.78057e-05	1.89462e-06	1
test_8_12	2	random	0.1	10	10	10034.1	10014.1	20	1	0.0314277	0.00314667	1
test_8_12	2	random	0.1	10	10	10034.1	10014.1	20	1	0.0293377	0.00586709	2
test_8_12	2	random	0.1	10	10	10034.1	10014.1	20	1	0.0365298	0.0109439	3
test_8_12	1	random	0.1	10	10	10034.1	10014.1	20	1	0.0126391	0.00126562	1
test_8_12	0	random	0.1	10	10	0	0	0	0	6.78976e-05	1.77343e-06	1
test_8_12	2	random	0.3	10	10	10034.1	10014.1	20	1	0.0440607	0.00442869	1
test_8_12	6	random	0.3	10	10	10040	10020	20	1	0.140457	0.0140199	1
test_8_12	4	random	0.3	10	10	10034.1	10014.1	20	1	0.0955061	0.00966703	1
test_8_12	7	random	0.3	10	10	20076.5	20036.5	40	2	0.136237	0.040999	3
test_8_12	2	random	0.3	10	10	10034.1	10014.1	20	1	0.0354262	0.0035687	1
test_8_12	6	random	0.3	10	10	20068.2	20028.2	40	2	0.12808	0.0260056	2
test_8_12	2	random	0.3	10	10	10034.1	10014.1	20	1	0.0401122	0.0040407	1
test_8_12	2	random	0.3	10	10	10034.1	10014.1	20	1	0.0359176	0.00722321	2
test_8_12	4	random	0.3	10	10	10034.1	10014.1	20	1	0.0983744	0.0197076	2
test_8_12	3	random	0.3	10	10	10034.1	10014.1	20	1	0.055221	0.0055989	1
test_8_12	1	random	0.3	10	10	10034.1	10014.1	20	1	0.0165873	0.00165244	1
test_8_12	2	random	0.3	10	10	10034.1	10014.1	20	1	0.0437647	0.00440755	1
test_8_12	2	random	0.3	10	10	10034.1	10014.1	20	1	0.028648	0.00285683	1
test_8_12	2	random	0.3	10	10	10034.1	10014.1	20	1	0.0273545	0.0109486	4
test_8_12	0	random	0.3	10	10	0	0	0	0	6.96171e-05	1.82982e-06	1
test_8_12	2	random	0.3	10	10	10034.1	10014.1	20	1	0.0313539	0.00314154	1
test_8_12	2	random	0.3	10	10	10034.1	10014.1	20	1	0.0292734	0.00585395	2
test_8_12	2	random	0.3	10	10	10034.1	10014.1	20	1	0.0365556	0.0109461	3
test_8_12	1	random	0.3	10	10	10034.1	10014.1	20	1	0.0126437	0.00126605	1
test_8_12	0	random	0.3	10	10	0	0	0	0	6.8574e-05	1.76973e-06	1
test_8_12	2	random	0.5	10	10	10034.1	10014.1	20	1	0.0432272	0.00433171	1
test_8_12	6	random	0.5	10	10	10040	10020	20	1	0.137229	0.0136317	1
test_8_12	4	random	0.5	10	10	10034.1	10014.1	20	1	0.0924644	0.00927845	1

test_8_12	7	random	0.5	10	10	20076.5	20036.5	40	2	0.13445	0.0402372	3
test_8_12	2	random	0.5	10	10	10034.1	10014.1	20	1	0.0349051	0.00353573	1
test_8_12	6	random	0.5	10	10	20068.2	20028.2	40	2	0.124763	0.0253863	2
test_8_12	2	random	0.5	10	10	10034.1	10014.1	20	1	0.0394038	0.00396255	1
test_8_12	2	random	0.5	10	10	10034.1	10014.1	20	1	0.0356056	0.00712634	2
test_8_12	4	random	0.5	10	10	10034.1	10014.1	20	1	0.0953593	0.0191788	2
test_8_12	3	random	0.5	10	10	10034.1	10014.1	20	1	0.0542584	0.00555285	1
test_8_12	1	random	0.5	10	10	10034.1	10014.1	20	1	0.0159952	0.00159386	1
test_8_12	2	random	0.5	10	10	10034.1	10014.1	20	1	0.0427739	0.00433105	1
test_8_12	2	random	0.5	10	10	10034.1	10014.1	20	1	0.027737	0.00276594	1
test_8_12	2	random	0.5	10	10	10034.1	10014.1	20	1	0.0271019	0.0108288	4
test_8_12	0	random	0.5	10	10	0	0	0	0	6.87541e-05	1.83802e-06	1
test_8_12	2	random	0.5	10	10	10034.1	10014.1	20	1	0.0309073	0.00308819	1
test_8_12	2	random	0.5	10	10	10034.1	10014.1	20	1	0.0288713	0.00578595	2
test_8_12	2	random	0.5	10	10	10034.1	10014.1	20	1	0.0358698	0.010771	3
test_8_12	1	random	0.5	10	10	10034.1	10014.1	20	1	0.0121807	0.00122402	1
test_8_12	0	random	0.5	10	10	0	0	0	0	6.74911e-05	1.8096e-06	1
test_8_12	2	random	0.9	10	10	10034.1	10014.1	20	1	0.0432923	0.00433269	1
test_8_12	6	random	0.9	10	10	10040	10020	20	1	0.137593	0.0136669	1
test_8_12	4	random	0.9	10	10	10034.1	10014.1	20	1	0.0929842	0.00927437	1
test_8_12	7	random	0.9	10	10	20076.5	20036.5	40	2	0.133919	0.0400376	3
test_8_12	2	random	0.9	10	10	10034.1	10014.1	20	1	0.0348918	0.00353283	1
test_8_12	6	random	0.9	10	10	20068.2	20028.2	40	2	0.124305	0.0252269	2
test_8_12	2	random	0.9	10	10	10034.1	10014.1	20	1	0.0394111	0.0039677	1
test_8_12	2	random	0.9	10	10	10034.1	10014.1	20	1	0.0355892	0.007122	2
test_8_12	4	random	0.9	10	10	10034.1	10014.1	20	1	0.0958436	0.0191801	2
test_8_12	3	random	0.9	10	10	10034.1	10014.1	20	1	0.0539511	0.00554781	1
test_8_12	1	random	0.9	10	10	10034.1	10014.1	20	1	0.0159929	0.00160037	1
test_8_12	2	random	0.9	10	10	10034.1	10014.1	20	1	0.0428703	0.00433142	1
test_8_12	2	random	0.9	10	10	10034.1	10014.1	20	1	0.0277881	0.00277321	1

test_8_12	2	random	0.9	10	10	10034.1	10014.1	20	1	0.0270324	0.0107961	4
test_8_12	0	random	0.9	10	10	0	0	0	0	6.76029e-05	1.83007e-06	1
test_8_12	2	random	0.9	10	10	10034.1	10014.1	20	1	0.0306867	0.00307783	1
test_8_12	2	random	0.9	10	10	10034.1	10014.1	20	1	0.0288521	0.00578071	2
test_8_12	2	random	0.9	10	10	10034.1	10014.1	20	1	0.0359659	0.010778	3
test_8_12	1	random	0.9	10	10	10034.1	10014.1	20	1	0.0121864	0.00122325	1
test_8_12	0	random	0.9	10	10	0	0	0	0	7.26902e-05	1.77028e-06	1
test_8_12	2	random	0	100	10	10034.1	10014.1	20	1	0.435242	0.0440488	1
test_8_12	6	random	0	100	10	10040	10020	20	1	1.39467	0.139611	1
test_8_12	4	random	0	100	10	10034.1	10014.1	20	1	0.944083	0.0959224	1
test_8_12	7	random	0	100	10	20076.5	20036.5	40	2	1.35368	0.40466	3
test_8_12	2	random	0	100	10	10034.1	10014.1	20	1	0.349562	0.0356157	1
test_8_12	6	random	0	100	10	10040.6	10020.6	20	1	1.27527	0.889564	7
test_8_12	2	random	0	100	10	10034.1	10014.1	20	1	0.395433	0.0401994	1
test_8_12	2	random	0	100	10	10034.1	10014.1	20	1	0.355462	0.0715703	2
test_8_12	4	random	0	100	10	10034.1	10014.1	20	1	0.97208	0.194323	2
test_8_12	3	random	0	100	10	10034.1	10014.1	20	1	0.546289	0.0553993	1
test_8_12	1	random	0	100	10	10034.1	10014.1	20	1	0.164191	0.016487	1
test_8_12	3	random	0	100	10	10034.1	10014.1	20	1	0.540435	0.0557147	1
test_8_12	2	random	0	100	10	10034.1	10014.1	20	1	0.282657	0.0284657	1
test_8_12	2	random	0	100	10	10034.1	10014.1	20	1	0.269548	0.108588	4
test_8_12	1	random	0	100	10	10034.1	10014.1	20	1	0.177099	0.0178525	1
test_8_12	2	random	0	100	10	10034.1	10014.1	20	1	0.309185	0.0310144	1
test_8_12	2	random	0	100	10	10034.1	10014.1	20	1	0.289154	0.0584302	2
test_8_12	2	random	0	100	10	10034.1	10014.1	20	1	0.359276	0.108123	3
test_8_12	2	random	0	100	10	10034.1	10014.1	20	1	0.263236	0.0264003	1
test_8_12	0	random	0	100	10	0	0	0	0	6.8571e-05	1.76229e-06	1
test_8_12	2	random	0.1	100	10	10034.1	10014.1	20	1	0.434758	0.0440245	1
test_8_12	6	random	0.1	100	10	10040	10020	20	1	1.38931	0.138688	1
test_8_12	4	random	0.1	100	10	10034.1	10014.1	20	1	0.945944	0.0961351	1

test_8_12	7	random	0.1	100	10	20076.5	20036.5	40	2	1.35627	0.406255	3
test_8_12	2	random	0.1	100	10	10034.1	10014.1	20	1	0.349099	0.0353503	1
test_8_12	6	random	0.1	100	10	10040.6	10020.6	20	1	1.28148	0.893932	7
test_8_12	2	random	0.1	100	10	10034.1	10014.1	20	1	0.395439	0.0402322	1
test_8_12	2	random	0.1	100	10	10034.1	10014.1	20	1	0.354707	0.071264	2
test_8_12	4	random	0.1	100	10	10034.1	10014.1	20	1	0.973922	0.195421	2
test_8_12	3	random	0.1	100	10	10034.1	10014.1	20	1	0.546146	0.0552883	1
test_8_12	1	random	0.1	100	10	10034.1	10014.1	20	1	0.163881	0.0164771	1
test_8_12	3	random	0.1	100	10	10034.1	10014.1	20	1	0.540582	0.0557768	1
test_8_12	2	random	0.1	100	10	10034.1	10014.1	20	1	0.282306	0.0284816	1
test_8_12	2	random	0.1	100	10	10034.1	10014.1	20	1	0.269113	0.10856	4
test_8_12	1	random	0.1	100	10	10034.1	10014.1	20	1	0.176731	0.0177859	1
test_8_12	2	random	0.1	100	10	10034.1	10014.1	20	1	0.30954	0.0312012	1
test_8_12	2	random	0.1	100	10	10034.1	10014.1	20	1	0.289422	0.0585774	2
test_8_12	2	random	0.1	100	10	10034.1	10014.1	20	1	0.359918	0.108235	3
test_8_12	2	random	0.1	100	10	10034.1	10014.1	20	1	0.26342	0.0264569	1
test_8_12	0	random	0.1	100	10	0	0	0	0	6.80404e-05	1.77065e-06	1
test_8_12	2	random	0.3	100	10	10034.1	10014.1	20	1	0.434771	0.0440019	1
test_8_12	6	random	0.3	100	10	10040	10020	20	1	1.38793	0.138712	1
test_8_12	4	random	0.3	100	10	10034.1	10014.1	20	1	0.942998	0.0957601	1
test_8_12	7	random	0.3	100	10	20076.5	20036.5	40	2	1.36245	0.407465	3
test_8_12	2	random	0.3	100	10	10034.1	10014.1	20	1	0.349953	0.0354583	1
test_8_12	6	random	0.3	100	10	10040.6	10020.6	20	1	1.27575	0.889904	7
test_8_12	2	random	0.3	100	10	10034.1	10014.1	20	1	0.394832	0.0401767	1
test_8_12	2	random	0.3	100	10	10034.1	10014.1	20	1	0.355496	0.0716043	2
test_8_12	4	random	0.3	100	10	10034.1	10014.1	20	1	0.973086	0.195243	2
test_8_12	3	random	0.3	100	10	10034.1	10014.1	20	1	0.546619	0.0552429	1
test_8_12	1	random	0.3	100	10	10034.1	10014.1	20	1	0.163879	0.0164846	1
test_8_12	3	random	0.3	100	10	10034.1	10014.1	20	1	0.540033	0.0557102	1
test_8_12	2	random	0.3	100	10	10034.1	10014.1	20	1	0.282398	0.0284751	1

test_8_12	2	random	0.3	100	10	10034.1	10014.1	20	1	0.269132	0.10848	4
test_8_12	1	random	0.3	100	10	10034.1	10014.1	20	1	0.177882	0.0179031	1
test_8_12	2	random	0.3	100	10	10034.1	10014.1	20	1	0.309932	0.0312397	1
test_8_12	2	random	0.3	100	10	10034.1	10014.1	20	1	0.288978	0.0583666	2
test_8_12	2	random	0.3	100	10	10034.1	10014.1	20	1	0.359575	0.108257	3
test_8_12	2	random	0.3	100	10	10034.1	10014.1	20	1	0.263975	0.0265527	1
test_8_12	0	random	0.3	100	10	0	0	0	0	6.78855e-05	1.81551e-06	1
test_8_12	2	random	0.5	100	10	10034.1	10014.1	20	1	0.427364	0.0431466	1
test_8_12	6	random	0.5	100	10	10040	10020	20	1	1.36081	0.134864	1
test_8_12	4	random	0.5	100	10	10034.1	10014.1	20	1	0.916878	0.092209	1
test_8_12	7	random	0.5	100	10	20076.5	20036.5	40	2	1.33138	0.39719	3
test_8_12	2	random	0.5	100	10	10034.1	10014.1	20	1	0.344586	0.0351241	1
test_8_12	6	random	0.5	100	10	10040.6	10020.6	20	1	1.24361	0.864471	7
test_8_12	2	random	0.5	100	10	10034.1	10014.1	20	1	0.391258	0.0398074	1
test_8_12	2	random	0.5	100	10	10034.1	10014.1	20	1	0.35145	0.070893	2
test_8_12	4	random	0.5	100	10	10034.1	10014.1	20	1	0.946952	0.189522	2
test_8_12	3	random	0.5	100	10	10034.1	10014.1	20	1	0.536941	0.0553131	1
test_8_12	1	random	0.5	100	10	10034.1	10014.1	20	1	0.15729	0.0158397	1
test_8_12	3	random	0.5	100	10	10034.1	10014.1	20	1	0.528837	0.0549859	1
test_8_12	2	random	0.5	100	10	10034.1	10014.1	20	1	0.274423	0.0275576	1
test_8_12	2	random	0.5	100	10	10034.1	10014.1	20	1	0.266642	0.107492	4
test_8_12	1	random	0.5	100	10	0	0	0	0	0.172144	0.0173095	1
test_8_12	2	random	0.5	100	10	10034.1	10014.1	20	1	0.304689	0.0306905	1
test_8_12	2	random	0.5	100	10	10034.1	10014.1	20	1	0.28533	0.0575247	2
test_8_12	2	random	0.5	100	10	10034.1	10014.1	20	1	0.354846	0.106746	3
test_8_12	2	random	0.5	100	10	10034.1	10014.1	20	1	0.25901	0.0259812	1
test_8_12	0	random	0.5	100	10	0	0	0	0	6.82604e-05	1.75813e-06	1
test_8_12	2	random	0.9	100	10	10034.1	10014.1	20	1	0.425888	0.0429666	1
test_8_12	6	random	0.9	100	10	10040	10020	20	1	1.36151	0.135162	1
test_8_12	4	random	0.9	100	10	10034.1	10014.1	20	1	0.917353	0.0921957	1

test_8_12	7	random	0.9	100	10	20076.5	20036.5	40	2	1.3325	0.39741	3
test_8_12	2	random	0.9	100	10	10034.1	10014.1	20	1	0.344644	0.0350975	1
test_8_12	6	random	0.9	100	10	10040.6	10020.6	20	1	1.24337	0.864428	7
test_8_12	2	random	0.9	100	10	10034.1	10014.1	20	1	0.3889	0.0396744	1
test_8_12	2	random	0.9	100	10	10034.1	10014.1	20	1	0.350156	0.0703363	2
test_8_12	4	random	0.9	100	10	10034.1	10014.1	20	1	0.948319	0.190147	2
test_8_12	3	random	0.9	100	10	10034.1	10014.1	20	1	0.536094	0.0552499	1
test_8_12	1	random	0.9	100	10	10034.1	10014.1	20	1	0.157554	0.0158454	1
test_8_12	3	random	0.9	100	10	10034.1	10014.1	20	1	0.528872	0.0546737	1
test_8_12	2	random	0.9	100	10	10034.1	10014.1	20	1	0.274299	0.027495	1
test_8_12	2	random	0.9	100	10	10034.1	10014.1	20	1	0.266504	0.107153	4
test_8_12	1	random	0.9	100	10	0	0	0	0	0.171738	0.0173217	1
test_8_12	2	random	0.9	100	10	10034.1	10014.1	20	1	0.30501	0.03084	1
test_8_12	2	random	0.9	100	10	10034.1	10014.1	20	1	0.284594	0.0575808	2
test_8_12	2	random	0.9	100	10	10034.1	10014.1	20	1	0.355124	0.107187	3
test_8_12	2	random	0.9	100	10	10034.1	10014.1	20	1	0.259302	0.0260236	1
test_8_12	0	random	0.9	100	10	0	0	0	0	6.75146e-05	1.78653e-06	1

Acronyms

CF	Central Frequency
CL	Candidate list
Gb/s	Gigabits per second
GCO	Global Concurrent Optimization module
GHz	Gigahertz
ILP	Integer linear programming problem
LSP	Label Switched Path
LSP	Label Switched Path
OS	Optical Spectrum
OXC	Optical Cross Connect
RCL	Restricted candidate list
RSA	Routing and Spectrum Allocation
TED	Traffic Engineering Database
THz	Terahertz
VINCI	Vulnerability-aware incremental capacity problem.

References

[As13] A. Asensio, A. Castro, L. Velasco, J. Comellas, "An Elastic Netwoks OMNET++ -based Simulator," in Proc. IEEE International Conference on Transparent Optical Networks (ICTON), 2013.

[Aw01] D. Awduche et al. "RSVP-TE: Extensions to RSVP for LSP Tunnels," IETF RFC 3209, 2001.

[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.

[Ca14] A. Castro, L. Velasco, J. Comellas, and G. Junyent, "On the benefits of Multi-path Recovery in Flexgrid Optical Networks," accepted in Springer Photonic Network Communications, 2014.

[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, May 2011.

[CISCO] CISCO blogs: "2014 Cisco VNI Forecast," https://blogs.cisco.com/

[Cr13] E. Crabbe, J. Medved, R. Varga, and I. Minei, "PCEP Extensions for Stateful PCE," IETF draft, Mar. 2013

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

[Fa06] A. Farrel, J.P. Vasseur, and J. Ash, "A Path Computation Element (PCE)- Based Architecture," IETF RFC 4655, 2006.

[Fe95] T. Feo and M. Resende, "Greedy Randomized Adaptive Search Procedures," Journal of Global Optimization, Vol. 51, pp.109-133, Jun. 1995.

[G964] Spectral grids for WDM applications: DWDM frequency grid. ITU-T G.694.1 (ed. 2.0) (2012)

[Gi14] Ll. Gifre, A. Castro, M. Ruiz, N. Navarro, and L. Velasco, "An In-

Operation Planning Tool Architecture for Flexgrid Network Re-Optimization," in Proc. IEEE International Conference on Transparent Optical Networks (ICTON), 2014.

[Gr03] Wayne Grover, "Mesh-based Survivable Transport Networks: Options and Strategies for Optical, MPLS, SONET and ATM Networking," Prentice Hall, 2003.

[Gu04] Maier, A. Pattavina, L. Barbato, F. Cecini and Mario Martinelli, "Routing Algorithms in WDM Networks under Mixed Static and Dynamic Lambda-Traffic", Photonic Network Communications, vol. 8, no. 1, pp. 69-87, 2004.

[ITU05] Telecom Development Bureau, Teletraffic Engineering Handbook, ITU, 2005.

[Ji09] M. Jinno, et al., "Spectrum-efficient and scalable elastic optical path network: architecture, benefits, and enabling technologies," IEEE Commun Mag., vol. 47, pp. 66-73, 2009.

[Ki13] D. King and A. Farrel, "A PCE-Based Architecture for Application-Based Network Operations," IETF draft, work in progress, 2013.

[Le09] Y. Lee, et al., "Path Computation Element Communication Protocol (PCEP) Requirements and Protocol Extensions in Support of Global Concurrent Optimization," IETF RFC 5557, 2009

[Li09] W. Liu, H. Sirisena, K. Pawlikowski, A. McInnes, "Utility of Algebraic Connectivity Metric in Topology Design of Survivable Networks," in Proc. DRCN, 2009.

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

[Ru14.1] M. Ruiz et al., "Planning Fixed to Flexgrid Gradual Migration: Drivers and Open Issues," IEEE Communications Magazine, vol. 52, pp. 70-76, 2014.

[Va09] JP. Vasseur, JL. Le Roux, "Path Computation Element (PCE) Communication Protocol (PCEP)," IETF RFC 5440, 2009

[Ve14.1] L. Velasco, D. King, O. Gerstel, R. Casellas, A. Castro, and V. López, "In-Operation Network Planning," *IEEE Communications Magazine*, vol. 52, pp. 52-60, 2014.

[Ve14.2] L. Velasco, A. Castro, M. Ruiz, and G. Junyent, "Solving Routing and Spectrum Allocation Related Optimization Problems: from Off-Line to In-Operation Flexgrid Network Planning," (Invited Tutorial) IEEE/OSA Journal of Lightwave Technology (JLT), vol. 32, pp. 2780-2795, 2014.

[Wa11] Y. Wang, X. Cao, and Y. Pan, “A study of the routing and spectrum allocation in spectrum-sliced elastic optical path networks,” in Proc. IEEE INFOCOM, 2011, pp. 1503–1511.