PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

# Tesis de Doctorado
## en Informática

**GRASP heuristics for Wide Area Network design**

Franco Robledo Amoza

Febrero 2005

**Tutor: Héctor Cancela (FI-UDELAR),**
**Gerardo Rubino (IRISA-INRIA)**

GRASP heuristics for Wide Area Network design
Franco Robledo Amoza

ISSN 0797-6410
Tesis de Doctorado en Informática
Reporte Técnico RT 05-01
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República.
Montevideo, Uruguay, febrero de 2005

# Resumen

Una red de área extendida (*Wide Area Network* - WAN) puede ser considerada como un conjunto de sitios interconectados por líneas de comunicación. Topologicaménte, una red WAN esta organizada en dos niveles: la Red Dorsal (*Backbone*) y la Red de Acceso (*Access Network*) compuesta de un cierto número de sub-redes de acceso locales. Cada sub-red de acceso local usualmente tiene topología de árbol, teniendo como raíz un nodo de la Red Dorsal (un sitio *switch*). Los sitios terminales (o clientes) se conectan directamente al sitio dorsal correspondiente a una sub-red de acceso o bien a un sitio concentrador de la misma. La Red Dorsal tiene usualmente topología de malla y su propósito es permitir comunicación eficiente y confiable entre los nodos de la Red Dorsal que actúan como puntos de entrada para las sub-redes de acceso locales.

En esta tesis atacamos el problema del diseño de una red WAN descomponiéndolo en dos sub-problemas interrelacionados: el diseño de la Red de Acceso (*the Access Network Design Problem* - ANDP) y el diseño de la Red Dorsal (*the Backbone Network Design Problem* - BNDP). En ambos modelos consideramos solamente costos de construcción, por ejemplo, los costos de dragado para el tendido de líneas y la puesta en servicio del cableado de la red.

Modelamos el ANDP como una variante del Problema de Steiner en Grafos (*the Steiner Problem in Graphs* - SPG), y el BNDP en base al Problema General de Steiner en Grafos con requerimientos de nodo conectividad (*the Generalized Steiner Problem with Node-Connectivity Constraints* - GSP-NC). Además, estudiamos un caso particular del BNDP en el cuál tenemos requerimientos de 2-nodo-conectividad entre pares de sitios switch fijos de la Red Dorsal. Este problema lo denominamos BND2NS (*the 2-Node-Survivable Backbone Network Design Problem*). El ANDP, BNDP y BNDP2NS son problemas NP-Hard.

Nuestro objetivo fue resolver el ANDP, BNDP y BNDP2NS mediante el diseño de heurísticas eficientes. Optamos por la meta-heurística GRASP (*Greedy Randomized Adaptive Search Procedure*) como base para el diseño de algoritmos a medida para estos problemas. GRASP es una metodología potente que ha demostrado ser altamente eficiente al ser aplicada en otros problemas de optimización combinatoria. Desarrollamos algoritmos GRASP para los tres problemas, introduciendo diferentes alternativas tanto para la fase de construcción como para la búsqueda local. Los algoritmos diseñados explotan fuertemente propiedades teóricas que incluyen la descomposición topológica de las soluciones factibles. Introducimos además un modelo neural subyacente (*Random Neural Network* - RNN) utilizado en una de las búsquedas locales para el ANDP.

En los tres problemas, las heurísticas GRASP diseñadas fueron testeadas sobre grandes conjuntos de instancias de prueba, con topologías de diferentes características, incluyendo instancias con centenares de nodos. Los resultados computacionales obtenidos fueron altamente prometedores, alcanzado en muchos casos la optimalidad o bien soluciones factibles minimales óptimas locales de bajo costo.

**Palabras Claves:** topological design, access network, backbone network, survivability, GRASP, RNN.

# Abstract

A wide area network (WAN) can be considered as a set of sites and a set of communication lines that interconnect the sites. Topologically a WAN is organized in two levels: the *backbone network* and the *access network* composed of a certain number of *local access networks*. Each local access network usually has a tree-like structure, rooted at a single site of the backbone, and connects users (terminal sites) either directly to this backbone site or to a hierarchy of intermediate concentrator sites which are connected to the backbone site. The backbone network has usually a meshed topology, and its purpose is to allow efficient and reliable communication between the switch sites that act as connection points for the local access networks.

In this thesis we tackled the problem of designing a WAN by breaking it down into two inter-related sub-problems: the Access Network Design Problem (ANDP) and the Backbone Network Design Problem (BNDP). In both models we considered only the construction costs, e.g. the costs of digging trenches and placing a fiber cable into service.

We modeled the ANDP as a variant of the *Steiner Problem in Graphs* (SPG), and the BNDP on the basis of the *Generalized Steiner Problem with Node-Connectivity Constraints* (GSP-NC). In addition, we studied the specific case of BNDP when there exist 2-node-survivability requirements between pairs of backbone fixed nodes. We call it BNDP2NS and it is analogous to the *Steiner 2-node-survivable network problem* (STNSNP). ANDP, BNDP, and BNDP2NS are NP-Hard problems.

Our goal was to attack the ANDP, BNDP, and BNDP2NS models heuristically. We opted for the GRASP (Greedy Randomized Adaptive Search Procedure) methodology for solving them. GRASP is a powerful method which has been used with success to find good quality solutions to many combinatorial optimization problems. We developed GRASP algorithms for these three problems, designing different alternative algorithms for the construction and local each phases. The algorithms exploited theoretical properties including feasible solution decompositions. We also introduced an algorithm based on the RNN (Random Neural Network) model, which was used in the ANDP local search phase.

For the three problems, the GRASP heuristics were tested over large testing problem sets containing heterogenous topologies with different characteristics, including instances with hundreds of nodes. The computational results were highly promising, accomplishing the optimality in many cases or good quality local-optimal solutions.

**Key words:** topological design, metaheuristic, access network, backbone network, survivability, GRASP, RNN.

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation and General Context

Telecommunication networks have become strategic resources for private and state-owned institutions and its economic importance continuously increases. There are series of recent tendencies that have a considerable impact on the economy evolution such as growing integration of networks in the productive system, integration of different services in the same communication system, important modifications in the telephone network structure (voice and data integration, mobility, telephony development on IP platforms, etc). Such evolutions accompany a significant growth of the design complexity of these systems. The integration of different sorts of traffics and services, the necessity of a more accurate management of the service quality, in particular on IP platforms (but on which has not been anticipated the evolution management and the technologies coexistence), are factors that make this type of systems very hard to design, to dimension and therefore to optimize. This situation is complemented with a very high competitiveness context, on an area of critical strategic importance.

In this work, we will focus on modern communication network planning. This field has considerably developed recently mostly owing to the introduction of optic fiber technologies which have very good performance. The planning and design of telecommunication networks is a very complex and generally expensive task. It integrates optimization process loops, analysis activities and quantitative evaluations. The planning team must consider the already existent or anticipated needs, the costs of the different elements that compose the systems, the restrictions on the performance, the reliability, the evolutiveness, the service quality, etc., besides specific restrictions on each particular system and, as a function of these, design a network as adapted as possible to the technical and to the economic plan. In the case of small size networks, the team may consist of a single person while in large-scale networks as wide telecommunication networks (a WAN: Wide Area Network) the planning team may be constituted by several people working at different

organization levels.

The conception of a WAN is a process in which dozens of sites with different characteristics require to be connected in order to satisfy certain reliability and performance restrictions with minimal cost. This design process involves the terminal sites location, the concentrators location, the backbone (central network or kernel) design, the routing procedures, as well as the lines and nodes dimensioning. A key aspect on WAN design is the high complexity of the problem, as much in its globality as in the principal sub-problems in which it is necessary to decompose it. Due to the high investment levels a cost decrease of very few percentage points while preserving the service quality results in high economic benefits.

Typically, a WAN network global topology can be decomposed into two main components: the *access network* and the *backbone network*. These components have very different properties, and consequently they introduce specific design problems (although they are strongly interdependent). On one hand, this causes complicated problems (particularly algorithmic ones); on the other hand, it leads to stimulating and difficult research problems.

A WAN access network is composed of a certain number of access sub-networks, having tree-like topologies; and the flow concentration nodes allow to diminish the costs. These integrated flows reach the backbone which has a meshed topology, in order to satisfy security, reliability, vulnerability, survivability and performance criteria. Consequently, the backbone is usually formed by high capacity communication lines such as optic fiber links. In general, this WAN topological feature is valid in the case of a datagram based network (as in IP technology) and also in circuit commutation (as in the current telephone network or other technologies like X25, Frame Relay o ATM).

Globally, the designing team manages an important amount of data to propose a model that fulfills the preestablished requirements. For instance, it has information about the set of the terminal sites positions (the company customers, the service subscribers, etc.) and about the characteristics (most of the time estimated) of the inter-sites flows (volume, temporary behaviour, etc). Also there is information about the performance restrictions (for example delays), about the service quality (for example of video data quality), and on aspects such as reliability, vulnerability, connectivity, security and availability. On the network components aspect, the designer has a list of possible components according to the involved network nature, with its characteristics and costs. The technical nature of the considered network leads to specific routing procedures that should be taken into account for searching efficient solutions, or if possible optimal solutions. In general there are many other complementary data such as which sites are suitable to install a concentrator in, which ones are not suitable for that, which backbone sites must have switch servers, special characteristics or security restrictions for some flows, etc.

Based on this data set, the designer must specify the access network and the backbone network topologies as well as the characteristics of the different sites and connections, the traffic routing, etc. The result of this process are specific optimization models for the design of both subnetworks and for the global WAN

design problem. This global set of problems typically include the evaluation of performance, reliability, etc.

## 1.2 A general WAN Design Process

Modelling a WAN design by means of the formulation of a single mathematical optimization problem is very intricate due to the interdependence of its large amount of parameters. Therefore the design of a WAN is usually divided into different sub-problems. A good example of a possible decomposition approach for the WAN design process is the following [108]:

**I)** Access and backbone network topologies design. Specific knowledge about the cost of laying lines between different network sites (terminals, concentrators and backbone) is assumed. Frequently, these costs are independent of the type of line that will effectively be installed since they model the fixed costs (cost of digging trenches in the case of optic fiber, installing cost, placing a fiber cable into service, etc.). A high percentage from the total construction network budget is spent in this phase [126].

**II)** Dimensioning of the lines that will connect the different sites of the access and backbone networks, and the equipment to be settled in the mentioned sites.

**III)** Definition of the routing strategy of the flow on the backbone network.

These three sub-problems have different types of constraints:

- for the sub-problem $(\mathbf{I})$:
  - The terminal sites (the clients) must be connected either directly to a backbone node, or through a hierarchy of intermediate concentrator sites which are connected to the backbone. Usually, there exist additional restrictions such as limiting the number of concentrators connected in cascade, so that in the case of a trunk line cut a significative number of terminal sites will not be affected.
  - The backbone must satisfy reliability restrictions that allow it to remain operational (connected) when failures occur in its servers or links. These reliability restrictions are often expressed in terms of the network connectivity. For instance, telecommunication network topologies which have proved being highly performant are the 2-node-connected optic fiber networks. Its physical components have very low failure rates; and the network itself is resilient in the presence of a failure in a node or link. In the same direction, 3-node-connected topologies have been used in optic fiber networks connecting critical sites of a aircraft carrier [78, 126].

- Once the topological structure of the WAN is designed, its components are dimensioned in order to fulfill the performance requirements. A routing plan design and the projection of flows over the backbone must be done so that the performance restrictions imposed be respected. In this way, it can be noticed that $(\mathbf{II})$ and $(\mathbf{III})$ are not independent. Taking into account the technologies used, some of the usual performance restrictions are:
  - the traffic delay should not be greater than a certain prefixed limit. This restriction is imposed to the access network as well as to the backbone.
  - the blocking factor (the probability of a new connection to not succeed) must be lower than a certain prefixed value.
  - the packet loss rate must be relatively low. A level of $10^{-4}$ constitutes the agreed maximal level of the packet loss rate for a network normally working, for today standards.

We give below a generic WAN planning process as well as references related to other works in this area, including topics such as hierarchical network design, multitechnology network design, etc. Taken from [108]:

1) Backbone nodes localization. This implies producing a hypothesis $H$ regarding the backbone sites localization or modifying the precedent hypothesis. These hypothesis must consider the switches installation in the core of the most dense zones.

2) Access network conception:

   a) Depending on the hypothesis $H$, an access topology is constructed by optimally placing the concentrator equipments. If this is not possible, at least a local optimum should be reached as the result of applying clustering strategies.

   b) Determination of the needed capacities in the access network (links and nodes).

   c) Determination of the access network performances by adjusting them to the required level (specified in $2b$).

   d) Determination of the access network reliability by tunning corresponding parameters to meet the required level (specified in $2b$).

   e) Compute the access network cost.

   f) Determination of the reduced matrix of point-to-point traffic between the backbone switch nodes which are entries of the access sub-networks induced by $2a$. These flows must be routed over the backbone topology once this last one has been designed.

3) Backbone network conception:

   a) Based on the hypothesis $H$, a backbone network topology is built adjusted to the reliability demands.

   b) A routing strategy is defined. The point-to-point flows are projected into the network designed in $3a$. Thus, the paths of the backbone on which will circulate the effective traffic are obtained.

   c) Determination of the needed capacities in the backbone network (links and nodes).

   d) Determination of the backbone network performance in order to check if the required levels are fulfilled. If necessary the network of $3a$ or the capacities of $3c$ are redefined.

   e) Determination of the backbone network reliability by adjusting it to the required level. If necessary the topology computed in $3a$ is redefined.

   f) Determination of the network fairness in order to achieve some required level. If necessary the topology computed in $3a$ is redefined.

   g) Computation of the backbone network cost.

4) Results consolidation and global balance:

   a) Determination of the global performances involving the access and the backbone networks simultaneously. If appropriate, return to $2$ or $3$ depending on where performance restrictions violations happen (i.e. in the access network, the backbone, or both).

   b) Determination of the global reliability involving the access and the backbone networks. If appropriate, return to $2$ or $3$ depending on where reliability restrictions violations happen (i.e. in the access network, the backbone, or both).

   c) Compute the overall cost (composed of the access network cost and the backbone network cost). If the WAN cost is approved, the obtained topology is returned as a solution. Otherwise, return to $1$ in order to produce a new hypothesis $H$.

Based on performance evaluation procedures and dimensioning rules common to both network levels (access network and backbone), in [108] each sub-problem is studied and specific algorithms to solve them are proposed. The connections cost taking into account the geographic distances among the involved sites and the annual connection tariffs provided by the telecom operators are estimated. For the topological design of the access network the authors use clustering approaches regarding to the backbone switch nodes whereas for the topological design of the backbone network they apply a variant of the Steiglitz heuristic, denominated Hierarchical Method. Even if the testing cases presented are relatively small, the suggestedd methodology can be useful as a reference about the way of decomposing the WAN topological planning

process into several sub-problems. For other related works concerning the optimal design of a multi-level hierarchical network the reader can consult the references [10, 8, 11, 81, 24, 25, 38, 79, 94, 128]. Mainly, they are centered in network planning contemplating also several aspects of network dimensioning. Other problems in this area can be found in [3, 16, 53, 55, 56, 107, 129], where the authors propose several models for designing low-cost network topologies with additional constraints such as fault tolerance and performance restrictions, considering in addition in some of them network components dimensioning.

In this thesis, we will concentrate on phase $(\mathbf{I})$ of the decomposition of a WAN design process. More precisely, we are interested in the topology planning process concerning the access network and the backbone network. Our motivation comes from the necessity of devising efficient approximated algorithms for these topological design highly-combinatorial problems. Due to the NP-hard nature of the problem and even though there exist some results, there is still room for improving industrial practices applied today. In this sense, we believe it is of strategic importance designing powerful quantitative analysis techniques, potentially easy to integrate into tools. We introduce combinatorial optimization models to formally define the topological design of the access and backbone networks, and we propose different approximated algorithms to solve them which are based on the well-known GRASP (Greedy Randomized Adaptive Search Procedure) methodology [45].

## 1.3 Access and Backbone Network Design Problems

We will define these problems in terms of graph theory; for this purpose we introduce the following notation:

- $S_T$ is the set of terminal sites (clients) to be connected to the backbone.
- $S_C$ is the set of feasible concentrator sites of the access network. On each one of these sites, an intermediate server equipment might be placed. From this one, a trunk line is laid towards the backbone or other concentrator site.
- $S_D$ is the set of feasible switch sites of the backbone network. On each one of these sites, a powerful server might be placed and from it, connection lines towards other backbone server equipments.
- $V = S_T \cup S_C \cup S_D$ are all the feasible sites of the WAN network.
- $A = \{a_{ij}\}_{i,j \in V}$ is a matrix which gives for any pair of sites $i, j \in V$, the cost $a_{ij} \geq 0$ of laying a line between them. When the direct connection between $i$ and $j$ is not possible, we define $a_{ij} = \infty$.
- $U = \{(i, j) | i, j \in V, a_{ij} < \infty\}$ is the set of all the feasible connections between the different sites of the WAN network.

- $G = (V, U)$ is the simple graph which models every node and feasible connection of the WAN.

The General Access Network Design Problem (GANDP) consists of finding a minimum-cost subgraph $H \subset G$ such that all the sites of $S_T$ are communicated with some node of the backbone. This connection can be direct or through intermediate concentrators. The use of terminal sites as intermediate nodes is not allowed; this implies that they must have degree one in the solution.

We simplified the GANDP problem by collapsing the backbone into a fictitious node. We call it the Access Network Design Problem (ANDP) and the equivalence between both problems, GANDP and ANDP, is proved in Appendix A. The ANDP is NP-Complete (we will demonstrate it by reducing the Steiner Problem in Graphs to it in Chapter 3).

Given a subset of switch sites $S_D^{(I)} \subseteq S_D$ and a non-negative integer matrix $R = \{r_{ij}\}_{i,j \in S_D^{(I)}}$, the Backbone Network Design Problem (denoted by BNDP) consists of finding a minimum-cost subgraph $H \subseteq G(S_D)$ such that $S_D^{(I)} \subset H$ and $\forall i, j \in S_D^{(I)}$ there exist at least $r_{ij}$ node-disjoint paths connecting $i$ with $j$ in $H$. This problem can be modelled as the Generalized Steiner Problem in Graphs with Node-Connectivity constraints (denoted by GSP-NC) which is NP-Complete in the general case [136].

For further details on the formulations of the Generalized Steiner Problem in its both versions, edge-connectivity (denoted by GSP-EC) and node-connectivity, the reader may consult [134, 135, 136].

A particular case of the BNDP is when $r_{ij} = 2, \forall i, j \in S_D^{(I)}$. This is known in the literature as the Steiner 2-node-survivable network problem (denoted by STNSNP) [6].

We call $S_D^{(I)}$ the set of fixed switch nodes. These will necessarily have to be integrated to the solution, either because they are access sub-network entry points to the backbone or due to specific conception requirements. The sites of $S_D \setminus S_D^{(I)}$ are optional (commonly named Steiner nodes) and may be use to reduce the backbone cost.

Our aim in this thesis is the study of the ANDP and BNDP problems. For the BNDP case we study the general case when the base model is the GSP-NC as well as the particular case when the base model is the STNSNP, which we denote BNDP2NS (2-node-survivable BNDP). In the next subsection, we introduce a survey of works related to the ANDP, BNDP and BNDP2NS.

## 1.4 Related Work

As we already mentioned when we are talking of networks in this thesis, we are interested only in their topology, that is, we see a network as a set of sites and links that are placed between sites. Survivability in this context means that between any two sites there exists a pre-specified number of paths (consisting of nodes and links) that have no node or link in common. The only costs considered are costs associated with the network topology like the cost of digging trenches in case of optic fiber. The problems ANDP, BNDP,

and BNDP2NS correspond to this context. In practice, the topology of a network with low placement costs is created first, and in a second optimization stage, traffic and routing costs are considered [126].

We concentrate first in the literature related to ANDP. In [2, 4, 12, 49, 57, 54, 72, 68, 96, 99, 110, 111], the authors propose different approximate algorithms for the topological design of local and large-scale access networks. They are based on different approaches, and consider different parameters and restrictions, including aspects such as: the design of the access network is restricted to specific topologies; the number of concentrators to be placed is limited; network components dimensioning, etc. The resolution techniques used in these works include: Lagrangian Relaxation mixed with the Sub-gradient Method [99], Simulated Annealing [96], Linear Programming Relaxation [4], Lagrangian Heuristic, Greedy Heuristics [72], Branch-and-Bound mixed with Lagrangian Relaxation, Branch-and-Bound with Benders decomposition [110, 111], Neural Networks [2], Tabu Search [68], Genetic Algorithms, plus other specific methods.

Next we will focus on the GSP-NC and STNSNP (which are the reference models of base for our BNDP y BNDP2NS problems) and their related survivability models like those presented in [135, 136, 126].

Winter [135, 134, 136] demonstrated that the GSP-NC can be solved in linear time if the network is series-parallel, outerplanar or a Halin graph. Nextly, we will give a summary of the survivability problems related to the GSP-NC and STNSNP. Gröstchel, Monma and Stoer [74] consider a particular case of the GSP-NC working on a slightly different context. They called it the NCON problems. In [126], Stoer gives an extensive survey for the NCON and the ECON (the version with edge-connectivity constraints), and some particular cases. The NCON (resp. ECON) is formalized as follows. Given an undirected graph $N = (X, U)$ such that each edge $e \in U$ has a fixed weight $c_e$ representing the cost of establishing the direct link connection. In particular, each node $i \in X$ has an associated nonnegative integer $r_i$, the **type** of $i$ (the survivability requirement or "importance" of a node is modeled by node types). Let $H = (W, F)$ be a subgraph of $N$. We say that $H$ satisfies the node-survivability conditions (also called node-connectivity constraints or requirements), if, for each pair $i, j \in X$ of distinct nodes, $H$ contains at least $r_{ij} = \min\{r_i, r_j\}$ node-disjoint paths communicating $i$ with $j$. Similarly, we say that $H$ satisfies the edge-survivability conditions (also called edge-connectivity constraints or requirements), if, for each pair $i, j \in X$ of distinct nodes, $H$ contains at least $r_{ij} = \min\{r_i, r_j\}$ edge-disjoint paths communicating $i$ with $j$. These conditions ensure that some communication path between $i$ and $j$ will survive a prespecified level of node (or edge) failures.

Let us observe that the GSP-NC model generalizes the model given above since in the GSP-NC there exist general survivability requirements $r_{ij}$ that are specified for each pair $i, j$ of fixed nodes independently. Nevertheless, Grötchel, Monma and Stoer [75, 77, 76, 78] introduce the use of node types to define survivability requirements based on the premise that these adequately express the relative importance placed on maintaining connectivity between offices. They classify the different problem types according to the

largest occurring node type and according to whether the node types represent node or edge connectivity requirements. In this way, given a graph $N = (X, U)$ and a vector $r = (r_i)_{i \in X}$, by assuming (without loss of generality) that there exist at least two node types of type $k$ (which is defined as the largest node type), they speak of the $k$NCON problem (resp. $k$ECON) when the objective is to find a minimum-cost network that satisfies the node survivability conditions (resp. the edge survivability conditions). If the highest value of $k$ is not specified, these problems are called NCON and ECON respectively. In particular, if all node types have the same value $k$, the problem NCON (resp. ECON) is reduced to find $k$-node-survivable (resp. $k$-edge-survivable) networks having minimum cost.

Let us note that there exist many specializations of the survivability problems which can be formulated by varying its parameters as follows:

- As mentioned previously, the GSP-NC and GSP-EC are more general models of survivability than NCON and ECON, since the connectivity requirements are associated to pairs of nodes in independent form and not necessarily involving all the nodes of $X$.

- In the NCON and ECON, we have $r_{ij} = \min\{r_i, r_j\}$ for given nodes types $r_i$, $r_j$, which in turn may be:

  - general ($k$ECON or $k$NCON problem),
  - uniform ($k$-edge or $k$-node connected graphs),
  - in $\{0, 1\}$ (Steiner trees)

- general or euclidean or uniform costs.

There exist polynomially solvable cases of the NCON and ECON problem. They result from relaxing the original problem with restrictions like uniform costs, 0/1 costs, restricted node types, and special underlying graphs such as outerplanar, series-parallel, and Halin graphs. All these particular cases are referenced and briefly exposed in [126]. On the other hand, lower bounds and heuristics with worst-case guarantees for $k$ECON problems were found for restricted costs, e.g., uniform costs or costs satisfying the triangle inequality, as well as very important results on the structure of optimal survivable networks for this cost structure. Details of these works can be seen in [13, 34, 29, 48, 50, 69, 71, 102] and in a summarized form in [126]. In [126], Stoer also summarizes heuristic procedures to solve general $k$NCON and $k$ECON problems. Monma and Shallcross [103] give heuristics for the 2ECON and 2NCON problems. Frederickson and Jájá [50] propose a heuristic for the 2NCON problem with worst-case guarantee of $3/2$ under costs satisfying triangle inequality. Consider the NCON problem where instead a vector $r = (r_i)_{i \in X}$ we have a matrix $R = (r_{ij})_{ij \in X}$; this variant had been posed already in 1969 by Steiglitz, Weiner, and Kleitman [125],

but they did not give it a specific name. They developed a simple heuristic for this problem which basically consists of a randomized starting routine and an optimizing routine where local transformations are applied to a feasible solution. Ko and Monma [89] propose heuristics for the design of $k$-edge or $k$-node connected networks. Goemans and Bertsimas [69] propose a heuristic for the ECON problem with worst-case guarantee. In addition, Goemans and Williamson [71] proposed an approximation algorithm which can be applied to the GSP-EC (Generalized Steiner Problem with Edge-Connectivity Constraints) allowing the use of multiple parallel edges. Khuller and Vishkin [88] propose an algorithm for the $k$ECON problem with a worst-case guarantee of $2$ and under the restriction that parallel edges are not allowed in the solution and all types of nodes are equal. Recently, Balakrishnan, Magnanti and Mirchandani [9] presented a family of new mixed-integer programming formulations for the GSP-EC, whose associated linear programming relaxations can be tighter than those of the usual cutset formulation. They provide several combinatorial heuristics for these formulations, which satisfy that the bounds on the heuristic costs relative to the optimal values of the integer program and the linear programming relaxation of the tighter formulation are stronger than some previously known performance bounds for combinatorial heuristics. For further details of these works (and their performance tests) the reader may consult the cited references.

Unfortunately, there exist few exact algorithms for the NCON and ECON for general costs. Christofides and Whitlock [36] introduce a cutting plane algorithm together with branch-and-bound for ECON problems where instead a vector $r = (r_i)_{i \in X}$ we have a matrix $R = (r_{ij})_{ij \in X}$. Chopra and Gorres [33] give a cutting plane algorithm mixed with branch-and-bound for solving 2ECON problems.

In the literature there are several works related to approximation algorithms for the GSP and different particular cases. Next, we will introduce a survey of the main existing algorithms based on this approach. In [112] the authors show how to obtain approximately optimal solutions to 2-edge-connected versions of the problems addressed in [71]. Subsequent papers [70, 51, 133] extended these methods to give approximation algorithms for the GSP-EC without link duplication. Agrawal, Klein and Ravi [1] developed an algorithm for the GSP-EC with performance guarantee of $2\lceil \log_2(r_{max} + 1) \rceil$, where $r_{max}$ is the highest requirement value. More recently Jain [83] presented a factor $2$ approximation algorithm for the GSP-EC. Kortsarz, Krauthgamer and Lee [92] introduced the first strong lower bound on the approximability of the GSP-NC when there are no Steiner nodes.

An important special case of the GSP-NC occurs when we are searching the minimum-cost $k$-node-connected subgraph spanning all the nodes. In first place, let us see the general case. In [29, 40, 32, 92, 93, 114, 113] the authors propose several approximation algorithms for the problem of finding a minimum-cost $k$-node-connected spanning subgraph, besides they give their respective approximation ratios. For $k \leq 7$ an approximation ratio of $\lceil (k+1)/2 \rceil$ is known (see [86] for $k = 2$, [5] for $k = 2, 3$, [82] for $k = 4, 5$, and [93] for $k = 6, 7$). Other approximations for $k = 2$ can be seen in [14, 39]. Furthermore, in [40], [31] and [93] the authors respectively supply approximation algorithms for the following special cases: the graph

has complete Euclidean topology, uniform costs, and metric costs (i.e. when the costs satisfy the triangle inequality).

Another problem related to the GSP-NC (resp. GSP-EC) is the node-connectivity augmentation problem, where the goal is to find a minimum-cost set of edges that augments an $m$-node-connected (resp. $m$-edge-connected) graph into a $k$-node-connected (resp. $k$-edge-connected) graph. Some of the principal references in this area are [27, 30, 46, 52, 84, 95, 87, 104, 109, 106, 130, 132]. These papers provide different approximation algorithms with their respective approximation ratios. Some of these works study the particular case when the costs are uniform, which is commonly known as minimum-size connectivity problem.

Finally, let us see works related to the STNSNP. In [6] Baïou mentions different problems related directly to the STNSNP. In particular, the problems known as:

- the Steiner 2-edge-connected subgraph problem (STECSP), and
- the Steiner 2-node-connected subgraph problem (STNCSP), and
- the Steiner 2-edge-survivable network problem (STESNP).

The STNSNP (resp. STESNP) also corresponds to the problem $k$NCON (resp. $k$ECON) in the case where $r_i \in \{0, 2\}$, $\forall i \in X$. Given a graph $N = (X, U)$, a subset $T \subseteq X$ and a matrix $C$ of connection costs associated to $U$; the objective in the STNCSP (resp. STECSP) is to find a minimum-cost 2-node-connected (resp. 2-edge-connected) subgraph spanning the set of nodes $T$. If the matrix $C$ is positive, the sets of optimal solutions associated to the STNSNP and STNCSP are equal. Idem the sets of optimal solutions associated to the STESNP and STECSP. If all the nodes are fixed (there are no Steiner nodes) the problems STESNP and STECSP coincide, and also the STNSNP with the STNCSP. Moreover, it is easy to see that all feasible solution of the STNCSP (resp. STECSP) is also feasible for the STNSNP (resp. STESNP). In [37] the authors developed a linear algorithm to solve the STNCSP in the case of graphs without $W_4$ (a wheel graph with four nodes) and Halin graphs. The author of the present thesis has previously developed a parallel method (of worst case exponential complexity) for the general case [23]. Other works related to particular cases of the STNCSP, e.g. when $T = X$ or uniform costs, already have been mentioned above.

We find in the literature other works related to our BNDP. In [15, 26, 28, 41, 67, 66, 91, 97] the authors provide different approaches for the topological design of a backbone network. Most of these works are not only focused in the topological design, but they also consider aspects such as network dimensioning, routing mechanisms, etc. They are based on different optimization models which include selection of network topology and other additional objectives. We can see these ones as network planning processes where the goal is to find backbone topologies with lowest possible overall network price, while keeping all requirements (such as availability, maximal number of, maximal blocking probability, etc.) satisfied. The

resolution techniques used in these works include: Genetic Algorithms, Branch-and-Bound method mixed with the algorithm of Ford-Fulkerson, Tabu Search, Greedy Heuristic combined with Tabu Search heuristic as improver, Lagrangian Relaxation embedded in a sub-gradient optimization procedure, Dual-Based lower bounding procedure incorporated in a Branch-and-Bound algorithm, Dual-Based solution procedure, Hybrid approach of a genetic algorithm and local search algorithms as improver, Tabu-Search heuristic with a post-optimization algorithm, and other specific heuristics.

## 1.5 Manuscript Organization and Main Results

This thesis work is organized as follows. Chapter 2 introduces notation, basic definitions of graph theory, and general descriptions of the GRASP metaheuristic and the Random Neural Network (RNN) model. In Chapter 3 we study the ANDP problem. We propose different approximated algorithms to solve the ANDP. These algorithms were designed based on the GRASP methodology. In particular, we provide several alternatives for the GRASP components: two algorithms for the construction phase and two algorithms for the local search phase (one of these based on the RNN model). In this way, we yield four different versions of GRASP algorithms to be applied to the ANDP. To proof practically our GRASP algorithms, we generate a large ANDP test-set by customizing SPG instances extracted from the SteinLib library [90]. The optimal costs of the original SPG instances provide lower bounds for the optimal costs of the corresponding ANDP instances. Nevertheless, except for the cases in which we achieved the lower bound and therefore the optimality, we do not know the optimal values for the generated ANDP instances. The experimental results showed a good quality of the built solutions, obtaining in most of the cases optimal or near-optimal solutions with low gaps with respect to the lower bounds. The main results presented in this chapter have been published in [19, 20, 22, 122, 123]. In Chapter 4 we study the BNDP problem and propose different approximated algorithms to solve it. As in the ANDP, we used GRASP methodology. We introduce an algorithm for the construction phase as well as three algorithms for the local search phase. These three procedures are structurally different so that each of them explores a particular sub-space of neighbor solutions. This allow to define different exploration strategies by suitably combining the local search algorithms. As test-set for the BNDP, we used instances generated constructively with known optimum, instances generated by adding randomly Steiner nodes to TSP instances extracted from the TSPLib repository and specific GSP instances from the literature. The algorithms obtained good results, attaining the optimum in many cases or near-optimum solutions with low gaps (less than 0.7%) with respect to the optimum values. In addition, for the cases without known optimal value (those derived from TSP instances), we achieved good quality minimal feasible solutions when comparing them with tight lower bounds for the optimal 2-node-connected solutions for the original TSP instances. The results concerning BDNP have also been published

in [17, 18, 21]. In Chapter 5, we study our BNDP2NS. Based on specific structural properties of the 2-node-survivable networks, we designed GRASP algorithms to solve approximately the BNDP2NS. More precise, we introduce an algorithm for the construction phase based on a topological characterization of 2-connected graphs. Furthermore, we give other two alternative algorithms for the construction phase. As test-set for the BNDP2NS, we used problems from the TSPLib transforming them into BNDP instances by adding a large number of Steiner nodes (in all cases more than 29% of the total). Although we do not know the optimum solutions (i.e. their respective optimal values) of the resulting instances, particularly for the Euclidian BNDP2NS instances we can bound the optimal values within a interval determined by a theorem introduced by Monma et al. [102]. This interval only depends on the optimal value corresponding to the original TSP instance. The results obtained in the testing phase show that our GRASP algorithms find in most cases good quality solutions. In particular, for the Euclidian BNDP2NS instances, we improved significatively in many cases the optimal 2-node-connected solution spanning the fixed-nodes; more precisely, we reached smaller values than the lower bounds for the latter. In Chapter 6, we give final conclusions and some guidelines for future work. Appendix A contains the proof of ANDP NP-completeness. Appendix B provides tables summarizing experimental results obtained for the ANDP. In Appendix C we introduce some propositions used to build BNDP instances with known optimum cost.

## 1.6 List of publications issued from this thesis work

For easy lecture, we give here the list of publications related to this thesis (with the same reference number as they are found in the Reference list at the end).

[17] H. Cancela, F. Robledo, and G. Rubino. A GRASP algorithm for designing a Wide Area Network backbone. In *Proceedings of the International Network Optimization Conference (INOC'03)*, pages 138–143, Evry/Paris, France, October 2003.

[18] H. Cancela, F. Robledo, and G. Rubino. Network design with node connectivity constraints. In *Proceedings of the IFIP/ACM Latin America Networking Conference (LANC'03)*, pages 13–20, La Paz, Bolivia, October 2003.

[19] H. Cancela, F. Robledo, and G. Rubino. Finding Steiner trees with degree 1 terminal nodes. *IEICE Electronics Express (ELEX)*, 1(9):258–262, 2004.

[20] H. Cancela, F. Robledo, and G. Rubino. A GRASP algorithm with RNN based local search for designing a WAN access network. In *Electronic Notes in Discrete Mathematics - special issue including the Proceedings of the Latin-American Conference on Combinatorics, Graphs and Applications (LACGA'04)*, volume 18C, pages 53–58, 2004.

[21] H. Cancela, F. Robledo, and G. Rubino. A GRASP algorithm with tree based local search for designing a Wide Area Network backbone. *Journal of Computer Science and Technology*, 4(1):52–58, 2004.

[22] H. Cancela, F. Robledo, and G. Rubino. Designing low-cost access network topologies. In *Proceeding of the International Network Optimization Conference (INOC'05)*, University of Lisbon, Portugal, October 2005.

[122] F. Robledo. A GRASP algorithm with MST based local search for designing a WAN access network. In *Proceedings of the 6 ème Journées Doctorales Informatique et Réseau (JDIR'04)-France Télécom R&D*, Lannion, France, November 2004.

[123] F. Robledo, R. Maba, I. Manzo, and D. Nachman. Using GRASP for designing low-cost access topologies. In *International Conference on Industrial Logistics (ICIL'05)*, Universidad de la República-Facultad de Ingeniería, Montevideo, Uruguay, February 2005.

# Chapter 2

# Background

In this chapter, we introduce basic notation and definitions that will be used in the next chapters. In addition, we give a short description of the GRASP (Greedy Randomized Adaptive Search Procedure) methodology [44, 45, 47] as well as a general description of the RNN (Random Neural Network) model [58, 59].

## 2.1 Basic Definitions

At this point, we introduce basic some definitions of graph theory and other definitions frequently used in works related to survivability models.

**Undirected Graph.** An undirected graph is a pair $G = (V, E)$ where $V$ is a non-empty set and $E$ is a part of $V \times V$; thus, the elements of $E$ are 2-element subsets of $V$. The elements of $V$ are the vertices (or nodes) of the graph $G$, the elements of $E$ are its edges (or lines). A graph $G$ is simple when there exists at most one edge between any pair of nodes. A graph $G$ is undirected when the edges are pairs of nodes not ordered. From now, we use graph as a synonymous of simple and undirected graph, as a default. Set $V$ is denominated set of nodes and $E$ is denominated set of edges.

**Ends of an Edge.** A node is incident to an edge $e$ if $v \in e$; then $e$ is an edge at $v$. The two nodes incident to an edge are its endpoints or ends, and an edge joins (or connects) its ends. An edge $\{u, v\}$ is also written here as $(u, v)$ or $(v, u)$. The set of all the edges in $E$ at a node $v$ is denoted by $E(v)$.

**Induced Graph.** Given a graph $G = (V, E)$, if $U \subseteq V$ is a subset of nodes then $G(U)$ denotes the graph on $U$ whose edges are precisely the edges of $G$ with both ends in $U$.

**Adjacent Node.** Given a graph $G = (V, E)$ and $u \in V$, a node $v \in V$ is adjacent (or neighbor) to $v$ in $G$ if $(u, v) \in E$.

**Adjacent Edge.** Two edges $e \neq f$ are adjacent if they have an end in common.

**Complete Graph.** If all the nodes of a graph $G$ are pairwise adjacent, then $G$ is complete. A complete

graph with $n$ nodes is usually denoted by $K_n$.

**Neighborhood of a Node.** Given a graph $G = (V, E)$ the set of neighbors of a node $v$ in $G$ is denoted by $N_G(v)$ or more briefly by $N(v)$.

**Degree of a Node.** The degree $d_G(v)$ or $d(v)$ of a node $v$ is the number $|E(v)|$ of edges at $v$; it is equal to the number of neighbors of $v$. A node of degree $0$ is said to be isolated.

**Path.** A path is a non-empty graph $P = (V, E)$ of the form:

$$V = \{v_1, v_2, \ldots, v_k\},\ E = \{(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)\},$$

where the $v_i$ are all distinct. The nodes $v_1$ and $v_k$ are linked by $P$ and are called its endpoints; the nodes $v_2, \ldots, v_{k-1}$ are the inner nodes of $P$ (or internal nodes). We often refer to a path by the natural sequence of its nodes, writing, say, $P = v_1 v_2 \ldots v_k$, and calling $P$ a path from $v_1$ to $v_k$. We can also denote the path by the sequence of its edges: $P = e_1 e_2 \ldots e_{k-1}$, where $e_1 = (v_1, v_2)$, $e_2 = (v_2, v_3), \ldots, e_{k-1} = (v_{k-1}, v_k)$.

**Cycle.** Given a path $P = (v_1 \ldots v_k)$, the graph $C$ obtained by concatenating $P$ with $v_k v_1$ is called a cycle. We often denote a cycle by its (cyclic) sequence of nodes; the above cycle $C$ might be then written as $v_1 v_2 \ldots v_k v_1$.

**Independent Paths.** We say that two paths $p_1$, $p_2$ included in a graph $G = (V, E)$ are independent if the intersection of their sets of nodes is empty. We denote $p_1 \cap p_2 = \emptyset$.

**Node-disjoint Paths (with the same endpoints).** Given two paths $p_1$, $p_2$ including in $G = (V, E)$ and having the same endpoints $u, v \in V$, we say that $p_1$ and $p_2$ are node-disjoint if the intersection of their sets of internal nodes is empty. Thus, $p_1 \cap p_2 = \{u, v\}$.

**Subgraph.** Given a graph $G = (V, E)$, $H = (V', E')$ is a subgraph of $G$ if $V' \subseteq V$, $E' \subseteq E$ and $\forall (u, v) \in E'$, we have $u, v \in V'$. We also write $H \subseteq G$.

**Connected Graph.** A graph $G = (V, E)$ is connected if for all pair of nodes $u, v \in V$ there exists a path from $u$ to $v$ in $G$.

**Tree.** A connected graph $G = (V, E)$ is a tree if for all edges $e \in E$, $G' = (V, E \setminus \{e\})$ is unconnected.

**Forest.** A forest is a graph whose connected components are trees.

**Connected Component.** A connected component of a graph is a subgraph that is connected and that is maximal with respect to this property.

**Spanning Tree.** Given a connected graph $G = (V, E)$, a subgraph $H = (V, E')$ is a spanning tree of $G$ if $H$ is connected and for all edge $e \in E'$, $H' = (V, E' \setminus \{e\})$ is unconnected.

**2-Node-Connected Graph.** A graph $G = (V, E)$ is 2-node-connected if $\forall u, v \in V$ there exists at least two node-disjoints paths connecting them in $G$.

**2-Node-Connected Component.** A 2-node-connected component of a graph is a subgraph that is 2-node-connected and maximal with respect to this property.

**Nodes locally $k$-node-connected.** Given a graph $G = (V, E)$, two nodes $u, v \in V$ are locally $k$-node-connected if there exists at least $k$-node-disjoints paths connecting them in $G$.

**$k$-Node-Connectivity.** A graph $G = (V, E)$ is $k$-node-connected if $\forall u, v \in V$ there exists at least $k$-node-disjoints paths connecting them in $G$.

**$k$-Node-Survivability.** Given a graph $G = (V, E)$ and a subset $T \subseteq V$, $G$ is $k$-node-survivable with respect to $T$ if $\forall u, v \in T$ there exists at least $k$-node-disjoints paths connecting them in $G$.

**2-Node-Survivable Component.** Given a graph $G = (V, E)$, a subset $T \subseteq V$ and $T_1 \subset T$, a 2-node-survivable component with respect to $T_1$ is a subgraph of $G$ that is 2-node-survivable spanning $T_1$.

**Bridge.** Given a graph $G = (V, E)$, an edge $e \in E$ is bridge if the graph $G' = (V, E \setminus \{e\})$ has more connected components than $G$.

**Articulation Set.** Given a graph $G = (V, E)$, we call $Z \subset V$ an articulation set of $G$, if the induced subgraph $G(V \setminus Z)$ has more connected components than $G$. Synonymously, we say that $Z$ is a *separating set* in $G$.

**Articulation Node.** Given a graph $G = (V, E)$, we call a single node $z \in V$ an articulation node of $G$, if the induced subgraph $G(V \setminus \{z\})$ has more connected componentes than $G$.

## 2.2 Notation

Now, we will introduce some additional notation that will be used in the following chapters.

- Given two graphs $G = (V, E)$ and $G' = (V', E')$, we denote $G \cup G' = (V \cup V', E \cup E')$, $G \cap G' = (V \cap V', E \cap E')$ and $G \setminus G' = (V \setminus V', E \setminus E')$.

- Given a graph $G$ and a path $p$ we denote $G \cup \{p\}$ (or $G \cup p$) the resulting graph when adding to $G$ the path $p$. More generally, if $P$ is a set of paths, $G \cup P$ denotes the resulting graph of adding to $G$ all the paths from $P$.

- Analogously, if $p$ has its endpoints in $G$, the graph $G \setminus p$ denotes the resulting graph when removing from $G$ all the edges and nodes of $p$ except its endpoints. Moreover, if $P$ is a set of paths having the same endpoints in $G$, then $G \setminus P$ is the graph obtained by removing from $G$ all the the edges and nodes belonging to paths of $P$ except its endpoints.

- Given two paths $p_1$ and $p_2$ such that the endpoints of $p_2$ are in $p_1$, $p_1 \setminus p_2$ is the subgraph obtained when removing from $p_1$ all the edges and nodes of $p_2$ except its endpoints.

- Given a graph $G = (V, E)$ and a cost function $C$ associated with $E$ ($C : E \rightarrow \bar{\mathbb{R}}^+$), we introduce the operator $\textsc{cost}(\cdot)$ defined by:

$$\textsc{cost}(H) = \sum_{\forall (i,j) \in H} c_{i,j},\ \forall H \subseteq G,$$

  where $c_{ij} = C((i, j))$ and $H$ is a subgraph of $G$. In addition, if $A$ is another cost function on $E$, we denote $\textsc{cost}_{|A}(H)$ in order to distinguish this operation from other cost functions.

- Given a graph $H$, we introduce the following operators:
  - $\textsc{nodes}(H)$ is the set of nodes of $H$,
  - $\textsc{edges}(H)$ is the set of edges of $H$,
  - given a path $p$, $\textsc{internal\_nodes}(p)$ is the set of nodes of $p$ except its endpoints.

## 2.3 Greedy Randomized Adaptive Search Procedure

The Greedy Randomized Adaptive Search Procedure (GRASP) is a well known metaheuristic, which has been applied for solving many hard combinatorial optimization problems with very good results [42, 47, 100, 98, 105, 124, 116, 117, 119]. Extensive surveys of the associated literature are given in [45, 118, 47].

Before describing the main ideas of GRASP, we formulate a generic combinatorial optimization problem based on the description introduced in [118]. Let us consider:

i) $N = \{n_1, \ldots, n_m\}$ is the finite basic set containing the potential elements which will be able to integrate a feasible solution.

ii) $F$ denotes the set of feasible solutions satisfying: $F \subseteq 2^N$.

iii) $f : 2^N \rightarrow \mathbb{R}$ is the objective function. Without losing generality, we will assume the minimization version, i.e. the aim is to find a global optimal solution $S^* \in F$ such that $f(S^*) \leq f(S),\ \forall S \in F$.

These points will be determined, when specifying the optimization problem to be studied. For example, in the case of the Minimum Vertex Covering Problem:

- $N = \{v_1, \ldots, v_n\}$ is the set of nodes to be considered,
- $E$ is the set of edges connecting the nodes of $N$,
- $F$ is composed of all the subsets of $N$ such that if $S \in F$ any edge in $E$ has at least one endpoint in $S$,

- $f(S)$ is the number of nodes belonging to $S$.

A GRASP is an iterative process, where each iteration consists of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood (in some sense to be defined when adapting the method to each specific problem) is explored during the second phase, looking for an improvement. The best solution over all GRASP iterations is returned as the result.

We describe now a generic GRASP implementation, whose pseudo-code can be seen in Figure 2.1. This generic implementation serves as a template to be mapped into the problems introduced in Chapters 3, 4, and 5 where specific GRASP methods customized to our problems are proposed.

The GRASP heuristic has three main parameters: the number of iterations $MaxIter$, the candidate list size $ListSize$, and a third implicit parameter, the initial seed $Seed$ for the pseudo-random number generator. The first parameter corresponds to the number of iterations in the outer loop of the algorithm. The second parameter will be seen in more detail when explaining the construction phase, but roughly speaking, it is a measure of how many alternatives will be taken into account at each constructive step.

In some GRASP versions the size of the restricted candidate list is recomputed dynamically (i.e. the value of $ListSize$ is not fixed), being used in this case a threshold parameter denoted by $\alpha$. Later, we will explain in detail both variants.

Looking again at the pseudo-code, it can be seen that GRASP iterations are carried out in lines 2-8. Each GRASP iteration consists of the construction phase (line 3), the local search phase (line 4) and, if necessary, the solution update (lines 5-7).

```
Procedure GRASP(ListSize,MaxIter,Seed);

1  Read_Input_Instance();
2  for k = 1 to MaxIter do
3        InitialSolution ← Construct_Greedy_Randomized_Solution(ListSize, Seed);
4        LocalSearchSolution ← Local_Search(InitialSolution);
5        if cost(LocalSearchSolution) < cost(BestSolutionFound) then
6              Update_Solution(BestSolutionFound, LocalSearchSolution);
7        end_if;
8  end_for;
9  return BestSolutionFound;
```

Figure 2.1: GRASP pseudo-code.

In the construction phase, a feasible solution is built. Figure 2.2 shows a generic pseudo-code for the construction phase. The solution is usually represented as a set of elements (the precise meaning of these elements depends on the specific problem); the construction phase starts from an empty set and iteratively adds an element until the set corresponds to a feasible solution. At each step of the construction phase, a restricted candidate list (denoted by RCL) is determined by ordering all non already selected elements with

respect to a greedy function that measures the (myopic) benefit of including them in the partial solution. In general, this function evaluates the incremental increase in the cost function $f(\cdot)$ when incorporating each new element into the solution under construction. Specifically, by applying this function, we build the RCL containing those elements whose addition to the current partial solution induce the smallest incremental costs (this is the greedy component of GRASP). The next element to be included into the partial solution is randomly chosen (uniformly or in some biased form) from the RCL (this is the probabilistic component of GRASP). In this way, GRASP allows for different solutions to be obtained at each GRASP iteration. When the chosen element is added to the partial solution, the benefits associated with every element not yet added to the partial solution are updated in order to reflect the change induced by the insertion of the new element. Thus, the heuristic recomputes the RCL and reevaluates the incremental costs (this is the adaptive component of GRASP). Once the construction phase is finished, the solution built is returned.

```
Procedure Construct_Greedy_Randomized_Solution(ListSize,Seed);

1  Solution ← ∅;
2  Incremental costs evaluation for the candidate elements;
3  while not_feasible(Solution) do;
4          RCL ← the restricted candidate list;
5          s ← select randomly an element from the RCL;
6          Solution ← Solution ∪ {s};
7          Incremental costs reevaluation;
8  end_while;
9  return Solution;
```

Figure 2.2: Construct_Greedy_Randomized_Solution pseudo-code.

The solutions generated by the construction phase are not guaranteed to be locally optimal with respect to simple neighborhood definitions. Hence, it can be beneficial to apply a local search to attempt to improve each constructed solution. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better one taken from its neighborhood. It finalizes once there is no better solution found in the neighborhood. Figure 2.3 shows a generic pseudo-code for the local search phase. It has as input a feasible solution $Solution$ and searches for a better solution within a neighborhood $N(Solution)$ previously defined. In most of the cases, the local search phase takes as entry the feasible solution $Solution$ delivered by the construction phase, but for certain applications, we could have several local search phases working in a combined form by exploring different neighborhoods, implying thus that their entries are not necessarily the solutions given by the construction phase.

The success when applying the local search phase is strongly related with the following points:

- the suitable choice of a neighborhood structure,

```
Procedure Local_Search(Solution);

1  while not_locally_optimal(Solution) do;
2        Find Neighbor_Solution ∈ N(Solution) satisfying f(Neighbor_Solution) < f(Solution);
3        Solution ← Neighbor_Solution;
4  end_while;
5  return Solution;
```

Figure 2.3: Local_Search pseudo-code.

- efficient neighborhood search techniques,
- easy evaluation of the cost function when exploring the neighborhood,
- the quality of the starting solution.

The construction phase plays an important role with respect to this last point, since it must produce good starting solutions for this local search sub-procedure. Depending on the problem, the used neighborhoods are generally not complex. There exist two basic different strategies to explore a neighborhood, which are:

**best-improvement:** all neighbors are investigated and the current solution is (possibly) replaced by the best neighbor.

**first-improvement:** when finding the first better neighbor solution (i.e. whose cost value is smaller than that of the current solution), the current solution is replaced by this one.

In [118], the authors mention that empirically (when applying both strategies on many applications), in most of the cases, both strategies reach the same final solution, but in general the *first-improvement* takes a smaller computational time. Besides, they observe that is more frequent the premature convergence to a non-global local optimum by using *best-improvement* than *first-improvement*.

One important characteristic of GRASP is its low parametrization; few parameters need to be set and tuned. This implies that the main effort can be focused on implementing efficient data structures to obtain fast iterations. Let us analyze the influence of the GRASP parameters and the RCL construction.

A GRASP algorithm finalizes once performed $MaxIter$ iterations. Clearly, the probability of finding a new solution improving the currently best one decreases with the number of iterations already computed, the quality of the best solution found may only improve with the latter. In general, the computation times from iteration to iteration are relatively similar, therefore the total computation time depends linearly on $MaxIter$. Thus, when increasing $MaxIter$, the global computation time will be increased as well as the probability of finding better solutions.

At any GRASP iteration, let us denote by $c(e)$ the incremental cost associated with the insertion of element $e \in E$ into the solution under construction and by $c_{min}$ and $c_{max}$ the smallest and the largest incremental costs respectively. There are two main variants to compute the RCL used in the construction phase. Next, we will describe both approaches.

i) Given a positive integer $ListSize$, the RCL is composed of the $ListSize$ elements of $E$ with the best (i.e. smallest) incremental costs. In this case, we say that the RCL is cardinality-based. The size of the RCL can be smaller than $ListSize$ since, depending on the instance, we could not get to compute exactly the $ListSize$ best elements.

ii) The second variant uses a threshold parameter denoted by $\alpha \in [0, 1]$. In this case the size of the RCL is dynamically adapted according to the quality of the elements to be added (we say that the RCL is value-based). Fixed $\alpha$, the RCL is formed by all "feasible" elements $e \in E$ which can be inserted into the partial solution under construction without losing feasibility and whose quality is superior to the threshold value; that is to say:

$$e \in \text{RCL} \Leftrightarrow c(e) \in [c_{min}, c_{min} + \alpha(c_{max} - c_{min})].$$

If we set $\alpha = 0$ the resulting algorithm is purely greedy, and with $\alpha = 1$ we obtain a random construction. Hence, we can infer that $\alpha$ regulates the amounts of greediness and randomness in the construction phase.

For further details of GRASP the reader may consult the references [42, 44, 45, 98, 118, 119], which provide an extensive analysis of the GRASP metaheuristic based on many applications. Topics discussed include: successful implementation techniques, parameter tuning strategies, alternative solution construction mechanisms, techniques to speed up the local search, reactive GRASP, cost perturbations, bias functions, memory and learning, local search on partially constructed solutions, hashing, filtering, implementation strategies of memory-based intensification and post-optimization techniques using path-relinking, hybridizations with other metaheuristics and parallelization strategies.

## 2.4 Random Neural Network

The Random Neural Network (RNN) is a novel model introduced by Gelenbe [58, 59, 64]; it can be applied in optimization as well as learning settings. The RNN differs substantially from the existing connexionist models. The main different with respect to other existing models is that it can be solved numerically (by computing a fixed point equation) very easily and without using step-by-step Monte Carlo simulations. Specifically, in optimization problems, the RNN has been shown to be efficient and computationally very

fast compared to the classical existing models [61, 60, 65, 62, 63]. In particular, the RNN model has already been applied to the following NP-Hard combinatorial optimization problems:

- *Traveling Salesman Problem* (TSP). In this case, it has been shown in [63] that dynamical RNN obtains optimal or near-optimal solutions in most of the benchmark instances tested.

- *Minimum Vertex Covering Problem* (MVCP). In [65], the authors compare the performances of RNN, the conventional Greedy Algorithm, the Hopfield network, and Simulated Annealing, when applied to the MVCP; the results reveal that the results obtained by RNN heuristic are significantly superior to the others in terms of overall optimization.

- *Steiner Problem in Graphs* (SPG). In [62], the authors use RNN as improver of solutions delivered by the classical SPG heuristics: *Minimum Spanning Tree Heuristic* (MSTH) and the *Average Distance Heuristic* (ADH) applied to a test set composed of randomly generated graphs. They report that the obtained solutions have a better quality than the corresponding starting heuristic, attaining a difference in cost of less that one or two percentage points away from the optimal.

The RNN model works in the following way. Signals in the form of impulses (or spikes) of unit amplitude circulate among the neurons. Positive signals represent excitation, whereas negative signals represent inhibition of the receiving neuron. In this way, an excitatory impulse is interpreted as a "+1" signal, while an inhibitory impulse is interpreted as a "-1" signal. Each neuron $i$ has associated a state $k_i(t)$, which is its potential at time $t$, represented by a non-negative integer. The state of the $n$-neuron network at time $t$, is represented by the vector of non-negative integers $k(t) = (k_1(t), \ldots, k_n(t))$. The values of the state vector and the $i-th$ neuron's state are usually denoted by $k$ and $k_i$ respectively. By definition, a neuron $i$ is excited iff its potential is strictly positive. If a neuron $i$ is excited it can transmit impulses to other neurons or out of the network (we say neuron $i$ "fires"). The neuron's potential is increased when an excitation signal arrives to the neuron and is decreased when an inhibition signal arrives. Neural potential also decreases when the neuron fires. If a neuron $i$ emits an impulse, whether it be an excitation or an inhibition, it will lose potential by one unit, going from the state whose value is $k_i$ to the state of value $k_i - 1$.

Each neuron $i$, if it is excited, emits impulses at random, separated by intervals distributed as an exponential distribution with constant rate. That is to say, the impulses will be sent out as a Poisson process with rate $r_i$, with independent, identical exponentially distributed inter-impulse intervals. The impulses transmitted will arrive at neuron $j$ as excitation signals with probability $p_{ij}^+$, and as inhibitory signals with probability $p_{ij}^-$. A neuron's transmitted impulse may also leave the network with probability $d_i$, therefore $d_i + \sum_{j=1}^{n}(p_{ij}^+ + p_{ij}^-) = 1$. Let $i$ and $j$ be two neurons, the excitatory firing rate from $i$ to $j$ and the inhibitory firing rate from $i$ to $j$ are given by: $\varrho_{ij}^+ = r_i p_{ij}^+$ and $\varrho_{ij}^- = r_i p_{ij}^-$ respectively. Thus, the firing rate of neuron $i$ is $r_i = \sum_{j=1}^{n}(\varrho_{ij}^+ + \varrho_{ij}^-)$. The matrices $\mathcal{W}^+ = \{\varrho_{ij}^+\}$ and $\mathcal{W}^- = \{\varrho_{ij}^-\}$ can be viewed as being analogous

to the synaptic weights in connectionist models, although they specifically represent rates of excitatory and inhibitory impulse emission. Let us note that they are non-negative since their values are products of rates and probabilities.

Moreover, exogenous excitatory and inhibitory signals arrive at neuron $i$ according to Poisson processes with rates $\alpha_i$ and $\beta_i$ respectively. Let us notice that this is a recurrent network model, i.e. a network which is allowed to have feedback loops, and the network has an arbitrary topology. Figure 2.4 shows a typical neuron in the RNN using the model parameters that have been exposed above. In the figure, only the transitions to and from a single neuron $i$ are considered. Symmetrically, any other neuron has the same model that $i$.

Next, we summarize the dynamics of the RNN model by considering the possible state transitions. The neuron's state $k_i(t)$ can be modified when occurring certain transitions, more precisely:

i) The potential $k_i(t)$ of a neuron will decrease by one whenever it fires an excitation signal or an inhibition signal. Also, when an exogenous inhibitory signal arrives from outside the network to neuron $i$, its potential diminishes to $k_i(t) - 1$. In addition, neuron $i$ will decrease its potential $k_i(t)$ by one whenever it receives an inhibitory impulse from another neuron $j$.

ii) When arriving an exogenous excitatory signal from outside, or an excitatory impulse from another neuron within the network will result in incrementing the neuron potential by one, obtaining thus $k_i(t) + 1$.

iii) The value of $i - th$ neuron's state remains unchanged when none of the previous events occur.

In addition, there are two boundary conditions which prevent some of the transitions from occurring; these are:

- A neuron can fire only when it has positive potential (as mentioned previously).
- When the neuron has a potential of zero, the arrival the new inhibitory signals does not decrease its value further.

The analysis of the RNN model is focused on two measures: the probability distribution of network state $p(k,t) = Pr[k(t) = k]$ and the marginal probability that neuron $i$ is excited $q_i(t) = Pr[k_i(t) > 0]$. The behavior of the model is described by an infinite system of Chapman-Kolmogorov equations for discrete state-space continuous time Markovian systems. The information in the RNN is represented by the frequency at which the signals travel. The neurons may be considered as *frequency modulators and demodulators*. Intuitively, each neuron of this model is also a frequency modulator, since neuron $i$ sends out excitatory and inhibitory signals at rates (or frequencies) $q_i(t)r_ip_{ij}^+$, $q_i(t)r_ip_{ij}^-$ to any neuron $j$. In this

way, when neuron $i$ is excited, it will send signals to neuron $j$ at a frequency $\varrho_{ij} = \varrho_{ij}^{+} + \varrho_{ij}^{-}$. These signals will be emitted at exponentially distributed random intervals. In turn, each neuron behaves as a non-linear frequency demodulator since it transforms the incoming excitatory and inhibitory signal trains' rates into an "amplitude", which is $q_i(t)$, the probability that neuron $i$ is excited at time $t$.

The asymptotic analysis of the RNN model consists of computing the stationary probability distribution, i.e. the limits:

$$p(k) = \lim_{t \to +\infty} p(k,t),$$
$$q_i = \lim_{t \to +\infty} q_i(t),\ \forall i \in (1,\ldots,n).$$

Gelenbe proves the following theorem and introduces equations which provide the way to iteratively compute the stationary probability distribution for the RNN model.

**Theorem 2.4.1** *The vector $q = (q_1,\ldots,q_n)$ satisfies the system of nonlinear equations:*

$$q_i = \frac{\lambda_i^{+}}{r_i + \lambda_i^{-}},\ \forall i \in 1,\ldots,n. \tag{2.1}$$

*where $\lambda_i^{+}$ and $\lambda_i^{-}$ satisfy the following simultaneous equations:*

$$\lambda_i^{+} = \sum_j q_j \varrho_{ji}^{+} + \alpha_i, \tag{2.2}$$
$$\lambda_i^{-} = \sum_j q_j \varrho_{ji}^{-} + \beta_i. \tag{2.3}$$

In addition, another theorem introduced by Gelenbe guarantees the vector $q$ computation, since $\lambda_i^{+}$ and $\lambda_i^{-}$ can always be computed. By replacing $\lambda_i^{+}$ and $\lambda_i^{-}$ in equation $(2.1)$, we obtain a fixed point equation $F(q) = q$, whose explicit expression is given by:

$$q_i = \frac{\sum_{j,j\neq i}(q_j \varrho_{ji}^{+} + \alpha_i)}{r_i + \sum_{j,j\neq i}(q_j \varrho_{ji}^{-} + \beta_i)},\ \forall i \in 1,\ldots,n.$$

When iteratively solving the equation $F(q) = q$, if in certain iteration we get a value $q_i > 1$, then we force $q_i = 1$ until the convergence (we say neuron $i$ is saturated). Empirically, in most of the cases convergence is reached in few iterations. Furthermore, let us notice that the fixed point equation $F(q) = q$ can be solved in parallel, allowing thus to reduce significantly the computation time. For further details of RNN, the reader may consult the references [7, 58, 59, 64].

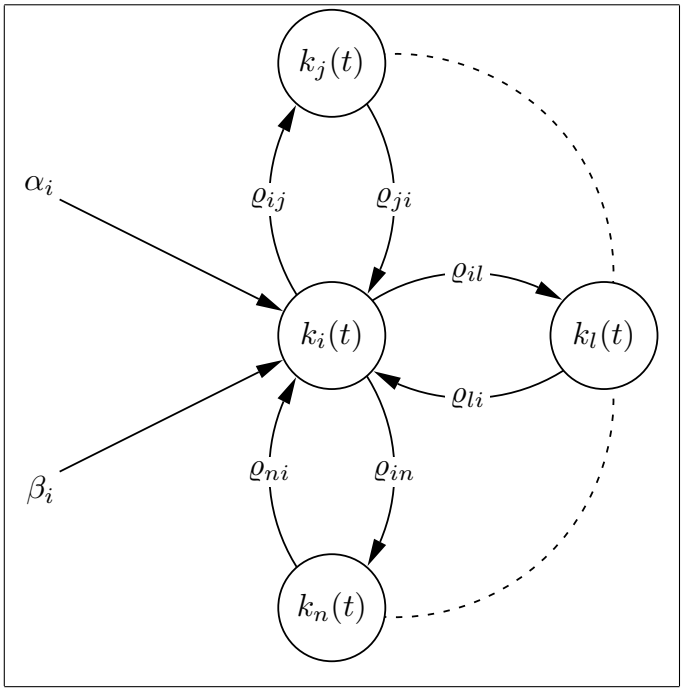



Figure 2.4: Representation of a neuron in the RNN model.

# Chapter 3

# The Access Network Design Problem

## 3.1 Introduction

A wide area network (WAN) can be seen as a set of sites and a set of communication lines that interconnect the sites. A typical WAN is organized as a hierarchical structure integrating two levels: the *backbone network* and the *access network* composed of a certain number of *local access networks* [108]. Figure 3.1 shows an example of a WAN topology. Each local access network usually has a tree-like structure, rooted at a single site of the backbone, and connects users (terminal sites) either directly to this backbone site or to a hierarchy of intermediate concentrator sites which are connected to the backbone site. The backbone network has usually a meshed topology, and its purpose is to allow efficient and reliable communication between the switch sites that act as connection points for the local access networks.

Assume the backbone network fixed. Let $S_C$ be the set of sites where concentrator equipment can be installed in order to diminish the cost of the access network and $S_T$ the set of terminal sites (the clients). Informally, if we consider the network of feasible connections on the WAN as a weighted (by costs), undirected graph, the Access Network Design Problem (denoted by ANDP) consists of finding a subgraph of minimum cost such that $\forall s_t \in S_T$ there exists a path from $s_t$ to the backbone network. In Appendix A, we prove the equivalence between the problem of designing the global access topology when considering all the switch nodes, and the problem of designing the global access topology where the backbone network is shrunk to a simple fictitious node. To simplify the presentation, the analysis and the algorithms, we collapse the backbone into a single fixed node $z$. We will focus over this last model. We introduce the notation used to formalize the problem:

- $S = S_T \cup S_C \cup \{z\}$ is the set of all nodes.
- $C = \{c_{ij}\}_{i,j \in S}$ is the matrix which gives for any pair of sites of $S$, the cost of laying a line between them. When the direct connection between $i$ and $j$ is not possible, we set $c_{ij} = \infty$.

- $E = \{(i,j); \forall i,j \in S$ such that $c_{ij} < \infty\}$ is the set of feasible connections between sites of $S$.
- $G_A = (S,E)$ is the graph of feasible connections on the Access Network.

**Definition 3.1.1 (Access Network Design Problem - ANDP)** *We define the Access Network Design Problem ANDP$(G_A(S,E),C)$ as the problem of finding a subgraph $\mathcal{T} \subset G_A$ of minimum cost such that $\forall s_t \in S_T$ there exists a unique path from $s_t$ to node $z$ and such that terminal sites can not be used as intermediate nodes (they must thus have degree 1 in the solution). We will denote by $\Gamma_{ANDP}$ the space of feasible solutions associated with the problem.*

This problem belongs to the NP-Hard class, this will be proved (in the next section) by reducing the Steiner Problem in Graphs to it.

In this chapter we propose several polynomial time heuristics based on the GRASP methodology for approximately solving the ANDP. In Section 3.2 we introduce the proof of NP-completeness for ANDP. In Section 3.3 we give two different alternative algorithms for the construction phase. In Section 3.4 we provide two algorithms for the local search phase. In particular, we present a RNN model which is used in a hybrid way GRASP-RNN, more precisely, the RNN approach is suitably customized in order to work as a variant of local search. Section 3.5 presents the GRASP metaheuristics yielded by combining the construction algorithms with the local search algorithms. Section 3.6 includes computational results obtained on a large test-set of problem instances, including topologies with hundreds of nodes. Discussions, conclusions and future work are the purpose of Section 3.7.

## 3.2 ANDP NP-Completeness

Before introducing the demonstration of ANDP NP-completeness, we will define formally the *Steiner Problem in Graphs*.

**Definition 3.2.1** *Let $G = (V,E)$ be a connected undirected graph, where $V$ is the set of nodes and $E$ denotes the set of edges. Given a non-negative weight function $C : E \rightarrow \mathbb{R}$ associated with its edges and a subset $T \subseteq V$ of terminal nodes, the Steiner problem $SPG(V,E,C,T)$ consists in finding a minimum weighted connected subgraph of $G$ spanning all terminal nodes in $T$.*

**Proposition 3.2.2 (ANDP NP-completeness)** *The ANDP belongs to the NP-Complete class.*

**Proof.** We will demonstrate that the ANDP is NP-Complete by reducing the Steiner Problem in Graphs (SPG) to it.
Let $SPG(V,E,T,C)$ be a Steiner Problem instance. For this, we build an ANDP instance as follows.

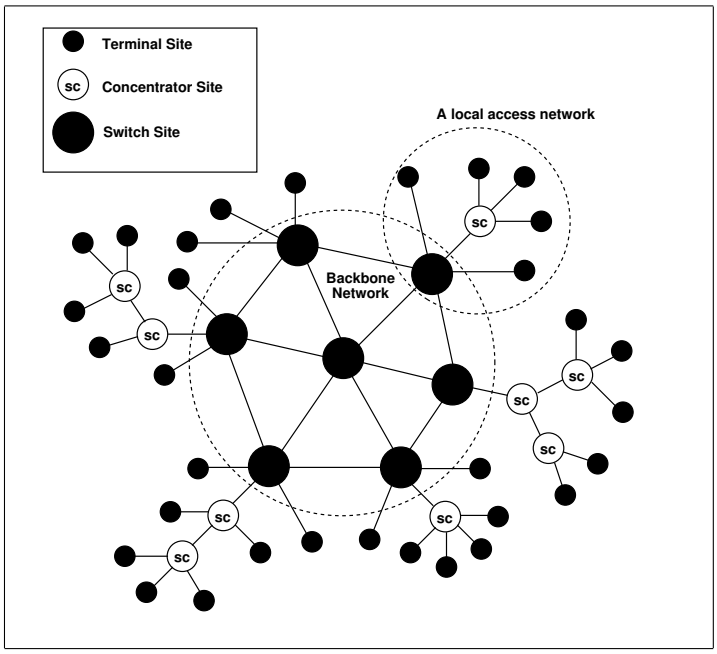



Figure 3.1: A WAN topology.

1) we select randomly a terminal $\hat{t} \in T$ and we associate with it the node $z$ in the ANDP.

2) each terminal $i \in (T \setminus \{\hat{t}\})$ has associated two sites in the ANDP: a terminal site $s_t^i$ and a concentrator site $s_c^i$. We introduce an edge with zero cost between $s_t^i$ and $s_c^i$ in the ANDP.

3) each Steiner node $i \in (V \setminus T)$ has associated a concentrator site $\bar{s}_c^i$ in the ANDP.

4) an edge $(i, j) \in E$, with $i, j \in (T \setminus \{\hat{t}\})$, has associated an edge between $s_c^i$ and $s_c^j$ with the same cost in the ANDP.

5) an edge $(i, j) \in E$, with $i \in (T \setminus \{\hat{t}\})$ and $j \in (V \setminus T)$, has associated an edge between $s_c^i$ and $\bar{s}_c^j$ with the same cost in the ANDP.

6) an edge $(i, j) \in E$, with $i, j \in (V \setminus T)$, has associated an edge between $\bar{s}_c^i$ and $\bar{s}_c^j$ with the same cost in the ANDP.

7) an edge $(i, \hat{t}) \in E$, with $i \in T$, has associated an edge between $s_c^i$ and $z$ with the same cost in the ANDP.

8) an edge $(i, \hat{t}) \in E$, with $i \in (V \setminus T)$, has associated an edge between $\bar{s}_c^i$ and $z$ with the same cost in the ANDP.

In the resulting ANDP instance we have two classes of concentrators: those associated with the terminal nodes and those associated with the Steiner nodes, denoted by $W_C$ and $\bar{S}_C$ respectively. The set of terminal sites, the set of feasible connections and the matrix of costs on the access network are denoted by $S_T$, $U$ and $A$ respectively. We have then $ANDP(S_T \cup W_C \cup \bar{S}_C \cup \{z\}, U, A)$. Clearly, the transformation process is polynomially computable. Now, we must prove that any minimal feasible solution of the SPG induces a minimal feasible solution of the corresponding ANDP (having the same cost) and reciprocally.
($\Rightarrow$) Let $\mathcal{T}$ be a minimal feasible solution of the SPG. Let $\mathcal{H}$ be the network built by applying the transformations $1-8$ to $\mathcal{T}$. Then,

i) it is easy to see that, by construction, the network $\mathcal{H}$ is a minimal feasible solution for the ANDP (deleting an edge of $\mathcal{H}$ the feasibility is lost). Figure 3.2 illustrates this correspondence; for the first graph, the black nodes represent the terminal nodes in the SPG whereas the white nodes represent Steiner nodes. In the corresponding ANDP instance, the greatest black node is the fixed site $z$, the other black nodes are terminal sites, the white and round nodes are concentrator sites and the square nodes are concentrator sites introduced when applying step $3$ of the previous transformation process.

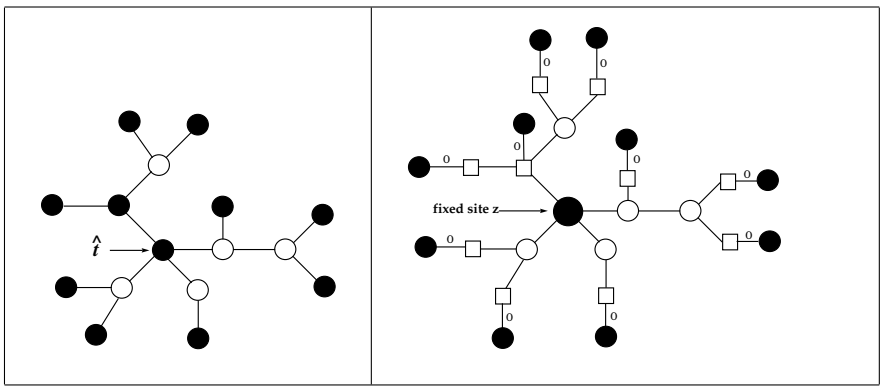



Figure 3.2: A SPG minimal feasible solution and its corresponding ANDP topology.

ii) $\text{COST}(\mathcal{T}) = \text{COST}(\mathcal{H})$ since the new connections introduced by the transformation (step 2) have all cost zero,

concluding therefore the direct.
($\Leftarrow$) Let $\mathcal{H}$ be a minimal feasible solution of the ANDP. We built the corresponding minimal feasible solution of the SPG by means of the inverse transformation process:

a) $z$ has associated a unique terminal node $\hat{t} \in T$ in the SPG.

b) each edge $(s_t^i, s_c^i) \in \mathcal{H}$, with $s_t^i \in S_T$ and $s_c^i \in W_C$, has associated in the SPG a unique terminal node $i \in T$. Notice that the edge $(s_t^i, s_c^i)$ necessarily belongs to any feasible solution of the ANDP since $s_c^i$ is the unique adjacent site to $s_t^i$ in the network $\mathcal{G} = (S_T \cup W_C \cup \bar{S}_C \cup \{z\}, U)$ .

c) each site $\bar{s}_c^i \in \mathcal{H}$, with $\bar{s}_c^i \in \bar{S}_C$, has associated in the SPG a unique Steiner node $i \in (V \setminus T)$.

d) each edge $(s_c^i, s_c^j) \in \mathcal{H}$, with $s_c^i, s_c^j \in W_C$, has associated in the SPG a unique edge $(i, j) \in E$, $i, j \in T$.

e) each edge $(s_c^i, \bar{s}_c^j) \in \mathcal{H}$, with $s_c^i \in W_C$, $\bar{s}_c^j \in \bar{S}_C$, has associated in the SPG a unique edge $(i, j) \in E$, $i \in T$, $j \in (V \setminus T)$.

f) each edge $(\bar{s}_c^i, \bar{s}_c^j) \in \mathcal{H}$, with $\bar{s}_c^i, \bar{s}_c^j \in \bar{S}_C$, has associated in the SPG a unique edge $(i, j) \in E$, $i, j \in (V \setminus T)$.

g) each edge $(s_c^i, z) \in \mathcal{H}$, with $s_c^i \in W_C$, has associated in the SPG a unique edge $(i, \hat{t}) \in E$, $i \in T$.

h) each edge $(\bar{s}_c^i, z) \in \mathcal{H}$, with $\bar{s}_c^i \in \bar{S}_C$, has associated in the SPG a unique edge $(i, \hat{t}) \in E$, $i \in (V \setminus T)$.

Let $\mathcal{T}$ be the resulting network of the inverse transformation by applying steps $\mathrm{a} - \mathrm{h}$. This is clearly a feasible and minimal solution for the SPG. Figure 3.3 illustrates this transformation. In the access topology the sites are labeled in the following way: $z$ is the fixed site modeling the backbone network, a label $s_c$ indicates a concentrator site belonging to $\bar{S}_C$, and the black (and square) nodes represent the pairs of sites $(s_t^i, s_c^i)$, with $s_t^i \in S_T$ and $s_c^j \in W_C$. For the corresponding SPG instance, we have that the black nodes represent the terminal nodes whereas the white nodes represent Steiner nodes. Let $\mathcal{B} = \{(s_t^i, s_c^i) \in \mathcal{H} | s_t^i \in$

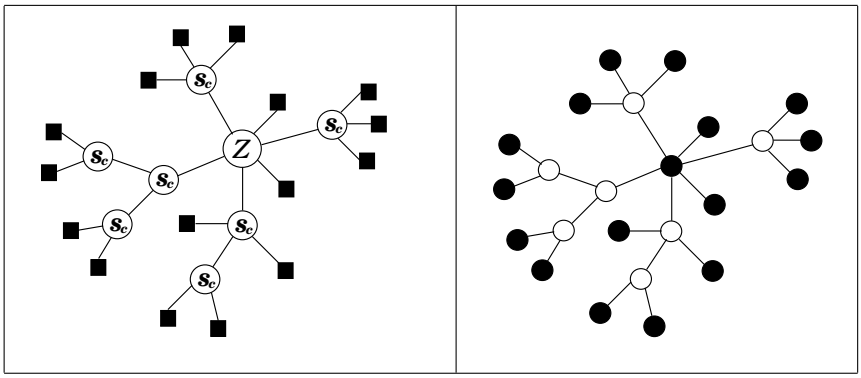



Figure 3.3: An ANDP minimal feasible solution and its corresponding SPG topology.

$S_T, s_c^i \in W_C\}$ be the set of edges in $\mathcal{H}$ associated with the terminal nodes in $\mathcal{T}$. Since the set $\mathcal{B}$ has cost zero and the other edges of $\mathcal{H}$ have the same cost that their corresponding ones in $\mathcal{T}$, we have the relation:

$$\textsc{cost}(\mathcal{T}) = \textsc{cost}(\mathcal{H}) - \textsc{cost}(\mathcal{B}) = \textsc{cost}(\mathcal{H}),$$

as required and completing the reciprocal.
Finally, since Karp proved [85] that SPG is NP-Complete, therefore ANDP is also NP-Complete.

**QED**

In the following sections, we will customize the different GRASP components to design different algorithms for solving approximately the ANDP. In particular, we design two path-based construction phases, a minimum spanning tree based local search phase and a RNN (Random Neural Network) based local search phase. Combining these two construction phases with the two local search phases yields four versions of GRASP for the ANDP. Next, we present the construction phase with the two proposed strategies.

## 3.3 ANDP Construction Phase Algorithms

Here, we introduce two construction phases for the ANDP. Both approaches are path-based but the used methodologies are structurally different. The proposed algorithms are called $\mathrm{ANDP\_ConstPhase1}$ and $\mathrm{ANDP\_ConstPhase2}$ and they will be described below in detail.

### 3.3.1 Algorithm ANDP_ConstPhase1

The $\mathrm{ANDP\_ConstPhase1}$ is a greedy randomized algorithm which works by iteratively choosing a terminal site and connecting it to the current partial solution by means of one of $k$ shortest paths.

More in detail, the algorithm (shown in Figure 3.4) takes as inputs the network $G_A$ of feasible connections on the access network, the matrix of connection costs $C$ and the GRASP parameter $k$ (the candidate list size). In order to introduce randomness in the selection process of the terminal sites to be added to the current solution, we assign (in line 1) a unique identifier $n_t$ to each terminal site $s_t \in S_T$. Line 2 initializes the current solution $\mathcal{T}_{sol}$ with the node $z$ (recall that $z$ models the backbone). The set $Y$ containing the terminal sites already added to $\mathcal{T}_{sol}$ is initialized empty. Iteratively the construction phase adds new terminal sites to the current solution $\mathcal{T}_{sol}$. On each iteration, the algorithm chooses randomly (in line 4) a terminal site $\bar{s}_t$ not yet included in $\mathcal{T}_{sol}$ and computes (in line 5) the $k$ shortest paths from $\bar{s}_t$ to $\mathcal{T}_{sol}$ using an algorithm proposed by Yen [137]. We remark that when computing these paths, terminal sites are not taken into account as intermediate nodes since they must have degree one in the solution. Line 5 stores these paths in a restricted candidate list $\mathcal{L}_p$. A path $p$ is randomly (and uniformly) selected from $\mathcal{L}_p$ in line 6 and added to $\mathcal{T}_{sol}$ in line 7. This process is repeated until all the terminal sites have been added; then the feasible solution $\mathcal{T}_{sol}$ is returned in line 9.

Let us observe that every feasible solution built by the $\mathrm{ANDP\_ConstPhase1}$ has a tree topology and therefore is minimal (deleting an edge the feasibility is lost). The following proposition formalizes this

```
Procedure ANDP_ConstPhase1;
Input: G_A = (S, E), C, k;

1  ∀s_t ∈ S_T a unique identifier n_t is assigned;
2  T_sol ← {z}; Y ← ∅;
3  while (Y \ S_T) ≠ ∅ do
4     s̄_t ← ArgMax{n_t | s_t ∈ (S_T \ Y)};
5     L_p ← the k shortest valid paths from s̄_t to T_sol;
6     p ← Select_Random(L_p);
7     T_sol ← T_sol ∪ {p}; Y ← Y ∪ {s̄_t};
8  end_while;
9  return T_sol;
end ANDP_ConstPhase1;
```

Figure 3.4: ANDP_ConstPhase1 pseudo-code.

point.

**Proposition 3.3.1** *If $\Gamma_{ANDP}$ is not empty, then the algorithm* $\mathrm{ANDP\_ConstPhase1}$ *builds a minimal feasible solution for the ANDP.*

**Proof.** Consider an ANDP instance such that $\Gamma_{ANDP} \neq \emptyset$. We will demonstrate the proposition by induction in $|Y|$. For this, we prove that in each iteration a new terminal site is added to $\mathcal{T}_{sol}$ and the resulting network has tree topology having as endpoints sites of $S_T$.

*Basic Step:* $|Y| = 0$. We compute lines 1-2. The network $\mathcal{T}_{sol}$ is composed of the isolated node $z$, satisfying thus the property.

*Induction Step:* $0 < |Y|$. As inductive hypothesis (I.H.) we have that if $|Y| < k \leq |S_T|$, then executing lines 4-7 the resulting network $\mathcal{T}_{sol}$ will have tree topology, fulfilling the previous conditions and containing a new terminal site. As inductive thesis the property is satisfied when $|Y| = k$ for certain iteration.

Let us suppose that in some iteration we have $|Y| = k$. We will analyze the following cases.

*Case 1.* $|Y| = k < |S_T|$. In this case, the condition in line 3 is TRUE and therefore the algorithm will execute the lines 4-7. In the previous iteration, we had $|Y| < k$ and by I.H. after executing lines 4-7 the network $\mathcal{T}_{sol}$ has tree topology. In the current iteration, line 4 selects a terminal site $\bar{s}_t$ not yet added to $\mathcal{T}_{sol}$ and lines 5-6 compute a path $p$ from $\bar{s}_t$ to $\mathcal{T}_{sol}$. Line 7 adds $p$ to the tree $\mathcal{T}_{sol}$. We remark that the candidate list $\mathcal{L}_p$ is computed (in line 5) of way of not using other terminal sites as intermediate nodes. Hence, it is easy to see that the terminal sites are endpoints in the current tree $\mathcal{T}_{sol}$. Thus, the resulting network has tree topology. In addition the set $Y$ is updated by adding to it the site $\bar{s}_t$.

*Case 2.* $|Y| = |S_T|$. In the previous iteration we had $|Y| < |S_T|$ and by I.H. after executing lines 4-7 the network $\mathcal{T}_{sol}$ has tree topology. In the current iteration, the condition in line 3 is FALSE and therefore $\mathcal{T}_{sol}$ is a tree containing $z$ and having the set $S_T$ as endpoints, as required and completing the proof.

**QED**

Figure 3.5 illustrates a ANDP_ConstPhase1 iteration.

- The first network models the current solution $\mathcal{T}_{sol}$ and a new terminal site $\bar{s}_t$ to be added to it. The broken lines with origin in $\bar{s}_t$ are the edges of the $k$ non-disjoint shortest paths from $\bar{s}_t$ to $\mathcal{T}_{sol}$. One of them will be selected in order to connect $\bar{s}_t$ to $\mathcal{T}_{sol}$.
- The second network shows $\mathcal{T}_{sol}$ with a path belonging to the restrict candidate list $\mathcal{L}_p$. Observe that (in this case) when adding this path, we introduce a new concentrator site to the solution.
- The third network shows another path from $\bar{s}_t$ to $\mathcal{T}_{sol}$. In this case, the path is a simple edge and therefore we do not introduce any concentrator site.

### 3.3.2 Algorithm ANDP_ConstPhase2

The $\mathrm{ANDP\_ConstPhase2}$ is an alternative algorithm for the ANDP construction phase. We design this algorithm based on the idea of Takahashi-Matsuyama algorithm [127], which is different from the $\mathrm{ANDP\_ConstPhase}$ algorithm in the way in which the restricted candidate list is built.

The $\mathrm{ANDP\_ConstPhase2}$ uses an auxiliary structure $\mathcal{P}$ in which are stored all the shortest paths from sites of $(S_T \cup \{z\})$ to sites in $S$ without using terminal sites as intermediate nodes. Figure 3.6 shows the preprocessing of this structure. Line 1 computes the subgraph induced by $(S_C \cup \{z\})$, which is denoted by $\mathcal{H}$. Lines 2-5 compute for all pairs of sites $i, j \in (S_T \cup \{z\})$ the shortest path connecting them in $G_A$ so that the terminal sites are not used as intermediate sites. For this, we compute in line 3 the auxiliary network $\mathcal{G} = \mathcal{H} \cup G_A(N(i) \cup N(j))$, and the shortest path communicating $i$ with $j$ on $\mathcal{G}$ is computed in line 4. Similarly, in lines 6-9 we compute for all pair of sites $i \in (S_T \cup \{z\})$ and $j \in S_C$ the shortest path connecting them without using intermediate terminal sites. Again, we compute in line 7 an auxiliary network $\mathcal{G} = \mathcal{H} \cup G_A(N(i))$, and the shortest path communicating $i$ with $j$ on $\mathcal{G}$ is computed in line 8. The matrix of computed paths $\mathcal{P}$ is returned in line 10.

The algorithm $\mathrm{ANDP\_ConstPhase2}$ (shown in Figure 3.4) takes as inputs the network $G_A$ of feasible connections on the access network, the matrix of connection costs $C$, the GRASP parameter $k$ (the candidate list size), and the matrix of paths $\mathcal{P}$ computed by $\mathrm{Preprocessing\_Algorithm}$. Lines 1-2 select randomly (and uniformly) a site from $(S_T \cup \{z\})$ as initial network. The solution in construction is denoted by $\mathcal{T}_{sol}$ and the auxiliary set $Y \subseteq (S_T \cup \{z\})$ denotes the sites already included to the current solution $\mathcal{T}_{sol}$. Let us note that all the sites of $(S_T \cup \{z\})$ necessarily must be in the solution. Iteratively the construction phase adds new sites of $(S_T \cup \{z\})$ to the current solution $\mathcal{T}_{sol}$. Each iteration works of the following way. In line 4 the algorithm searches for the $k$ nodes of $(S_T \cup \{z\}) \setminus Y$ which are nearest to the current solution $\mathcal{T}_{sol}$; the

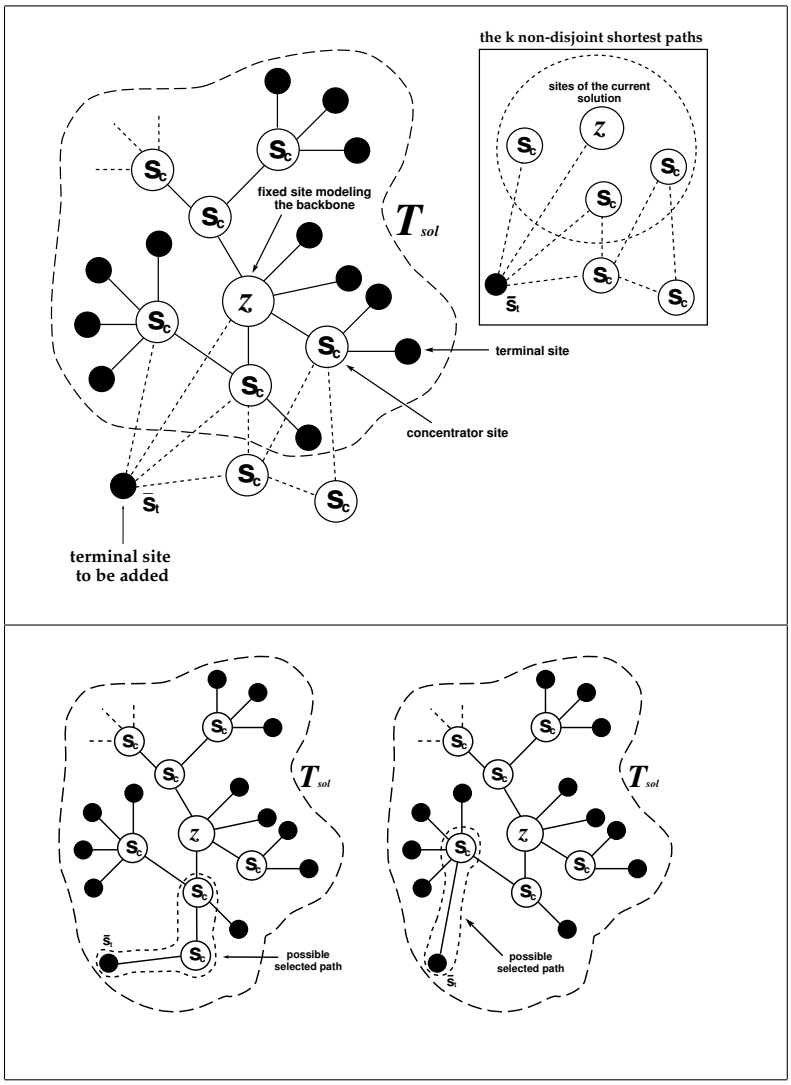



Figure 3.5: Example of ANDP_ConstPhase1 iteration.

corresponding shortest paths are extracted from $\mathcal{P}$ and stored in a restricted candidate list $\mathcal{L}_p$. A path $p$ is randomly (and uniformly) selected from $\mathcal{L}_p$ in line 6. Let $u$ be the endpoint of $p$ such that $u \notin \mathcal{T}_{sol}$. In line 7, we add $p$ to the current solution $\mathcal{T}_{sol}$ and the set $Y$ is updated by adding $u$ to it. This process is repeated until all the sites of $(S_T \cup \{z\})$ have been added to $\mathcal{T}_{sol}$. The built feasible solution $\mathcal{T}_{sol}$ is returned in line 9 (unless the space of feasible solutions is empty, in which case at least one terminal site will not be able to be connected with the solution in construction).

The construction phase algorithm $\mathrm{ANDP\_ConstPhase2}$ also builds a minimal feasible solution for the ANDP. The following proposition demonstrates this property.

**Proposition 3.3.2** *If $\Gamma_{ANDP}$ is not empty the algorithm $\mathrm{ANDP\_ConstPhase2}$ builds a minimal feasible solution for the ANDP.*

```
Preprocessing_Algorithm;
Input: G_A = (S, E), C;

1  H ← G_A(S_C ∪ {z});
2  for each i, j ∈ (S_T ∪ {z}) do
3      G ← H ∪ G_A(N(i) ∪ N(j));
4      P_{i,j} ← the shortest path from i to j in G;
5  end_for_each;
6  for each i ∈ (S_T ∪ {z}) and j ∈ S_C do
7      G ← H ∪ G_A(N(i));
8      P_{i,j} ← the shortest path from i to j in G;
9  end_for_each;
10 return P;
end Preprocessing_Algorithm;
```

Figure 3.6: Computation of set $\mathcal{P}$.

```
Procedure ANDP_ConstPhase2;
Input: G_A = (S, E), C, k, P;

1  v ← SelectRandom(S_T ∪ {z});
2  T_sol ← {v}; Y ← {v};
3  while Y \ (S_T ∪ {z}) ≠ ∅ do
4      L_p ← the shortest paths from the k nearest sites of (S_T ∪ {z}) \ Y to T_sol using P;
5      p ← Select_Random(L_p);
6      u ← the endpoint of p non-belonging to T_sol;
7      T_sol ← T_sol ∪ {p}; Y ← Y ∪ {u};
8  end_while;
9  return T_sol;
end ANDP_ConstPhase2;
```

Figure 3.7: ANDP_ConstPhase2 pseudo-code.

**Proof.** Again, let us consider an ANDP instance such that $\Gamma_{ANDP} \neq \emptyset$. We will demonstrate the proposition by induction in $|Y|$ (that is to say, on the number of sites from $S_T \cup \{z\}$ already added to $\mathcal{T}_{sol}$). Specifically, we will prove that on each iteration a site belonging to $S_T \cup \{z\}$ is added to $\mathcal{T}_{sol}$ and the resulting network has tree topology having as endpoints sites of $S_T$.

*Basic Step:* $|Y| = 0$. In lines 1-2, we select randomly a site of $S_T \cup \{z\}$ in order to initialize $\mathcal{T}_{sol}$. In addition, in the same line, the set $Y$ is initialized with the selected site.

*Induction Step:* $1 < |Y|$. The induction step is presented as follows. As inductive hypothesis we have that the property is fulfilled when $|Y| < k \leq |S_T| + 1$ for certain iteration. As inductive thesis the property is fulfilled when $|Y| = k$ for certain iteration. Let us analyze the following cases.

*Case 1.* $|Y| = k < |S_T| + 1$. In this case, the condition in line 3 is TRUE and therefore the algorithm will execute the lines 4-7. In the previous iteration, we had $|Y| < k$ and by I.H. after executing lines 4-7 the

network $\mathcal{T}_{sol}$ has tree topology having as endpoints sites of $S_T$. In the current iteration, line 4 computes the $k$ shortest paths from $(S_T \cup \{z\}) \setminus Y$ to $\mathcal{T}_{sol}$ without using terminal sites as intermediate nodes. For this, we use the matrix $\mathcal{P}$ which contains the shortest paths from $S_T \cup \{z\}$ to sites in $S$ not using intermediate terminal sites. Line 5 selects randomly one of them (which is denoted by $p$) and line 6 computes its endpoint non-belonging to $\mathcal{T}_{sol}$ (which is denoted by $u$). Clearly, if $z \in \mathcal{T}_{sol}$ the site $u$ is a terminal, otherwise $u$ can be a terminal or $z$. In any case, we add $p$ to $\mathcal{T}_{sol}$ in line 7, in addition $Y$ is updated adding $u$ to it. By construction, the resulting network has tree topology containing $z$ and all its leaves are terminal sites.
*Case 2.* $|Y| = |S_T| + 1$. The condition in line 3 is FALSE. In the previous iteration we had $|Y| = |S_T|$ and by I.H. after executing lines 4-7 the resulting network $\mathcal{T}_{sol}$ is a tree whose endpoints are the sites of $S_T$.

**QED**

Figure 3.8 illustrates a ANDP_ConstPhase2 iteration.

- The first network models the current solution $\mathcal{T}_{sol}$. Let us note that (in this case) the fixed site $z$ was not yet added to $\mathcal{T}_{sol}$.
- The second network shows $\mathcal{T}_{sol}$ when adding to it a new terminal site. Notice that the path connecting this terminal with $\mathcal{T}_{sol}$ contains the fixed site $z$. Thus, for the next iterations, when computing the shortest paths from the terminal sites not yet added to the current solution, we must also consider the fixed site $z$.
- The third network shows $\mathcal{T}_{sol}$ when adding to it a new terminal site by means of a simple edge.

## 3.4 ANDP Local Search Phase Algorithms

Generally the feasible solution built by the construction phase algorithm is not even a local optimum. In this way, the GRASP metaheuristic applies a local search phase in order to improve this solution. For the ANDP we designed two different local search phases. One based on minimum spanning tree approach and the other on Random Neural Network approach. In this section, we give a detailed description for both approaches. Before describing the first local search phase, we introduce some notation, a proposition, and a suitable structure for the neighborhood.

**Notation 3.4.1** *Given a network $\mathcal{H}$ and the matrix of connection costs $C$, we denote by $\mathrm{MST}(\mathcal{H}, C)$ a minimum spanning tree for the network $\mathcal{H}$.*

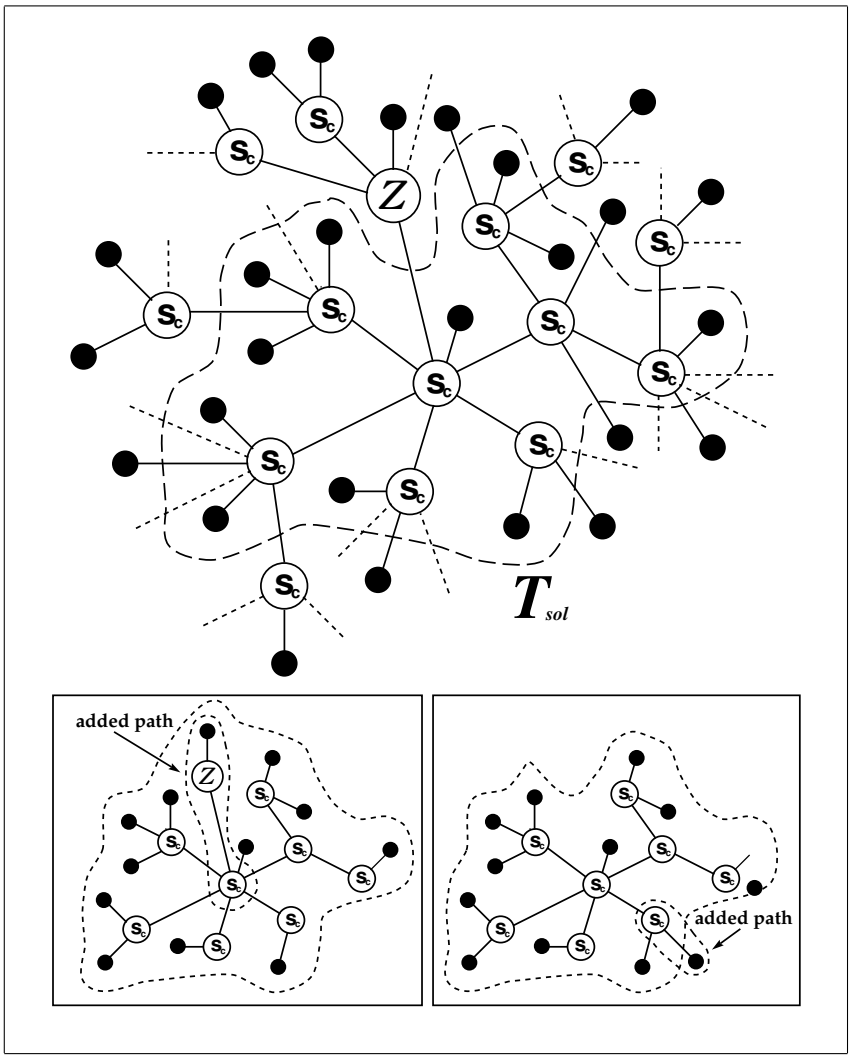



Figure 3.8: Example of ANDP_ConstPhase2 iteration.

**Proposition 3.4.2** *Let $\bar{S}_C \subseteq S_C$ be a subset of concentrator sites and $\mathcal{H} \subset G$ the sub-network induced by $(\{z\} \cup \bar{S}_C)$. The best feasible solution having $\bar{S}_C$ as set of concentrator sites is given by the network: $\mathcal{G} = U \cup \mathcal{T}_{MST}$, where $\mathcal{T}_{MST}$ is a minimum spanning tree for $\mathcal{H}$ and $U$ is the set of edges that connects the set of terminal sites $S_T$ to $\mathcal{T}_{MST}$ with minimum cost.*

**Proof.** Trivial, the set $U$ is the same for any other sub-network $\hat{\mathcal{G}} \subset \mathcal{H}$ spanning the sites $(\{z\} \cup \bar{S}_C)$.

**QED**

Let us notice that the network $\mathcal{G}$ (defined above) can be computed in polynomial time. In the following, solutions of the $ANDP(G_A(S, E), C)$ will be characterized by the associated set of concentrator sites and one of the corresponding minimum spanning trees. Accordingly, the search for the optimal access network will be reduced to search for the optimal set $S_C^{(\mathrm{opt})}$ of concentrator sites.

**Definition 3.4.3 (Neighborhood Structure)** *Let $\bar{S}_C \subseteq S_C$ be a subset of concentrator sites associated with a feasible solution of the $ANDP(G_A(S,E),C)$. The neighbors of a solution characterized by its set $\bar{S}_C$ of concentrator sites are defined by all the sets of concentrator sites which can be obtained by adding to $\bar{S}_C$ a new concentrator site, or by eliminating from $\bar{S}_C$ one of its concentrator sites.*

According to this definition; given a feasible solution $\mathcal{T}$ whose set of concentrator sites is $\bar{S}_C$, and given a concentrator site $s_c \in (S_C \setminus \bar{S}_C)$ we can compute the following neighbor solutions.

i) Let us denote by $\mathcal{H}_1 \subset G_A$ the sub-network induced by $\{z\} \cup \bar{S}_C \cup \{s_c\}$. Let $\mathcal{T}_1$ be a minimum spanning tree for $\mathcal{H}_1$. The neighbor solution produced by the insertion of $s_c$ is the network $\mathcal{T} = \mathcal{T}_1 \cup U$, where $U$ is the set of edges that connect the set of terminal sites $S_T$ to the network $\mathcal{T}_1$ with minimum cost. To compute $\mathcal{T}_1$ we use an algorithm proposed by Minoux [101] which has $O(|S| - |S_T|)$ average time and the set $U$ is computed in time $O(|S_T| \cdot |\bar{S}_C|)$. Thus, we reduce significantly the running times in comparison with the classic algorithms used to compute minimum spanning trees, such as Prim or Kruskal algorithms.

ii) Let us denote by $\mathcal{H}_2 \subset G_A$ the sub-network induced by $\{z\} \cup (\bar{S}_C \setminus \{s_c\})$. Since $\mathcal{H}_2$ cannot be connected, in order to guarantee feasibility, we only consider the connected component $\hat{\mathcal{H}} \subseteq \mathcal{H}_2$ such that $z \in \hat{\mathcal{H}}$. Clearly, the concentrator sites non-belonging to $\mathcal{H}_2$ cannot be used since they can induce a non-feasible solution. Let $\mathcal{T}_2$ be a minimum spanning tree for $\hat{\mathcal{H}}$. The neighbor solution produced by the elimination of $s_c$ is the network $\mathcal{T} = \mathcal{T}_2 \cup U$, where $U$ is the same that in $(\mathrm{i})$. To compute $\mathcal{T}_2$ we use the Prim algorithm which has $O(|\bar{E}| + |\bar{S}_C| \log(|\bar{S}_C|)$ time-complexity, being $|\bar{E}|$ the number of edges in $\mathcal{H}_2$.

Next, we present the descriptions of both strategies which we will use in combined form in the MST based local search phase.

### 3.4.1 Neighborhood Strategy by concentrator site insertion

The algorithm (shown in Figure 3.9) takes as inputs the network $G_A$, the matrix $C$, the current solution $\mathcal{T}_{sol}$ and its set of concentrator sites $\bar{S}_C$. The $\mathrm{Insertion\_Neighborhood}$ searches for a better solution by means of the insertion of a new concentrator site into $\bar{S}_C$. In line $1$ we initialize the best neighbor solution with the current solution $\mathcal{T}_{sol}$. Lines $2-10$ searches for the best neighbor solution adding a new concentrator site $s_c \in (S_C \setminus \bar{S}_C)$ into $\bar{S}_C$. In line $3$, the sub-network $\mathcal{H}$ induced by the set of sites $(\{z\} \cup \bar{S}_C \cup \{s_c\})$ is computed. Notice that, this network includes all the possible topologies that have as set of concentrator sites a subset of $\bar{S}_C \cup \{s_c\}$. In line $4$ the minimum spanning tree for this network is computed using an algorithm by Minoux [101]. In line $5$, we connect to $\mathcal{T}$ (computed in line 4) all the terminal sites of $S_T$ by means

of edges of minimum cost from $S_T$ to $\mathcal{T}$. Clearly, the built network is feasible for the ANDP. Iteratively, the pendant concentrators are removed in line 6. In line 7, we compare whether the cost of the resulting network is smaller than the cost of the current best neighbor. If the new neighbor solution improves the best cost, then in line 8 the best cost, the best set of concentrator sites, and the best neighbor solution are updated. Once all the possible insertions of a new concentrator site have been considered, the procedure Insertion_Neighborhood returns the best found neighbor solution, its cost, and its set of concentrator sites.

```
Procedure Insertion_Neighborhood;
Input: G_A = (S, E), C, T_sol, S̄_C;
1   best ← COST(T_sol); Ŝ_C ← S̄_C; T_best ← T_sol;
2   for all s_c ∈ S_C \ S̄_C do
3     H ← sub-network induced by ({z} ∪ S̄_C ∪ {s_c});
4     T ← MST computed by Minoux_Algorithm(H, C);
5     Compute ∀s_t ∈ S_T:
          e ← the edge of minimum cost from s_t to T;
          T ← T ∪ {e};
6     Iteratively remove all concentrators from T with
      degree 1;
7     if (COST(T) < best) then
8         best ← COST(T); Ŝ_C ← S̄_C ∪ {s_c}; T_best ← T;
9     end_if;
10  end_for;
11  return best, Ŝ_C, T_best;
end Insertion_Neighborhood;
```

Figure 3.9: Neighborhood by insertion moves.

**Proposition 3.4.4 (**Insertion_Neighborhood **correctness)** *Let $\mathcal{T}_{sol}$ be a feasible solution for the ANDP and $\bar{S}_C$ its set of concentrator sites. The algorithm* Insertion_Neighborhood *builds the best neighbor solution whose set of concentrator sites belongs to $\mathcal{A} = \{\bar{S}_C \cup \{s_c\} | s_c \in S_C \setminus \bar{S}_C\}$.*

**Proof.** When running Insertion_Neighborhood, for each concentrator site $s_c \in S_C \setminus \bar{S}_C$ loop 2-10 computes the following:

i) $\mathcal{H}$ is the subnetwork induced by $\{z\} \cup \bar{S}_C \cup \{s_c\}$.

ii) $\mathcal{T}$ is the minimum spanning tree for $\mathcal{H}$.

iii) line 5 connects the sites of $S_T$ to $\mathcal{T}$ by means of edges the minimum cost. By Proposition 3.4.2, the resulting network is the best feasible solution having $\hat{S}_C \cup \{s_c\}$ as set of concentrators. In this way, once finalized loop 2-10, we have computed the best feasible solution belonging to the set:

$$\{\mathcal{T} \in \Gamma_{ANDP} | \text{such that } \text{CONCENTRATORS}(\mathcal{T}) \in \mathcal{A}\} .$$

In addition, in line 6 each computed feasible solution is improved by removing the pendant concentrators; the best of them is assigned to $\mathcal{T}_{best}$ in lines 7-9, and returned in line 11.

**QED**

Figure 3.10 is an example of a insertion move computed by $\mathrm{Insertion\_Neighborhood}$.

- The first network is the graph of feasible connections on the access network.
- The second network models a small access network $\mathcal{T}_{sol}$. This one is composed of ten concentrator sites (which are labeled with $s_c$) and sixteen terminal sites (the black nodes).
- The third network represents the solution $\mathcal{T}_{sol}$ without the terminal sites, and a new concentrator site (non-belonging to $\mathcal{T}_{sol}$) with its feasible connections (modeled by the broken lines) towards $z$ and the other concentrator sites belonging to $\mathcal{T}_{sol}$.
- The fourth network represents a minimal spanning tree for the network $\mathcal{H}$ (defined in 3.9). This tree can be computed by applying Minoux algorithm, which reuses efficiently the current topology in order to find the new spanning tree.
- The fifth network represents the new neighbor solution obtained by reconnecting with minimum cost the terminal sites to the computed minimum spanning tree. Note that there exists a redundant concentrator site (having degree 1) which can be removed preserving the feasibility. Moreover, two terminal sites were reassigned to another concentrator site.

### 3.4.2 Neighborhood Strategy by concentrator site elimination

The algorithm (shown in Figure 3.11) takes as inputs the network $G$, the matrix $C$, the current solution $\mathcal{T}_{sol}$ and its set of concentrator sites $\bar{S}_C$. The procedure denominated $\mathrm{Elimination\_Neighborhood}$ (shown in Figure 3.11) tries to find a better solution by means of the elimination of a concentrator site belonging to $\bar{S}_C$. As in the previous procedure, the best neighbor solution is initialized with the current solution $\mathcal{T}_{sol}$. The lines $2-11$ search for the better neighbor feasible solution eliminating a concentrator site $s_c \in \bar{S}_C$. In line 3 the sub-network $\mathcal{H}$ induced by the set of sites $(\{z\} \cup (\bar{S}_C \setminus \{s_c\}))$ is computed. Since this network can be unconnected, we compute in line 4 the connected component $\hat{\mathcal{H}} \subseteq \mathcal{H}$ containing the fixed node $z$ (which models the backbone network). It is easy to see that the concentrator sites not belonging to $\hat{\mathcal{H}}$ cannot be considered since they can induce a non-feasible solution for the ANDP. A minimum spanning tree for $\hat{\mathcal{H}}$ is computed in line 5. In line 6, we connect to $\mathcal{T}$ (computed in line 5) all the terminal sites of $S_T$ by means of edges of minimum cost from $S_T$ to $\mathcal{T}$. The resulting topology is feasible for the ANDP.

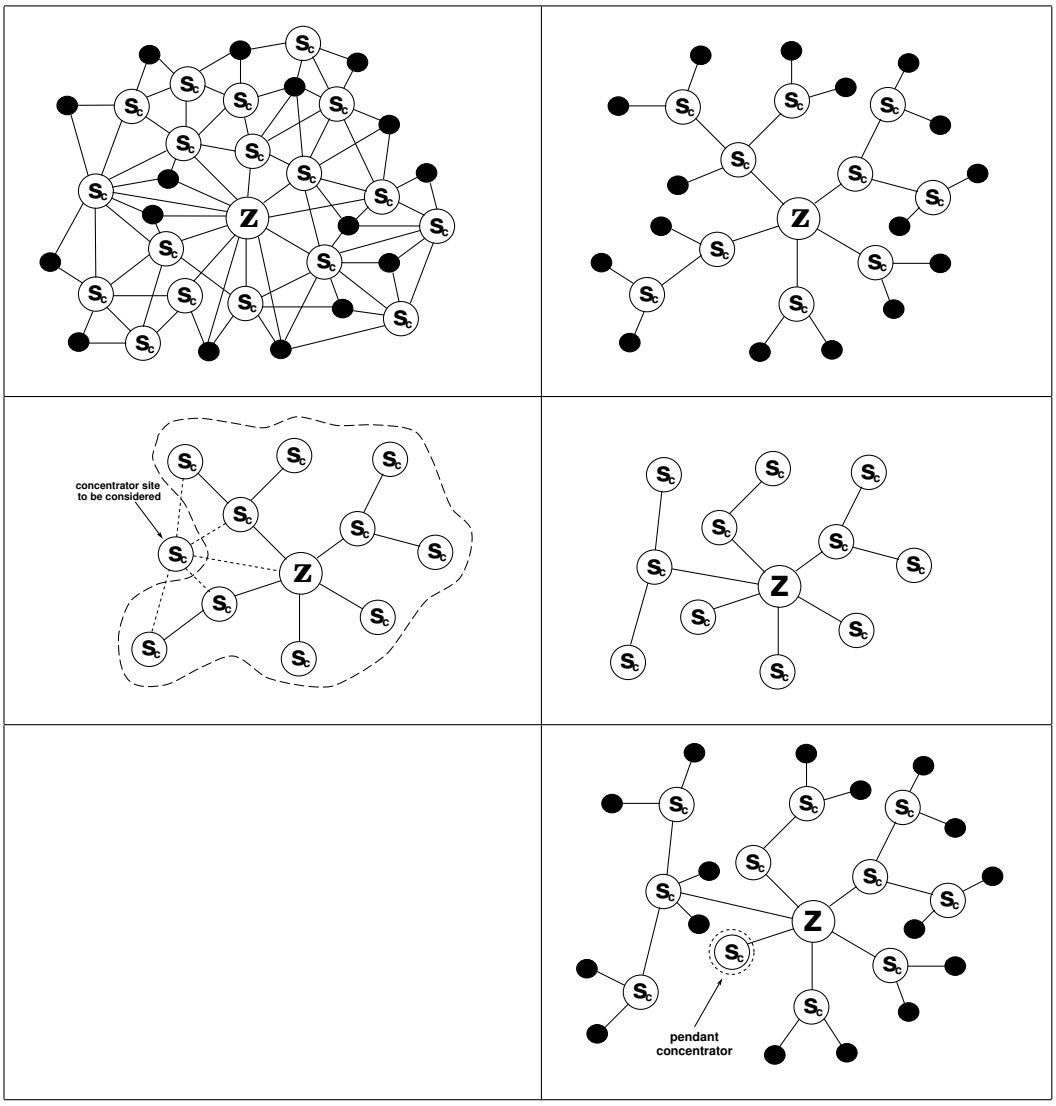



Figure 3.10: Example of Insertion_Neighborhood iteration.

The pendant concentrators are iteratively removed in line 7. In line 8, we compare whether the cost of the resulting network is smaller to the cost of the current best neighbor. If we found a better solution, in line 9 the best cost, the best set of concentrator sites, and the best solution are updated. Once all the possible eliminations of concentrator sites have been considered, the procedure $\mathrm{Elimination\_Neighborhood}$ returns the best found neighbor solution, its cost, and its set of concentrator sites.

**Proposition 3.4.5 (**$\mathrm{Elimination\_Neighborhood}$ **correctness)** *Let $\mathcal{T}_{sol}$ be a feasible solution for the ANDP and $\bar{S}_C$ its set of concentrator sites. The algorithm* $\mathrm{Elimination\_Neighborhood}$ *builds the best neighbor solution whose set of concentrator sites belongs to $\mathcal{B} = \{\bar{S}_C \setminus \{s_c\} | s_c \in \bar{S}_C\}$.*

```
Procedure Elimination_Neighborhood;
Input: G_A = (S, E), C, T_sol, S̄_C;
1   best ← COST(T_sol); Ŝ_C ← S̄_C; T_best ← T_sol;
2   for all s_c ∈ S̄_C do
3     H ← sub-network induced by ({z} ∪ (S̄_C \ {s_c}));
4     Ĥ ← connected comp. of H such that z ∈ Ĥ;
5     T ← MST(Ĥ, C);
6     Compute ∀s_t ∈ S_T:
          e ← the edge of minimum cost from s_t to T;
          T ← T ∪ {e};
7     Iteratively remove all concentrators from T with
      degree 1;
8     if (COST(T) < best) then
9         best ← COST(T); Ŝ_C ← (S̄_C \ {s_c}); T_best ← T;
10    end_if;
11  end_for;
12  return best, Ŝ_C, T_best;
end Elimination_Neighborhood;
```

Figure 3.11: Neighborhood by elimination moves.

**Proof.** When running $\mathrm{Elimination\_Neighborhood}$, for each concentrator site $s_c \in \bar{S}_C$ loop 2-11 computes the following:

i) $\mathcal{H}$ is the subnetwork induced by $\{z\} \cup (\bar{S}_C \setminus \{s_c\})$.

ii) $\mathcal{T}$ is the minimum spanning tree for the connected component $\hat{\mathcal{H}} \subseteq \mathcal{H}$ such that $z \in \hat{\mathcal{H}}$.

iii) line 6 connects to $\mathcal{T}$ the sites of $S_T$ by means of edges the minimum cost. Let us notice that the concentrator sites non-belonging to $\hat{\mathcal{H}}$ cannot be considered since when connecting a terminal site to one of them, we obtain a non-feasible solution (in the resulting network there are no paths connecting it with the node $z$). In this way, once finalized loop 2-11, we have computed the best feasible solution belonging to the set:

$$\{\mathcal{T} \in \Gamma_{ANDP} | \text{such that } \textsc{concentrators}(\mathcal{T}) \in \mathcal{B}\}\,.$$

In addition, in line 7 each computed feasible solution is improved by removing the pendant concentrators; the best of them is assigned to $\mathcal{T}_{best}$ in lines 8-10 and returned in line 12.

**QED**

Figure 3.12 is an example of a elimination move computed by $\mathrm{Elimination\_Neighborhood}$.

- The first network is the same access network presented previously.

- The second network represents the solution $\mathcal{T}_{sol}$ without one of its concentrators and without all the terminal sites. In addition, the broken lines represent all the possible feasible connections reconnecting the isolated concentrators (which appear when removing the concentrator site) to the fixed site $z$ and other concentrators present in the solution. We suppose here that it is possible to reconnect the isolated concentrator.

- The third network represents a minimal spanning tree for the network $\hat{\mathcal{H}}$ (defined in 3.11). This tree can be computed by applying Prim algorithm.

- Again, the fourth network represents the new neighbor solution obtained by reconnecting with minimum cost the terminal sites to the computed minimum spanning tree. Note that a terminal site was reassigned to another concentrator site.

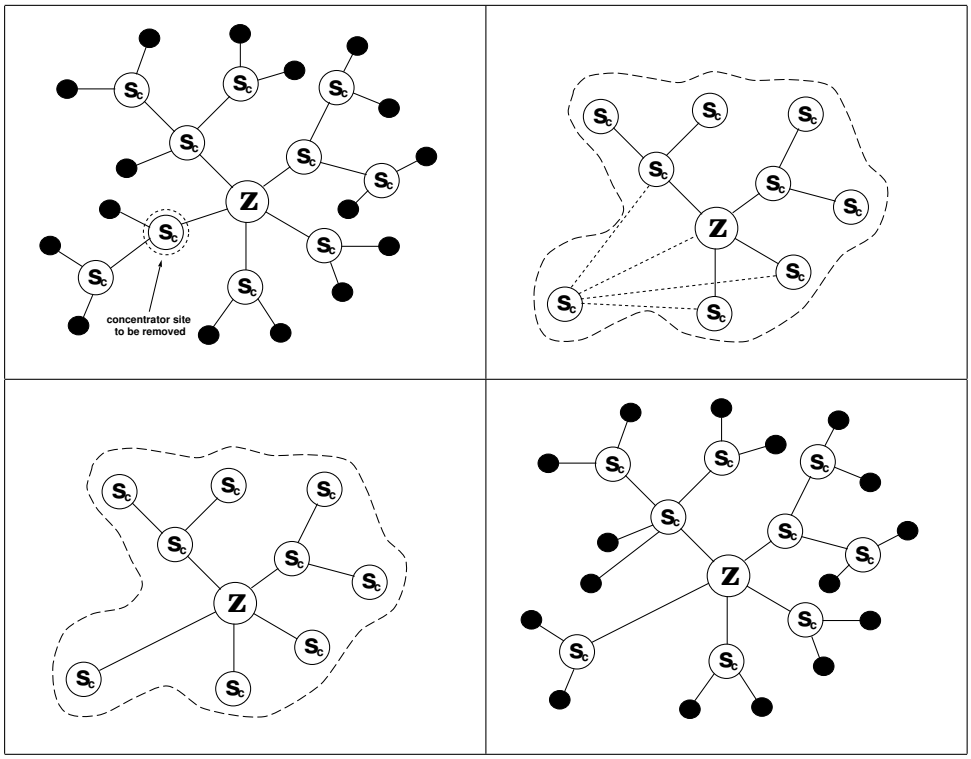



Figure 3.12: Example of Elimination_Neighborhood iteration.

### 3.4.3 MST based local search

Now, we will present the description of our MST based local search, which was designed by combining suitably the strategies exposed above.

Given a feasible solution $\mathcal{T}_{sol}$, the algorithm tries to find a better neighbor solution computing minimum spanning trees on insertions and eliminations of concentrator sites to/from the current solution. For this, we use in combined form the local search strategies described above. The proposed algorithm is called $\mathrm{ANDP\_MST\_Local\_Search}$ and its pseudo-code is shown in Figure 3.13. The algorithm has as input the network $G_A = (S, E)$ of feasible connections on the access network, the matrix of connection costs $C$, and a feasible solution $\mathcal{T}_{sol}$.

In line 1, we compute the set of concentrator sites belonging to $\mathcal{T}_{sol}$; this is denoted by $\bar{S}_C$. In line 2 the cost of $\mathcal{T}_{sol}$ is assigned to $cost\_sol$. In line 3, we call the procedure $\mathrm{Insertion\_Neighborhood}$ in order to find for neighbor solutions with smaller cost. As we mentioned previously, this procedure searches for a better neighbor solution by means of the insertion of a new concentrator site not belonging to $\bar{S}_C$. In line 4 we compare the cost of the current solution with the cost of the solution delivered by $\mathrm{Insertion\_Neighborhood}$. If a neighbor solution with smaller cost has been found by $\mathrm{Insertion\_Neighborhood}$, then the local search resumes from this new current solution executing from line 2. If no neighbor solution of better cost is found by $\mathrm{Insertion\_Neighborhood}$, then in line 5 we call the procedure $\mathrm{Elimination\_Neighborhood}$, which evaluates the search of neighbor solutions with smaller cost by means of the elimination of a concentrator site belonging to the current solution. In line 6 we compare the cost of the current solution with the cost of the solution found by $\mathrm{Elimination\_Neighborhood}$. Again, if a neighbor solution with smaller cost has been found by $\mathrm{Elimination\_Neighborhood}$, then the local search resumes from this new current solution executing from line 2. If no neighbor solution of better cost is found by $\mathrm{Elimination\_Neighborhood}$, then the best found neighbor feasible solution and its set of concentrator sites are returned in line 7.

Observe that the $\mathrm{GRASP\_ANDP}$ uses first the local search strategy based on concentrator site insertions and the local search strategy based on concentrator site eliminations is performed only when there are no improvements by insertions. This is motivated in the fact that the time complexity of $\mathrm{Insertion\_Neighborhood}$ is smaller than the time complexity of $\mathrm{Elimination\_Neighborhood}$.

**Procedure ANDP_MST_Local_Search;**
**Input:** $G_A = (S, E)$, $C$, $\mathcal{T}_{sol}$;

**1** $\bar{S}_C \leftarrow \mathrm{Concentrators}(\mathcal{T}_{sol})$;
**2** $cost\_sol \leftarrow \mathrm{COST}(\mathcal{T}_{sol})$;
**3** $[best, \bar{S}_C, \mathcal{T}_{sol}] \leftarrow \mathrm{Insertion\_Neighborhood}(G, C, \bar{S}_C, \mathcal{T}_{sol})$;
**4** **if** $(best < cost\_sol)$ **then goto line** 2;
**5** $[best, \bar{S}_C, \mathcal{T}_{sol}] \leftarrow \mathrm{Elimination\_Neighborhood}(G, C, \bar{S}_C, \mathcal{T}_{sol})$;
**6** **if** $(best < cost\_sol)$ **then goto line** 2;
**7** **return** $\bar{S}_C$, $\mathcal{T}_{sol}$;
**end ANDP_MST_Local_Search;**

Figure 3.13: MST based Local Search Algorithm.

**Proposition 3.4.6** *Let $\mathcal{T}_{sol}$ be the feasible solution returned by the algorithm* $\mathrm{ANDP\_MST\_Local\_Search}$ *and $\bar{S}_C$ its set of concentrator sites. Then, the following points are fulfilled:*

*i) $\nexists s_c \in \bar{S}_C$ such that the feasible solution induced by $\bar{S}_C \setminus \{s_c\}$ improves $\mathcal{T}_{sol}$.*

*ii) $\nexists s_c \in S_C \setminus \bar{S}_C$ such that the feasible solution induced by $\bar{S}_C \cup \{s_c\}$ improves $\mathcal{T}_{sol}$.*

**Proof.** By construction, when running $\mathrm{ANDP\_MST\_Local\_Search}$, it easy to see that the delivered solution $\mathcal{T}_{sol}$ satisfies the following points:

a) $\mathcal{T}_{sol}$ could not be improved by $\mathrm{Insertion\_Neighborhood}$, i.e. condition in line 4 is FALSE.

b) $\mathcal{T}_{sol}$ could not be improved by $\mathrm{Elimination\_Neighborhood}$, i.e. condition in line 6 is FALSE.

By Proposition 3.4.4, we have that $\mathcal{T}_{sol}$ is better than any feasible solution with set of concentrator sites belonging to: $\mathcal{A} = \{\bar{S}_C \cup \{s_c\} | s_c \in S_C \setminus \bar{S}_C\}$, implying thus $(i)$. In addition, by Proposition 3.4.5 $\mathcal{T}_{sol}$ is better than any feasible solution whose set of concentrator sites belongs to: $\mathcal{B} = \{\bar{S}_C \setminus \{s_c\} | s_c \in \bar{S}_C\}$, implying thus $(ii)$, as required and completing the proof.

**QED**

### 3.4.4 RNN based local search

We propose a local search which differs substantially from the classic local searches. We use a RNN model in the local search phase with the aim of capturing global connectivity information about the access network and to determine the order in which the concentrator sites non-present in the solution delivered by the construction phase are chosen one at time in order to try improve the solution delivered by the greedy construction phase. We denominate as $\mathrm{ANDP\_RNN\_Local\_Search}$ to the designed algorithm. Next, we give a detailed description for the underlaying neural network used and the corresponding algorithm.

The underlying RNN is defined as follows. There exists a neuron for each node of $S$. The values for the excitatory and inhibitory rates are defined as:

- $\varrho^{+}_{ij} = \frac{\bar{c}}{c_{ij}}$ if $(i,j) \in E$, where $\bar{c}$ is the average edge cost in the graph $G_A$,
- $\varrho^{-}_{ij} = 1$ if $(i,j) \notin E$ and
- $r_i = \sum_j (\varrho^{+}_{ij} + \varrho^{-}_{ij})$,

The other rates are zero and exogenous signals do not exist. The goal from these setting is to obtain information inherent to the access network global connectivity. With the neural network customized in

this manner, if a neuron is excited and has low connection costs with their neighboring neurons (i.e. high excitatory rates), it will have a greater excitatory influence on its neighborhood (the adjacent neurons). For the nodes that necessarily will integrate the solution (these are $S_T \cup \{z\}$) the associated neurons are artificially excited at each local search iteration by means of the assignment $q_i = 1, \forall i \in S_T \cup \{z\}$, this guarantees to solve the fixed point equation given by the Gelenbe theorem (which provides explicit equations for the stationary probability distribution associated with the RNN model). The idea on this customization is iteratively to choose as candidates to improve the current solution those concentrator sites (not yet considered) whose associated neurons have highest excitation stationary probability. It is to say, we select sequentially neurons that have not been analyzed previously with greatest value of $q$.

The algorithm takes as inputs the matrices $\mathcal{W}^+$ and $\mathcal{W}^-$ of excitatory and inhibitory rates respectively (which will be used by the RNN model), the solution $\mathcal{T}_{sol}$ (computed in the Construction Phase), the network $G_A$ of feasible connections on the access network and the matrix of connection costs $C$. The algorithm (shown in Figure 3.14) searches a better solution using the underlying neural network with the objective to determine iteratively the order in which to analyze each concentrator node non-present in $\mathcal{T}_{sol}$ and to evaluate the benefit of its inclusion in the current solution.

In line 1, we compute the set $I$ of concentrator sites belonging to $\mathcal{T}_{sol}$. In addition the set $J$ containing the sites whose associated neurons are artificially excited is initialized with the set $S_T \cup \{z\}$. We will consecutively add to $J$ the concentrator sites chosen at each iteration. Line 2 initializes the best neighbor solution $\mathcal{T}_{best}$ with $\mathcal{T}_{sol}$. Iteratively, the concentrator node (non belonging to $\mathcal{T}_{sol}$ and that has not been analyzed previously) selected as potential improver will be that one whose associated neuron has greater value of $q_j$ (asymptotically the most excited). For this, each iteration works as follows.

In line 4, we excite artificially all the neurons associated with sites of $J$ by assigning $q_s = 1$, $\forall s \in J$. This implies that their potentials are always positive being able to excite other neurons without losing potential. Line 5 computes the vector of asymptotic probabilities $q = (q_1, \ldots, q_n)$ by solving the fixed point equation given by Gelenbe theorem. In line 6 we select the concentrator site non-belonging to $J$ (i.e. not yet analyzed) whose associated neuron has highest value of $q_j$. This is the neuron asymptotically most excited. Let $s_c$ denote the selected site. In the same line the set $J$ is updated by adding the site $s_c$ to it. In line 7, we check if $s_c$ does not belong to $I$ (since clearly only concentrator sites that do not belong to the original solution can diminish the cost of the current solution). If it is the case in lines 8-14 we evaluate the benefit of including the site $s_c$ as potential improver of the best neighbor solution. We compute in line 8 the set $\hat{S}_C$ containing the concentrator sites of $J$ (that is to say $\hat{S}_C = S_C \cap J$). Notice that $s_c \in \hat{S}_C$. Line 9 computes the network induced by the set $I \cup \hat{S}_C \cup \{z\}$. Line 10 computes the connected component $\hat{\mathcal{H}} \subseteq \mathcal{H}$ such that $z \in \hat{\mathcal{H}}$. Clearly, the concentrator sites non-belonging to $\hat{\mathcal{H}}$ cannot be considered when computing a neighbor solution since they can induce a non-feasible topology. Thus, line 11 computes a minimum spanning tree for $\hat{\mathcal{H}}$, which is denoted by $\mathcal{T}$. In line 12, we connect to $\mathcal{T}$ the terminal sites of

$S_T$ by means of edges of minimum cost. Iteratively, the pendant concentrators (i.e. concentrators with degree 1) are deleted from T in line 13. Line 14 compares whether the cost of the resulting network $\mathcal{T}$ is smaller than the cost of the current best neighbor $\mathcal{T}_{best}$. If this is the case, the current best neighbor solution is updated with $\mathcal{T}$. Once all the concentrators non belonging to $\mathcal{T}_{sol}$ have been evaluated, the best solution found $\mathcal{T}_{best}$ is returned in line 17.

**Procedure ANDP_RNN_Local_Search;**
**Input:** $\mathcal{W}^+ = \{\varrho_{ij}^+\}$, $\mathcal{W}^- = \{\varrho_{ij}^-\}$, $G_A = (S, E)$, $C$, $\mathcal{T}_{sol}$;

1 $I \leftarrow \text{CONCENTRATORS}(\mathcal{T}_{sol})$; $J \leftarrow S_T \cup \{z\}$;
2 $\mathcal{T}_{best} \leftarrow \mathcal{T}_{sol}$;
3 **while** $(I \cup J) \neq S$ **do**
4 $q_s \leftarrow 1, \forall s \in J$;
5 **Compute the solution of the equation:**
$F(q) = q$ **given by the Gelenbe Theorem;**
6 $s_c \leftarrow \text{ArgMax}\{q_s | s \in (S_C \setminus J)\}$; $J \leftarrow J \cup \{s_c\}$;
7 **if** $(s_c \notin I)$ **then**
8 $\hat{S}_C \leftarrow \text{CONCENTRATORS}(J)$;
9 $\mathcal{H} \leftarrow$ **subgraph induced by** $(I \cup \hat{S}_C \cup \{z\})$;
10 $\hat{\mathcal{H}} \leftarrow$ **connected comp. of** $\mathcal{H}$ **such that** $z \in \hat{\mathcal{H}}$;
11 $\mathcal{T} \leftarrow$ **minimal spanning tree for** $\hat{\mathcal{H}}$;
12 **Compute** $\forall s_t \in S_T$**:**
$e \leftarrow$ **the edge of minimum cost from** $s_t$ **to** $\mathcal{T}$;
$\mathcal{T} \leftarrow \mathcal{T} \cup \{e\}$;
13 **Iteratively remove all concentrator sites**
**from** $\mathcal{T}$ **with degree 1;**
14 **if** $\text{COST}(\mathcal{T}) < \text{COST}(\mathcal{T}_{best})$ **then** $\mathcal{T}_{best} \leftarrow \mathcal{T}$;
15 **end_if;**
16 **end_while;**
17 **return** $\mathcal{T}_{best}$;
**end ANDP_RNN_Local_Search;**

Figure 3.14: RNN based Local Search Algorithm.

**Notation 3.4.7** *Let $\mathcal{T}_{sol}$ be the input solution of* ANDP_RNN_Local_Search *and $I$ its set of concentrator sites. We will denote by $V[1..m]$ (with $m = |S_C \setminus I|$) to a vector containing the sites of $S_C \setminus I$ which are sequentially analyzed (one at a time) according to the order determined by the underlaying RNN. That is to say, in $V[\cdot]$ we have the concentrator sites of $J \setminus I$ sorted by insertion order.*

**Proposition 3.4.8** *Let $\mathcal{T}_{sol}$ be the input solution of* ANDP_RNN_Local_Search *and $I$ its set of concentrator sites. The feasible solution $\mathcal{T}_{best}$ returned by* ANDP_RNN_Local_Search *satisfies the following points:*

*a) If $q_j$ is the associated asymptotic probability when computing $V[j]$, then:*

$$q_j > q_s,\ \forall s \in S_C \setminus (I \cup V[1..j-1]),\ \forall j \in 1 \ldots m.$$

*b)* $\text{COST}(\mathcal{T}_{best}) \leq \text{COST}(\mathcal{T}_k)$, $\forall k \in 1 \ldots m$, *where* $\mathcal{T}_k$ *is the feasible solution characterized by the set of concentrator sites* $I \cup V[1..k]$.

**Proof.** Point $(a)$ is trivially deduced from lines 4-6. We will prove $(b)$ by induction in $k$ (the number of sites of $S_C \setminus I$ already added to $J$). Initially $\mathcal{T}_{best} = \mathcal{T}_{sol}$ (line 2).
*Basic Step:* $k = 0$. This case corresponds when the algorithm did not yet execute lines 7-15.
*Induction Step:* $0 < k \leq m$. In certain iteration condition in line 7 is TRUE and therefore the algorithm executes the following:

i) lines 8-9 compute the subnetwork $\mathcal{H}$ induced by $I \cup V[1..k] \cup \{z\}$.

ii) lines 10-11 compute a minimum spanning tree $\mathcal{T}$ for the connected component $\hat{\mathcal{H}} \subseteq \mathcal{H}$ such that $z \in \hat{\mathcal{H}}$. Notice that the concentrator sites non-belonging to $\hat{\mathcal{H}}$ cannot be considered since they induce non-feasible solutions.

iii) line 12 connects to $\mathcal{T}$ the sites of $S_T$ by means of edges the minimum cost.

By Proposition 3.4.2, the computed network $\mathcal{T}$ is the feasible solution characterized by the set of concentrator sites $I \cup V[1..k]$. On the other hand, by inductive hypothesis, we have that network $\mathcal{T}_{best}$ satisfies:

$$\text{COST}(\mathcal{T}_{best}) \leq \text{COST}(\mathcal{T}_j),\ \forall j \in 1 \ldots k-1.$$

Line 13 removes iteratively from $\mathcal{T}$ the pendant concentrators since they are redundant in the solution. If the resulting network $\mathcal{T}$ is better than $\mathcal{T}_{best}$, this is updated in line 14. Hence, once executed line 15 the following inequality is reached:

$$\text{COST}(\mathcal{T}_{best}) \leq \min\left(\text{COST}(\mathcal{T}), \min\{\text{COST}(\mathcal{T}_j),\ \forall j \in 1 \ldots k-1\}\right) = \min\{\text{COST}(\mathcal{T}_j),\ \forall j \in 1 \ldots k\},$$

as required and completing the proof.

**QED**

The algorithm basically builds $n_C - |I|$ neighbor solutions and selects the best among these. As mentioned above, the RNN is used to determine the order in which we must consider the concentrator sites from $S_C \setminus I$ to improve the current solution.

## 3.5 The GRASP algorithms for the ANDP

We now describe the general GRASP algorithm for approximately solving the ANDP. Figure 3.15 shows the corresponding pseudo-code. The generic procedures $\text{Construction\_Phase}$ and $\text{Local\_Search}$ can be instanced of the following way:

- Construction_Phase: by ANDP_ConstPhase1 or ANDP_ConstPhase2.

- Local_Search: by ANDP_MST_Local_Search or ANDP_RNN_Local_Search.

In the following, Construction_Phase will reference indifferently to ANDP_ConstPhase1 or ANDP_ConstPhas and in the same way Local_Search will reference to ANDP_MST_Local_Search or ANDP_RNN_Local_Search. Next, we introduce a detailed description of the algorithm GRASP_ANDP.

```
Procedure GRASP_ANDP;
Input: G_A, k, seed, MaxIter;

1   P ← Preprocessing_Algorithm(G_A, C);
2   Compute the RNN parameters: W+ = {ϱ+_ij}; W− = {ϱ−_ij};
3   min_cost ← ∞;
4   for i = 1, ..., MaxIter do
5     T_sol ← Construction_Phase(G_A, C, k);
6     T_sol ← Local_Search(G_A, C, T_sol);
7     cost_sol ← COST(T_sol);
8     if (cost_sol < min_cost) then
9       T^(opt) ← T_sol; min_cost ← cost_sol;
10    end_if;
11  end_for;
12  return T^(opt);
end GRASP_ANDP;
```

Figure 3.15: General Version of the algorithm GRASP_ANDP.

The algorithm takes as inputs the graph $G_A$ of feasible connections on the access network, the matrix of connection costs $C$, the GRASP parameters $k$ (used in the construction phase), a seed for the pseudo random number generator $seed$ and the number of iterations $MaxIter$ to be performed. Line 1 calls Preprocessing_Algorithm in order to compute the auxiliary structure $P$ (used by ANDP_ConstPhase2). Line 2 computes the RNN parameters $\mathcal{W}^+$ and $\mathcal{W}^-$ (as presented above, they are the matrices of excitatory and inhibitory rates respectively, which are used by ANDP_RNN_Local_Search). The cost of the best found feasible solution is initialized with the value infinite ($\infty$) in line 3. The algorithm is repeated $MaxIter$ times exploring the space of feasible solutions and searching for the optimal feasible solution for the ANDP. Each iteration works as follows.

In line 5, a greedy randomized feasible solution $\mathcal{T}_{sol}$ is built using the algorithm Construction_Phase (i.e. ANDP_ConstPhase1 or ANDP_ConstPhase2). Line 6 calls Local_Search (i.e. ANDP_MST_Local_Search or ANDP_RNN_Local_Search) in order to find for better neighbor feasible solutions. Depending on the chosen algorithm, it searches for a better neighbor feasible solution by applying one of the following approaches:

i) a local search which consists in finding minimum spanning trees that integrate potential optimal topologies for the ANDP.

ii) a local search based on a RNN model which uses a customized underlaying neural network to improve the current solution.

In line 7 we compute the cost of the neighbor solution $\mathcal{T}_{sol}$ found in line 6. In line 8, we compare the cost of the current solution with the one delivered by $\mathrm{Local\_Search}$. If a neighbor solution with smaller cost has been found by $\mathrm{Local\_Search}$, then, we update in line 9 the best found feasible solution and the minimum cost. Once finalized the loop 4-11, the best found feasible solution $\mathcal{T}^{(\mathrm{opt})}$ is returned in line 12. Figures 3.16 and 3.17 show the execution diagrams corresponding to the generic algorithm $\mathrm{GRASP\_ANDP}$, when choosing the MST based local search or the RNN based local search respectively.

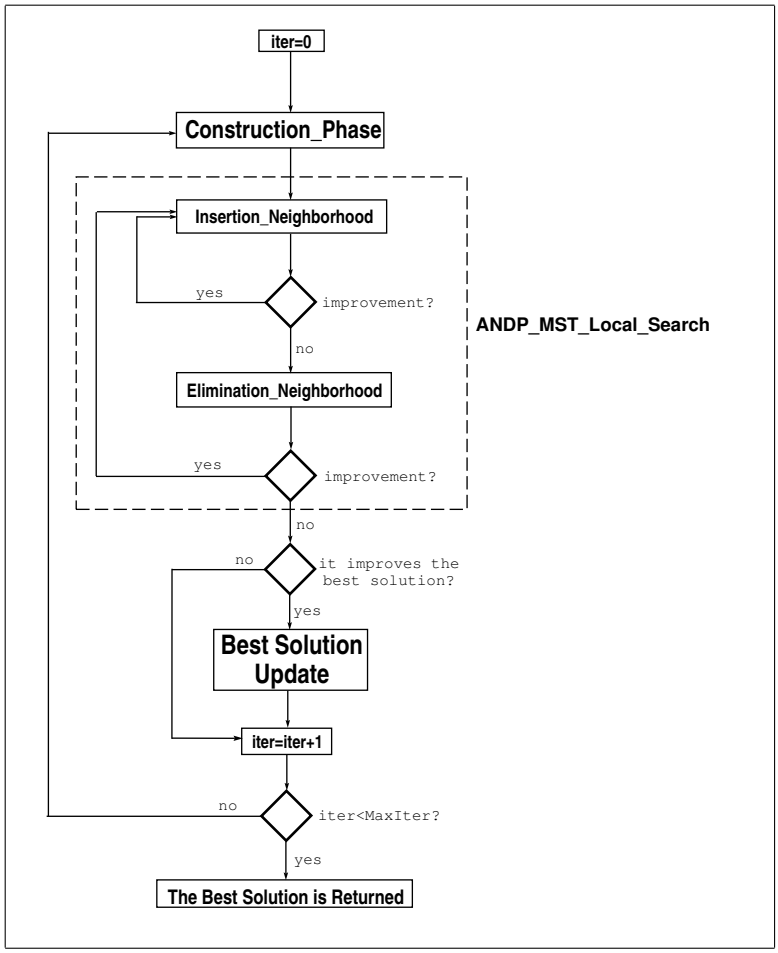



Figure 3.16: $\mathrm{GRASP\_ANDP}$ execution diagram with MST based local search.

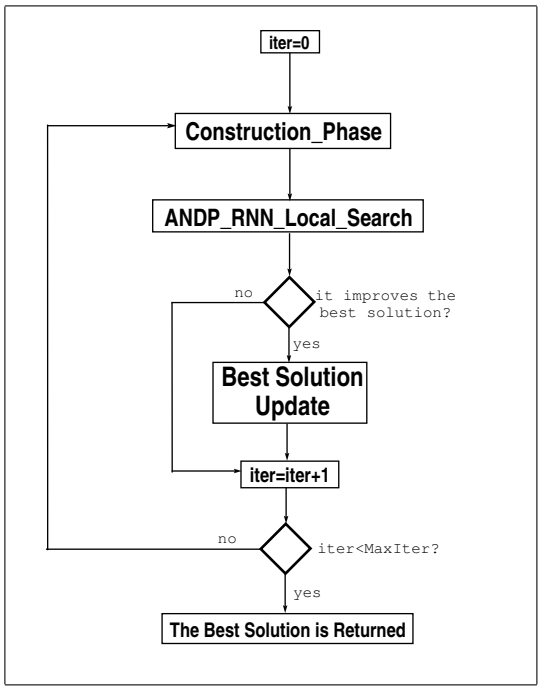



Figure 3.17: $\mathrm{GRASP\_ANDP}$ execution diagram with RNN based local search.

## 3.6 Performance Tests

We present here the experimental results obtained with the $\mathrm{GRASP\_ANDP}$ algorithm by using the different combinations of algorithms for the construction phase and the local search phase. The algorithms were implemented in ANSI C. The experiments were made on a Pentium IV with 1.7 GHz, and 1 Gbytes of RAM, running under Windows XP. In a first phase the candidate list size $k$ was chosen in the set $\{5, 10, 15, 20, 30\}$, and the maximum number of iterations $MaxIter$ in the interval $[50..500]$. These values were chosen from GRASP reference literature. In particular, we tuned the value for the candidate list size of the following way. We considered a reduced group of ANDP instances with different topological characteristics and over it we ran our $\mathrm{GRASP\_ANDP}$ considering its four possible versions and varying the value of $k$. We selected $k = 10$ as the value with better results since it outperformed in all the cases the results obtained with the other values.

In a second phase we tested the $\mathrm{GRASP\_ANDP}$ on widely diverse ANDP classes. We used a large test set, by modifying the Steiner Problem in Graphs (SPG) instances from SteinLib [90]. This library contains many problem classes of widely different graph topologies. We extracted most of the problems in the classes: C, MC, X, PUC, I080, I160, I320, I640, P6E, P6Z, WRP3 and WRP4. We customized the SPG problems, transforming them into ANDP instances by means of the following changes. For each considered problem:

i) we selected the terminal node with greatest degree as the $z$ node (modelling the backbone),

ii) the Steiner nodes model the concentrator sites, and the terminal nodes model the terminal sites,

iii) all the edges between terminal sites were deleted (as they are not allowed in feasible ANDP solutions).

Moreover, if the resulting topology was unconnected, the problem instance was discarded. If the resulting topology was connected, we ran the $\mathrm{Construction\_Phase}$ (i.e. one of the construction phase algorithms) in order to discard the instance if its space of feasible solutions was empty. By this process, we obtained 210 ANDP instances having nonempty space of feasible solutions.

Let us notice that since in the ANDP the terminals cannot be used as intermediate nodes (which implies also that edges between pairs of terminals are not allowed), the cost of a SPG optimum is a lower bound for the optimum of the corresponding ANDP.

Tables 3.1 to 3.4 show a summary of computational results obtained by applying $\mathrm{GRASP\_ANDP}$ to the customized SteinLib classes, they correspond to the combinations:

**Heuristic $\mathcal{H}_1$:** $\mathrm{ANDP\_ConstPhase1}$ and $\mathrm{ANDP\_MST\_Local\_Search}$,

**Heuristic $\mathcal{H}_2$:** $\mathrm{ANDP\_ConstPhase2}$ and $\mathrm{ANDP\_MST\_Local\_Search}$,

**Heuristic $\mathcal{H}_3$:** $\mathrm{ANDP\_ConstPhase1}$ and $\mathrm{ANDP\_RNN\_Local\_Search}$,

**Heuristic $\mathcal{H}_4$:** $\mathrm{ANDP\_ConstPhase2}$ and $\mathrm{ANDP\_RNN\_Local\_Search}$.

The results shown in these tables were obtained with the combination of parameters $k = 10$ and $MaxIter = 100$. In each one of them the first column contains the names of the original SteinLib classes and the entries from left to right are:

- the number of customized instances (NI),
- the size of the selected instances in terms of number of nodes and edges respectively,
- the number of instances where the lower bound was obtained reaching therefore the optimum (NOPT),
- the average of the improvement of the results of the local search phase over the construction phase (Avg. LSI),
- the average running time per iteration (Avg. secs/itr),
- and the average of the gap of the GRASP solution with respect to the lower bound (Avg. LB_GAP).

The average improvement is computed as Avg. LSI= $\sum_{p \in Set} \mathrm{LSI}(p)/\mathrm{NI}$, where for problem $p$,

$$\mathrm{LSI}(p) = \frac{100}{MaxIter} \times \left( \sum_{i=1}^{MaxIter} \frac{(CC_i - LC_i)}{CC_i} \right),$$

$CC_i$ and $LC_i$ being the costs of the solutions delivered in iteration $i$ by the Construction Phase and the Local Search Phase respectively. The average gap is Avg. LB_GAP= $\sum_{p \in Set} \mathrm{LB\_GAP}(p)/\mathrm{NI}$, where for problem $p$,

$$\mathrm{LB\_GAP}(p) = 100 \times \frac{Best\_Cost\_Found - Lower\_Bound}{Lower\_Bound},$$

and $Lower\_Bound$ is the optimum value corresponding to the original SPG instance.

| Testset | NI | Nodes | Edges | NOPT | Avg. LSI | Avg. secs/itr | Avg. LB_GAP |
|---|---|---|---|---|---|---|---|
| C | 6 | 500 | 625-2500 | - | 18.17% | 13.83 | 0.43% |
| MC | 3 | 97-150 | 4656-11175 | 1 | 21.56% | 5.01 | 3.33% |
| X | 2 | 52-58 | 1326-1653 | - | 15.13% | 1.43 | 52.33% |
| PUC | 4 | 64-128 | 192-750 | 2 | 19.65% | 2.47 | 0.14% |
| I080 | 70 | 80 | 120-3160 | 15 | 18.87% | 1.42 | 7.58% |
| I160 | 22 | 160 | 240-2544 | 7 | 21.31% | 3.02 | 3.57% |
| I320 | 15 | 320 | 480-10208 | 3 | 22.54% | 11.27 | 2.57% |
| I640 | 15 | 640 | 960-4135 | 2 | 22.06% | 34.08 | 3.71% |
| P6E | 10 | 100-200 | 180-370 | 2 | 22.12% | 1.98 | 17.08% |
| P6Z | 5 | 100-200 | 180-370 | 1 | 20.36% | 1.54 | 27.33% |
| WRP3 | 27 | 84-925 | 149-1800 | 7 | 20.22% | 22.42 | 0.00032% |
| WRP4 | 31 | 110-938 | 188-1869 | 4 | 24.16% | 32.09 | 0.00786% |
| ALL | 210 | - | - | 44 | 20.75% | 12.30 | 5.37% |

Table 3.1: Results with ANDP_ConstPhase1 and ANDP_MST_Local_Search (Heuristic $\mathcal{H}_1$).

| Testset | NI | Nodes | Edges | NOPT | Avg. LSI | Avg. secs/itr | Avg. LB_GAP |
|---|---|---|---|---|---|---|---|
| C | 6 | 500 | 625-2500 | - | 14.83% | 11.27 | 0.32% |
| MC | 3 | 97-150 | 4656-11175 | 1 | 17.96% | 4.17 | 3.33% |
| X | 2 | 52-58 | 1326-1653 | - | 14.25% | 1.46 | 34.32% |
| PUC | 4 | 64-128 | 192-750 | 2 | 18.42% | 1.82 | 0.14% |
| I080 | 70 | 80 | 120-3160 | 17 | 14.27% | 1.03 | 4.42% |
| I160 | 22 | 160 | 240-2544 | 7 | 18.98% | 2.98 | 3.28% |
| I320 | 15 | 320 | 480-10208 | 3 | 17.63% | 9.12 | 2.35% |
| I640 | 15 | 640 | 960-4135 | 2 | 16.59% | 29.83 | 3.01% |
| P6E | 10 | 100-200 | 180-370 | 2 | 17.65% | 1.78 | 14.78% |
| P6Z | 5 | 100-200 | 180-370 | 1 | 17.02% | 1.24 | 18.64% |
| WRP3 | 27 | 84-925 | 149-1800 | 7 | 16.76% | 19.64 | 0.00031% |
| WRP4 | 31 | 110-938 | 188-1869 | 5 | 18.44% | 26.91 | 0.00723% |
| ALL | 210 | - | - | 47 | 16.47% | 10.47 | 3.73% |

Table 3.2: Results with ANDP_ConstPhase2 and ANDP_MST_Local_Search (Heuristic $\mathcal{H}_2$).

| Testset | NI | Nodes | Edges | NOPT | Avg. LSI | Avg. secs/itr | Avg. LB_GAP |
|---|---|---|---|---|---|---|---|
| C | 6 | 500 | 625-2500 | - | 19.95% | 12.13 | 0.41% |
| MC | 3 | 97-150 | 4656-11175 | 1 | 23.34% | 3.14 | 6.64% |
| X | 2 | 52-58 | 1326-1653 | - | 11.00% | 0.73 | 39.56% |
| PUC | 4 | 64-128 | 192-750 | 2 | 21.04% | 1.27 | 0.14% |
| I080 | 70 | 80 | 120-3160 | 13 | 18.22% | 1.49 | 10.71% |
| I160 | 22 | 160 | 240-2544 | 7 | 23.82% | 4.03 | 3.86% |
| I320 | 15 | 320 | 480-10208 | 2 | 21.12% | 10.14 | 2.89% |
| I640 | 15 | 640 | 960-4135 | 2 | 20.59% | 29.63 | 4.67% |
| P6E | 10 | 100-200 | 180-370 | 2 | 23.75% | 1.83 | 16.49% |
| P6Z | 5 | 100-200 | 180-370 | 1 | 22.01% | 1.10 | 23.22% |
| WRP3 | 27 | 84-925 | 149-1800 | 7 | 20.18% | 20.32 | 0.00323% |
| WRP4 | 31 | 110-938 | 188-1869 | 3 | 25.14% | 28.25 | 0.00513% |
| ALL | 210 | - | - | 40 | 20.91% | 11.07 | 6.33% |

Table 3.3: Results with $\mathrm{ANDP\_ConstPhase1}$ and $\mathrm{ANDP\_RNN\_Local\_Search}$ (Heuristic $\mathcal{H}_3$).

| Testset | NI | Nodes | Edges | NOPT | Avg. LSI | Avg. secs/itr | Avg. LB_GAP |
|---|---|---|---|---|---|---|---|
| C | 6 | 500 | 625-2500 | - | 15.17% | 10.12 | 0.27% |
| MC | 3 | 97-150 | 4656-11175 | 1 | 19.33% | 4.43 | 3.69% |
| X | 2 | 52-58 | 1326-1653 | - | 8.12% | 0.52 | 32.27% |
| PUC | 4 | 64-128 | 192-750 | 2 | 17.32% | 1.02 | 0.11% |
| I080 | 70 | 80 | 120-3160 | 17 | 13.21% | 0.57 | 6.73% |
| I160 | 22 | 160 | 240-2544 | 7 | 17.14% | 3.09 | 3.08% |
| I320 | 15 | 320 | 480-10208 | 3 | 16.12% | 9.03 | 2.12% |
| I640 | 15 | 640 | 960-4135 | 2 | 17.69% | 26.96 | 2.56% |
| P6E | 10 | 100-200 | 180-370 | 2 | 16.11% | 1.54 | 14.02% |
| P6Z | 5 | 100-200 | 180-370 | 1 | 16.42% | 1.03 | 19.12% |
| WRP3 | 27 | 84-925 | 149-1800 | 8 | 15.76% | 16.83 | 0.00096% |
| WRP4 | 31 | 110-938 | 188-1869 | 5 | 17.84% | 20.05 | 0.00132% |
| ALL | 210 | - | - | 48 | 15.54% | 8.68 | 4.39% |

Table 3.4: Results with $\mathrm{ANDP\_ConstPhase2}$ and $\mathrm{ANDP\_RNN\_Local\_Search}$ (Heuristic $\mathcal{H}_4$).

In Appendix B, we introduce tables which summarize the best results obtained for each customized instance. Before comparing the different GRASP heuristics for the ANDP, we define the average of the difference between the best solution cost obtained by $\mathcal{H}_i$, $i \in 1..4$, with respect to the obtained one by $\mathcal{H}_j$, $j \in 1..4$, $j \neq i$, as:

$$\text{Avg. DCOST}_{ij} = \frac{100}{NI} \times \left( \sum_{p \in Set} \frac{(CH_i(p) - CH_j(p))}{CH_j(p)} \right),$$

where for problem $p$, $CH_i(p)$ and $CH_j(p)$ are the costs of the solutions delivered by $\mathcal{H}_i$ and $\mathcal{H}_j$ respectively. In addition, we introduce the notation:

- $\mathrm{NLB}_{ij}$ is the number of instances where the lower bound was attained by the heuristic $\mathcal{H}_i$ but not by the heuristic $\mathcal{H}_j$,

- $\mathrm{NO}_{ij}$ the number of instances where the heuristic $\mathcal{H}_i$ overcame the heuristic $\mathcal{H}_j$.

Based on these definitions, we provide the following comparison tables. In Tables 3.5 and 3.6 we compare the construction phase algorithms when the local search algorithms are the same; Table 3.5 compares the heuristics $\mathcal{H}_1$ and $\mathcal{H}_2$ whereas Table 3.6 compares the heuristics $\mathcal{H}_3$ and $\mathcal{H}_4$. Tables 3.7 and 3.8 compare the local search algorithms when the construction phase algorithms are the same; Table 3.7 compares the heuristics $\mathcal{H}_1$ and $\mathcal{H}_3$ whereas Table 3.8 compares the heuristics $\mathcal{H}_2$ and $\mathcal{H}_4$. Finally, Tables 3.9 and 3.10 compare the heuristics $\mathcal{H}_1$ with $\mathcal{H}_4$, and $\mathcal{H}_2$ with $\mathcal{H}_3$ respectively.

| **Testset** | $\mathrm{NLB}_{12}$ | $\mathrm{NLB}_{21}$ | $\mathrm{NO}_{12}$ | $\mathrm{NO}_{21}$ | Avg. $\mathrm{DCOST}_{12}$ | Avg. $\mathrm{DCOST}_{21}$ |
|---|---|---|---|---|---|---|
| C | - | - | 0 | 2 | 0.09% | -0.07% |
| MC | 0 | 0 | 0 | 0 | 0.0% | 0.0% |
| X | - | - | 0 | 1 | 15.12% | -14.53% |
| PUC | 0 | 0 | 0 | 0 | 0.0% | 0.0% |
| I080 | 0 | 2 | 2 | 6 | 2.46% | -2.25% |
| I160 | 0 | 0 | 1 | 4 | 0.24% | -0.19% |
| I320 | 0 | 0 | 0 | 3 | 0.18% | -0.16% |
| I640 | 0 | 0 | 0 | 3 | 0.58% | -0.53% |
| P6E | 0 | 0 | 0 | 2 | 1.86% | -1.73% |
| P6Z | 0 | 0 | 0 | 2 | 6.95% | -6.24% |
| WRP3 | 0 | 0 | 1 | 6 | 0.0000086% | -0.0000075% |
| WRP4 | 0 | 1 | 0 | 5 | 0.00051% | -0.00047% |
| Total | 0 | 3 | 4 | 34 | 1.30% | -1.19% |

Table 3.5: Comparison between $\mathcal{H}_1$ and $\mathcal{H}_2$.

| **Testset** | $\mathrm{NLB}_{34}$ | $\mathrm{NLB}_{43}$ | $\mathrm{NO}_{34}$ | $\mathrm{NO}_{43}$ | Avg. $\mathrm{DCOST}_{34}$ | Avg. $\mathrm{DCOST}_{43}$ |
|---|---|---|---|---|---|---|
| C | - | - | 0 | 2 | 0.11% | -0.10% |
| MC | 0 | 0 | 0 | 1 | 2.38% | -2.28% |
| X | - | - | 0 | 1 | 6.23% | -5.78% |
| PUC | 0 | 0 | 0 | 1 | 0.021% | -0.020% |
| I080 | 0 | 4 | 0 | 8 | 3.32% | -3.06% |
| I160 | 0 | 0 | 0 | 2 | 0.67% | -0.61% |
| I320 | 0 | 0 | 0 | 3 | 0.63% | -0.57% |
| I640 | 0 | 0 | 0 | 2 | 1.72% | -1.64% |
| P6E | 0 | 0 | 0 | 2 | 2.14% | -1.96% |
| P6Z | 0 | 0 | 0 | 1 | 3.52% | -3.23% |
| WRP3 | 0 | 1 | 0 | 5 | 0.0017% | -0.0016% |
| WRP4 | 0 | 2 | 0 | 6 | 0.0029% | -0.0019% |
| Total | 0 | 7 | 0 | 36 | 1.62% | -1.50% |

Table 3.6: Comparison between $\mathcal{H}_3$ and $\mathcal{H}_4$.

Considering the possible combinations of algorithms for the construction phase and the local search phase, the results show that the four resulting heuristics find in most cases good quality solutions. Next, we will discuss the obtained results.

| **Testset** | $\mathrm{NLB}_{13}$ | $\mathrm{NLB}_{31}$ | $\mathrm{NO}_{13}$ | $\mathrm{NO}_{31}$ | Avg. $\mathrm{DCOST}_{13}$ | Avg. $\mathrm{DCOST}_{31}$ |
|---|---|---|---|---|---|---|
| C | - | - | 1 | 1 | 0.017% | -0.017% |
| MC | 0 | 0 | 1 | 0 | -2.55% | 2.75% |
| X | - | - | 0 | 1 | 10.72% | -8.75% |
| PUC | 0 | 0 | 0 | 0 | 0% | 0% |
| I080 | 2 | 0 | 4 | 0 | -2.01% | 2.63% |
| I160 | 0 | 0 | 3 | 0 | -0.18% | 0.24% |
| I320 | 1 | 0 | 2 | 0 | -0.23% | 0.26% |
| I640 | 0 | 0 | 2 | 0 | -0.64% | 0.71% |
| P6E | 0 | 0 | 0 | 2 | 0.45% | -0.38% |
| P6Z | 0 | 0 | 0 | 2 | 3.73% | -2.68% |
| WRP3 | 0 | 0 | 3 | 1 | -0.00186% | 0.00237% |
| WRP4 | 1 | 0 | 2 | 2 | 0.00219% | -0.00117% |
| Total | 4 | 0 | 18 | 9 | -0.57% | 0.84% |

Table 3.7: Comparison between $\mathcal{H}_1$ and $\mathcal{H}_3$.

| **Testset** | $\mathrm{NLB}_{24}$ | $\mathrm{NLB}_{42}$ | $\mathrm{NO}_{24}$ | $\mathrm{NO}_{42}$ | Avg. $\mathrm{DCOST}_{24}$ | Avg. $\mathrm{DCOST}_{42}$ |
|---|---|---|---|---|---|---|
| C | - | - | 0 | 2 | 0.039% | -0.033% |
| MC | 0 | 0 | 1 | 0 | -0.17% | 0.28% |
| X | - | - | 0 | 1 | 1.63% | -1.45% |
| PUC | 0 | 0 | 0 | 1 | 0.024% | -0.022% |
| I080 | 0 | 0 | 4 | 3 | 1.52% | -2.08% |
| I160 | 0 | 0 | 0 | 2 | 0.17% | -0.14% |
| I320 | 0 | 0 | 0 | 3 | 0.20% | -0.18% |
| I640 | 0 | 0 | 0 | 3 | 0.43% | -0.28% |
| P6E | 0 | 0 | 0 | 2 | 0.68% | -0.54% |
| P6Z | 0 | 0 | 1 | 2 | 0.26% | -0.31% |
| WRP3 | 0 | 1 | 2 | 2 | 0.00047% | -0.00062% |
| WRP4 | 0 | 0 | 2 | 6 | 0.00338% | -0.00101% |
| Total | 0 | 1 | 10 | 27 | 0.62% | -0.78% |

Table 3.8: Comparison between $\mathcal{H}_2$ and $\mathcal{H}_4$.

The heuristic $\mathcal{H}_4$ reached the lower bound and therefore the optimality in 48 instances, followed in order by $\mathcal{H}_2$, $\mathcal{H}_1$ and $\mathcal{H}_3$ with 47, 44 and 40 instances respectively. With respect to the 210 instances considered, these values correspond to $22\%$ of the cases using the heuristics $\mathcal{H}_4$ and $\mathcal{H}_2$, more than $20\%$ using $\mathcal{H}_1$, and more than $19\%$ using $\mathcal{H}_3$. In general, the gaps related to the lower bounds were low in the four heuristics, and did not surpass the $5\%$ in 7 out of 12 classes. Even though we do not know the global optimal solution costs of the ANDP instances generated, some of the feasible solutions found might eventually be globally optimal if we consider the low gaps obtained as well as the percentage of cases in which we reached the lower bound.

We also notice that the percentages of local search improvement strongly depend on which solution construction algorithm is applied. For example, when setting the algorithm $\mathrm{ANDP\_MST\_Local\_Search}$ as local search and comparing the heuristics $\mathcal{H}_1$ and $\mathcal{H}_2$, we notice that in every class the average improvement

| **Testset** | $\mathrm{NLB}_{14}$ | $\mathrm{NLB}_{41}$ | $\mathrm{NO}_{14}$ | $\mathrm{NO}_{41}$ | Avg. $\mathrm{DCOST}_{14}$ | Avg. $\mathrm{DCOST}_{41}$ |
|---|---|---|---|---|---|---|
| C | - | - | 0 | 2 | 0.13% | -0.09% |
| MC | 0 | 0 | 1 | 0 | -0.28% | 0.33% |
| X | - | - | 0 | 1 | 15.21% | -15.08% |
| PUC | 0 | 0 | 0 | 1 | 0.03% | -0.02% |
| I080 | 0 | 2 | 3 | 7 | 0.64% | -0.48% |
| I160 | 0 | 0 | 1 | 4 | 0.32% | -0.27% |
| I320 | 0 | 0 | 0 | 3 | 0.30% | -0.20% |
| I640 | 0 | 0 | 0 | 4 | 0.85% | -0.65% |
| P6E | 0 | 0 | 0 | 3 | 2.01% | -1.12% |
| P6Z | 0 | 0 | 0 | 1 | 6.05% | -5.14% |
| WRP3 | 0 | 1 | 3 | 2 | -0.00031% | 0.00052% |
| WRP4 | 0 | 1 | 1 | 6 | 0.00601% | -0.00213% |
| Total | 0 | 4 | 9 | 34 | 0.71% | -0.56% |

Table 3.9: Comparison between $\mathcal{H}_1$ and $\mathcal{H}_4$.

| **Testset** | $\mathrm{NLB}_{23}$ | $\mathrm{NLB}_{32}$ | $\mathrm{NO}_{23}$ | $\mathrm{NO}_{32}$ | Avg. $\mathrm{DCOST}_{23}$ | Avg. $\mathrm{DCOST}_{32}$ |
|---|---|---|---|---|---|---|
| C | - | - | 2 | 0 | -0.05% | 0.08% |
| MC | 0 | 0 | 2 | 0 | -2.01% | 3.22% |
| X | - | - | 1 | 0 | -3.11% | 3.87% |
| PUC | 0 | 0 | 0 | 0 | 0% | 0% |
| I080 | 4 | 0 | 9 | 4 | -3.07% | 4.32% |
| I160 | 0 | 0 | 3 | 1 | -0.31% | 0.47% |
| I320 | 1 | 0 | 4 | 1 | -0.28% | 0.38% |
| I640 | 0 | 0 | 4 | 0 | -0.78% | 0.93% |
| P6E | 0 | 0 | 2 | 0 | -0.71% | 1.11% |
| P6Z | 0 | 0 | 2 | 0 | -1.92% | 2.54% |
| WRP3 | 0 | 0 | 3 | 2 | -0.00141% | 0.00217% |
| WRP4 | 2 | 0 | 4 | 2 | -0.0007% | 0.0018% |
| Total | 7 | 0 | 36 | 10 | -1.27% | 1.78% |

Table 3.10: Comparison between $\mathcal{H}_2$ and $\mathcal{H}_3$.

of the local search phase (Avg. LSI) was lower in $\mathcal{H}_2$ than in $\mathcal{H}_1$. This shows that the feasible solution constructed by $\mathrm{ANDP\_ConstPhase2}$ is generally of better quality (i.e. closest to a local optimum) than the one built by $\mathrm{ANDP\_ConstPhase1}$. Moreover, computing the weighted averages of Avg. LSI over all the classes we have a $20.75\%$ and $16.47\%$ of average improvement in $\mathcal{H}_1$ and $\mathcal{H}_2$ respectively. This observations are also valid when comparing the heuristics $\mathcal{H}_3$ and $\mathcal{H}_4$ which use the algorithm $\mathrm{ANDP\_RNN\_Local\_Search}$ as local search. In this last case the weighted average of Avg. LSI over all the classes are of $20.91\%$ and $15.54\%$ of average improvement for $\mathcal{H}_3$ and $\mathcal{H}_4$ respectively.

On the other hand, when setting the algorithm $\mathrm{ANDP\_ConstPhase1}$ as construction of the solution and comparing the heuristics $\mathcal{H}_1$ and $\mathcal{H}_3$, we notice that in seven classes the average improvement of the local search phase was bigger in $\mathcal{H}_3$ than in $\mathcal{H}_1$. However, both heuristics had very similar values considering the weighted average per class. In the same way, when setting the algorithm $\mathrm{ANDP\_ConstPhase2}$ as

construction of the solution and comparing the heuristics $\mathcal{H}_2$ and $\mathcal{H}_4$, we notice that in 9 classes the average improvement of the local search phase was bigger in $\mathcal{H}_2$ than in $\mathcal{H}_4$ and its weighted averages with a difference lower to $1\%$. Therefore, we are able to say that independently of the construction used, both local search algorithms significantly improved the feasible solutions constructed, being this improvement on average always superior to $8\%$ and superior to $14\%$ in 11 classes.

Let us analyze now the execution times. When measuring the heuristics $\mathcal{H}_2$ and $\mathcal{H}_4$ execution times we considered the pre-processing time of the paths matrix $\mathcal{P}$ used by the algorithm $\mathrm{ANDP\_ConstPhase2}$, in order to compare them to the execution times of the heuristics $\mathcal{H}_1$ and $\mathcal{H}_3$. Considering the average execution times over all classes, we obtained that the heuristic with best times was $\mathcal{H}_4$ followed by $\mathcal{H}_2$, $\mathcal{H}_3$ and $\mathcal{H}_1$. Notice that those average values do not differ in more than 4 seconds per iteration. Besides it should be noticed that the execution times strongly depend on the class to which the problem belongs. Comparing any two of the heuristics used, we found in every case situations where an algorithm has better average time than the other one for a certain class and inversely. For example: $\mathcal{H}_2$ and $\mathcal{H}_3$ have for class C average time by iteration of 12.13 seconds and 10.12 seconds respectively, whereas for class MC have average time by iteration of 3.14 seconds and 4.43 seconds respectively.

Let us analyze the comparisons between heuristics.

- Table 3.5 shows that the heuristic $\mathcal{H}_2$ improves on average $1.19\%$ the solution delivered by the heuristic $\mathcal{H}_1$. In every instance where $\mathcal{H}_1$ reached the lower bound, $\mathcal{H}_2$ also reached it. Moreover $\mathcal{H}_2$ reached the lower bound in three ANDP instances where $\mathcal{H}_1$ could not reach it. Nevertheless, it is to be noticed that in four instances $\mathcal{H}_1$ obtained better feasible solutions than $\mathcal{H}_2$, what makes impossible stating that $\mathcal{H}_2$ overcomes $\mathcal{H}_1$.

- Table 3.6 shows that the heuristic $\mathcal{H}_4$ improves on average $1.5\%$ the solution delivered by the heuristic $\mathcal{H}_3$. In every case where $\mathcal{H}_3$ reached the lower bound, $\mathcal{H}_4$ also reached it. Besides $\mathcal{H}_4$ achieved the lower bound in seven ANDP instances where $\mathcal{H}_3$ could not achieve it. In 36 instances $\mathcal{H}_4$ found better solutions than $\mathcal{H}_3$ whereas $\mathcal{H}_3$ did not overcame $\mathcal{H}_4$ at any instance.

- Table 3.7 shows that the heuristic $\mathcal{H}_1$ improves on average $0.57\%$ the solution delivered by the heuristic $\mathcal{H}_3$. In every case where $\mathcal{H}_3$ reached the lower bound, $\mathcal{H}_1$ reached it too. Moreover $\mathcal{H}_1$ achieved the lower bound in four ANDP instances where $\mathcal{H}_3$ could not achieve them. However, in nine instances $\mathcal{H}_3$ obtained better feasible solutions than $\mathcal{H}_1$ while in 18 instances $\mathcal{H}_1$ overcame $\mathcal{H}_3$.

- Table 3.8 shows that the heuristic $\mathcal{H}_4$ improves on average $0.78\%$ the solution delivered by $\mathcal{H}_2$. In every case where $\mathcal{H}_2$ reached the lower bound, $\mathcal{H}_4$ reached it too. In addition $\mathcal{H}_4$ succeed in reaching the lower bound in an ANDP instance where $\mathcal{H}_2$ could not reach it. On the other hand, in ten instances $\mathcal{H}_2$ obtained better feasible solutions than $\mathcal{H}_4$ while in 27 instances $\mathcal{H}_4$ overcame $\mathcal{H}_2$.

- Table 3.9 shows that the heuristic $\mathcal{H}_4$ improves on average $0.56\%$ the solution delivered by the heuristic $\mathcal{H}_1$. In every case where $\mathcal{H}_1$ reached the lower bound, $\mathcal{H}_4$ also reached it. Besides this $\mathcal{H}_4$ reached the lower bound in four ANDP instances where $\mathcal{H}_1$ did not succeed in achieving it. In nine instances $\mathcal{H}_1$ obtained better feasible solutions than $\mathcal{H}_4$ whereas in 34 instances $\mathcal{H}_4$ overcame $\mathcal{H}_1$.

- Table 3.10 shows that the heuristic $\mathcal{H}_2$ improves on average $1.27\%$ the best solution found by the heuristic $\mathcal{H}_3$. In every case where $\mathcal{H}_3$ reached the lower bound, $\mathcal{H}_2$ reached it too. In addition to this $\mathcal{H}_2$ achieved the lower bound in seven ANDP instances where $\mathcal{H}_3$ could not reach them. In ten instances $\mathcal{H}_3$ obtained better feasible solutions than $\mathcal{H}_2$ while in 36 instances $\mathcal{H}_2$ overcame $\mathcal{H}_3$.

To conclude the comparison we can say that the four heuristics produced good results. All of them reached optimal solutions in more than $20\%$ of test cases and attaining on average low gaps with respect to their lower bounds. In addition, from the empirical data, we also notice that some of the heuristics are incomparable since there are ANDP instances where one heuristic beats the other and inversely. At last, it is worth mentioning the good performance of the RNN model as optimizer of feasible topologies.

## 3.7 Conclusions

By modelling the access network design problem as a variant of the Steiner problem in graphs, we were able to develop four Greedy Randomized Search Adaptive Procedures which can give a good quality approximate solution. The implementation of our algorithms was tested on a number of different problems with heterogeneous characteristics. In particular, we built a set of ANDP instances transforming 210 SPG instances (extracted from SteinLib) to our problem. The optimal values for the selected SPG instances are lower bounds for the corresponding ANDP.

The four versions of GRASP algorithms found good quality feasible solutions, reaching the optimum in 40, 44, 47, and 48 cases in a total of 210 instances (over $20\%$ of the test-set in all the cases). Even though we only know the lower bounds given by the optimal values of the SPG original problems, when computing the weighted average over all the classes, the average gaps of the solutions obtained related to this bounds were lower than $7\%$. It is reasonable supposing that the gaps related to the global optimums of the ANDP instances be even lower since the feasible solutions of a ANDP are also feasible solutions of the original SPG, but not reciprocally. In this sense, remember that in any ANDP instances generated, all the edges between terminal nodes pairs were eliminated (because in our ANDP such connections are not allowed) having the additional constraint that the terminal nodes must have degree one in the solution.

On the other hand, as above exposed, the local search algorithms notably improved the solutions delivered by the construction algorithms, strongly varying the percentage of improvement according to the SPG

class of problem from which the ANDP instance comes. In addition to this, such improvements are strongly attached to the quality of the solution delivered by the construction phase.

We noticed that, as expected, the execution times of the proposed algorithms are strongly dependant on the number of sites, edges, and the terminal sites.

To sum up, as far as we are concerned, the results obtained with the GRASP algorithms proposed are very good as we consider that computing the global optimal solution of an ANDP is a NP-hard problem.

Actually, our ANDP model has some limitations; for instance, we have not considered constraints related to the depth of the resulting topology. As future work, it is possible to incorporate these restrictions with the aim of producing access networks topologically more reliable by limiting the amount of chained concentrator nodes. Moreover, it is possible to search for new methods which improve either the initial construction or the local search phases of the GRASP.

# Chapter 4

# The Backbone Network Design Problem

## 4.1 Introduction

In general, a typical WAN backbone network has a meshed topology, and its purpose is to allow efficient and reliable communication between the switch sites of the network that act as connection points for the local access networks (eventually incorporating other switch sites for efficiency purposes).

The topological design of a Backbone network basically consists of finding a minimum cost topology which satisfies some additional requirements, generally chosen to improve the survivability of the network (that is, its capacity to resist the failures of some of its components). One way to do this is to specify a connectivity level, and to search for topologies which have at least this number of disjoint paths (either edge disjoint or node disjoint) between pairs of switch sites. In the most general case, the connectivity level can be fixed independently for each pair of switch sites (heterogeneous connectivity requirements). This problem can be modelled as a *Generalized Steiner Problem with Node-Connectivity* (denoted by GSP-NC) and it is an NP-Complete problem [125, 135, 136]. Some references in this area are [1, 6, 9, 33, 69, 78, 89]. As we mentioned in Section 1.4, most of these works are either focused on the edge-disjoint flavor of the problem, or on the exploration of particular cases, for example, when it is required to have two disjoint paths between all pairs of distinguished switch sites, which is called the 2-survivability problem [6]. In [126], an extensive survey (1992) over high survivability models is introduced.

Topologies verifying edge-disjoint path connectivity constraints assure that the network can survive to failures in the connection lines; whereas node-disjoint path constraints assure that the network can survive to failures both in switch sites as well as in the connection lines. In this chapter we discuss several customizations of the GRASP methodology for finding a low cost Backbone network topology that satisfies connection requirements on the number of node-disjoint paths, working upon the Generalized Steiner Problem with Node-Connectivity model.

The remainder of this chapter is organized as follows. Section 4.2 introduces the notation, the auxiliary definitions to be used and the formal definition of our Backbone Network Design Problem (denoted by BNDP). In Section 4.3 we introduce an algorithm for the construction phase plus a slight variant of it. In Section 4.4 we provide three different alternative algorithms for the local search phase. These methods were designed with the goal of working in a combined form, allowing thus the exploration of different neighborhood structures. Hence, by combining them suitably, we obtain different versions of GRASP. Section 4.5 discuss the combinations of the previous construction and local search phases within the GRASP methodology for approximately solving the BNDP. Section 4.6 presents experimental results obtained when applying the GRASP algorithms on an extensive test-set of BNDP instances, containing problems with distinctive topological characteristics. Finally, we conclude with a discussion in Section 4.7, which includes some guidelines for future work.

## 4.2 Notation, Problem Formalization and Auxiliary Definitions

We introduce the notation used to formalize the problem:

- $S_D$ is the set of sites where the switch equipments can be installed; these sites also will be called potential switch sites or backbone sites. The number of the switch sites is given by $n_D = |S_D|$.

- $S_D^{(I)}$ is a distinguished subset of switch sites, which will always be included in the backbone network topology (usually because they are the access points for some local access subnetworks). We usually call these the *fixed sites* of the backbone. The nodes in $S_D \setminus S_D^{(I)}$ are called Steiner nodes or non-fixed sites.

- $C = \{c_{ij}\}_{i,j \in S_D}$ is the matrix which gives for any pair of sites of $S_D$, the (non-negative) cost of laying a line between them. When the direct connection between $i$ and $j$ is not possible, we take $c_{ij} = \infty$.

- $R = \{r_{ij}\}_{i,j \in S_D^{(I)}}$ is an integer matrix of requirements of connection between pairs of sites of $S_D^{(I)}$. We will require $r_{ij}$ node disjoint paths between fixed sites $i$ and $j$, where $r_{ij}$ usually is strictly greater than 1.

- $E = \{(i,j); \forall i,j \in S_D$ such that $c_{ij} < \infty\}$, this is the set of feasible connections between switch sites belonging to $S_D$.

- $G_B = (S_D, E)$ is the undirected simple graph modeling the feasible connections on the Backbone Network.

**Definition 4.2.1 (Backbone Network Design Problem-BNDP)** *We define the Backbone Network Design Problem $BNDP(S_D, E, C, R)$ as the problem of finding a subgraph $\mathcal{H}_B$ of $G_B$ of minimum cost such that the nodes of $\mathcal{H}_B$ include those in $S_D^{(I)}$ and $\mathcal{H}_B$ satisfies the connection requirements specified in $R$. We will denote by $\Gamma_{BNDP}$ the space of feasible solutions associated with the problem.*

Let us observe that this definition is equivalent to the GSP-NC definition presented in Appendix C. We introduce some supplementary auxiliary definitions here which will be used in the descriptions of the proposed algorithms.

**Definition 4.2.2 (key-node)** *Given a BNDP instance and a feasible solution $\mathcal{G}_{sol} \in \Gamma_{BNDP}$, we define a **key-node** as a non-fixed site (that is, a site in $S_D \setminus S_D^{(I)}$) with degree at least three in $\mathcal{G}_{sol}$.*

**Definition 4.2.3 (key-path)** *Given a BNDP instance and a feasible solution $\mathcal{G}_{sol} \in \Gamma_{BNDP}$, we define a **key-path** as a path in $\mathcal{G}_{sol}$ such that all its intermediate sites are non-fixed sites with degree two in $\mathcal{G}_{sol}$, and whose endpoints are either fixed sites or key-nodes.*

**Notation 4.2.4** *Let $\mathcal{G}_{sol} \in \Gamma_{BNDP}$ be a feasible solution. If each edge of $\mathcal{G}_{sol}$ belongs to some path between two fixes switch sites, then it is possible to decompose $\mathcal{G}_{sol}$ in key-paths (i.e, there is a set of key-paths such that every edge of $\mathcal{G}_{sol}$ belongs to one and only one key-path). We will denote by $K(\mathcal{G}_{sol}) = (p_1, \ldots, p_h)$ the decomposition of $\mathcal{G}_{sol}$ in key-paths, ordered by decreasing cost.*

**Definition 4.2.5 (key-tree)** *Given a BNDP instance, a feasible solution $\mathcal{G}_{sol} \in \Gamma_{BNDP}$ and a key-node $v \in \mathcal{G}_{sol}$, we define the **key-tree** associated with $v$ as the tree in $\mathcal{G}_{sol}$ which is conformed by all the key-paths that have $v$ as one of its endpoints. Topologically, we can see it as a set of chains having $v$ as endpoint in common (the key-node $v$).*

The previous definitions are inspired on key-path and key-node definitions used in [131] in the context of SPG. We present a small instance example, based on the network shown in Figure 4.2. In this network there are six fixed switch sites, colored black and labeled $s_1$, $s_2$, $s_3$, $s_4$, $s_5$ and $s_6$, and four non-fixed switch sites, colored white. The connections that can be used to build a solution are shown in the figure, annotated with their costs. The connection requirements are the following (supposing the $R$ elements ordered by their index number):

This corresponds to asking for three node-disjoint paths between $s_2$ and $s_3$, three node-disjoint paths between $s_2$ and $s_4$, and two node-disjoint paths between a node of $\{s_1, s_5, s_6\}$ with any other fixed site.

Figure 4.3 presents a minimal feasible solution (of cost 29) to this problem instance. As it can be seen, not all non-fixed sites are needed in order to satisfy the connection requirements. Observe that in the solution there are two key-nodes and therefore two key-trees.

$$R = \begin{pmatrix} - & 2 & 2 & 2 & 2 & 2 \\ 2 & - & 3 & 3 & 2 & 2 \\ 2 & 3 & - & 2 & 2 & 2 \\ 2 & 3 & 2 & - & 2 & 2 \\ 2 & 2 & 2 & 2 & - & 2 \\ 2 & 2 & 2 & 2 & 2 & - \end{pmatrix}$$

Figure 4.1: Connection requirement matrix.

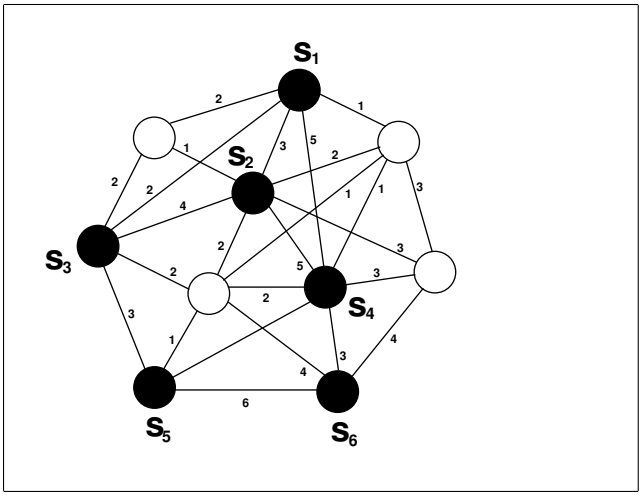



Figure 4.2: Example of a BNDP instance.

To apply GRASP to the BNDP, it is necessary to customize the different GRASP components described in Section 2.3. In the next two sections, we propose a path based construction phase and three local searches which are based on different neighborhood definitions. By combining suitably these local searches, we yield a series of GRASP versions which will be explained in detail in Section 4.5. Firstly, we will present the construction phase algorithms.

## 4.3 BNDP Construction Phase Algorithms

For this phase, the method proposed can be seen as an extension of the Takahashi-Matsuyama algorithm [127], which is a heuristic for computing a (low cost) Steiner tree, and works by searching for shortest paths between pairs of nodes not already connected. Our extension has a quite different objective, as we need to efficiently compute $k$ low-cost node-disjoint paths from $i$ to $j$, with $i, j \in S_D^{(I)}$. The construction

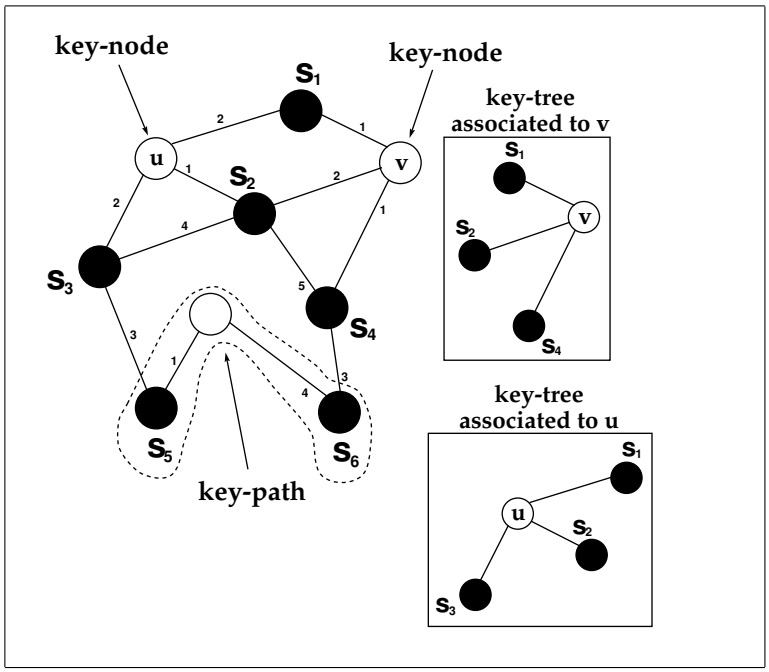



Figure 4.3: A solution to the graph example given in Figure 4.2.

phase algorithm is denominated $\mathrm{ConstPhase}$. In addition we give a slight variant of this one, to which we denominated $\mathrm{ConstPhase}*$. Both algorithms are described below.

### 4.3.1 Algorithm ConstPhase

The algorithm builds iteratively a network satisfying the matrix $R = \{r_{ij}\}_{i,j \in S_D^{(I)}}$ of connection requirements between fixed sites, i.e. given $i, j \in S_D^{(I)}$ there exists $r_{ij}$ node-disjoint paths connecting them in the network.

The algorithm (shown in Figure 4.4) takes as inputs the graph $G_B$ of feasible connections on the backbone network, the matrix of connection costs $C$, the matrix of connection requirements $R$, and the GRASP parameter $k$. In line 1 we initialize:

- the current solution $\mathcal{G}_{sol}$ with the sites of $S_D^{(I)}$ and an empty set of links,
- the matrix $M = \{m_{ij}\}_{i,j \in S_D^{(I)}}$ (indicating the connection requirements not yet satisfied between fixed sites) with $m_{ij} = r_{ij}$, $\forall i, j \in S_D^{(I)}$,
- the set $\mathcal{P} = \{\mathcal{P}_{ij}\}_{i,j \in S_D^{(I)}}$ (used to store the $r_{ij}$ computed paths between two fixed sites $i, j \in S_D^{(I)}$) with $\mathcal{P}_{ij} = \emptyset$, $\forall i, j \in S_D^{(I)}$,

**Procedure ConstPhase($G_B$,$C$,$R$,$k$);**

1 $\mathcal{G}_{sol} \leftarrow (S_D^{(I)}, \emptyset)$; $m_{ij} \leftarrow r_{ij}\ \forall i,j \in S_D^{(I)}$; $\mathcal{P}_{ij} \leftarrow \emptyset\ \forall i,j \in S_D^{(I)}$; $A_{ij} \leftarrow 0\ \forall i,j \in S_D^{(I)}$;
2 **while** $\exists m_{ij} > 0$ **such that** $A_{ij} < \mathrm{MAX\_ATTEMPT}$ **do**
3 **Let** $i,j \in S_D^{(I)}$ **be a randomly chosen pair of fixed switch sites such that** $m_{ij} > 0$ ;
4 $\bar{\mathcal{G}} \leftarrow (G_B \setminus \mathcal{P}_{ij})$;
5 **Let** $\bar{C}$ **be the matrix given by:** $\bar{c}_{uv} \leftarrow \begin{cases} 0 & \textbf{if } (u,v) \in \mathcal{G}_{sol}, \\ c_{uv} & \textbf{if } (u,v) \in (\bar{\mathcal{G}} \setminus \mathcal{G}_{sol}). \end{cases}$ ;
6 $\mathcal{L}_p \leftarrow$ **the** $k$ **shortest paths from** $i$ **to** $j$ **on** $\bar{\mathcal{G}}$**, considering the matrix** $\bar{C}$;
7 **if** $\mathcal{L}_p = \emptyset$ **then** $A_{ij} \leftarrow A_{ij} + 1$; $\mathcal{P}_{ij} \leftarrow \emptyset$; $m_{ij} \leftarrow r_{ij}$;
8 **else**
9 **if** $\exists \hat{p} \in \mathcal{L}_p$ **such that** $\mathrm{COST}_{|\bar{C}}(\hat{p}) = 0$ **then** $p \leftarrow \hat{p}$;
10 **else** $p \leftarrow \mathrm{Select\_Random}(\mathcal{L}_p)$; $\mathcal{G}_{sol} \leftarrow \mathcal{G}_{sol} \cup \{p\}$;
11 $\mathcal{P}_{ij} \leftarrow \mathcal{P}_{ij} \cup \{p\}$; $m_{ij} \leftarrow m_{ij} - 1$;
12 $[\mathcal{P}, M] \leftarrow \mathrm{General\_Update\_Matrix}(\mathcal{G}_{sol}, \mathcal{P}, M, p, i, j)$;
13 **end_if;**
14 **end_while;**
15 **return** $\mathcal{G}_{sol}$, $\mathcal{P}$;
**end ConstPhase;**

Figure 4.4: ConstPhase pseudo-code.

- and the auxiliary matrix $A = \{A_{ij}\}_{i,j \in S_D^{(I)}}$ (used to record when there has not been found a path between two fixed sites) with $A_{ij} = 0$, $\forall i,j \in S_D^{(I)}$.

Loop 2-14 is repeated until all the fixed sites have satisfied their connection requirements or for a certain pair of fixed sites $i,j \in S_D^{(I)}$ have not been found $r_{ij}$ node-disjoint paths connecting them after $\mathrm{MAX\_ATTEMPT}$ attempts.

Each iteration works of the following way. Line 3 selects randomly (and uniformly) a pair $i,j \in S_D^{(I)}$ such that $m_{ij} > 0$ (there exists at least one requirement not yet computed among them). Line 4 computes the network $\bar{\mathcal{G}} = (G_B \setminus \mathcal{P}_{ij})$. Note that this network does not contain any edge and node of $\mathcal{P}_{ij}$ excepting $i$ and $j$; therefore, every path communicating $i$ with $j$ in $\bar{\mathcal{G}}$ will be node-disjoint with respect to the paths already added to $\mathcal{P}_{ij}$. Line 5 computes an auxiliary matrix $\bar{C}$ of connection costs where any connection $(u,v) \in \mathcal{G}_{sol}$ has cost zero. This allows to reuse the already existing edges in $\mathcal{G}_{sol}$ (without considering their costs), when computing new node-disjoint paths between two fixed sites. Line 6 computes the $k$ shortest paths from $i$ to $j$ on $\bar{\mathcal{G}}$ using the matrix $\bar{C}$. These paths are stored in the restricted candidate list $\mathcal{L}_p$. Line 7 checks if $\mathcal{L}_p$ is empty. If this is the case (assuming that $G_B$ satisfies the matrix $R$, that is to say, the space of feasible solutions is non-empty), we re-initialize $\mathcal{P}_{ij}$ and $m_{ij}$ since $\mathcal{P}_{ij}$ contains a separating set between $i$ and $j$ on $G_B$ and therefore there does not exist a path from $i$ to $j$ in $\bar{\mathcal{G}}$, they are in different connected components. Otherwise, if $i$ and $j$ are in the same connected component in $\bar{\mathcal{G}}$, in order to not increase the cost of $\mathcal{G}_{sol}$, in line 9 we search a path $\hat{p} \in \mathcal{L}_p$ such that $\mathrm{COST}_{|\bar{C}}(\hat{p}) = 0$ (its cost with respect to $\bar{C}$). If this is successful, we assign to $p$ the found path. Otherwise, line 10 selects randomly (and uniformly) a path $p$

from $\mathcal{L}_p$ and it is added to $\mathcal{G}_{sol}$ in the same line. Since in both cases we obtained a new path $p$ node-disjoint with respect to the paths in $\mathcal{P}_{ij}$, in line 11 the set $P_{ij}$ is updated by adding the path $p$ to it and $m_{ij}$ is decremented by one. On the other hand, when adding this new path, new node-disjoint paths can have been introduced between some pairs of fixed sites, the auxiliary procedure $\mathrm{General\_Update\_Matrix}$ is called in line 12 in order to update some connection requirements of the fixed sites belonging to $p$ and the set $\mathcal{P}$. The description of this procedure (given in Sub-Section 4.3.1) explains in detail the introduced updates.

Once finalized the loop 2-14, the built feasible solution $\mathcal{G}_{sol}$ is returned in line 16. Figure 4.5 illustrates when a new node-disjoint path is added between two fixed switch sites $i, j \in S_D^{(I)}$. Let us note that by construction the new path $p$ connecting $i$ and $j$ is node-disjoint with respect to the set of paths $\mathcal{P}_{ij}$ already present in $\mathcal{G}_{sol}$.

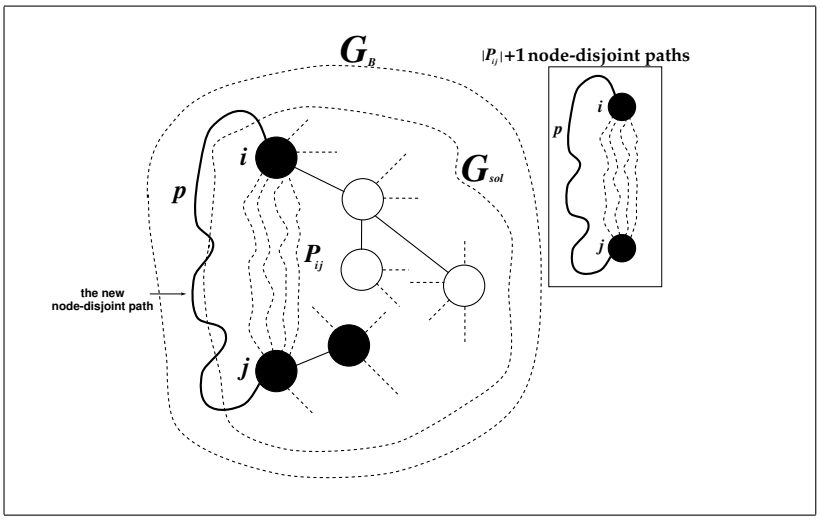



Figure 4.5: Computation of a new node-disjoint path between the fixed sites $i$ and $j$.

**General_Update_Matrix description**

The algorithm (shown in Figure 4.6) receives as input the current solution (in construction) $\mathcal{G}_{sol}$, the matrix $\mathcal{P}$ of computed paths, the matrix $M$ indicating the requirements not yet satisfied, two fixed sites $i$ and $j$, and the path $p$ computed among them. The loop 1-15 analyzes each fixed site belonging to the path $p$ in order to update certain connection requirements with other fixed sites. More specifically, the algorithm iteratively analyzes each $k \in S_D^{(I)}, k \neq i, j$ such that $k \in p$ and it checks if in the computed path $p$ there exists a sub-path connecting $k$ with $i$ (resp. $j$) node-disjoint with respect to the already present in $\mathcal{P}_{ik}$ (resp. $\mathcal{P}_{kj}$). If this is the case, the set $\mathcal{P}_{ik}$ (resp. $\mathcal{P}_{kj}$) is updated by adding $p_{(i,k)}$ (resp. $p_{(k,j)}$) to it, and $m_{ik}$ and $m_{ki}$ (resp. $m_{kj}$ and $m_{jk}$) are decremented by one.

```
Procedure General_Update_Matrix(G_sol,P,M,p,i,j);

1  for each k ∈ S_D^(I), k ≠ i,j such that k ∈ p do
2    if m_ik > 0 then
3      if (NODES(P_ik) ∩ NODES(p_(i,k)) = {i,k}) then
4        P_ik ← P_ik ∪ {p_(i,k)};
5        m_ik ← m_ik − 1; m_ki ← m_ki − 1;
6      end_if;
7    end_if;
8    if m_kj > 0 then
9      if (NODES(P_kj) ∩ NODES(p_(k,j)) = {k,j}) then
10       P_kj ← P_kj ∪ {p_(k,j)};
11       m_kj ← m_kj − 1; m_jk ← m_jk − 1;
12     end_if;
13   end_if;
14 end_for_each;
15 return P, M;
end General_Update_Matrix;
```

Figure 4.6: General_Update_Matrix pseudo-code.

**Proposition 4.3.1** *Once the algorithm* $\mathrm{General\_Update\_Matrix}$ *finalizes the following points are satisfied* $\forall i, j \in S_D^{(I)}$*:*

*i)* $\mathcal{P}_{ij} = \emptyset$ *iff* $m_{ij} = r_{ij}$.

*ii) If* $m_{ij} = k$ *(with* $k \in 0..r_{ij}$*) then there exist at least* $r_{ij} - k$ *node-disjoint paths from* $i$ *to* $j$ *in* $\mathcal{G}_{sol}$.

*iii) The relation* $|P_{ij}| = r_{ij} - m_{ij}$ *is satisfied in each* $ConstPhase$ *iteration.*

**Proof.** Firstly, let us assume that when $\mathrm{General\_Update\_Matrix}$ is called in line 12 of $\mathrm{ConstPhase}$, $\mathcal{P}$ and $M$ satisfy points $i - iii$.
Let $i, j \in S_D^{(I)}$ be the input fixed switch sites and $p$ the path connecting $i$ with $j$ computed by $\mathrm{ConstPhase}$. Loop 1-14 analyzes $\forall k \in S_D^{(I)}, k \in p, k \neq i, j$ the following cases.
*Case 1: Lines 2-7.* If $m_{ik} > 0$ we know that there exit $r_{ij} - m_{ij}$ node-disjoint paths communicating $i$ and $j$ in $\mathcal{G}_{sol}$. In addition, if $m_{ij} = r_{ij}$ we have that $\mathcal{P}_{ij} = \emptyset$. If condition $\mathrm{NODES}(\mathcal{P}_{ik}) \cap \mathrm{NODES}(p_{(i,k)}) = \{i, k\}$ is true, the sub-path $p_{(i,k)}$ is added to $\mathcal{P}_{ik}$ (in line 4) since it is node-disjoint with respect to the already present in $\mathcal{P}_{ik}$. The values of $m_{ik}$ and $m_{ki}$ are decremented by one in line 5 preserving thus the points $\mathrm{i} - \mathrm{iii}$.
*Case 2: Lines 8-13.* Idem to the previous case.

**QED**

The following proposition demonstrates the constructive correctness of the algorithm $\mathrm{ConstPhase}$.

**Proposition 4.3.2** *If $A_{ij} < \mathrm{MAX\_ATTEMPT}, \forall i,j \in S_D^{(I)}$ then the graph returned by $\mathrm{ConstPhase}$ is a feasible solution for the BNDP satisfying the matrix of connection requirements $R$.*

**Proof.** In the proof, we assume that there exists a subnetwork $\mathcal{G}_{sol} \subseteq G_B$ satisfying the matrix $R$.
In line 1 the algorithm initializes:

- $\mathcal{G}_{sol}$ with the set of fixed switch sites $S_D^{(I)}$ and an empty set of links,
- the auxiliary matrix $M$ (indicating connections that we know satisfies network $\mathcal{G}_{sol}$) with $m_{ij} = r_{ij}$, $\forall i,j \in S_D^{(I)}$,
- the matrix $\mathcal{P}$ (which will store the computed paths from $i$ to $j$ on $\mathcal{G}_{sol}$, $\forall i,j \in S_D^{(I)}$) with empty.

Suppose that for certain iteration the condition in line 2 is TRUE. In line 3 we choose randomly a pair $i,j \in S_D^{(I)}$ of fixed switch sites such that $m_{ij} > 0$. Line 4 computes the auxiliary network $\bar{\mathcal{G}} = (G_B \setminus \mathcal{P}_{ij})$. Notice that, if there exits a path from $i$ to $j$ in $\bar{\mathcal{G}}$, this one is node-disjoint with respect to the paths of $\mathcal{P}_{ij}$. In addition, line 5 computes an auxiliary cost matrix $\bar{\mathcal{C}}$ associated with $\bar{\mathcal{G}}$, by assigning cost zero to the connections that already are in $\mathcal{G}_{sol}$. Knowing that $\mathrm{General\_Update\_Matrix}$ preserves the condition: $|P_{ij}| = r_{ij} - m_{ij}$, lines 5-13 searches for a new path from $i$ to $j$ on $\mathcal{G}_{sol}$ considering $\bar{\mathcal{C}}$. Let us analyze the following situations.
*Case 1: $\nexists p \subset \bar{\mathcal{G}}$ connecting $i$ with $j$.* In this case, line 7 re-initializes $\mathcal{P}_{ij}$ and $m_{ij}$ since $\mathcal{P}_{ij}$ contains a separating set between $i$ and $j$ on $G_B$. The construction is resumed from line 2.
*Case 2: $\exists p \subset \bar{\mathcal{G}}$.* In this case, we differentiate the following subcases:

i) If there exits a path $\hat{p} \subset \bar{\mathcal{G}}$ such that $\hat{p} \subset \mathcal{G}_{sol}$ then $\hat{p}$ will be select as the new node-disjoint path between $i$ and $j$ (line 9). This allows to satisfy a new node-connectivity requirement between $i$ and $j$, without increasing the cost of the current solution.

ii) Otherwise, in line 9 we select a path $p$ from the list of paths $\mathcal{L}_p$ (computed in line 7). As $p \notin \mathcal{G}_{sol}$ the current solution $\mathcal{G}_{sol}$ is updated in line 9.

In both cases ($2.i$ and $2.ii$), the indicator $m_{ij}$ is decremented by one in line 11.

Based on the construction process described above, it easy to see that once finalized loop 2-14, if $m_{ij} = 0, \forall i,j \in S_D^{(I)}$ (i.e. $A_{ij} < \mathrm{MAX\_ATTEMPT}, \forall i,j \in S_D^{(I)}$) then the built solution $\mathcal{G}_{sol}$ satisfies the matrix $R$.

**QED**

It is possible to consider a variant of $\mathrm{ConstPhase}$ where we select randomly (and uniformly) a path from $\mathcal{L}_p$ without checking if there already exists a path $\hat{p}$ contained in $\mathcal{L}_p$ such that $\mathrm{COST}_{|\bar{C}}(\hat{p}) = 0$ (line 9

**Procedure ConstPhase*(**$G_B$**,**$C$**,**$R$**,**$k$**);**

**1** $\mathcal{G}_{sol} \leftarrow (S_D^{(I)}, \emptyset)$; $m_{ij} \leftarrow r_{ij}\ \forall i,j \in S_D^{(I)}$; $\mathcal{P}_{ij} \leftarrow \emptyset\ \forall i,j \in S_D^{(I)}$; $A_{ij} \leftarrow 0\ \forall i,j \in S_D^{(I)}$;
**2** **while** $\exists m_{ij} > 0$ **such that** $A_{ij} < \mathrm{MAX\_ATTEMPT}$ **do**
**3** **Let** $i,j \in S_D^{(I)}$ **be a randomly chosen pair of fixed switch sites such that** $m_{ij} > 0$ ;
**4** $\bar{\mathcal{G}} \leftarrow (G_B \setminus \mathcal{P}_{ij})$;
**5** **Let** $\bar{C}$ **be the matrix given by:** $\bar{c}_{uv} \leftarrow \begin{cases} 0 & \textbf{if } (u,v) \in \mathcal{G}_{sol}, \\ c_{uv} & \textbf{if } (u,v) \in (\bar{\mathcal{G}} \setminus \mathcal{G}_{sol}). \end{cases}$ ;
**6** $\mathcal{L}_p \leftarrow$ **the** $k$ **shortest paths from** $i$ **to** $j$ **on** $\bar{\mathcal{G}}$**, considering the matrix** $\bar{C}$;
**7** **if** $\mathcal{L}_p = \emptyset$ **then** $A_{ij} \leftarrow A_{ij} + 1$; $\mathcal{P}_{ij} \leftarrow \emptyset$; $m_{ij} \leftarrow r_{ij}$;
**8** **else**
**9** $p \leftarrow \mathrm{Select\_Random}(\mathcal{L}_p)$; $\mathcal{G}_{sol} \leftarrow \mathcal{G}_{sol} \cup \{p\}$;
**10** $\mathcal{P}_{ij} \leftarrow \mathcal{P}_{ij} \cup \{p\}$; $m_{ij} \leftarrow m_{ij} - 1$;
**11** $[\mathcal{P}, M] \leftarrow \mathrm{General\_Update\_Matrix}(\mathcal{G}_{sol}, \mathcal{P}, M, p, i, j)$;
**12** **end_if;**
**13** **end_while;**
**14** **return** $\mathcal{G}_{sol}$, $\mathcal{P}$;
**end ConstPhase*;**

Figure 4.7: ConstPhase* pseudo-code.

of $\mathrm{ConstPhase}$ pseudo-code). This variant is shown in Figure 4.7. This way, we can reduce the execution time for the $\mathrm{ConstPhase}$, the tradeoff being that we may lose opportunities for using zero cost paths.

**Proposition 4.3.3** *If* $A_{ij} < \mathrm{MAX\_ATTEMPT},\ \forall i,j \in S_D^{(I)}$ *then the graph returned by* $\mathrm{ConstPhase}*$ *is a feasible solution for the BNDP satisfying the matrix of connection requirements* $R$*.*

**Proof.** It is similar to the proof for the $\mathrm{ConstPhase}$ with the difference that we do not check if $\exists \hat{p} \in \mathcal{L}_p$ such that $\mathrm{cost}_{|\bar{C}}(\hat{p}) = 0$. Directly, a path from $\mathcal{L}_p$ is selected randomly and uniformly (line 9).

**QED**

## 4.4 BNDP Local Search Phase Algorithms

Usually the solution built by the construction phase is not even a local optimum. For this reason the GRASP metaheuristic applies a local search phase in order to improve this solution. We propose three local search algorithms for the BNDP, which are based on different neighborhood structures. Specifically, we designed a key-path based local search, a path based local search and a tree based local search, which can work in complementary form, running in combined way. We will give in Section 4.5 the GRASP algorithms that can be obtained by combining these local search algorithms. Next, we describe a key-path based local search algorithm for the BNDP. We call it $\mathbf{LocalSearch1}$.

### 4.4.1 Algorithm LocalSearch1

We start by defining a Neighborhood Structure which will be used by the **LocalSearch1** algorithm.

**Definition 4.4.1 (key-path based Neighborhood Structure)** *Let $\mathcal{G}_{sol}$ be a feasible solution satisfying the matrix of connection requirements $R$. Given a key-path $p \subset \mathcal{G}_{sol}$, we define a neighbor solution of $\mathcal{G}_{sol}$ as: $\hat{\mathcal{G}}_{sol} = (\mathcal{G}_{sol} \setminus p) \cup \hat{p}$, where $\hat{p}$ is another key-path connecting the endpoints of $p$ and maintaining the feasibility in the new network $\hat{\mathcal{G}}_{sol}$.*

*The Key-Path Neighborhood of $\mathcal{G}_{sol}$ is composed of the neighbor solutions obtained by applying the previous operation to each of the different key-paths in $\mathcal{K}(\mathcal{G}_{sol}) = (p_1, \ldots, p_h)$.*

The **LocalSearch1** algorithm builds iteratively neighbor solutions by replacing key-paths from the current solution by other key-paths which have the same endpoints. As we will see in Proposition 4.4.2, the feasibility of the built neighbor solution is preserved in each iteration. The process is repeated until the key-path replacements do not induce a better feasible solution.

```
Procedure LocalSearch1(G_B,C,G_sol);

1  improve ← TRUE;
2  while improve do
3     improve ← FALSE;
4     K(G_sol) ← the decomposition in key-paths of G_sol;
5     while not(improve) and ∃ key-paths not yet analyzed do
6        Let p ∈ K(G_sol) be a key-path not yet analyzed with ends u and v;
7        Ĥ ← the subgraph induced by NODES(p) ∪ (S_D \ NODES(G_sol));
8        p̂ ← the shortest path from u to v on Ĥ;
9        if COST(p̂) < COST(p) then
10          G_sol ← (G_sol \ p) ∪ p̂;
11          improve ← TRUE;
12       end_if;
13    end_while;
14 end_while.
15 return G_sol;
end LocalSearch1;
```

Figure 4.8: LocalSearch1 pseudo-code.

The algorithm (shown in Figure 4.8) takes as inputs the graph $G_B$ of feasible connections on the backbone network, the matrix of connection costs $C$, and the current feasible solution $\mathcal{G}_{sol}$. In line 1 we initialize with TRUE the indicator variable $improve$ used to indicate improvements obtained by the key-path replacements. Loop 2-13 searches for neighbor solutions analyzing each key-path in the current solution $\mathcal{G}_{sol}$ and replacing this by another key-path in order to improve its cost without losing the feasibility.

Each iteration works of the following way. In line 3 $improve$ is set to $\mathrm{FALSE}$. Line 4 computes the decomposition in key-paths of $\mathcal{G}_{sol}$, which we denote by $\mathcal{K}(\mathcal{G}_{sol})$. The internal loop 5-13 analyzes one at a time the key-paths from $\mathcal{K}(\mathcal{G}_{sol})$ with the aim of finding a new key-path of smaller cost to replace the corresponding original key-path. Line 6 selects randomly (and uniformly) a key-path $p \in \mathcal{K}(\mathcal{G}_{sol})$ not yet analyzed. We denote by $u$ and $v$ the ends of the selected key-path $p$. In line 7 we compute the sub-network induced by the set of sites $\textsc{nodes}(p) \cup (S_D \setminus \textsc{nodes}(\mathcal{G}_{sol}))$, which is denoted by $\hat{\mathcal{H}}$. Notice that in $\hat{\mathcal{H}}$ there are no sites from $(\mathcal{G}_{sol} \setminus p)$ excepting $u$ and $v$. Therefore, all path connecting $u$ with $v$ in $\hat{\mathcal{H}}$ reestablishes the feasibility of $(\mathcal{G}_{sol} \setminus p)$. Consequently, line 8 computes the shortest path from $u$ to $v$ on $\hat{\mathcal{H}}$, which is denoted by $\hat{p}$. Line 9 compares the costs of $\hat{p}$ and $p$. If $\hat{p}$ has cost smaller than $p$, in line 10 the current key-path $p$ is replaced by $\hat{p}$ in $\mathcal{G}_{sol}$ and in line 11 the indicator $improve$ is set to $\mathrm{TRUE}$ to restart the local search from line 2. Otherwise, if $\hat{p}$ has cost greater than $p$, the loop 5-13 continues with another key-path not yet analyzed or it finalizes since there are no more key-paths to analyze.

Once there are no more improvements by key-path replacements the loop 2-14 finalizes and the best neighbor solution found is returned in line 15.

Figure 4.9 illustrates a typical key-path replacement made by $\mathrm{LocalSearch1}$. The black nodes model the fixed switch sites whereas the white nodes the non-fixed switch sites. The broken lines represent paths between nodes. Note that the new key-path can have sites of $p$ as well as of $S_D \setminus \textsc{nodes}(\mathcal{G}_{sol})$.

The following proposition demonstrates the feasibility preservation in each $\mathrm{LocalSearch1}$ iteration.

**Proposition 4.4.2** *Given a feasible solution $\mathcal{G}_{sol}$ for the BNDP satisfying the matrix of connection requirements $R$, the algorithm* $\mathrm{LocalSearch1}$ *preserves the feasibility, returning a neighbor feasible solution satisfying $R$.*

**Proof.** Let us suppose that $\mathrm{LocalSearch1}$ does not preserve the feasibility. Necessarily, in certain iteration we would have:

- the current solution $\mathcal{G}_{sol}$ is feasible,
- the path $\hat{p}$ computed in lines 7-8 satisfies: $\textsc{cost}(\hat{p}) < \textsc{cost}(p)$, where $p \in \mathcal{K}(\mathcal{G}_{sol})$ is the current key-path. Hence, line 10 will be computed.
- the network: $\hat{\mathcal{G}} = (\mathcal{G}_{sol} \setminus p) \cup \hat{p}$ is not feasible, i.e. $\exists i, j \in S_D^{(I)}$ such that in $\hat{\mathcal{G}}$ there are no $r_{ij}$ node-disjoint paths connecting them.

By construction, this would imply: $\textsc{internal\_nodes}(\hat{p}) \cap \textsc{nodes}(\mathcal{G}_{sol} \setminus p) \neq \emptyset$, which is a contradiction since $\textsc{nodes}(\hat{p}) \subseteq (\textsc{nodes}(p) \cup (S_D \setminus \textsc{nodes}(\mathcal{G}_{sol}))$. Therefore the network $\hat{\mathcal{G}}$ is feasible satisfying the matrix of connection requirements $R$, and completing thus the proof.

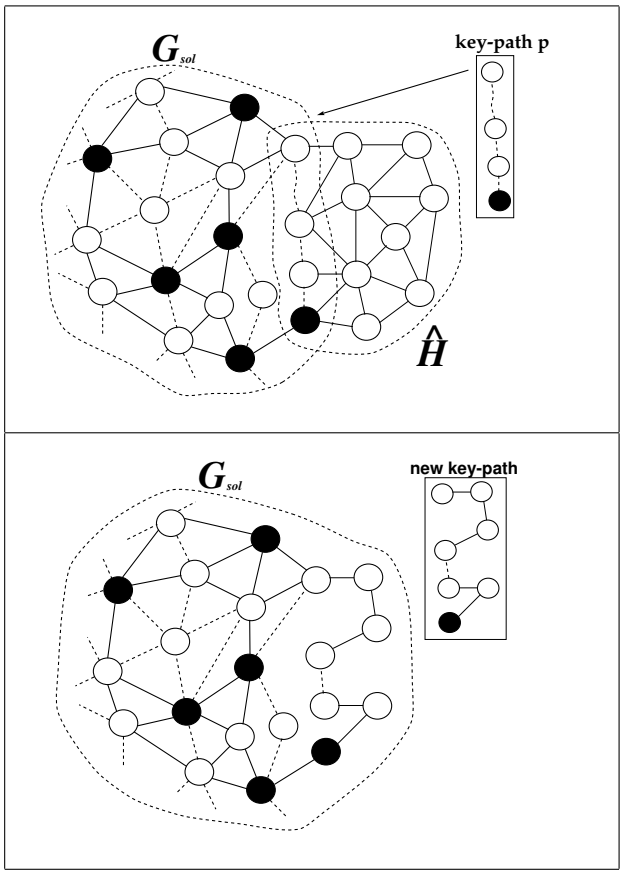



Figure 4.9: A generic key-path replacement computed by $\mathrm{LocalSearch1}$.

**QED**

The solutions delivered by $\mathrm{ConstPhase}$ and $\mathrm{ConstPhase}*$ not necessarily are minimal. If the constructed solution is minimal, the minimality will be preserved by $\mathrm{LocalSearch1}$. The following proposition demonstrates the minimality preservation in each $\mathrm{LocalSearch1}$ iteration.

**Proposition 4.4.3** *If the algorithm* $\mathrm{LocalSearch1}$ *receives as input a minimal feasible solution, the returned solution preserves the minimality.*

**Proof.** Let us denote $\mathcal{G}$ the solution delivered by $\mathrm{LocalSearch1}$. Again, by contradiction, let us suppose that $\mathcal{G}$ is not minimal. Necessarily, in certain iteration we would have that the current solution $\mathcal{G}_{sol}$ is minimal and by executing lines 6-10 the resulting network $\hat{\mathcal{G}} = (\mathcal{G}_{sol} \setminus p) \cup \hat{p}$ is not minimal. This implies that there exists an edge $e \in \hat{\mathcal{G}}$ such that $\hat{\mathcal{G}} \setminus \{e\}$ is feasible. We have the following cases:

i) $e \in \hat{p}$. We reach a contradiction, since $\hat{p}$ is a new key-path replacing to $p$ in $\mathcal{G}_{sol}$, i.e. $(\mathcal{G}_{sol} \setminus p) \cap \hat{p} = \{u, v\}$ (being $u, v$ the ends of $p$); therefore removing any edge from $\hat{p}$ the feasibility is lost.

ii) $e \notin \hat{p}$. As the connection requirements when removing $p$ are reestablished by the key-path $\hat{p}$, it is easy to see that if $\hat{\mathcal{G}} \setminus \{e\}$ is feasible then $\mathcal{G}_{sol} \setminus \{e\}$ also would be feasible, contradicting thus its minimality.

In this way, $(i)$ and $(ii)$ imply that $\mathcal{G}$ is a minimal feasible solution.

**QED**

Now, we propose a local search strategy based in key-paths replacement which is more complex than $\mathrm{LocalSearch1}$ but more flexible since when replacing a key-path we can use a set of switch sites that contains the set used by $\mathrm{LocalSearch1}$. Hence, we can see this new strategy as a generalization of $\mathrm{LocalSearch1}$. It is based on a path-based approach where iteratively, key-paths belonging to the current solution are replaced by other paths suitably constructed preserving the feasibility.

### 4.4.2 Algorithm LocalSearch2

Before describing in detail the local search algorithm, we introduce a suitable structure for the neighborhood and some auxiliary definitions.

**Definition 4.4.4 (path based Neighborhood Structure)** *Let $\mathcal{G}_{sol}$ be a feasible solution satisfying the matrix of connection requirements $R$. Given a key-path $p \subset \mathcal{G}_{sol}$, we define a neighbor solution of $\mathcal{G}_{sol}$ as: $\hat{\mathcal{G}}_{sol} = (\mathcal{G}_{sol} \setminus p) \cup \hat{p}$, where $\hat{p}$ is another path connecting the endpoints of $p$ and maintaining the feasibility in the new network $\hat{\mathcal{G}}_{sol}$. The substitute path $p$ is not necessarily a key-path and it can contain nodes and edges belonging to $\mathcal{G}_{sol}$ whenever the feasibility is preserved.*

*The Path Neighborhood of $\mathcal{G}_{sol}$ is composed of the neighbor solutions obtained by applying the previous operation to each of the different key-paths in $\mathcal{K}(\mathcal{G}_{sol}) = (p_1, \ldots, p_h)$.*

**Definition 4.4.5** *Let $p$ be a key-path belonging to the feasible solution $\mathcal{G}_{sol}$ and $\mathcal{P} = \{\mathcal{P}_{ij}\}_{i,j \in S_D^{(I)}}$ the set of node-disjoint paths between fixed switch sites (initially computed by $\mathrm{ConstPhase}$ or $\mathrm{ConstPhase}*$). We define:*

$$V_p(\mathcal{P}) = \left\{(i,j) \in S_D^{(I)} \times S_D^{(I)} | \exists \hat{p}_{ij} \in \mathcal{P}_{ij}, \text{ such that } p \subseteq \hat{p}_{ij}\right\}.$$

*This is the set of pairs of fixed switch sites of $S_D^{(I)}$ which "depend on" key-path $p$, that is to say, $p$ is part of at least one path $\hat{p}_{ij}$ in the set $\mathcal{P}_{ij}$ of the node-disjoint paths connecting $i$ and $j$.*

**Notation 4.4.6** *Given a key-path $p \in \mathcal{G}_{sol}$ and $i, j \in S_D^{(I)}$ such that $(i,j) \in V_p(\mathcal{P})$ we will denote by $\hat{p}_{ij}$ the path of $\mathcal{P}_{ij}$ containing $p$.*

**Definition 4.4.7** *Given a key-path $p \subset \mathcal{G}_{sol}$ and the set $\mathcal{P}$, we define the following set:*

$$\mathcal{X}_p(\mathcal{P}) = \bigcup_{\forall (i,j) \in V_p(\mathcal{P})} \text{NODES}(\mathcal{P}_{ij} \setminus \hat{p}_{ij}),$$

*where $\hat{p}_{ij} \in \mathcal{P}_{ij}$ (as above) denotes the path containing $p$. This is the union over every pair of fixed sites $i$, $j$ which depend on key-path $p$, of all the nodes belonging to paths in $\mathcal{P}_{ij}$ which do not contain $p$.*

Based on these definitions, we design a path-based local search algorithm for the BNDP, which we call $\text{LocalSearch2}$. Notice that, by definition, the space of feasible solutions induced by the Path Neighborhood Structure includes the space of feasible solutions induced by the Key-Path Neighborhood Structure defined previously. In particular, the proposed algorithm is used in combined way with the key-tree based local search algorithm $\text{LocalSearch3}$ (presented in Sub-Section 4.4.3) since both are designed on structurally different neighborhoods. Next, we give a detailed description of $\text{LocalSearch2}$ and some topological properties satisfied by the constructed neighbor solutions.

The algorithm builds iteratively neighbor solutions by replacing key-paths from the current solution by other paths suitably designed so that the feasibility is preserved (these paths are not necessarily key-paths since they can introduce new key-nodes). This process is repeated until the key-path replacements do not induce a better feasible solution.

In more detail, the algorithm (shown in Figure 4.10) takes as inputs the graph $G_B$ of feasible connections on the backbone network, the matrix of connection costs $C$, the current feasible solution $\mathcal{G}_{sol}$ and the set $\mathcal{P} = \{\mathcal{P}_{ij}\}_{i,j \in S_D^{(I)}}$ which contains all the computed paths between fixed switch sites. In line 1 we initialize with $\text{TRUE}$ the indicator variable $improve$ used to indicate improvements obtained by the key-path replacements. In line 2 we compute the decomposition in key-paths of $\mathcal{G}_{sol}$, denoted by $\mathcal{K}(\mathcal{G}_{sol})$. Loop 3-17 searches for neighbor solutions by analyzing each key-path in $\mathcal{K}(\mathcal{G}_{sol})$ and replacing (if it is possible) these ones by other paths in order to improve its cost without losing the feasibility. When we reach a better feasible solution by a key-path replacement the local search resumes from this new feasible solution.

Each iteration works of the following way. In line 4 the indicator $improve$ is set to $\text{FALSE}$. The internal loop 5-16 analyzes each key-path $p \in \mathcal{K}(\mathcal{G}_{sol})$ and using an auxiliary network searches for a substitute path which must preserve the feasibility when performing the replacement. Let $p$ be the current key-path and $u, v$ its ends. Let us consider the set of sites: $\hat{S} = \text{NODES}(p) \cup (S_D \setminus \mathcal{X}_p(\mathcal{P}))$; by definition of $\mathcal{X}_p(\mathcal{P})$, in the set $(S_D \setminus \mathcal{X}_p(\mathcal{P}))$ there are no sites of paths belonging to $\mathcal{P}$ that do not contain to $p$. That is to say, any site of $(S_D \setminus \mathcal{X}_p(\mathcal{P}))$ belongs to some path from $\mathcal{P}$ containing $p$, or does not belong to $\mathcal{P}$. Therefore, if $\hat{\mathcal{H}}$ is the subnetwork induced by $\hat{S}$ in $G_B$, any path connecting $u$ and $v$ in $\hat{\mathcal{H}}$ is a potential substitute for $p$, since (as we will see in Proposition 4.4.8) this one preserves the feasibility. In lines 6-7 we compute the network $\hat{\mathcal{H}}$ and the auxiliary matrix of connection costs $\hat{C}$ (in $\hat{C}$ we assign costs zero to the edges belonging to $\hat{\mathcal{H}} \cap (\mathcal{G}_{sol} \setminus p)$, in order to reuse them, without considering their costs, in the computation of the substitute

**Procedure LocalSearch2(**$G_B$**,**$C$**,**$\mathcal{G}_{sol}$**,**$\mathcal{P}$**);**

1 $improve \leftarrow \text{TRUE}$**;**
2 $\mathcal{K}(\mathcal{G}_{sol}) \leftarrow$ **the decomposition in key-paths of** $\mathcal{G}_{sol}$**;**
3 **while** $improve$ **do**
4 $improve \leftarrow \text{FALSE}$**;**
5 **for each key-path** $p \in \mathcal{K}(\mathcal{G}_{sol})$ **with ends** $u, v$ **do**
6 $\hat{\mathcal{H}} \leftarrow$ **the subgraph induced by** $\text{NODES}(p) \cup (S_D \setminus \mathcal{X}_p(\mathcal{P}))$**;**
7 **Compute** $\hat{C}$ **where** $\hat{c}_{ij} = \begin{cases} 0 & \textbf{if } (i,j) \in (\mathcal{G}_{sol} \setminus p), \\ c_{ij} & \textbf{otherwise}, \end{cases}$
8 $\bar{p}$ **the shortest path from** $u$ **to** $v$ **on** $\hat{\mathcal{H}}$**, considering** $\hat{C}$**;**
9 **if** $\text{COST}_{|\hat{C}}(\bar{p}) < \text{COST}(p)$ **then**
10 $\mathcal{G}_{sol} \leftarrow (\mathcal{G}_{sol} \setminus p) \cup \bar{p}$**;**
11 $\mathcal{P}$ **is updated as follows:** $\forall \hat{p} \in \mathcal{P}$ **such that** $p \subseteq \hat{p}$**, we have:** $\hat{p} \leftarrow (\hat{p} \setminus p) \cup \bar{p}$**;**
12 $improve \leftarrow \text{TRUE}$**;**
13 **if** $\exists s \in \bar{p}, s \neq u, v$ **such that** $degree(s) \geq 3$ **in** $\mathcal{G}_{sol}$ **then**
14 $\mathcal{K}(\mathcal{G}_{sol}) \leftarrow$ **the decomposition in key-paths of** $\mathcal{G}_{sol}$**;**
15 **end_if;**
16 **end_if;**
17 **end_for_each;**
18 **end_while.**
19 **return** $\mathcal{G}_{sol}$**,** $\mathcal{P}$**;**
**end LocalSearch2;**

Figure 4.10: LocalSearch2 pseudo-code.

path). Using $\hat{C}$ line 8 computes the shortest path from $u$ to $v$ on $\hat{\mathcal{H}}$, which is denoted by $\bar{p}$. As $\bar{p}$ can have edges in $(\mathcal{G}_{sol} \setminus p)$, in line 9 we compare only the costs of $\bar{p} \setminus (\mathcal{G}_{sol} \setminus p)$ and $p$. If $p$ has cost greater than $\bar{p} \setminus (\mathcal{G}_{sol} \setminus p)$, in line 10 the current solution $\mathcal{G}_{sol}$ is updated replacing $p$ by $\bar{p}$. In addition, considering set $\mathcal{P}$, every path $\hat{p} \in \mathcal{P}$ such that $p \subseteq \hat{p}$ is updated by replacing $p$ by $\bar{p}$ in line 11. In this way, let us note that $\mathcal{G}_{sol}$ and $\mathcal{P}$ are suitably updated in each iteration. The indicator $improve$ is set to $\text{TRUE}$ in line 12. If the path $\bar{p}$ introduces a new key-node in $\mathcal{G}_{sol}$, we recompute the key-paths decomposition in lines 13-15, and the local search is resumed from line 5 with this newly built neighbor solution. On the other hand, if the condition in line 9 is $\text{FALSE}$ (i.e. $\bar{p}$ is not better than $p$), another key-path from $\mathcal{K}(\mathcal{G}_{sol})$ will be analyzed by internal loop 5-17.

Once there are no more improvements by key-tree replacements the current solution $\mathcal{G}_{sol}$ and the set $\mathcal{P}$ are returned in line 19.

The following proposition demonstrates the feasibility preservation in each $\text{LocalSearch2}$ iteration.

**Proposition 4.4.8** *If* $\text{LocalSearch2}$ *receives as input a feasible solution* $\mathcal{G}_{sol}$ *satisfying the matrix of connection requirements* $R$*, the feasibility is preserved at any time.*

**Proof.** Let us suppose that $\mathrm{LocalSearch2}$ does not preserve the feasibility. Necessarily, in certain iteration we would have:

i) the current solution $\mathcal{G}_{sol}$ is feasible,

ii) the path $\hat{p}$ computed in lines 6-8 satisfies: $\mathrm{COST}(\bar{p} \setminus (\mathcal{G}_{sol} \setminus p)) < \mathrm{COST}(p)$, where $p \in \mathcal{K}(\mathcal{G}_{sol})$ is the current key-path, and therefore line 10 is computed.

iii) the resulting network: $\hat{\mathcal{G}} = (\mathcal{G}_{sol} \setminus p) \cup \hat{p}$ is not feasible, i.e. $\exists i, j \in S_D^{(I)}$ such that in $\hat{\mathcal{G}}$ there are no $r_{ij}$ node-disjoint paths connecting them.

Let $\hat{p}_{ij} \in \mathcal{P}_{ij}$ be such that $p \subseteq \hat{p}_{ij}$. We define the path $p_{aux} = (\hat{p}_{ij} \setminus p) \cup \bar{p}$. Since the nodes of $\mathcal{X}_p(\mathcal{P})$ are excluded from $\hat{H}$ (line 6) and therefore in $\hat{H}$ there are no nodes of $(\mathcal{P}_{ij} \setminus \hat{p}_{ij})$, we have that:

$$\mathrm{INTERNAL\_NODES}(p_{aux}) \cap \mathrm{INTERNAL\_NODES}(p_{ij}) = \emptyset,\ \forall p_{ij} \in (\mathcal{P}_{ij} \setminus \hat{p}_{ij}),$$

which contradicts $(\mathrm{iii})$. Hence, the network $\hat{\mathcal{G}}$ computed in line 10 is feasible satisfying the matrix of connection requirements $R$. To complete the proof, notice that:

- all the paths $\hat{p} \in \mathcal{P}$ containing $p$ are updated in line 10 replacing $p$ by $\bar{p}$, and accordingly there are $r_{ij}$ node-disjoints paths in $\mathcal{P}_{ij}$ connecting $i$ with $j$, $\forall i, j \in S_D^{(I)}$,
- lines 13-15 recompute the decomposition in key-paths of $\mathcal{G}_{sol}$ (which was updated in line 10 by $\hat{\mathcal{G}}$) if a new key-node was introduced.

In this way, the feasibility of the current topology is guaranteed in each local search iteration.

**QED**

Like $\mathrm{LocalSearch1}$, when the algorithm $\mathrm{LocalSearch2}$ receives as input a minimal feasible solution, the solution computed by it will be also minimal. The following proposition demonstrates the minimality preservation in each $\mathrm{LocalSearch2}$ iteration.

**Proposition 4.4.9** *If the algorithm* $\mathrm{LocalSearch2}$ *receives as input a minimal feasible solution, the returned solution preserves the minimality.*

**Proof.** Let us suppose that $\mathrm{LocalSearch2}$ does not preserve the minimality. Necessarily, for certain iteration we have:

- the current solution $\mathcal{G}_{sol}$ is minimal.
- there exists a path $\bar{p}$ replacing a key-path $p \in \mathcal{K}(\mathcal{G}_{sol})$, and moreover:

i) we have $\text{COST}(\bar{p} \setminus (\mathcal{G}_{sol} \setminus p)) < \text{COST}(p)$ and therefore the algorithm computes the network $\hat{\mathcal{G}} = (\mathcal{G}_{sol} \setminus p) \cup \bar{p}$ (line 10). In this proof, we will denote by $\mathcal{G}_{sol}$ to the current solution before its update.

ii) $\hat{\mathcal{G}}$ is not minimal (but it is feasible by Proposition 4.4.8).

Therefore, there exits an edge $e \in \hat{\mathcal{G}}$ such that $\hat{\mathcal{G}} \setminus \{e\}$ is feasible. We will analyze the following cases.
*Case A.* $e \in \mathcal{G}_{sol}$. Let us consider the network $\bar{\mathcal{G}} = (\hat{\mathcal{G}} \setminus \{e\}) \setminus \bar{p}$. Since in $\bar{p}$ (by construction of $\hat{\mathcal{H}}$) there are no nodes from $\mathcal{X}_p(\mathcal{P})$, applying $\mathcal{X}_p(\mathcal{P})$ definition, in $\bar{\mathcal{G}}$ we lose only one level of node-connectivity between pairs of sites of $V_p(\mathcal{P})$. The other pairs of fixed sites are not affected in their levels of node-connectivity. On the other hand, as $p \not\subset \bar{\mathcal{G}}$ and $p \cap \mathcal{X}_p(\mathcal{P}) = \emptyset$, in the network $\bar{\mathcal{G}} \cup p$ the lost node-connectivity levels between fixed sites from $V_p(\mathcal{P})$ are reestablished, implying therefore its feasibility. We have then the equality:

$$\bar{\mathcal{G}} \cup p = (\hat{\mathcal{G}} \setminus \{e\}) \setminus \bar{p} = ((\mathcal{G}_{sol} \setminus p) \cup \bar{p}) \setminus \{e\}) \setminus \bar{p} = \mathcal{G}_{sol} \setminus \{e\},$$

contradicting thus the minimality of $\mathcal{G}_{sol}$.
*Case B.* $e \notin \mathcal{G}_{sol}$. Firstly, it is easy to see that does not exist a path $p_2 \subset \mathcal{G}_{sol}$ from $u$ to $v$ such that $p_2 \cap \mathcal{X}_p(\mathcal{P}) = \emptyset$; otherwise, considering $\hat{\mathcal{C}}$, $p_2$ would be the shortest path from $u$ to $v$ in $\hat{\mathcal{H}}$ since it satisfies $\text{COST}_{|\hat{C}}(p_2) \overset{\text{by } \hat{C} \text{ def.}}{\overset{\uparrow}{=}} 0$, contradicting thus $\bar{p}$ being the shortest path in $\hat{\mathcal{H}}$. Hence, by $\mathcal{G}_{sol}$ minimality and $\hat{\mathcal{G}} \setminus \{e\}$ feasibility, it must exist a path $p_2 \subset (\mathcal{G}_{sol} \cup (\bar{p} \setminus \{e\}))$ such that $p_2 \cap (\bar{p} \setminus \{e\}) \neq \emptyset$ and $p_2 \cap \mathcal{X}_p(\mathcal{P}) = \emptyset$. By $\hat{C}$ definition, we have then:

$$\text{COST}_{|\hat{C}}(p_2) \overset{(p_2 \setminus \mathcal{G}_{sol}) \subset \bar{p}}{\overset{\uparrow}{<}} \text{COST}_{|\hat{C}}(\bar{p}),$$

which is a contradiction.
To conclude, cases A and B imply that $\hat{\mathcal{G}} \setminus \{e\}$ is non-feasible.

**QED**

We will compare the neighborhood structures associated with $\text{LocalSearch1}$ and $\text{LocalSearch2}$. Before, we introduce the following notation.

Given a feasible solution $\mathcal{G}$ for a BNDP instance and a key-path $p \subset \mathcal{G}$, let us denote by:

- $N_1(\mathcal{G}, p)$ the sub-space of neighbor solutions considered by $\text{LocalSearch1}$ when carrying out the replacement of $p$.

- $N_2(\mathcal{G}, p)$ the sub-space of neighbor solutions considered by $\text{LocalSearch2}$ when carrying out the replacement of $p$.

**Property 4.4.10** *Let $\mathcal{H}$ be a feasible solution and $p \subset \mathcal{H}$ a key-path. Let us suppose that* $\mathrm{LocalSearch1}$ *and* $\mathrm{LocalSearch2}$ *will analyze the replacement of $p$ on $\mathcal{H}$. Then, the following inequality is satisfied:*

$$\textsc{cost}(\mathcal{G}^2_{best}) \leq \textsc{cost}(\mathcal{G}^1_{best}),$$

*being $\mathcal{G}^1_{best}$ and $\mathcal{G}^2_{best}$ the resulting networks once computed the key-path replacement by* $\mathrm{LocalSearch1}$ *and* $\mathrm{LocalSearch2}$ *respectively.*

**Proof.** By definition of $\mathcal{X}_p(\mathcal{P})$ (see $\mathrm{LocalSearch1}$ description), we have that $N_1(\mathcal{H}, p) \subseteq N_2(\mathcal{H}, p)$. Moreover, since by construction $\mathrm{LocalSearch1}$ and $\mathrm{LocalSearch2}$ attain the best feasible solutions $\mathcal{H}^1_{best} \in N_1(\mathcal{H}, p)$ and $\mathcal{H}^2_{best} \in N_2(\mathcal{H}, p)$ respectively, we obtain the relation:

$$\textsc{cost}(\mathcal{H}^2_{best}) \leq \textsc{cost}(\mathcal{H}^1_{best}),$$

as required, and completing the proof.

**QED**

Next, we provide a third local search strategy which is structurally different with respect to the local search algorithms exposed above.

### 4.4.3 Algorithm LocalSearch3

Before introducing the local search strategy based on key-trees replacement, we define a suitable structure for the neighborhood.

**Definition 4.4.11 (key-tree based Neighborhood Structure)** *Let $\mathcal{G}_{sol}$ be a feasible solution satisfying the matrix of connection requirements $R$. Given a key-node $v \in \mathcal{G}_{sol}$ and its associated key-tree $\mathcal{T}_v \subset \mathcal{G}_{sol}$, we define a neighbor solution of $\mathcal{G}_{sol}$ as: $\hat{\mathcal{G}}_{sol} = (\mathcal{G}_{sol} \setminus \mathcal{T}_v) \cup \mathcal{T}$, where $\mathcal{T}$ is another key-tree spanning the endpoints of $\mathcal{T}_v$ and maintaining the feasibility in the new network $\hat{\mathcal{G}}_{sol}$.*

*The Key-Tree Neighborhood of $\mathcal{G}_{sol}$ is composed of the neighbor solutions obtained by applying iteratively the previous operation to each of the different key-trees in $\mathcal{G}_{sol}$.*

Based on this neighborhood structure, we design another local search algorithm for the BNDP, which we called $\mathrm{LocalSearch3}$. The proposed algorithm differs substantially from $\mathrm{LocalSearch1}$ and $\mathrm{LocalSearch2}$ since this one is based on key-tree replacements and the others on key-path replacements. However, we can use them running in combined form, exploiting thus the potentialities of both strategies. Next, we introduce a detailed description of $\mathrm{LocalSearch3}$ and some topological properties satisfied by the constructed neighbor solutions.

```
Procedure LocalSearch3(G_B,C,G_sol);

1  improve ← TRUE;
2  while improve do
3     improve ← FALSE;
4     Let X be the set of key-nodes in G_sol;
5     S̄ ← S_D \ NODES(G_sol);
6     while not(improve) and ∃ key-nodes not yet analyzed do
7        Let v ∈ X be not yet analyzed;
8        [G_sol, improve] ← General_RecConnect(G_B, C, G_sol, v, S̄);
9     end_while;
10 end_while;
11 return G_sol;
end LocalSearch3;
```

Figure 4.11: LocalSearch3 pseudo-code.

The algorithm builds iteratively neighbor solutions by replacing key-trees from the current solution by other key-trees which are suitably designed so that the feasibility is preserved. This process is repeated until the key-tree replacements do not induce a better feasible solution.

The algorithm (shown in Figure 4.11) takes as inputs the graph $G_B$ of feasible connections on the backbone network, the matrix of connection costs $C$, and the current feasible solution $\mathcal{G}_{sol}$. In line 1 we initialize with $\mathrm{FALSE}$ the indicator variable $improve$ used to indicate improvements obtained by the key-tree replacements. Loop 2-10 searches for neighbor solutions analyzing each key-node in the current solution $\mathcal{G}_{sol}$ and replacing (if it is possible) their respective key-trees by other key-trees in order to improve its cost without losing the feasibility. When we reach a better feasible solution by a key-tree replacement the local search resumes from this new feasible solution.

Each iteration works of the following way. In line 3 $improve$ is set to $\mathrm{FALSE}$. Line 4 computes the set $X$ of key-nodes of $\mathcal{G}_{sol}$. Line 5 computes the set $\bar{S}$ of non-fixed switch sites non-belonging to $\mathcal{G}_{sol}$. The internal loop 6-9 analyzes one at a time the key-nodes from $X$ with the aim of finding a suitable key-tree of smaller cost to replace the corresponding key-tree. Line 7 selects a site $v \in X$ randomly (and uniformly). In line 8 we execute the algorithm called $\mathrm{General\_RecConnect}$ in order to find a substitute key-tree for the key-tree associated with $v$, so that it has smaller cost than this one and preserves the feasibility (we give below a detailed description of this algorithm and Proposition 4.4.12 proves that it preserves the feasibility). If this search is successful, the $\mathrm{General\_RecConnect}$ delivers a better neighbor solution and the current solution $\mathcal{G}_{sol}$ is updated with it in the same line. In addition, $improve$ is set to $\mathrm{TRUE}$, to restart the local search from line 2. Otherwise, if $\mathrm{General\_RecConnect}$ cannot find a substitute key-tree, the loop 6-9 considers another key-node not yet analyzed or it finalizes since there are no more key-nodes to analyze.

Once there are no more improvements by key-tree replacements the current solution $\mathcal{G}_{sol}$ is returned in line 11.

### General_RecConnect description

The algorithm $\mathrm{General\_RecConnect}$ is an auxiliary procedure used by the algorithm $\mathrm{LocalSearch2}$. Given the current solution $\mathcal{G}_{sol}$ and a key-node $v \in \mathcal{G}_{sol}$, $\mathrm{General\_RecConnect}$ tries to build a better key-tree $\mathcal{T}$ spanning the endpoints of $\mathcal{T}_v$, where $\mathcal{T}_v$ is the key-tree associated a $v$. To preserve the feasibility, the substitute key-tree $\mathcal{T}$ is built using only sites of $\mathcal{T}_v$ and the non-fixed switch sites non-belonging to $\mathcal{G}_{sol}$. In addition the edges between the endpoints of $\mathcal{T}_v$ are not considered.

**Procedure General_RecConnect(**$G_B$,$C$,$\mathcal{G}_{sol}$,$v$,$\bar{S}$**);**

**1** $cost \leftarrow \mathrm{Cost\_Key\_Tree}(v, \mathcal{G}_{sol})$**;**
**2** $Y \leftarrow \mathrm{Nodes\_Key\_Tree}(v, \mathcal{G}_{sol})$**;**
**3** $Z \leftarrow \mathrm{Ends\_Key\_Tree}(v, \mathcal{G}_{sol})$**;**
**4** $\hat{S} \leftarrow (Y \setminus Z) \cup \bar{S}$**;**
**5** $U \leftarrow \{(i,j) \in G_B | i \in Z, j \in \hat{S}\}$**;**
**6** $\hat{\mathcal{H}} \leftarrow$ **the subgraph induced by** $\hat{S}$ **in** $G_B$**;** $\hat{\mathcal{H}} \leftarrow \hat{\mathcal{H}} \cup U$**;**
**7** $\mathcal{T} \leftarrow \{v\}$**;**
**8** **while** $\exists u \in Z$ **such that** $u \notin \mathcal{T}$ **do**
**9** $u \leftarrow \mathrm{Select\_Random}(X)$ **where** $X = \{u \in Z | u \notin \mathcal{T}\}$**;**
**10** $\mathcal{H} \leftarrow \hat{\mathcal{H}} \setminus (Z \setminus \{u\})$**;**
**11** $p \leftarrow$ **the shortest path from** $u$ **to** $\mathcal{T}$ **considering** $\mathcal{H}$**;**
**12** $\mathcal{T} \leftarrow \mathcal{T} \cup p$**;**
**13** **end_while;**
**14** **Iteratively remove all** $s \in \hat{S}$ **from** $\mathcal{T}$ **with degree 1;**
**15** **if** $(\mathrm{COST}(\mathcal{T}) < cost)$ **then**
**16** $\mathcal{G}_{sol} \leftarrow (\mathcal{G}_{sol} \setminus (Y \setminus Z)) \cup \mathcal{T}$**;**
**17** $improve \leftarrow \mathrm{TRUE}$**;**
**18** **else** $improve \leftarrow \mathrm{FALSE}$**;**
**19** **return** $\mathcal{G}_{sol}$**,** $improve$**;**
**end General_RecConnect;**

Figure 4.12: General_RecConnect pseudo-code.

The algorithm (shown in Figure 4.12) takes as inputs the graph $G_B$ of feasible connections on the backbone network, the matrix of connection costs $C$, the current feasible solution $\mathcal{G}_{sol}$, the current key-node $v$, and the set $\bar{S}$ of non-fixed switch sites non-belonging to $\mathcal{G}_{sol}$. Let $\mathcal{T}_v$ be the key-tree associated with $v$. Line 1 computes the cost of $\mathcal{T}_v$. Line 2 computes the set $Y$ of sites belonging to $\mathcal{T}_v$. Line 3 computes the set $Z \subset Y$ of endpoints of $\mathcal{T}_v$. Line 4 computes the set of sites $\hat{S} = (Y \setminus Z) \cup \bar{S}$, which includes all the non-fixed sites non-belonging to $\mathcal{G}_{sol}$ and all sites of $\mathcal{T}_v$ excepting their endpoints. In line 5 we compute the

set $U$ containing all the connections from $G_B$ which have an end in $Z$ and the other in $\hat{S}$. Clearly in $U$ there are no connections between sites of $Z$. Let $\hat{\mathcal{H}}$ be the sub-network induced by $\hat{S}$ in $\mathcal{G}_{sol}$. Line 6 adds to $\hat{\mathcal{H}}$ the set $U$. Let us notice that any spanning tree computed on $\hat{\mathcal{H}}$ is a potential substitute for $\mathcal{T}_v$ in $\mathcal{G}_{sol}$ since, when replacing $\mathcal{T}_v$ by this one the feasibility is preserved. In line 7 we initialize the substitute key-tree $\mathcal{T}$ with the fixed site $v$. Loop 8-13 builds iteratively a new key-tree by adding one at a time the nodes of $Z$ to $\mathcal{T}$. Line 9 selects randomly (and uniformly) a site $u \in Z$ not yet added to $\mathcal{T}$. In line 10 we consider the auxiliary network $\mathcal{H} = \hat{\mathcal{H}} \setminus (Z \setminus u)$ to compute a path from $u$ to $\mathcal{T}$. The sites of $(Z \setminus u)$ are not considered when connecting $u$ to $\mathcal{T}$, since (as we will see in Proposition 4.4.12) these must be endpoints in $\mathcal{T}$. Line 11 computes the shortest path from $u$ to $\mathcal{T}$ on $\mathcal{H}$. Let $p$ be this path, in line 11 we add $p$ to $\mathcal{T}$. Once all sites of $Z$ have been added to $T$, loop 8-13 finalizes, and the pendant Steiner nodes are removed from $\mathcal{T}$ in line 14. Let us note that these are not necessary to guarantee the feasibility. Besides, it is easy to see that a substitute key-tree can be always constructed since $\mathcal{T}_v \subseteq \hat{\mathcal{H}}$. In line 15 we compare the costs of $\mathcal{T}$ and $\mathcal{T}_v$. If $\mathcal{T}$ is a better key-tree, the current solution $\mathcal{G}_{sol}$ is updated in line 16 by replacing $T_v$ by $\mathcal{T}$. The indicator $improve$ is set to $\mathrm{TRUE}$ in line 17 (this is used by $\mathrm{LocalSearch2}$ to know if a new neighbor solution has been built). Otherwise, if $\mathcal{T}$ has greater cost than $\mathcal{T}_v$, $improve$ is set to $\mathrm{FALSE}$ in line 18. The indicator $improve$ and the solution $\mathcal{G}_{sol}$ are returned in line 19.

Figure 4.13 illustrates a generic key-tree replacement computed by $\mathrm{General\_RecConnect}$ algorithm. Again, the black nodes represent fixed switch sites whereas the white nodes represent the non-fixed switch sites. The broken lines are paths between sites.

- The first graph is the current feasible solution $\mathcal{G}_{sol}$ into which we will replace the key-tree $\mathcal{T}_v$ by another key-tree constructed by $\mathrm{General\_RecConnect}$.
- The second graph is the result of replacing in $\mathcal{G}_{sol}$ the key-tree $\mathcal{T}_v$ by another key-tree. Notice that the substitute key-tree has again the non-fixed site $v$ as a key-node.
- The third graph is also obtained when replacing the key-tree $\mathcal{T}_v$ by another key-tree. In this case, $v$ is not a key-node for the resulting network.

Let us note that the key-node corresponding to the substitute key-tree can be different since when removing the pendant concentrators in line 14 the original key-node could be deleted. The following proposition demonstrates the constructive correctness of the algorithm $\mathrm{General\_RecConnect}$.

**Proposition 4.4.12** *Given a feasible solution $\mathcal{G}_{sol}$ for the BNDP satisfying the matrix of connection requirements $R$, the set $\bar{S}$ of non-fixed switch sites not including in $\mathcal{G}_{sol}$, and a key-node $v \in \mathcal{G}_{sol}$. The algorithm* $\mathrm{General\_RecConnect}$ *builds a neighbor solution by replacing the key-tree associated with $v$ by another tree which preserves the feasibility.*

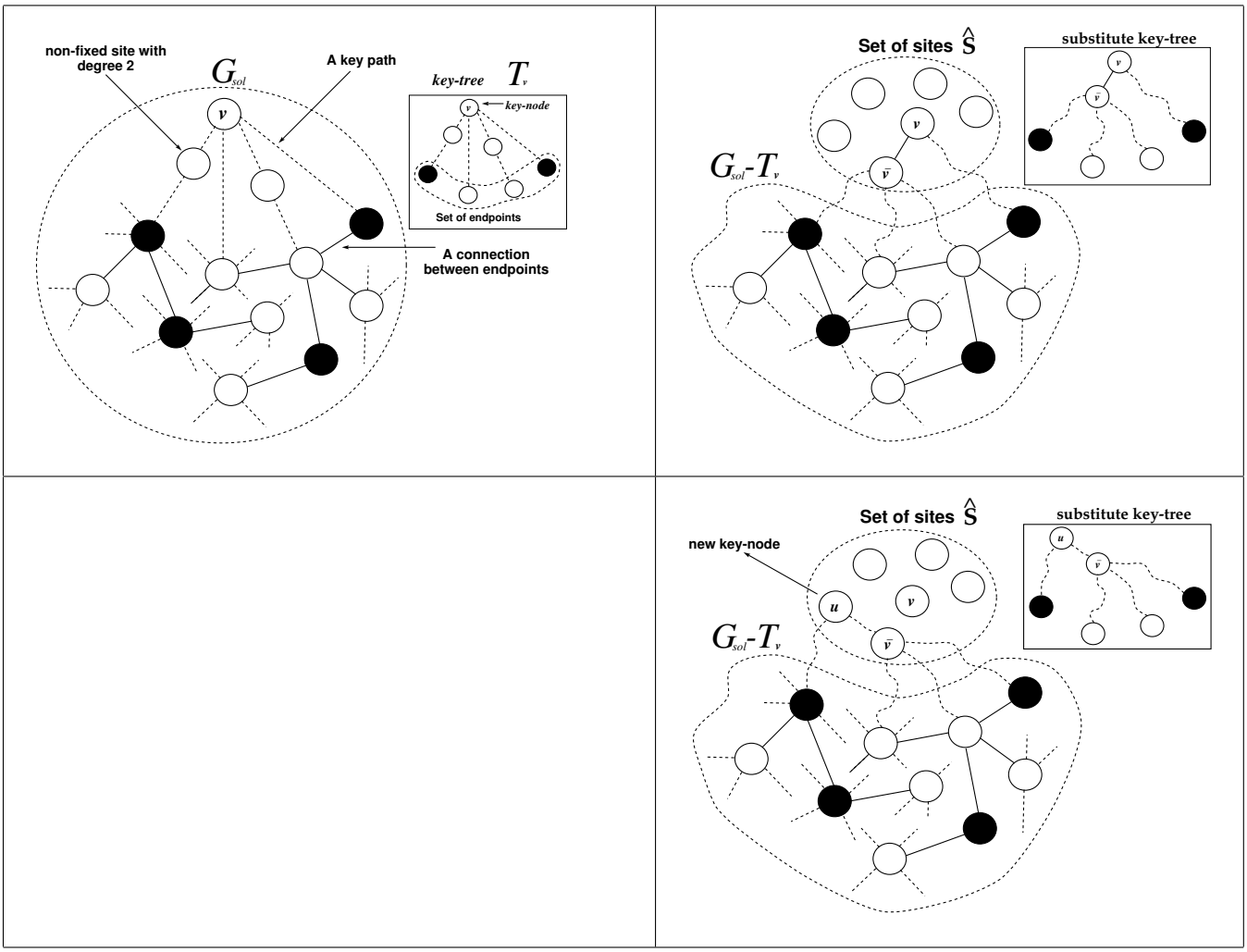



Figure 4.13: A generic key-tree replacement.

**Proof.** Let $\mathcal{T}_v$ be the key-tree associated with $v$. Lines 1-5 compute: the cost of $\mathcal{T}_v$, the set $Y$ of nodes in $\mathcal{T}_v$, the set $Z \subset \mathcal{T}_v$ of endpoints, the set $\hat{S} = (Y \setminus Z) \cup \bar{S}$, and the set $U = \{(i,j) \in G_B | i \in Z, j \in \hat{S}\}$. Line 6 computes the network: $\hat{\mathcal{H}} = U \cup G_B(\hat{S})$. Line 7 initializes $\mathcal{T}$ (the key-tree substitute) with the node $v$. It is easy to see that, by construction, once finalized loop $8-13$ and line 14, the network $\mathcal{T}$ has tree topology and furthermore:

i) $Z \subset \mathcal{T}$,

ii) the endpoints of $\mathcal{T}$ are exactly the nodes of $Z$,

iii) $\text{NODES}(\mathcal{T}) \cap \text{NODES}(\mathcal{G}_{sol}) = Z \cup J$, with $J \subseteq (Y \setminus Z)$,

iv) there exists a node $\hat{s} \in \hat{S}$ being root of the tree $\mathcal{T}$ (not necessarily $\hat{s} = v$).

If the condition in line 15 is true, the algorithm computes the network: $\hat{\mathcal{G}} = (\mathcal{G}_{sol} \setminus \mathcal{T}_v) \cup \mathcal{T}$ in line 16. Points $\mathrm{i} - \mathrm{iv}$ induce the feasibility of the network $\hat{\mathcal{G}}$ since replacing the key-tree $\mathcal{T}_v$ by $\mathcal{T}$ the lost node-connectivity requirements in $(\mathcal{G}_{sol} \setminus \mathcal{T}_v)$ are reestablished when adding $\mathcal{T}$. Hence, the network returned in line 19 is feasible for the BNDP satisfying the matrix $R$.

**QED**

Based on the previous proposition, the following proposition demonstrates the feasibility preservation in each $\mathrm{LocalSearch3}$ iteration.

**Proposition 4.4.13** *If* $\mathrm{LocalSearch3}$ *receives as input a feasible solution* $\mathcal{G}_{sol}$ *satisfying the matrix of connection requirements* $R$*, the feasibility is preserved during all the iterations of the algorithm.*

**Proof.** By contradiction, for certain iteration we have that $\mathcal{G}_{sol}$ is feasible fulfilling the matrix $R$, $X$ is its set of key-nodes, and there exists a key-node $u \in X$ such that $\mathrm{General\_RecConnect}$ returns a non-feasible solution. This contradicts Proposition 4.4.12. Hence, the algorithm preserves the feasibility at any time.

**QED**

As we mentioned previously, the solutions built by $\mathrm{ConstPhase}$ or $\mathrm{ConstPhase}*$ are not necessarily minimal. Anyway, if we reached a minimal topology in the construction phase or by another local search algorithm, the minimality preservation is guaranteed when running $\mathrm{LocalSearch3}$. The following proposition demonstrates that the minimality is preserved in each $\mathrm{LocalSearch3}$ iteration.

**Proposition 4.4.14** *If the algorithm* $\mathrm{LocalSearch3}$ *receives as input a minimal feasible solution, the returned solution preserves the minimality.*

**Proof.** Let us denote $\mathcal{G}$ the solution delivered by $\mathrm{LocalSearch3}$. Again, by contradiction, let us suppose that $\mathcal{G}$ is not minimal. Necessarily, in certain iteration we would have that the current solution $\mathcal{G}_{sol}$ is minimal and by executing lines 7-8 the resulting network is not minimal. Let $\hat{\mathcal{G}}$ be this solution, there exists an edge $e \in \hat{\mathcal{G}}$ such that $\hat{\mathcal{G}} \setminus \{e\}$ is feasible. Let us denote $\mathcal{T}_v$ and $\mathcal{T}$ the replaced key-tree and the new key-tree computed in line 8. Let $u$ be the key-node root of $\mathcal{T}$. We have the following cases:

i) $e \in \mathcal{T}$. Consider $\mathcal{T} \setminus \{e\}$; there exists a node $z \in Z$ (where $Z$ is the set of endpoints of $\mathcal{T}_v$) disconnected to the other sites belonging to $Z \setminus \{z\}$ and therefore its node-connectivity level with respect to these ones will be decreased by one in $\hat{\mathcal{G}} \setminus \{e\}$ losing thus the feasibility.

ii) $e \notin \mathcal{T}$. Clearly, the feasibility of $\hat{\mathcal{G}} \setminus \{e\}$ would imply the feasibility of $\mathcal{G}_{sol} \setminus \{e\}$, which is a contradiction.

Points $(i)$ and $(ii)$ imply that $\mathcal{G}$ is a minimal feasible solution.

**QED**

## 4.5 The GRASP algorithms for the BNDP

We now describe the general GRASP algorithm for approximately solving the BNDP. Figure 4.14 shows the corresponding pseudo-code. The generic procedures Construction_Phase and Local_Search can be instanced of the following way:

- Construction_Phase: by ConstPhase or ConstPhase$*$.
- Local_Search: by LocalSearch1 or LocalSearch2.

In the local search phase, the idea is to apply first key-path replacement moves (by running LocalSearch1 or LocalSearch2 for key-paths replacements) and the evaluation of key-tree replacement moves is performed only if there are no improving key-path replacement moves. Thus, we explore structurally different neighborhoods in combined form and the search is resumed from the beginning whenever we find one better neighbor feasible solution.

In the following, Construction_Phase will reference indifferently to ConstPhase or ConstPhase$*$, and in the same way Local_Search will reference to LocalSearch1 or LocalSearch2. Next, we introduce a detailed description of the algorithm GRASP_BNDP.

**Procedure GRASP_BNDP;**
**Input:** $G_B$, $C$, $R$, $k$, $seed$, $MaxIter$;

1 $min\_cost \leftarrow \infty$;
2 **for** $i = 1, \ldots, MaxIter$ **do**
3 $[\mathcal{G}_{sol}, \mathcal{P}] \leftarrow$ Construction_Phase$(G_B, C, R, k)$;
4 $cost\_sol \leftarrow \text{COST}(\mathcal{G}_{sol})$;
5 $\mathcal{G}_{sol} \leftarrow$ Local_Search$(G_B, C, \mathcal{G}_{sol}, \mathcal{P})$;
6 $best \leftarrow \text{COST}(\mathcal{G}_{sol})$;
7 **if** $(best < cost\_sol)$ **then goto line** 4;
8 $\mathcal{G}_{sol} \leftarrow$ LocalSearch3$(G_B, C, \mathcal{G}_{sol})$;
9 $best \leftarrow \text{COST}(\mathcal{G}_{sol})$;
10 **if** $(best < cost\_sol)$ **then goto line** 4;
11 **if** $(cost\_sol < min\_cost)$ **then**
12 $\mathcal{G}^{(\text{opt})} \leftarrow \mathcal{G}_{sol}$; $min\_cost \leftarrow cost\_sol$;
13 **end_if;**
14 **end_for;**
15 **return** $\mathcal{G}^{(\text{opt})}$;
**end GRASP_BNDP;**

Figure 4.14: General Version of the algorithm GRASP_BNDP.

The algorithm takes as inputs the graph $G_B$ of feasible connections on the backbone network, the matrix of connection costs $C$, the matrix of connection requirements $R$, the GRASP parameters $k$ (used in the construction phase), a seed for the pseudo random number generator $seed$ and the number of iterations

$MaxIter$ to be performed. The cost of the best found feasible solution is initialized with the value infinite ($\infty$) in line 1. The algorithm is repeated $MaxIter$ times exploring the space of feasible solutions and searching for the optimal feasible solution for the BNDP. Each iteration works of the following way.

In line 3, a greedy randomized feasible solution $\mathcal{G}_{sol}$ is built using the algorithm $\mathrm{Construction\_Phase}$ (i.e. $\mathrm{ConstPhase}$ or $\mathrm{ConstPhase}*$). In addition, it is also returned the set $\mathcal{P}$ of node-disjoint paths between fixed sites computed when building the solution. In line 4 the cost of $\mathcal{G}_{sol}$ is assigned to variable $cost\_sol$. In line 5 we call $\mathrm{Local\_Search}$ (i.e. $\mathrm{LocalSearch1}$ or $\mathrm{LocalSearch2}$) in order to find for neighbor feasible solutions with smaller cost. Depending on the algorithm, it searches for a better neighbor feasible solution by means of key-path replacement moves. In line 6 we compute the cost of the neighbor solution $\mathcal{G}_{sol}$ found in line 5. Line 7 compares the cost of the current solution with the one delivered by $\mathrm{Local\_Search}$. If a neighbor solution with smaller cost has been found by $\mathrm{Local\_Search}$, then the local search resumes from this new current solution executing from line 4. Otherwise, if no neighbor solution of better cost is found by $\mathrm{Local\_Search}$, then in line 8 we call the algorithm $\mathrm{LocalSearch3}$, which searches for neighbor solutions with smaller cost by applying key-tree replacement moves. In line 9 we compute the cost of the solution delivered by $\mathrm{LocalSearch3}$ in line 8. Again, if a neighbor feasible solution with smaller cost has been found by $\mathrm{LocalSearch3}$, then the local search resumes from this new current solution executing from line 4. Otherwise, if no neighbor solution with better cost is found by $\mathrm{LocalSearch3}$, then, if the solution found at the end of the local search phase is better than the best solution so far (line 11), we update in line 12 the best found feasible solution and the minimum cost. Once finalized the loop 2-14, the best found feasible solution $\mathcal{G}^{(\mathrm{opt})}$ is returned in line 15. Figure 4.15 is the execution diagram corresponding to the algorithm $\mathrm{GRASP\_BNDP}$.

## 4.6 Performance Tests

We present here the experimental results obtained with the $\mathrm{GRASP\_BNDP}$ algorithm in its different versions (depending on which construction phases and local search algorithms be instanced). The algorithms were implemented in ANSI C. The experiments were made on a Pentium IV with 1.7 GHz, and 1 Gbytes of RAM, running under Windows XP. In the performance testing phase all used instances were solved with the same GRASP parameter settings. In a previous tuning phase the candidate list size $k$ was chosen in the set $\{10, 15, 20, 30\}$. We tuned the value for the candidate list size by considering a reduced group of BNDP instances. As result of this tuning phase, we selected $k = 20$ as the value with better results since in the worse cases, it obtained the same solution costs that were obtained with the other values, beating them in many cases. In this way, we fixed $k = 20$ and $MaxIter = 300$ when running all the performance testing problems.

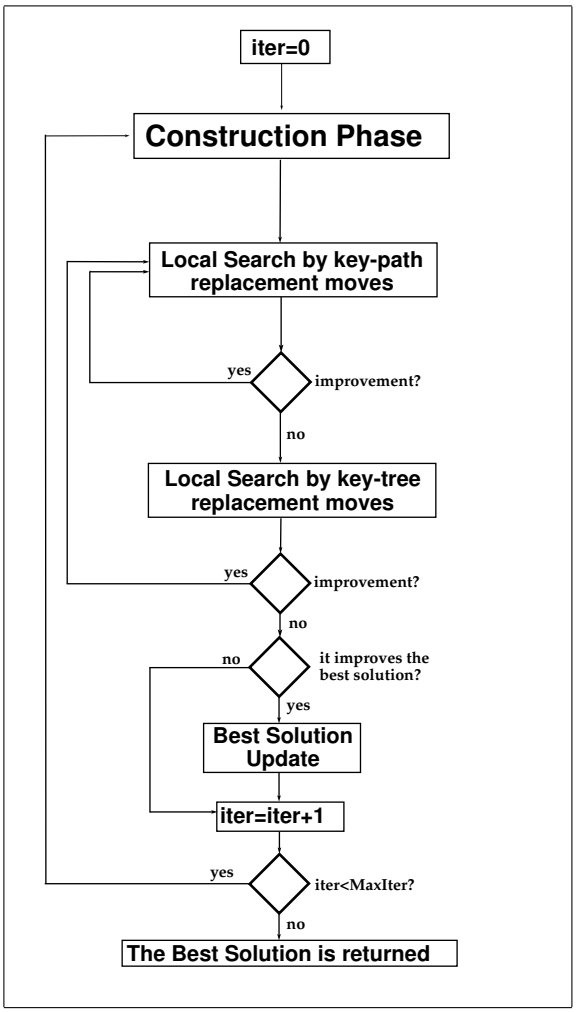



Figure 4.15: Execution Diagram associated with GRASP_BNDP.

### 4.6.1 BNDP test-set description

To our best knowledge, no library containing benchmark instances related to the BNDP (i.e. to the GSP-NC) exists. Nevertheless, there exist in the literature some related works where real problems with high survivability requirements are solved by means of the application of polyhedral algorithms [75, 78, 126]. We obtained some of these problems which were included in our test set.

The test-set for the GRASP_BNDP is composed of two sub-groups of instances: ones having known optimal cost (or at least a known lower bound) and others without known optimal solution nor lower bound. Altogether, we selected twenty-nine test problems as experimental suite, to investigate the effectiveness of the proposed method. For nine of them optimal solutions were known (or at least the optimal value), implying thus that the corresponding GRASP results could be more properly evaluated (i.e. we can compute the gaps with respect to the optimum costs). The other instances were generated by customizing *Traveling*

*Salesman Problems* (TSP) into BNDP instances. We describe in Table 4.1 the main characteristics of the twenty-nine problem instances. Figures 4.16 and 4.17 show the topologies associated with the test cases having a known optimal solution.

| **Problem instance** | **Nodes** | **Fixed nodes** | **Steiner nodes** | **Edges** | **Survivability requirements** |
|---|---|---|---|---|---|
| Instances with known optimal value | | | | | |
| Network 1 | 120 | 33 | 87 | 286 | 2-node-survivability |
| Network 2 | 83 | 22 | 61 | 262 | 4-node-survivability |
| Network 3 | 79 | 41 | 38 | 365 | heterogeneous (2 and 3 node connectivity) |
| Network 4 | 109 | 41 | 68 | 383 | heterogeneous (2, 3 and 4 node connectivity) |
| Network 5 | 121 | 27 | 94 | 386 | 2-node-survivability |
| Network 6 | 71 | 38 | 33 | 301 | heterogeneous (2 and 3 node connectivity) |
| Network 7 | 64 | 9 | 55 | 124 | 3-node-survivability |
| Network 8 | 38 | 38 | 0 | 71 | heterogeneous (1 and 2 node connectivity) |
| Network 9 | 116 | 116 | 0 | 173 | heterogeneous (1 and 2 node connectivity) |
| Instances without known optimal value nor tight lower bound | | | | | |
| Networks 10-14 | 150 | 100 | 50 | 11175 | heterogeneous (2 and 3 node connectivity) |
| Network 15 | 76 | 51 | 25 | 2850 | heterogeneous (2 and 3 node connectivity) |
| Network 16 | 114 | 76 | 38 | 6441 | heterogeneous (2 and 3 node connectivity) |
| Network 17 | 151 | 101 | 50 | 11325 | heterogeneous (2 and 3 node connectivity) |
| Network 18 | 114 | 76 | 38 | 6441 | heterogeneous (2 and 3 node connectivity) |
| Network 19 | 160 | 107 | 53 | 12720 | heterogeneous (2 and 3 node connectivity) |
| Network 20 | 186 | 124 | 62 | 17205 | heterogeneous (2 and 3 node connectivity) |
| Network 21 | 204 | 136 | 68 | 20706 | heterogeneous (2 and 3 node connectivity) |
| Network 22 | 216 | 144 | 72 | 23220 | heterogeneous (2 and 3 node connectivity) |
| Network 23 | 129 | 29 | 100 | 8256 | 3-node-survivability |
| Network 24 | 129 | 29 | 100 | 8256 | 3-node-survivability |
| Network 25 | 126 | 26 | 100 | 7875 | 3-node-survivability |
| Network 26 | 116 | 16 | 100 | 6670 | 3-node-survivability |
| Network 27 | 122 | 22 | 100 | 7381 | 3-node-survivability |
| Network 28 | 162 | 127 | 35 | 13041 | 2-node-survivability |
| Network 29 | 140 | 105 | 35 | 9730 | 2-node-survivability |

Table 4.1: Characteristics of the test cases.

In the following, due to the analogy between the BNDP and the GSP-NC, we will talk indifferently of Steiner nodes as non-fixed switch sites and fixed nodes as fixed switch sites. Next, we provide the description of each BNDP instance used in the performance testing phase.

Network 1 is a 2-node-survivability problem for which an optimal solution has been found by a backtracking algorithm [121]. Since there are few GSP-NC instances with connectivity requirements greater than two in the literature, we generated four instances having higher connectivity requirements (Networks 2, 3, 4 and 6) which were designed constructively in order to preserve a known optimal solution. Moreover, we generated one instance (Network 5, also with optimal feasible known solution) with a high number of Steiner nodes and a high density of edges on which we wanted to find a 2-node-survivable sub-network spanning the set of fixed nodes. We created this test instance with the aim of studying a relatively “dense”

network where the quotient between the number of Steiner nodes and the number of fixed nodes is higher than three. In Appendix C we give a detailed description of these six problem instances, with information about known optimal solutions, and their construction when relevant.

Network 7 represents a simplified version of a HSODTN (High Speed Optical Data Transmission Network) connecting different parts of a war ship. The reduced topology has 9 fixed switch sites (modelling strategic point in an aircraft carrier), 55 Steiner nodes (non-fixed switch sites) and 124 edges. A link between Steiner nodes has cost 1, a link between a Steiner node and a fixed node has cost 2, and a link between two fixed nodes has cost 4. The objective is to find a 3-node-survivable subnetwork with minimal cost (all connection requirement between fixed nodes are equal to 3). These model and other variants can be found in [78, 121, 126]. An optimal solution has been found by an exact parallel-distributed backtracking algorithm in [120].

Networks 8 and 9 are test cases respectively called LATA5S and LATADL; based on real networks from Bell Communications Research (later Bellcore, now Telcordia Technologies) [126]. Link costs are defined as geographical distances between nodes. The LATA5S problem has 38 nodes and 71 edges, and the LATADL problem has 116 nodes and 173 edges. In these problems, there are two classes of nodes: nodes of type 1, shown as circles in Figure 4.17, and nodes of type 2, shown as small squares. The connectivity requirements are that between two nodes of type 2 there must be two node-disjoint paths; and between a node of type 2 and a node of type 1, or between two nodes of type 1, there must be at least one path. For both instances, optimal solutions have been published in [126], with costs 4739 and 7400 respectively.

Networks 10 to 14 are BNDP instances built based on the TSP problems: kroA100, kroB100, kroC100, kroD100, and kroE100, extracted from TSPLIB. Specifically, we added iteratively fifty Steiner nodes to each one of the euclidian graphs associated with these instances. Starting from the original TSP graph (whose nodes will model the fixed sites), each Steiner node is added to the current graph one at a time, in addition its connection costs (with respect to the nodes already present in the graph in construction) are randomly chosen in the interval $\varrho \cdot [c_{min}, c_{max}]$; where $c_{min}$, $c_{max}$ are the minimum and maximum distances between two nodes of the original TSP graph, and $\varrho$ is a prefixed parameter. In particular, we set $\varrho = \frac{1}{8}$. The purpose of this parameter setting is potentially to generate Steiner nodes with lower connection costs (regarding their adjacent nodes) in comparison with the already existing links connecting the fixed nodes. Intuitively, when reducing the interval of possible costs for the new connections, in this way, we increase the probability that a Steiner node be a potential improver of feasible solutions. The resulting BNDP topology is a complete graph. We selected randomly eight fixed nodes, to which we associated 3-node-survivability requirements among them, and 2-node-survivability with respect to the other fixed nodes. These last ones have associated 2-node-survivability requirements among them.

Networks 15 to 22 are BNDP instances built based on the TSP problems: eil51, eil76, eil101, pr76, pr107, pr124, pr136, and pr144, extracted from TSPLIB. In the same way, by applying the process exposed

above, we generated BNDP instances by adding to these TSP instances: 25, 38, 50, 38, 53, 62, 68, and 72 Steiner nodes respectively. Let us note that in each resulting instance the number of fixed nodes is at least twice the number of Steiner nodes. On each resultant BNDP instance, we selected randomly six fixed nodes, to which we associated 3-node-survivability requirements among them, and 2-node-survivability with respect to the other fixed nodes. In addition, we established 2-node-survivability requirements among the other fixed nodes.

Networks 23 to 27 are BNDP instances built based on the TSP problems: bayg29, bays29, fri26, ulysses16, and ulysses22. Again, by using the same process that in the previous cases (but setting $\varrho = \frac{1}{3}$), we generated BNDP instances by adding to these problems 100 Steiner nodes. Let us notice that, in these designed networks, the number of Steiner nodes is at least three times the number of fixed nodes. For each resultant BNDP instance, the objective is to find a minimum-cost 3-node-survivable subnetwork spanning the fixed nodes.

Networks 28 and 29 are BNDP instances built based on the TSP problems bier127 and lin105. To each one of these cases, we added to it 35 Steiner nodes so that the resulting BNDP topology is a complete graph fulfilling the triangular inequality among its nodes (in Section 5.6.1, we explain in detail as generate an euclidian BNDP instance having Steiner nodes). Observe that, in the constructed networks, the number of fixed nodes is at least three times the number of Steiner nodes. The requirement for both BNDP instances is to find a minimum-cost 2-node-survivable subnetwork spanning the fixed nodes.

### 4.6.2 Numerical Results

Let us turn now to the study of the computational results. By combining the alternative algorithms for the construction phase and the local search phase, four versions of GRASP for the BNDP are yielded. We will distinguish them by means of the following notation:

**Heuristic $\mathcal{H}_1$:** it is the $\mathrm{GRASP\_BNDP}$ when instancing $\mathrm{Construction\_Phase}$ with $\mathrm{ConstPhase}$ and $\mathrm{Local\_Search}$ with $\mathrm{LocalSearch1}$,

**Heuristic $\mathcal{H}_2$:** it is the $\mathrm{GRASP\_BNDP}$ when instancing $\mathrm{Construction\_Phase}$ with $\mathrm{ConstPhase}$ and $\mathrm{Local\_Search}$ with $\mathrm{LocalSearch2}$,

**Heuristic $\mathcal{H}_3$:** it is the $\mathrm{GRASP\_BNDP}$ when instancing $\mathrm{Construction\_Phase}$ with $\mathrm{ConstPhase}*$ and $\mathrm{Local\_Search}$ with $\mathrm{LocalSearch1}$,

**Heuristic $\mathcal{H}_4$:** it is the $\mathrm{GRASP\_BNDP}$ when instancing $\mathrm{Construction\_Phase}$ with $\mathrm{ConstPhase}*$ and $\mathrm{Local\_Search}$ with $\mathrm{LocalSearch2}$.

As a result of the runs performed on the testing set, we noticed that the heuristics $\mathcal{H}_1$ and $\mathcal{H}_2$ overcame the results obtained by the heuristics $\mathcal{H}_3$ and $\mathcal{H}_4$ respectively. In other words, when using $\mathrm{ConstPhase}*$, in no case $\mathcal{H}_3$ obtained better results than $\mathcal{H}_1$ and either $\mathcal{H}_4$ in relation to $\mathcal{H}_2$. Furthermore, when comparing the GRASP results, $\mathcal{H}_1$ and $\mathcal{H}_2$ improved the topologies obtained by $\mathcal{H}_3$ and $\mathcal{H}_4$ in five and eleven BNDP instances respectively (improving them in average more than $3.16\%$ and $3.37\%$). Hence, in the following, we will concentrate in the analysis of the results obtained by $\mathcal{H}_1$ and $\mathcal{H}_2$.

In Tables 4.2 and 4.3 we show a summary of computational results obtained by applying the heuristics $\mathcal{H}_1$ and $\mathcal{H}_2$ on the Networks 1 to 9 (those with known optimal solution). These tables show some data about the performance of our GRASP algorithms for the mentioned instances and the structural characteristics of the optimal (or near-optimal) solutions it found. The column entries are from left to right:

- the average running time per iteration (secs./itr),
- the GRASP iteration number where the best feasible solution was found ($\mathrm{IT}$),
- the optimum cost ($\mathrm{COPT}$),
- the cost of the best feasible solution found by the GRASP algorithm ($\mathrm{BCF}$),
- the $\mathrm{GAP} = 100 \times \frac{\mathrm{BCF}-\mathrm{COPT}}{\mathrm{COPT}}$ (=percent relative error),
- the average of the improvement of the results of the local search phase over the construction phase ($\mathrm{LSI}$),
- the number of Steiner nodes and key-nodes of the best solution found by GRASP ($\mathrm{SN}$ and $\mathrm{KN}$ respectively),
- the number of edges of the best solution found by GRASP ($\mathrm{Edges}$).

| **Topology** | secs./itr | IT | COPT | BCF | GAP | LSI | SN | KN | Edges |
|---|---|---|---|---|---|---|---|---|---|
| Network 1 | 0.52 | 4 | 145 | 145 | 0.0% | 2.12% | 34 | 1 | 68 |
| Network 2 | 1.17 | 12 | 680 | 692 | 1.76% | 2.23% | 56 | 42 | 143 |
| Network 3 | 1.14 | 16 | 1848 | 1875 | 1.46% | 3.12% | 15 | 10 | 65 |
| Network 4 | 2.13 | 20 | 3980 | 4057 | 1.93% | 2.07% | 24 | 12 | 91 |
| Network 5 | 1.97 | 17 | 2393 | 2438 | 1.88% | 3.17% | 21 | 9 | 55 |
| Network 6 | 1.08 | 21 | 3031 | 3111 | 2.64% | 2.43% | 12 | 7 | 62 |
| Network 7 | 0.77 | 7 | 74 | 74 | 0.0% | 1.87% | 31 | 8 | 49 |
| Network 8 | 1.23 | 18 | 4739 | 4739 | 0.0% | - | - | - | 41 |
| Network 9 | 2.12 | 24 | 7400 | 7574 | 2.35% | - | - | - | 120 |
| | Average | | | | 1.33% | 2.43% | | | |

Table 4.2: Results associated with the best solutions found by $\mathcal{H}_1$ for the instances 1 to 9.

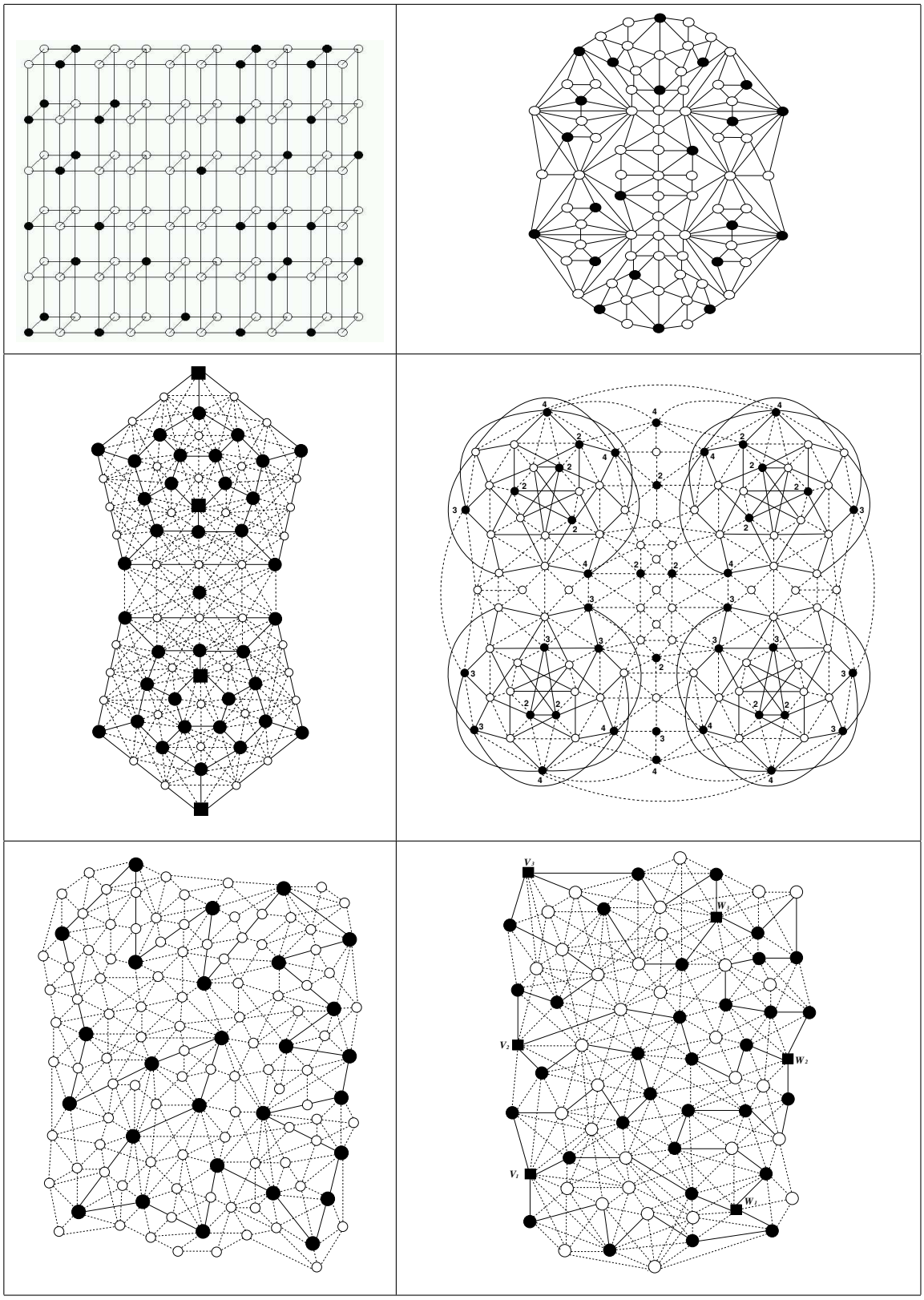



Figure 4.16: Topology of Networks 1, 2, 3, 4, 5, and 6.

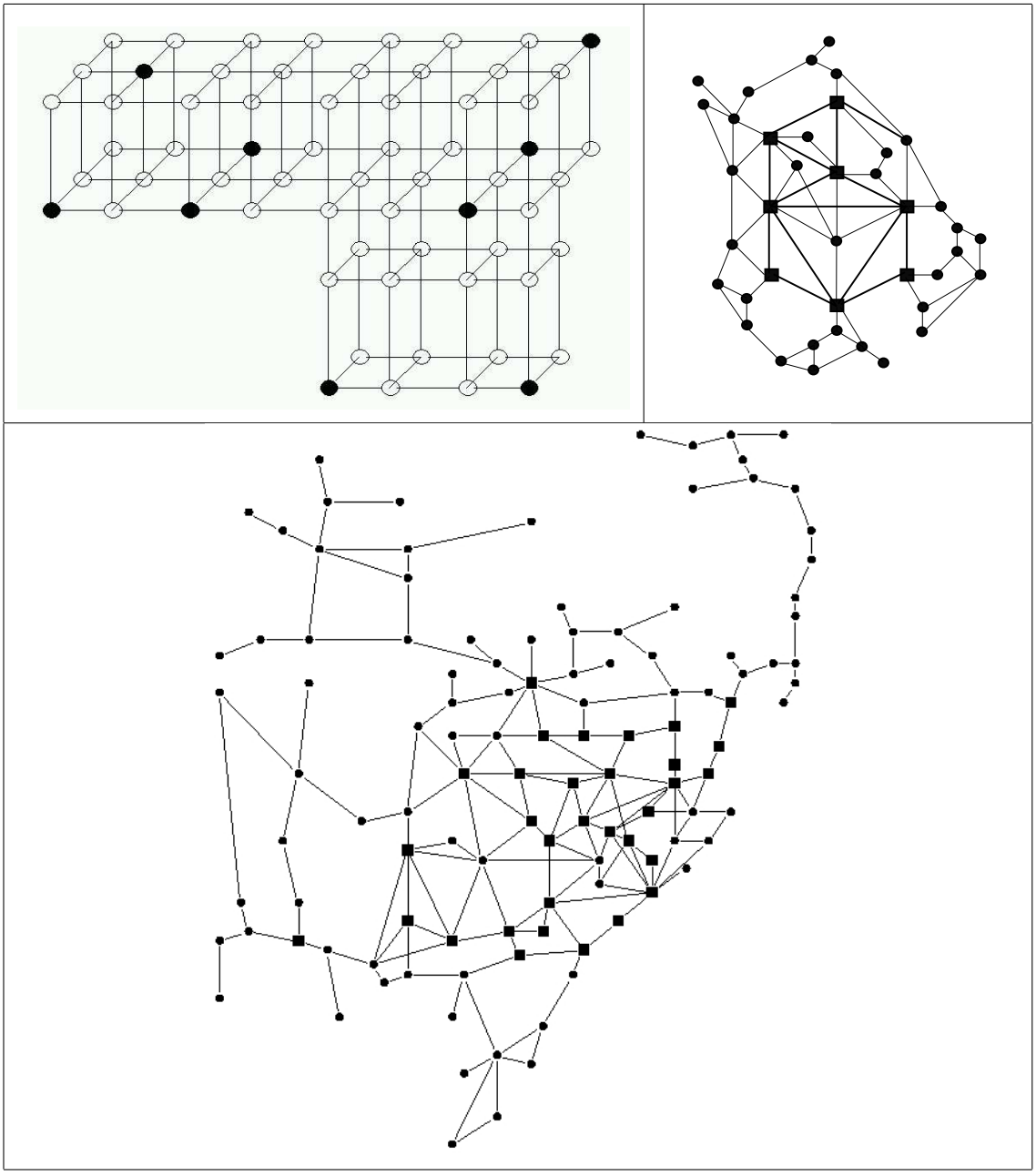


Figure 4.17: Topology of Networks 7, 8, and 9.

| **Topology** | secs./itr | IT | COPT | BCF | GAP | LSI | SN | KN | Edges |
|---|---|---|---|---|---|---|---|---|---|
| Network 1 | 0.78 | 2 | 145 | 145 | 0.0% | 2.21% | 34 | 1 | 68 |
| Network 2 | 2.48 | 5 | 680 | 680 | 0.0% | 5.32% | 54 | 40 | 139 |
| Network 3 | 2.57 | 7 | 1848 | 1848 | 0.0% | 4.85% | 13 | 10 | 62 |
| Network 4 | 3.23 | 9 | 3980 | 3980 | 0.0% | 5.43% | 22 | 11 | 89 |
| Network 5 | 2.45 | 5 | 2393 | 2393 | 0.0% | 5.34% | 18 | 8 | 52 |
| Network 6 | 2.91 | 6 | 3031 | 3031 | 0.0% | 5.43% | 9 | 7 | 58 |
| Network 7 | 0.98 | 2 | 74 | 74 | 0.0% | 6.16% | 31 | 8 | 49 |
| Network 8 | 1.75 | 4 | 4739 | 4739 | 0.0% | 3.12% | - | - | 41 |
| Network 9 | 3.01 | 14 | 7400 | 7445 | 0.6% | 4.47% | - | - | 118 |
| | Average | | | | 0.06% | 4.70% | | | |

Table 4.3: Results associated with the best solutions found by $\mathcal{H}_2$ for the instances 1 to 9.

Moreover, in Tables 4.4 and 4.5, we summarize the computational results obtained by applying the heuristics $\mathcal{H}_1$ and $\mathcal{H}_2$ on the Networks 10 to 29. These tables have the same entries that the previous tables, excepting the entries corresponding to $\mathrm{COPT}$ and $\mathrm{GAP}$, since we do not know their optimum costs nor tight lower bounds. However, we introduce another entries, denoted by $\mathrm{GAPTSP}$ and $\mathrm{GAP2NC}$, which are:

- the gap between the best solution found by GRASP with respect to the optimal TSP solution. That is, $\mathrm{GAPTSP} = 100 \times \frac{(\mathrm{BCF}-\mathrm{COPT\_TSP})}{\mathrm{COPT\_TSP}}$, where $\mathrm{COPT\_TSP}$ is the optimum TSP cost.
- the relative distance between the cost of the best solution found by GRASP and the tight lower bound proved in [102] for the optimal 2-node-connected solution spanning the set of fixed nodes that does not contain Steiner nodes. This lower bound is $\mathrm{LB2NC} = \frac{3}{4}\mathrm{COPT\_TSP}$ and therefore $\mathrm{GAP2NC} = 100 \times \frac{(\mathrm{BCF}-\mathrm{LB2NC})}{\mathrm{LB2NC}}$. We will provide more information on this lower bound in Chapter 5.

Next, we will discuss the computational results, focusing us firstly in the comparison of performance of both heuristics.

When analyzing the best costs found by the heuristics $\mathcal{H}_1$ and $\mathcal{H}_2$, we noticed that, in most of the BNDP instances, the heuristic $\mathcal{H}_2$ improved in significant form the quality of the solutions delivered by $\mathcal{H}_1$, excepting for Networks 1, 7, and 8, where both algorithms attained the optimality, and also Networks 26 and 27 where they achieved the same solution costs. In order to compare them, let us introduce the following notation:

- $\mathrm{BCF}_i^j$ is the best cost found by heuristic $\mathcal{H}_i$ ($i \in 1..2$) when solving Network $j$ ($j \in 1..|Set|$).
- $\mathrm{GAPBCF}$ is the relative improvement of the heuristic $\mathcal{H}_2$ with respect to the heuristic $\mathcal{H}_1$. That is,

$$\mathrm{GAPBCF} = \frac{100}{|Set|} \times \sum_{j \in Set} \frac{|\mathrm{BCF}_2^j - \mathrm{BCF}_1^j|}{\mathrm{BCF}_1^j}.$$

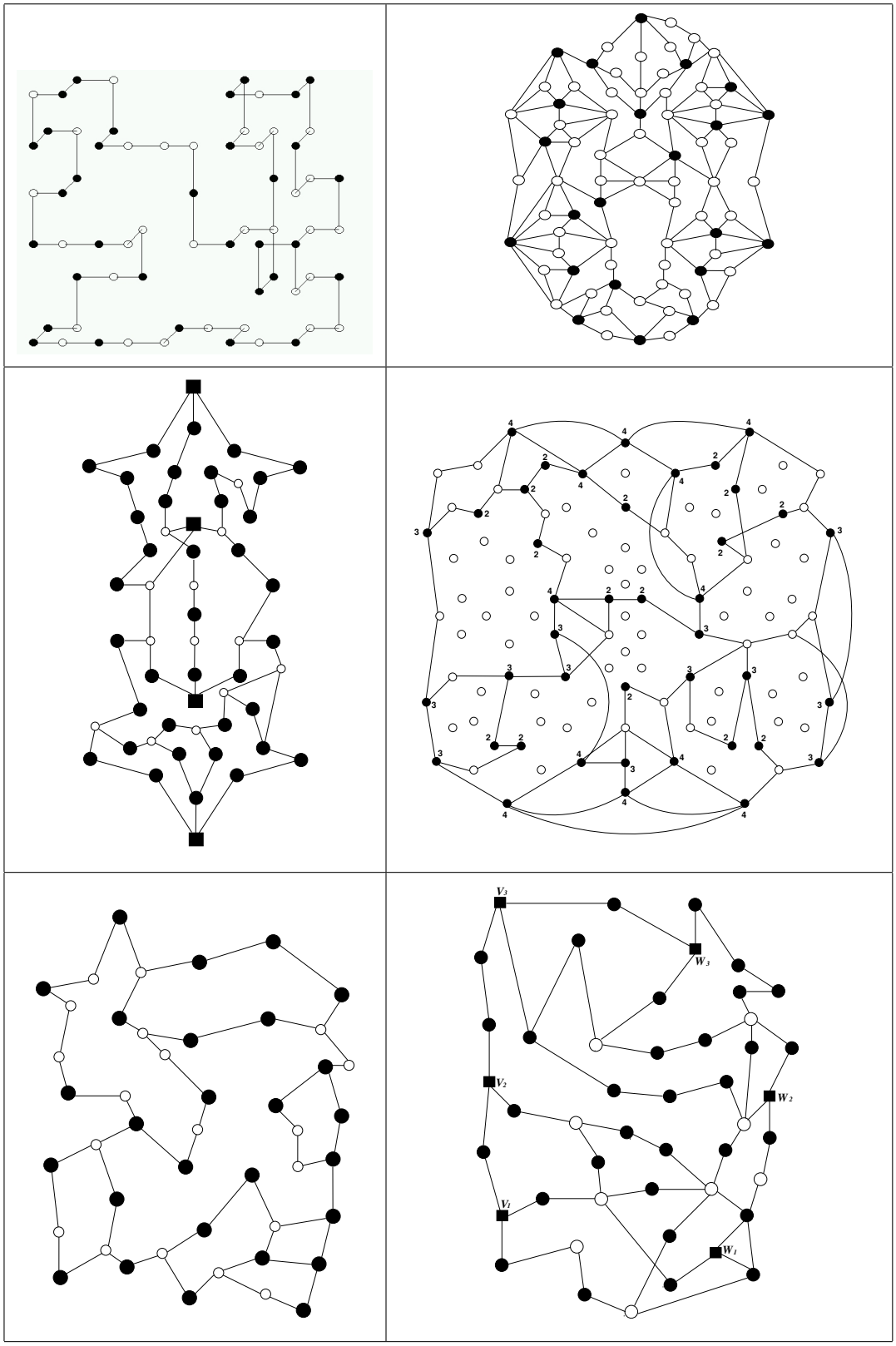



Figure 4.18: Optimal solutions for instances 1, 2, 3, 4, 5 and 6.

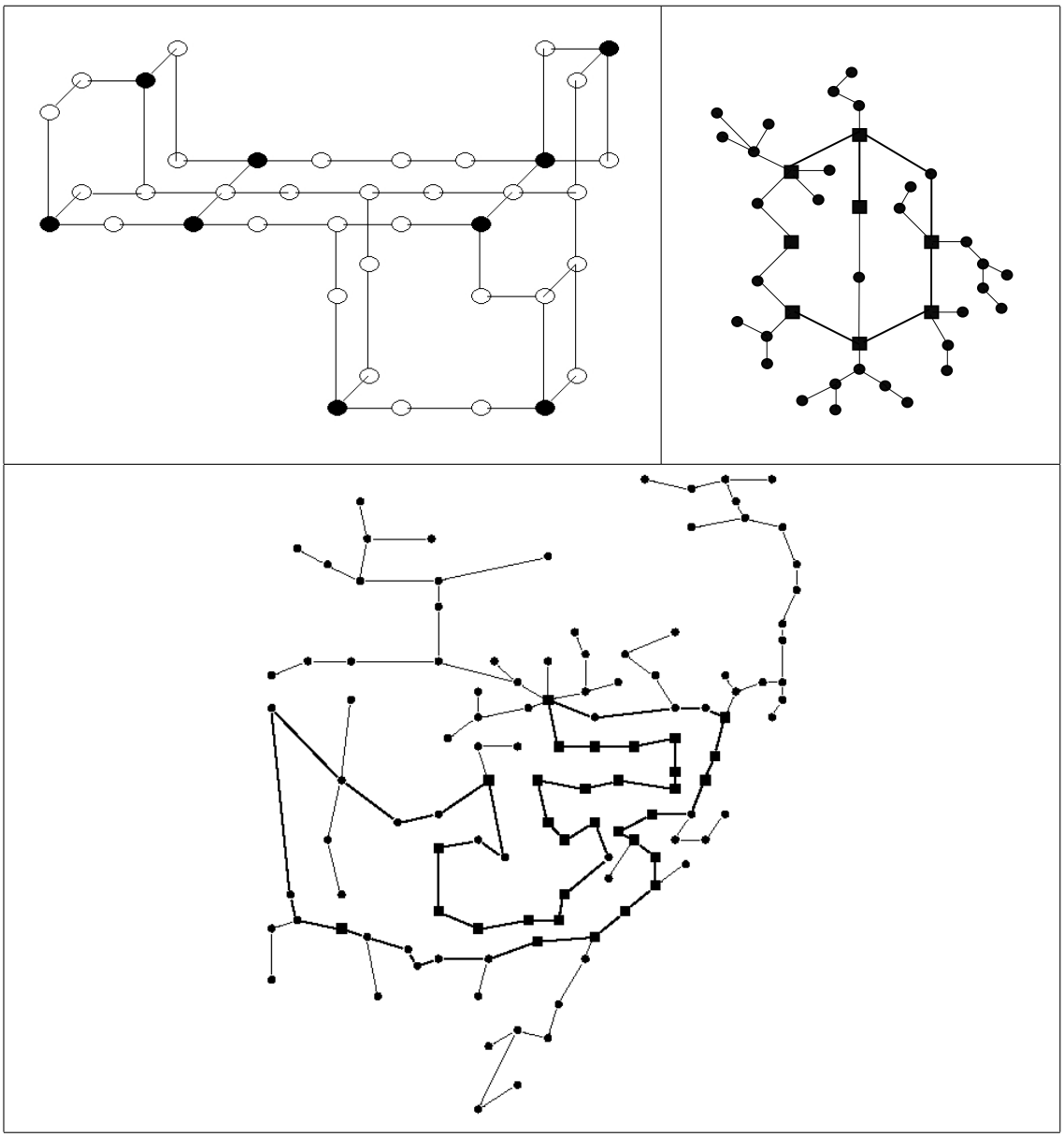


Figure 4.19: Optimal solutions for instances 7, 8, and 9.

| **Topology** | secs./itr | IT | BCF | GAPTSP | GAP2NC | LSI | SN | KN | Edges |
|---|---|---|---|---|---|---|---|---|---|
| BNDP INSTANCES WITH 2 AND 3 NODE-SURVIVABILITY | | | | | | | | | |
| Network 10 | 7.42 | 38 | 16223 | -23.77% | 1.64% | 7.19% | 6 | - | 118 |
| Network 11 | 7.35 | 43 | 16568 | -25.17% | -0.23% | 6.32% | 7 | - | 119 |
| Network 12 | 7.65 | 27 | 16603 | -19.98% | 6.69% | 5.17% | 6 | 1 | 118 |
| Network 13 | 7.63 | 65 | 15840 | -25.61% | -0.82% | 6.12% | 7 | 1 | 120 |
| Network 14 | 7.52 | 71 | 15975 | -27.61% | -3.48% | 6.24% | 7 | - | 119 |
| Network 15 | 4.82 | 24 | 387 | -9.15% | 21.13% | 6.37% | 6 | 2 | 67 |
| Network 16 | 6.02 | 39 | 556 | 3.35% | 37.79% | 5.12% | 8 | 1 | 94 |
| Network 17 | 8.11 | 56 | 448 | -28.78% | -5.03% | 3.98% | 7 | 1 | 118 |
| Network 18 | 6.21 | 61 | 83258 | -23.02% | 2.64% | 5.96% | 8 | - | 94 |
| Network 19 | 8.46 | 52 | 34752 | -21.56% | 4.59% | 6.56% | 4 | 2 | 121 |
| Network 20 | 9.18 | 48 | 43658 | -26.04% | -1.39% | 6.65% | 6 | - | 140 |
| Network 21 | 10.36 | 67 | 71214 | -26.41% | -1.88% | 4.21% | 5 | 1 | 151 |
| Network 22 | 12.12 | 72 | 42325 | -27.70% | -3.59% | 5.60% | 6 | - | 160 |
| | Average | | | -21.65% | 3.46% | 5.80% | | | |
| BNDP INSTANCES WITH 3-NODE-SURVIVABILITY | | | | | | | | | |
| Network 23 | 3.84 | 27 | 2458 | 52.67% | 103.56% | 17.68% | 9 | - | 59 |
| Network 24 | 3.83 | 26 | 3223 | 59.55% | 112.74% | 15.24% | 12 | 2 | 62 |
| Network 25 | 3.07 | 30 | 1634 | 74.39% | 132.52% | 14.33% | 11 | 2 | 57 |
| Network 26 | 2.98 | 19 | 9422 | 37.17% | 82.89% | 19.56% | 9 | - | 35 |
| Network 27 | 3.29 | 18 | 10224 | 45.79% | 94.38% | 18.54% | 13 | 3 | 44 |
| | Average | | | 53.91% | 105.22% | 17.07% | | | |
| BNDP INSTANCES WITH 2-NODE-SURVIVABILITY | | | | | | | | | |
| Network 28 | 8.40 | 53 | 91189 | -22.91% | 2.79% | 5.76% | 2 | 2 | 136 |
| Network 29 | 7.69 | 48 | 11056 | -23.11% | 2.52% | 6.03% | 2 | 2 | 114 |
| | Average | | | -23.01% | 2.66% | 5.90% | | | |

Table 4.4: Results associated with the best solutions found by $\mathcal{H}_1$ for the instances 10 to 29.

- $\text{LSI}_i^j$ is the local search average improvement when executing heuristic $\mathcal{H}_i$ ($i \in 1..2$) to solve the instance $j$ ($j \in 1..|Set|$).

- DIFLSI is the average of the difference between the local search improvement in $\mathcal{H}_2$ and the local search improvement in $\mathcal{H}_1$. It is computed by:

$$\text{DIFLSI} = \frac{1}{|Set|} \times \sum_{j \in Set} (\text{LSI}_2^j - \text{LSI}_1^j).$$

Considering all the test-set, we have $\text{GAPBCF} = 7.22\%$ percent of improvement of $\mathcal{H}_2$ on $\mathcal{H}_1$, which is an important reduction of design costs if we taken into account the real costs that appear when optimizing topologies associated with problems arising in practice. Besides, in 18 instances (out of 29) $\mathcal{H}_2$ improved more than $8\%$ the best solutions built by $\mathcal{H}_1$. In particular, when computing $\text{GAPBCF}$ for Networks 10 to 29, we obtain $\text{GAPBCF} = 9.91\%$, which confirms that $\mathcal{H}_2$ overcomes considerably the heuristic $\mathcal{H}_1$.

On the other hand, when comparing the improvement of the local searches, we have $\text{DIFLSI} = 4.6\%$ (being $\text{LSI}_2^j \geq \text{LSI}_1^j$, $\forall j \in 1 \ldots 29$). Since both heuristics use the same algorithm for the construction

| **Topology** | secs./itr | IT | BCF | GAPTSP | GAP2NC | LSI | SN | KN | Edges |
|---|---|---|---|---|---|---|---|---|---|
| BNDP INSTANCES WITH 2 AND 3 NODE-SURVIVABILITY | | | | | | | | | |
| Network 10 | 9.49 | 17 | 14782 | -30.54% | -7.39% | 14.34% | 5 | 1 | 117 |
| Network 11 | 9.68 | 14 | 15104 | -31.78% | -9.04% | 13.45% | 4 | 1 | 117 |
| Network 12 | 9.64 | 17 | 15132 | -27.07% | -2.76% | 16.12% | 4 | 1 | 117 |
| Network 13 | 9.84 | 14 | 14439 | -32.19% | -9.59% | 15.01% | 5 | - | 118 |
| Network 14 | 9.78 | 13 | 14554 | -34.05% | -12.07% | 12.33% | 5 | - | 118 |
| Network 15 | 6.47 | 19 | 309 | -27.46% | -3.29% | 9.63% | 6 | 2 | 63 |
| Network 16 | 8.06 | 22 | 422 | -21.56% | 4.58% | 9.20% | 6 | - | 88 |
| Network 17 | 10.37 | 27 | 416 | -33.89% | -11.82% | 14.12% | 5 | 2 | 112 |
| Network 18 | 8.14 | 25 | 75845 | -29.88% | -6.50% | 10.25% | 7 | 2 | 89 |
| Network 19 | 10.83 | 27 | 31668 | -28.52% | -4.69% | 8.56% | 6 | - | 118 |
| Network 20 | 12.02 | 26 | 39773 | -32.62% | -10.16% | 9.65% | 6 | 1 | 136 |
| Network 21 | 13.12 | 33 | 64885 | -32.95% | -10.60% | 10.41% | 5 | 2 | 148 |
| Network 22 | 14.97 | 32 | 38551 | -34.14% | -12.19% | 8.55% | 5 | 1 | 155 |
| | Average | | | -30.51% | -7.39% | 11.66% | | | |
| BNDP INSTANCES WITH 3-NODE-SURVIVABILITY | | | | | | | | | |
| Network 23 | 4.48 | 12 | 2244 | 39.36% | 85.81% | 20.56% | 7 | - | 53 |
| Network 24 | 4.76 | 9 | 2935 | 45.28% | 93.71% | 20.77% | 7 | - | 52 |
| Network 25 | 4.52 | 15 | 1487 | 58.74% | 111.66% | 22.53% | 6 | - | 46 |
| Network 26 | 3.92 | 13 | 9422 | 37.17% | 82.89% | 23.11% | 7 | 1 | 34 |
| Network 27 | 4.22 | 12 | 10224 | 45.79% | 94.38% | 22.60% | 8 | 1 | 43 |
| | Average | | | 45.26% | 93.69% | 21.91% | | | |
| BNDP INSTANCES WITH 2-NODE-SURVIVABILITY | | | | | | | | | |
| Network 28 | 10.58 | 38 | 85067 | -28.08% | -4.11% | 9.44% | 2 | 2 | 131 |
| Network 29 | 9.61 | 33 | 10277 | -28.53% | -4.70% | 10.03% | 3 | 3 | 111 |
| | Average | | | -28.30% | -4.41% | 9.73% | | | |

Table 4.5: Results associated with the best solutions found by $\mathcal{H}_2$ for the instances 10 to 29.

phase, this result indicates that, in average, the algorithm $\mathrm{LocalSearch2}$ obtains better neighbor solutions than the obtained ones by $\mathrm{LocalSearch1}$. As discussed in Subsection 4.4.2, the explanation of this fact is that when performing a key-path replacement, the set of Steiner nodes considered (as potential improvers) by $\mathrm{LocalSearch1}$ is totally included in the set of nodes considered by $\mathrm{LocalSearch2}$, and therefore the substitute path computed by $\mathrm{LocalSearch2}$ will have lower cost (or at the most the same) than the key-path computed by $\mathrm{LocalSearch1}$.

Let us note in addition that in all BNDP instances the number of GRASP iterations until reaching the best feasible solution was smaller in $\mathcal{H}_2$ than in $\mathcal{H}_1$. Specifically, on average, the heuristic $\mathcal{H}_1$ needed 35 GRASP iterations to achieve its best solution whereas $\mathcal{H}_2$ needed 16 GRASP iterations to obtain its best solution. This indicates that, in average, $\mathcal{H}_1$ required 19 iterations more than $\mathcal{H}_2$ until accomplishing its best local optimal solution.

Summarizing the comparison between $\mathcal{H}_1$ and $\mathcal{H}_2$, as conclusion of the exposed previously, we can say that in most cases, the topologies obtained by $\mathcal{H}_2$ were of a superior quality than the ones found by $\mathcal{H}_1$, beating them in many cases. Nevertheless, as the running times corroborate, the fact of finding better

feasible solutions is linked to the design of more complex algorithms as well as more complex data structure, which in general means higher execution times. Such is the case of our heuristic $\mathcal{H}_2$ with respect to the heuristic $\mathcal{H}_1$, where on average the execution times per iteration of $\mathcal{H}_2$ were $28.82\%$ percent superior than the ones obtained by $\mathcal{H}_1$. In addition, it can be seen from Tables 4.2, 4.3, 4.4, and 4.5, the running times gaps vary according to the considered topology. In the next section, we will only focus in the analysis of the computational results obtained by $\mathcal{H}_2$.

### 4.6.3 Performance Analysis for the GRASP heuristic $\mathcal{H}_2$

Firstly, for the instances with known optimum value, the heuristic $\mathcal{H}_2$ reached the optimality in Networks 1 to 8, and attaining a near-optimal solution for the Network 9 with a very small gap (below $0.61\%$) with respect to the optimal value. In addition, for these nine cases, we have $4.7\%$ percent of average of the local search improvement (and always over $2\%$ average improvement). Another interesting point was that for Networks 1, 7, and 8, in certain GRASP iterations, we obtained gap zero implying thus that the optimality was achieved in the construction phase. Figures 4.18 and 4.19 show known optimal solutions for BNDP instances 1 to 9. In particular, the shown topologies associated with Networks 1 to 8 were obtained by the GRASP heuristic $\mathcal{H}_2$. We remark that a known optimal topology for Network 9 (shown in Figure 4.19) was obtained by Grötschel et al. [126] by applying a cutting plane algorithm.

Let us observe that for Network 6 the topology of the optimal solution found satisfies that for all pair of terminals belonging to the set $\{v_1, v_2, v_3, w_1, w_2, w_3\}$ there exist three node-disjoint paths communicating them. This implies a better structure of the solution found with respect to the "primary optimal" solution (the one from which the test case was constructed, and which does not fulfill the mentioned property; this is discussed in Appendix C). Similarly, for Network 5, the topological structure of the solution found is not a cycle, contrary to its primary optimal solution. It has several key-nodes and therefore several internal cycles. This could be beneficial since if a fault in a link or node occurs, some nodes will maintain 2-node-connectivity among them.

Now, let us turn to the results obtained for the Networks 10 to 29, analyzing them according to their node-survivability requirements. Before, we remark that since we do not have tight lower bounds for these instances, we will compare them with the optimum TSP values and the lower bounds provided for the optimal 2-node-connected topologies spanning the fixed nodes without using Steiner nodes (their optimum costs will be denoted by $\mathrm{COPT2NC}$). In fact, in relation to this latter, we noticed that in previous works [6, 102, 126] some important particular cases of the BNDP such as the 2NCON and STNSNP also were compared (from a theoretical point-of-view and also numerically) with the optimal TSP values and the lower bound $\mathrm{LB2NC}$.

- **Instances 10 to 22.** We noticed that in all cases the gaps between the best GRASP solution with

respect to the optimal TSP solution were smaller than $-20\%$ having an average gap of $-30.51\%$ percent. Furthermore, in most cases (except for Network 16) the gaps between the best GRASP solution and the $\mathrm{LB2NC}$ value were smaller to $-2\%$ having an average gap of $-7.39\%$ percent. Taken into account that in these instances there exist relatively few fixed nodes with node-survivability requirements greater than two, (eight nodes in Networks 10 to 14 and six nodes in Networks 15 to 22, having all of them 2-3 node-survivability requirements) we think that even though we are comparing costs related to problems with different node-connectivity restrictions, the $\mathrm{COPT\_TSP}$ and $\mathrm{LB2NC}$ values are particularly useful to measure the influence of introducing Steiner nodes as potential enhancers of the quality of 2-node-survivable feasible solutions. In this sense, let us notice that the costs of the best found solutions were significantly smaller than the optimum TSP values and in 12 instances (out of 13) they were inferior than the $\mathrm{COPT2NC}$ values. In all cases the local search phase improved more than $8\%$ the solution delivered by the construction phase; over $10\%$ average improvement for most problem instances (and always over $8.5\%$ average improvement). On the other hand, by inspecting manually the best found GRASP solutions, we observed that each one of them was topologically minimal (i.e. when deleting an edge, we lose the feasibility), satisfying thus a necessary condition to be potentially a global optimal solution. Besides, all of them contain Steiner nodes, in particular, at least four key-nodes in each case.

- **Instances 23 to 27.** Let us note firstly that in these instances the node-survivability requirements are considerably higher with respect to the previous cases (here, we demand at least three node-disjoint paths between every pair of fixed nodes). Hence, it is reasonable to suppose that the $\mathrm{COPT\_TSP}$ and $\mathrm{COPT2NC}$ values will be relatively distant of the optimal BNDP costs, and as a consequence the gaps $\mathrm{GAPTSP}$ and $\mathrm{GAP2NC}$ will not provide us relevant information that allows to analyze the efficiency of our GRASP algorithm when applying it on these instances. As we can see from Tables 4.3 and 4.4, the values of these gaps corroborate numerically this fact. Anyway, we will concentrate in analyzing the qualitative improvements introduced by the local search.

  We noted that, in all cases, the local search phase improved more than $20\%$ the solution built by the construction phase and over $21.5\%$ average improvement. These are interesting results since they show the potentiality of the local search algorithms to enhance in considerable way the quality of the starting feasible solutions. On the other hand, when doing a structural analysis of the best solutions produced by our GRASP algorithm, we noticed that the attained solutions were minimal. In particular, we easily corroborated the minimality of the best found solutions, by considering the number of Steiner nodes present in their topologies conjointly with the aid of a Lemma proved by Harary [80] (which establishes that the minimum number of edges in a $k$-node-connected graph on $n$ nodes without parallel edges is $\lceil \frac{kn}{2} \rceil$).

- **Instances 28 and 29.** These two BNDP instances are particularly interesting because they are more closely linked to TSP problems; more exactly they are STNSNP instances. Let us recall that all feasible solution of a TSP instance also is feasible for a STNSNP instance derived from the TSP instance by adding Steiner nodes; but non conversely.

  Next, let us see some particular properties related to these BNDP instances. On the one hand, for these instances, we have that the optimum TSP values associated with the original problems provide upper bounds for the optimum BNDP values. This property is useful to analyze how well is our GRASP heuristic when building feasible solutions exploiting the existence of Steiner nodes as potential improvers. On the other hand, as we mentioned in Chapter 5, Monma et al. [102] proved that the $\mathrm{LB2NC}$ value is a good lower bound for the $\mathrm{COPT2NC}$ value. In this way, since in these BNDP instances our goal is to find an optimal Steiner 2-node-connected solution spanning the fixed nodes and taken into account that the 2-node-connected topologies without Steiner nodes are also feasible for these BNDP instances, we can deduce that the optimum BNDP values are smaller (or equal in the worst case) than the $\mathrm{COPT2NC}$ values and moreover potentially inferior than the $\mathrm{LB2NC}$ values. Even though this fact is a disadvantage, since we do not know a priori if the optimum BNDP value is greater or smaller than the $\mathrm{LB2NC}$ value, even so to compute the gap between the best GRASP solution with respect to the $\mathrm{LB2NC}$ value will allow us to investigate how efficient is the GRASP algorithm using optional nodes with the objective of improving (if it is possible) the quality of the optimal 2-node-connected topologies non-containing Steiner nodes. In particular, if we reach a negative value for $\mathrm{GAP2NC}$, this implies that we have improved the optimal 2-node-connected solution spanning the fixed nodes that does not contain Steiner nodes.

  Now, let us centre on the numerical results. We noticed that in both instances the gaps between the best GRASP solution with respect to the optimal TSP solution were smaller than $-28\%$ having an average gap of $-28.30\%$ percent. In addition, in both cases the gaps between the best GRASP solution and the $\mathrm{LB2NC}$ value were smaller than $-4.0\%$ having an average gap of $-4.41\%$ percent. These results imply that the feasible solutions attained by GRASP overcame in quality (i.e. with smaller costs) the corresponding optimal TSP solutions as well as the optimal 2-node-connected solutions that does not contain Steiner nodes. Concerning the structure of the best solutions achieved by the GRASP heuristic, both local-optimal topologies were minimal (corroborating manually this property by means of a rigorous inspection on the topology). Moreover, as one can see from Table 4.5, the GRASP solution associated with Network 28 has two Steiner nodes, being both key-nodes of degree three; and the GRASP solution associated with Network 29 has three Steiner nodes, being the three key-nodes of degree three. These topological characteristics are very important since a necessary optimality condition for a problem BNDP2NS satisfying triangular inequality is that any Steiner

node used in an optimal network is of degree three [102]. In this way, the feasible solutions reached for Networks 28 and 29 comply with necessary conditions for global optima.

In all cases the local search phase improved more than $8.5\%$ the solution constructed by the construction phase; over $9.7\%$ average improvement when considering both problem instances and over $9.4\%$ average improvement in each one of them.

To conclude the performance analysis of the GRASP heuristic $H_2$, we can say that the runs performed on the testing set proved to be successful, the optimal solution being obtained for the first eight instances, attaining a very near-optimal solution for Network 9, and reaching local-optimal minimal feasible solutions for Networks 10 to 29 (whose optimum costs are not known). In special, for these latter, the gaps used like comparative reference (namely $\mathrm{GAPTSP}$ and $\mathrm{GAP2NC}$) varied depending on the node-survivability requirements associated with each BNDP instance; occurring the greatest values for Networks 23 to 27, followed in decreasing order by the sub-sets composed of Networks 28 to 29, and Networks 10 to 22, respectively. Even if the connectivity requirements were stronger for Networks 10 to 22 than for Networks 28 to 29, interestingly enough, the gaps values for the former were smaller to those of the latter. This fact possibly happens due to two reasons:

i) Networks 10 to 27 have approximately $50\%$ percent of Steiner nodes whereas Networks 28 and 29 have $33\%$ and $27\%$ percent of Steiner nodes respectively. Potentially the space of feasible solutions associated with Networks 10 to 27 could be increased in a bigger proportion with respect to Networks 28 and 29.

ii) When building Networks 10 to 27, the costs of the new connections were chosen randomly based on the intervals of the original TSP costs, reduced by a factor of $\frac{1}{8}$. Clearly, when reducing the size of these intervals, increases the possibility that the Steiner nodes be components of good quality feasible solutions.

Let us note that, when averaging over all the testing set, we have $14.43\%$ average improvement. These are very good results, showing the potential of the local search phase in improving the starting solution. Besides, notice that most of the best GRASP solutions (except for Network 8 and 9) contain Steiner nodes. In addition in 18 instances (out of 29) the best GRASP topologies are also integrated by key-nodes. Particularly, when $S_D^{(I)} \neq \emptyset$ (nonempty set of Steiner nodes), our algorithm tries to exploit intensely the existence of key-nodes (and therefore of key-paths with at least one key-node as an endpoint) as components of a global optimal solution, and as above, to find minimal local-optimal topologies (as close as possible to the global optimal) with better possible survivability properties. Otherwise, when $S_D^{(I)} = \emptyset$, the GRASP local search pores over each link from the current solution, replacing it by a path that (as far as possible) contain

edges already present in the solution and trying to minimize the cost of introducing (if necessary) new edges to reestablish the connectivity.

## 4.7 Conclusions

By modelling the backbone network design problem (BNDP) with heterogeneous survivability requirements based on the Generalized Steiner Problem with node-connectivity requirements, we were able to develop several Greedy Randomized Search Adaptive Procedures designed to solve the BNDP. As a result of the performance testing phase, we concluded that, in particular, one of the yielded GRASP versions overcame the others with respect to the quality of the built solutions, beating them in many cases, and giving low-cost approximate solutions. This latter GRASP algorithm, denoted $\mathcal{H}_2$, is obtained instancing $\mathrm{Construction\_Phase}$ with $\mathrm{ConstPhase}$ and $\mathrm{Local\_Search}$ with $\mathrm{LocalSearch2}$.

The implementation of our algorithms was tested on a number of different problems with heterogeneous survivability requirements. In all cases, $\mathcal{H}_2$ was shown to find good quality solutions within few iterations; in all cases, except one, with known optimum value an optimal solution was found, and for the other instances minimal feasible solutions were reached improving in many cases the quality of optimal solutions corresponding to other related survivability problems such as TSP and 2NCON (this last one without considering Steiner nodes).

On the other hand, as discussed in Subsection 4.6.3, we can say that the local search phase enhances considerably the quality of the solutions constructed by the construction phase, varying the percentage of improvement according to the topological characteristics of each instance and also depending on the connectivity requirements. In relation to this, we noted that such improvements are linked to the quality of the solution delivered by the construction phase. More precisely, a good initial solution enhances the performance of the GRASP algorithm, because the local search phase will require less computational effort until accomplishing a local optimal feasible topology.

Globally, these are very promising results considering that to compute the best BNDP solution is a NP-Hard problem [126, 136]; in particular, the known exact algorithms have worst case computing time which grows exponentially with the number of terminal nodes and edges. The execution time of the proposed GRASP method is also dependent on the number of nodes, edges, and the connection requirements, but increases much slower.

These are (up to our knowledge) the first results on the use of a GRASP metaheuristic as topology planning method for designing a large-scale backbone network with high node-survivability constrains applicable to general graph classes.

As future work, it is possible to search for new methods which improve either the initial construction or

the local search phases of the GRASP. In addition, we are looking for more BNDP instances with known optimal costs, with the aim of comparing them with the costs associated with the solutions delivered by our GRASP algorithm.

# Chapter 5

# The 2-Node-Survivable Backbone Network Design Problem

## 5.1 Introduction

In this chapter, we will focus on a particular case of the BNDP: the design of 2-node-survivable backbone networks, which we will denote by BNDP2NS. Precisely, this has important applications in the problem of designing High Speed Optical Data Transmission Networks (HSODTN) to be robust to a single link or node failure.

Typically, in the design of large optical fiber networks, where there is high bandwidth and highly reliable links, the need of adding communication redundancy to the network comes up in order to increase its survivability, i.e., its capacity to resist failures. The survivability of a network is closely linked to its degree of connectivity, which is the minimum number of disjoint paths that exist between any pair of the nodes of the network; this connection measure may be given in terms of node-disjoint paths. Even if point to point links are highly reliable, due to the type of service provided by optical fiber, the consequences of failures of just one component may be disastrous. It is then necessary to obtain a higher degree of connectivity in the design of the optical fiber network, in order to increase its survivability. In the design of metropolitan optical fiber networks, a commonly applied requirement is to ensure the existence of at least two node-disjoint-paths between pairs of distinguished nodes of the network. In this way, when a failure occur in some component of the network (link or node), the network will remain in operational state, i.e. the resulting network is connected. The problem of finding a network topology verifying this restriction is known as the *Steiner 2-Node-Survivable Network Problem* (denoted by STNSNP) which is NP-Hard [6, 102]. Some reference in this area are [6, 37, 35, 102, 112, 125].

Our interest in the BNDP2NS is motivated by the need of finding low cost backbone topologies in the

general case within reasonable execution times. That is to say, the goal is to design efficient approximated algorithms for the BNDP2NS applicable to the most general class of graphs. As a result, we introduce several algorithms based on the GRASP methodology for finding a low cost 2-node-survivable Backbone network topology, working upon the STNSNP model.

The remainder of this chapter is organized as follows. Section 5.2 introduces the notation, the auxiliary definitions to be used and the formal definition of the 2-Node-Survivable Backbone Network Design Problem (BNDP2NS). In Section 5.3 we propose three different alternative algorithms for the construction phase. Section 5.4 provide two different algorithms for the local search phase. Like in the general case BNDP, the local search algorithms for the BNDP2NS were designed with the aim of complementing each other, allowing the search for better solutions within different neighborhood structures. Combining these options, yields different versions of GRASP. In Section 5.5 we propose several polynomial time heuristics based on GRASP methodology for approximately solving the BNDP2NS. Section 5.6 introduces a survey of the experimental results obtained by running the GRASP algorithms on a test-set of BNDP2NS instances, containing problem instances with different topological characteristics. Finally, in Section 5.7, we conclude with a discussion, including conclusions and future work.

## 5.2 Notation, Problem Definition and Auxiliary Definitions

We will use the same notation that in the previous chapter, but here we do not have a matrix of connection requirements $R$ since the goal is to design a low cost 2-node-survivable backbone network spanning the fixed switch sites from $S_D^{(I)}$.

**Definition 5.2.1 (Backbone Network Design Problem with 2-Node-Survivable Topology - BNDP2NS)** *We define the 2-Node-Survivable Backbone Network Design Problem $BNDP2NS(S_D, E, C)$ as the problem of finding a subgraph $\mathcal{H}_B$ of $G_B$ of minimum cost such that $\mathcal{H}_B$ is 2-node-survivable with respect to the set of fixed sites $S_D^{(I)}$. We will denote by $\Gamma_{BNDP2NS}$ the space of feasible solutions associated with the problem.*

We present a small BNDP2NS instance example, based on the network shown in Figure 5.1. In this network there are five fixed switch sites, colored black and labeled $s_1$, $s_2$, $s_3$, $s_4$, and $s_5$ and eight non-fixed switch sites, colored white. The connections that can be used to build a solution are shown, annotated with their costs.

Figure 5.2 presents a minimal 2-node-survivable feasible solution (of cost 22) to this problem instance. There are five non-fixed sites integrating the solution. Let us note that the solution is not a cycle, having a key-node and therefore a key-tree.

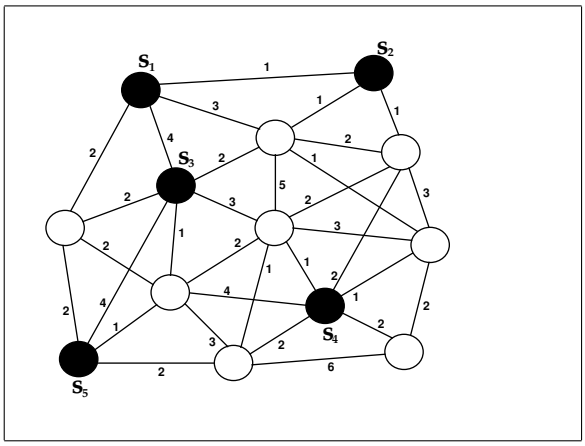



Figure 5.1: Example of a BNDP2NS instance.

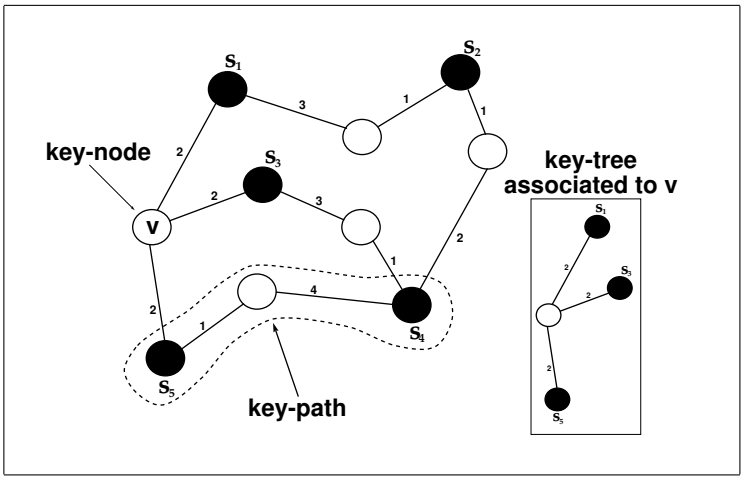



Figure 5.2: A solution to the graph example given in Figure 5.1.

We introduce some auxiliary definitions which will be used in the descriptions of the proposed algorithms.

**Definition 5.2.2 (H-path)** *Given a graph $H$, we call a path $p$ an $H-path$ if $p$ is non-trivial and meets $H$ exactly in its ends. In particular, the edge of any $H-path$ of length 1 is never an edge of $H$ .*

The following proposition is a characterization of the 2-connected graphs, a proof can be found in [43].

**Proposition 5.2.3** *A graph is 2-connected if and only if it can be constructed from a cycle by successively adding $H-paths$ to graphs $H$ already constructed.*

Notice that any minimal 2-node-survivable feasible solution belonging to $\Gamma_{BNDP2NS}$ is also 2-node-connected and 2-connected and therefore it has a decomposition in $H-paths$. In this way, we give the following definition.

**Definition 5.2.4 (H-paths decomposition)** *Given a 2-node-connected network $\mathcal{G}$. We define an $H-paths$ decomposition for $\mathcal{G}$ as a pair $(P,\mathcal{H})$, where $P=(p_1,\ldots,p_k)$ is a sequence of $H-paths$ and $\mathcal{H}=(H_1,\ldots,H_k)$ is a sequence of networks such that $H_0=\mathcal{C}$, with $\mathcal{C}$ a cycle, $H_j=H_{j-1}\cup p_j$, $\forall j\in 1..K$, and $H_k=\mathcal{G}$.*

Figure 5.3 illustrates an $H-paths$ decomposition associated with a 2-node-connected graph $\mathcal{G}$. In each subgraph $H_i$, $i\in 0..4$, the path with broken lines models the added $H-path$.

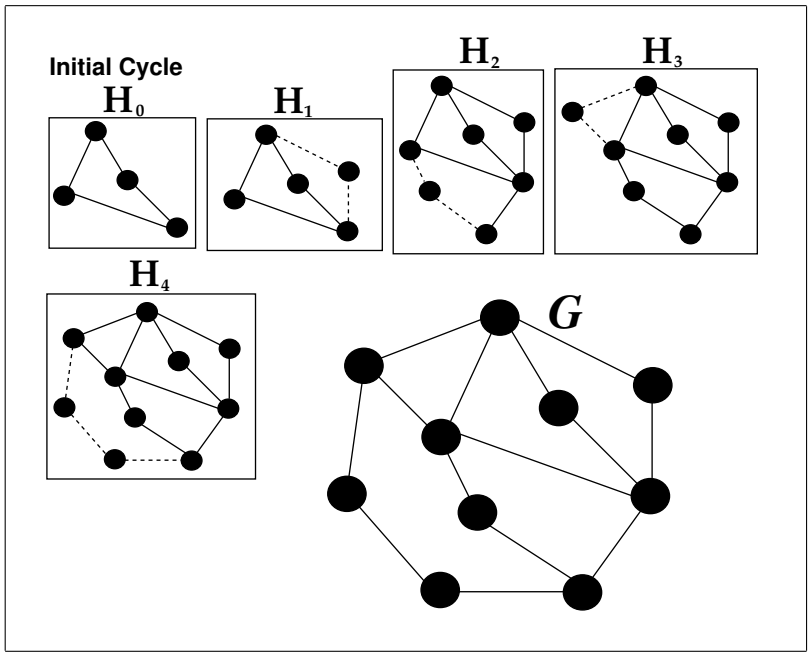



Figure 5.3: An $H-paths$ decomposition for the graph $\mathcal{G}$.

In the previous chapter, we introduced a GRASP algorithm for designing a backbone network with different node-survivability requirements between fixed sites from $S_D^{(I)}$. As mentioned above, an important particular case is when the aim is to design a 2-node-survivable backbone. Now, we introduce GRASP algorithms for approximately solving the BNDP2NS.

## 5.3 BNPD2NS Construction Phase Algorithms

In this point we will introduce the different construction phase algorithms designed as building blocks of a general GRASP algorithm for the BNDP (which will be presented in Section 5.5).

Firstly, we propose a construction phase algorithm, which based on Proposition 5.2.3, builds 2-node-survivable topologies spanning $S_D^{(I)}$. A second option for the construction phase is also proposed, which is a variant of the algorithm $\mathrm{ConstPhase}$ presented in 4.3. Furthermore, we introduce a third construction

phase algorithm, which is a minor variant for this latter, in particular the node-requirements between fixed nodes are computed in different order.

### 5.3.1 Algorithm ConstPhase1_2NS

Based on Proposition 5.2.3, the algorithm builds iteratively a 2-node-survivable network spanning the fixed switch sites of $S_D^{(I)}$. The feasible solution is computed building initially a cycle containing two sites of $S_D^{(I)}$ and by successively adding $\mathcal{G}_{sol} - paths$ (containing a site of $S_D^{(I)}$ not belonging to $\mathcal{G}_{sol}$) to networks $\mathcal{G}_{sol}$ already built until reaching feasibility.

**Procedure ConstPhase1_2NS(**$G_B$,$C$,$k$**);**

1 $\forall s_w^i \in S_D^{(I)}$ **a unique identifier** $n_i$ **is assigned;**
2 $v \leftarrow \text{Select\_Random}(S_D^{(I)})$**;**
3 $Y \leftarrow \{v\}$; $\mathcal{G}_{sol} \leftarrow \{v\}$**;**
4 **while** $Y \neq S_D^{(I)}$ **do**
5 $s_w \leftarrow \text{ArgMax}\{n_i | s_w^i \notin \mathcal{G}_{sol}\}$**;**
6 $\mathcal{L}_1 \leftarrow$ **the** $k$ **shortest paths from** $s_w$ **to** $\mathcal{G}_{sol}$**;**
7 $p_1 \leftarrow \text{Select\_Random}(\mathcal{L}_1)$**;**
8 $K_1 \leftarrow$ **the fixed switch sites of** $p_1$**;**
9 $N \leftarrow \mathcal{G}_{sol} \cup (p_1 \setminus s_w)$**;**
10 $\mathcal{H} \leftarrow G_B \setminus N$**;**
11 $\mathcal{L}_2 \leftarrow$ **the** $k$ **shortest paths from** $s_w$ **to** $\mathcal{G}_{sol}$ **on** $\mathcal{H}$**;**
12 $p_2 \leftarrow \text{Select\_Random}(\mathcal{L}_2)$**;**
13 $K_2 \leftarrow$ **the fixed switch sites of** $p_2$**;**
14 $\mathcal{G}_{sol} \leftarrow \mathcal{G}_{sol} \cup \{p_1, p_2\}$**;**
15 **Let** $u, v$ **be the endpoints of** $p_1$ **and** $p_2$ **in** $\mathcal{G}_{sol}$**;**
16 **if** $(|Y| > 1)$ **and** $(\exists$ **a key-path** $\bar{p}$ **from** $u$ **to** $v$ **on** $\mathcal{G}_{sol})$ **then**
$\mathcal{G}_{sol} \leftarrow \mathcal{G}_{sol} \setminus (\bar{p} \setminus \{u, v\})$**;**
17 $Y \leftarrow Y \cup K_1 \cup K_2 \cup \{s_w\}$**;**
18 **end_while;**
19 **return** $\mathcal{G}_{sol}$**;**
**end ConstPhase1_2NS;**

Figure 5.4: ConstPhase1_2NS pseudo-code.

The algorithm (shown in Figure 5.4) takes as inputs the graph $G_B$ of feasible connections on the backbone network, the matrix of connections cost $C$, and the GRASP parameter $k$. In order to introduce randomness in the selection process of the sites to be added to the current solution, we assign (in line 1) a unique identifier $n_i$ to each fixed switch site $s_w^i \in S_D^{(I)}$. Line 2 selects randomly (and uniformly) one of them. Line 3 initializes the current solution $\mathcal{G}_{sol}$ and the set $Y$ of fixed sites already added to $\mathcal{G}_{sol}$, with the selected fixed site. The loop from line 4 to 18 adds iteratively $\mathcal{G}_{sol} - paths$ to $\mathcal{G}_{sol}$ until $Y = S_D^{(I)}$, i.e. to reach the feasibility.

Each iteration works of the following way. Line 5 selects the fixed site not belonging to $\mathcal{G}_{sol}$ with greatest value of identifier. Let $s_w$ be this site. The $k$-shortest paths from $s_w$ to $\mathcal{G}_{sol}$ are computed in line 6 using the Yen algorithm [137]. They are stored in the restricted candidate list $\mathcal{L}_1$. In the same line, we check if $\mathcal{L}_1$ is empty, in which case the algorithm finalizes since we will not be able to construct a feasible solution. Line 7 selects a path $p_1 \in \mathcal{L}_1$ randomly (and uniformly). In line 8 the set $K_1$ of fixed sites belonging to $p_1$ is computed. Lines 9-10 compute an auxiliary network: $\mathcal{H} = G_B \setminus (\mathcal{G}_{sol} \cup (p_1 \setminus s_w))$. Notice that this network does not have nodes and edges of $\mathcal{G}_{sol}$ nor of $(p_1 \setminus s_w)$, which allows to compute a new node-disjoint path (with respect to $p_1$) from $s_w$ to $\mathcal{G}_{sol}$ on $\mathcal{H}$. Line 11 computes the $k$-shortest paths from $s_w$ to $\mathcal{G}_{sol}$ on $\mathcal{H}$ (which are stored in the restricted candidate list $\mathcal{L}_2$). Line 12 selects a path $p_2 \in \mathcal{L}_2$ randomly (and uniformly) and its set of fixed sites $K_2$ is computed in line 13. Let us observe that the path $p_1 \cup p_2$ is a $\mathcal{G}_{sol} - path$. Line 14 updates $\mathcal{G}_{sol}$ by adding the paths $p_1$ and $p_2$. Considering $\mathcal{G}_{sol}$, let $u$ and $v$ be the endpoints of $p_1$ and $p_2$ respectively. As we will see in Proposition 5.3.1, if in $\mathcal{G}_{sol}$ a key-path from $u$ to $v$ has been induced, it can be removed preserving the 2-node-survivability. Hence, if this is the case, this key-path will be removed to improve the cost of the current solution. Line 16 is incorporated with that purpose. Finally, the set $Y$ of fixed sites already added to $\mathcal{G}_{sol}$ is updated in line 17.

Once all fixed sites have been added, the built solution $\mathcal{G}_{sol}$ is returned in line 19. Figure 5.5 shows the adding of a new fixed switch site to the current solution $\mathcal{G}_{sol}$.

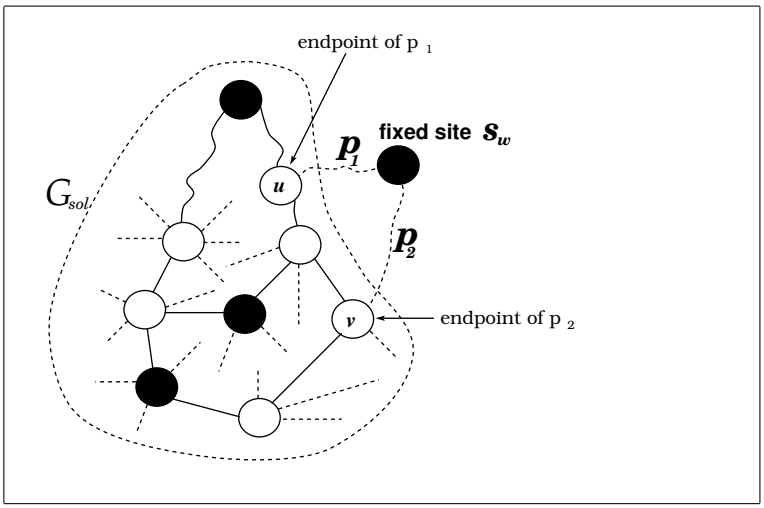


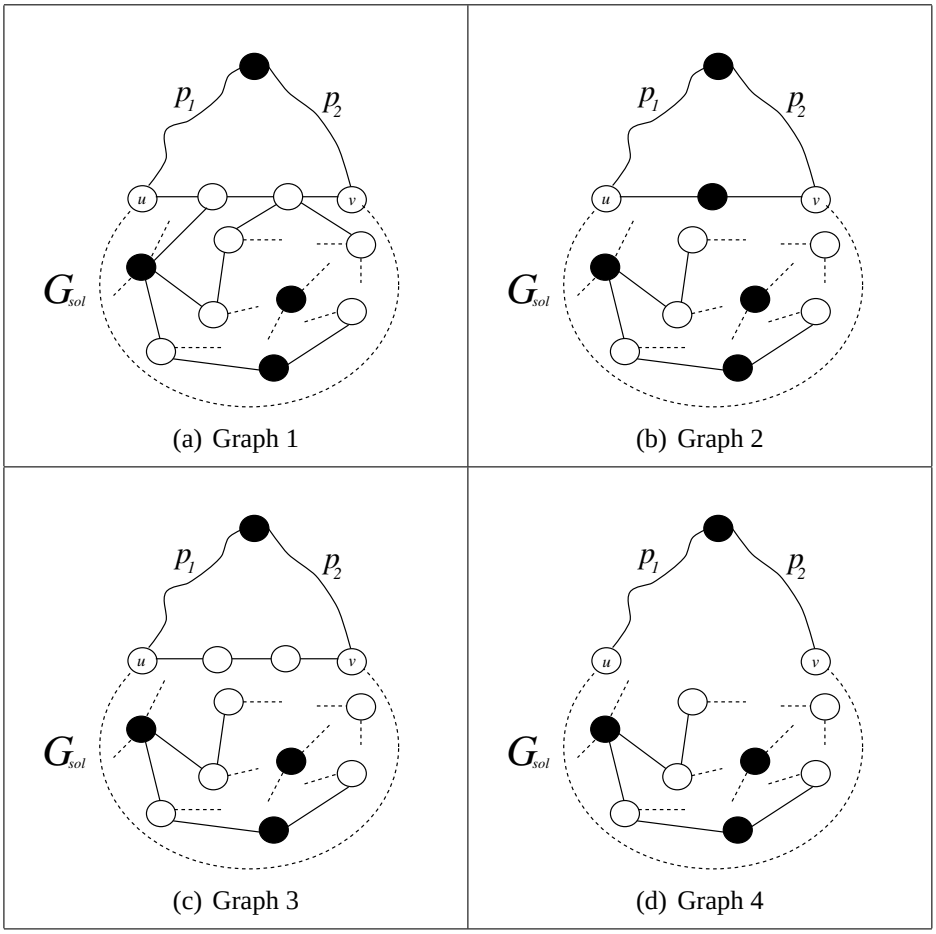

Figure 5.5: Example of a ConstPhase1_2NS iteration.

We present in Figure 5.6 the possible situations when a new fixed switch site is added to the current solution by the algorithm $\mathrm{ConstPhase1\_2NS}$. The black nodes represent the fixed sites whereas the white nodes represent the non-fixed switch sites.

- In the first two graphs there does not exist any key-path between $u$ and $v$ in $\mathcal{G}_{sol}$ since in any path

connecting them there are other key-nodes or fixed sites.

- In the third graph there exists a key-path connecting $u$ with $v$ in $\mathcal{G}_{sol}$, and the fourth graph is the result of deleting it. The obtained solution will be 2-node-survivable with respect to the fixed sites already added.




(a) Graph 1 (b) Graph 2

(c) Graph 3 (d) Graph 4

Figure 5.6: The insertion of a fixed switch site.

The following proposition demonstrates the constructive correctness of the algorithm $\mathrm{ConstPhase1\_2NS}$.

**Proposition 5.3.1** *If the algorithm* $\mathrm{ConstPhase1\_2NS}$ *returns a graph this will be a 2-node-survivable solution for the BNDP2NS.*

**Proof.** By induction in $|Y|$, we will demonstrate that at any time we have a 2-node-survivable network with respect to the subset of fixed switch sites $Y$.
*Basic Step:* $|Y| = 1$. In lines $2-3$, the current solution $\mathcal{G}_{sol}$ and the set $Y$ of nodes already added to $\mathcal{G}_{sol}$ are initialized with a fixed switch site $s \in S_D^{(I)}$. By convention, a simple node is a 2-node-survivable network.

*Induction Step:* $1 < |Y|$. The inductive step is presented of the following way.
As inductive hypothesis we have that if $|Y| < k \leq |S_D^{(I)}|$ for certain iteration, then executing lines $5-17$ the resulting network $\mathcal{G}_{sol}$ will be 2-node-survivable. As inductive thesis the property is fulfilled when $|Y| = k$ for certain iteration.
Let us suppose that in certain iteration we have $|Y| = k$. Let us analyze the following cases.
*Case 1.* $|Y| = k < |S_D^{(I)}|$. The condition in line 4 is TRUE and therefore the algorithm will execute the lines $5-17$. In the previous iteration we had $|Y| < k$ and by I.H. we know that after executing lines $5-17$ the network $\mathcal{G}_{sol}$ is 2-node-survivable. In the current iteration, in line 5 we select a fixed switch site $s_w$ not yet added to $\mathcal{G}_{sol}$ (that one with greater value of identifier). Lines $6-13$ computes two node-disjoint paths from $s_w$ to $\mathcal{G}_{sol}$. Let $p_1, p_2$ be these paths. Clearly, the network $\mathcal{H} = \mathcal{G}_{sol} \cup \{p_1, p_2\}$ is 2-node-survivable. Let $u, v$ be the endpoints of $p_1$ and $p_2$ in $\mathcal{H}$. If $|Y| = 1$ the network $\mathcal{H}$ is a cycle and therefore minimal. If $|Y| > 1$ and there exists a key-path $\bar{p}$ from $u$ to $v$ in $\mathcal{H}$, we can delete this path from $\mathcal{H}$ (except $u$ and $v$) preserving the 2-node-survivability and obtaining a better solution (lines 14-16). Notice that once added the paths $p_1$ and $p_2$ at most one redundant key-path is introduced. Hence, the updated solution $\mathcal{G}_{sol}$ will be 2-node-survivable when finalizing the current iteration. The set $Y$ is updated by adding to it $s_w$ and the fixed switch sites present in $p_1$ and $p_2$ (line 17).
*Case 2.* $|Y| = |S_D^{(I)}|$. In the previous iteration we had $|Y| < |S_D^{(I)}|$ and by I.H. we know that after executing lines $5-17$ the network $\mathcal{G}_{sol}$ is 2-node-survivable. In the current iteration, the condition in line 4 is FALSE and therefore $\mathcal{G}_{sol}$ is 2-node-survivable and spanning the set $S_D^{(I)}$.

**QED**

The following result will be useful to demonstrate certain topological properties related to the constructed solutions. A proof can be found in [102].

**Lemma 5.3.2 (Monma, Munson and Pulleyblank)** *Let $G = (V, E)$ be a two-connected graph with $G' = (V', E')$ a subgraph of $G$ induced by $V'$. Then replacing $E'$ by any collection of edges $E''$ defined on $V'$, where $G'' = (V', E'')$ is two-node-connected, results in a $G^* = (V, (E \setminus E') \cup E'')$ which is two-connected.*

Let us observe that, by construction, the solution $\mathcal{G}_{sol}$ delivered by $\mathrm{ConstPhase1\_2NS}$ has at least one fixed switch site with degree two. We introduce the following structural Lemma to show that a minimal 2-node-survivable solution must have at least one fixed switch site of degree two, or equivalently, that there do not exist minimal feasible solutions such that all the fixed sites have degree greater to two.

**Lemma 5.3.3** *If $\mathcal{G}_{sol}$ is a minimal 2-node-survivable network then there exists in $\mathcal{G}_{sol}$ at least one fixed switch site of degree two.*

**Proof.** By contradiction, let us suppose that in $\mathcal{G}_{sol}$ all the fixed switch sites have degree greater to two. Let $u, v \in S_D^{(I)}$ be two fixed switch sites. Let $p_1, p_2$ be two node-disjoint paths connecting $u$ and $v$ in $\mathcal{G}_{sol}$. We denote $\mathcal{C}$ the cycle conformed by $\{p_1, p_2\}$. Let us consider $\{x_1, x_2\}$ and $\{y_1, y_2\}$ the adjacent nodes to $u$ and $v$ respectively. In addition, let $x_3, y_3 \notin \mathcal{C}$ be two adjacent nodes to $u$ and $v$ respectively. Considering the path $p = ((x_3, u), p_1, (v, y_3))$, as $\mathcal{G}_{sol}$ is 2-node-connected (matrix $C$ is positive), there exists a path $p_3$ connecting $x_3$ and $y_3$ in $\mathcal{G}_{sol}$ so that $p_3 \cap p = \{x_3, y_3\}$. We will denote by $\bar{p}$ to the path $\bar{p} = ((u, x_3), p_3, (y_3, v))$. Let us analyze the following cases.

*Case 1.* $p_2 \cap p_3 = \emptyset$.

A) If $\exists k \in S_D^{(I)}$ such that $k \in p_2$, $k \neq u, v$, let $\hat{k}$ be the closest to $u$ on $p_2$. Since by hypothesis $\hat{k}$ has degree greater to 2 in $\mathcal{G}_{sol}$, there exists a node $\hat{s}$ adjacent to $\hat{k}$ in $\mathcal{G}_{sol}$ such that $\hat{s} \notin p_2$ (it is easy to see that if $\hat{s} \in \mathcal{G}_{sol}$ then it is not optimal). Let $\hat{k}_2 \in S_D^{(I)}$ be the next fixed switch closest to $u$ on $p_2$ (eventually $\hat{k}_2 = v$). Let us denote by $p_4$ the path conformed by: $p_4 = ((\hat{s}, \hat{k}), p_{2(\hat{k},u)})$, where $p_{2(\hat{k},u)} \subset p_2$ is the sub-path connecting $\hat{k}$ with $u$. As $\mathcal{G}_{sol}$ is 2-node-connected, there exists necessarily a path $\hat{p} \subset \mathcal{G}_{sol}$ from $\hat{s}$ to $\hat{k}_2$ such that: $\hat{p} \cap p_4 = \{\hat{s}\}$. Figure 5.7 illustrates this situation. Let us consider the network:

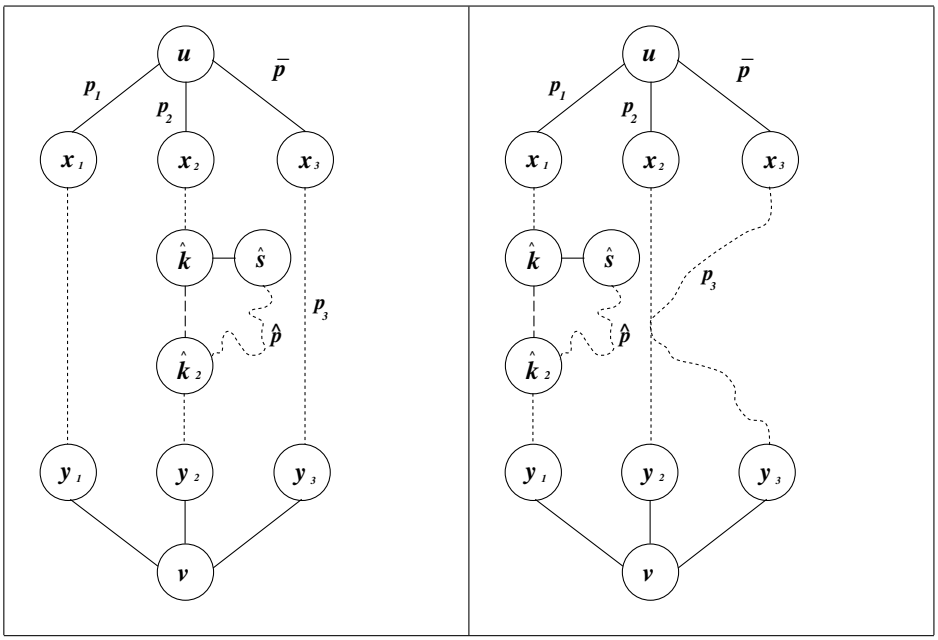



Figure 5.7: Cases 1.A and 2.A respectively.

$$\hat{\mathcal{H}} = \Big(p_1 \cup p_2 \cup \bar{p} \cup \hat{p} \cup \{(\hat{k}, \hat{s})\}\Big) \setminus p_{2(\hat{k},\hat{k}_2)},$$

where $p_{2(\hat{k},\hat{k}_2)} \subset p_2$ is the sub-path connecting $\hat{k}$ with $\hat{k}_2$. Notice that the network $\hat{\mathcal{H}}$ is 2-node-connected. If we denote $\mathcal{H} = p_1 \cup p_2 \cup \bar{p}$, applying Lemma 5.3.2, we have that:

- $\bar{\mathcal{G}}_{sol} = (\mathcal{G}_{sol} \setminus \mathcal{H}) \cup \hat{\mathcal{H}}$ is 2-node-connected (i.e. a 2-node-survivable feasible solution) and in addition,

- $\text{COST}(\bar{\mathcal{G}}_{sol}) < \text{COST}(\mathcal{G}_{sol})$,

contradicting therefore the optimality of $\mathcal{G}_{sol}$.
B) If $\nexists k \in S_D^{(I)}$ such that $k \in p_2$, $k \neq u, v$, clearly the network $\bar{\mathcal{G}}_{sol} = \mathcal{G}_{sol} \setminus \text{EDGES}(p_2)$ would be a better feasible solution than $\mathcal{G}_{sol}$, which contradicts its optimality.
*Case 2.* $p_2 \cap p_3 \neq \emptyset$.
A) It is similar to the previous case $(1.\text{A})$. Again, if $\exists k \in S_D^{(I)}$ such that $k \in p_1$, $k \neq u, v$, let $\hat{k}$ be the closest to $u$ on $p_1$. Since by hypothesis $\hat{k}$ has degree greater to 2 in $\mathcal{G}_{sol}$, there exists a node $\hat{s}$ adjacent to $\hat{k}$ in $\mathcal{G}_{sol}$ such that $\hat{s} \notin p_1$ (as above, if $\hat{s} \in \mathcal{G}_{sol}$ this is not optimal). Let $\hat{k}_2 \in S_D^{(I)}$ be the next fixed switch closest to $u$ on $p_1$ (eventually $\hat{k}_2 = v$). Let us denote by $p_4$ the path conformed by: $p_4 = ((\hat{s}, \hat{k}), p_{1(\hat{k},u)})$, where $p_{1(\hat{k},u)} \subset p_1$ is the sub-path connecting $\hat{k}$ with $u$. As $\mathcal{G}_{sol}$ is 2-node-connected, there exists necessarily a path $\hat{p} \subset \mathcal{G}_{sol}$ from $\hat{s}$ to $\hat{k}_2$ such that: $\hat{p} \cap p_4 = \{\hat{s}\}$. Figure 5.7 illustrates this situation. Let us consider the network:

$$\hat{\mathcal{G}} = \Big(\{(\hat{k}, \hat{s})\} \cup \hat{p} \cup p_1 \cup p_2 \cup \bar{p}\Big) \setminus p_{1(\hat{k},\hat{k}_2)},$$

where $p_{1(\hat{k},\hat{k}_2)} \subset p_1$ is the sub-path connecting $\hat{k}$ with $\hat{k}_2$. Let us observe that the network $\hat{\mathcal{G}}$ is 2-node-connected. Denoting $\mathcal{G} = p_1 \cup p_2 \cup \bar{p}$ and applying Lemma 5.3.2, we have that:

- $\bar{\mathcal{G}}_{sol} = (\mathcal{G}_{sol} \setminus \mathcal{G}) \cup \hat{\mathcal{G}}$ is 2-node-connected (i.e. a 2-node-survivable feasible solution) and in addition,
- $\text{COST}(\bar{\mathcal{G}}_{sol}) < \text{COST}(\mathcal{G}_{sol})$,

contradicting the optimality of $\mathcal{G}_{sol}$.
B) Finally, if $\nexists k \in S_D^{(I)}$ such that $k \in p_3$, $k \neq u, v$, then the network $\bar{\mathcal{G}}_{sol} = \mathcal{G}_{sol} \setminus \text{EDGES}(p_3)$ would be a better feasible solution than $\mathcal{G}_{sol}$, contradicting thus its optimality.

**QED**

We will prove that any minimal feasible solution can be built by the algorithm $\text{ConstPhase1\_2NS}$. Let us notice that, by construction, if there exists a minimal feasible solution with cycle topology, it is possible to find it, for example if $S_D^{(I)} \subset (p_1 \cup p_2)$ in step 14 of the first iteration. Nevertheless, we cannot trivially deduce this property when the minimal feasible topologies are not cycles. In this way, given a BNDP2NS instance, the following theorem and its respective corollary prove that by applying the algorithm $\text{ConstPhase1\_2NS}$ it is possible to reach any minimal feasible solution belonging to the space of feasible solutions. More precisely, if $\mathcal{G}_{sol}$ is a minimal feasible solution, there exists a decomposition in $H - paths$ for $\mathcal{G}_{sol}$ such that this one can iteratively be constructed by $\text{ConstPhase1\_2NS}$.

**Theorem 5.3.4** *Consider a minimal 2-node-survivable solution $\mathcal{G}_{sol}$ which is not a cycle. There exists a decomposition in $H - paths$ for $\mathcal{G}_{sol}$ fulfilling the following points:*

*A) The initial cycle $\mathcal{C}$ has at least two fixed switch sites.*

*B) If $P = (p_1, \ldots, p_k)$ is the sequence of $H - paths$ and $\mathcal{H} = (H_1, \ldots, H_k)$ is the sequence of the resulting networks such that $H_0 = \mathcal{C}$, $H_j = H_{j-1} \cup p_j$, $\forall j \in 1..K$, and $H_k = \mathcal{G}_{sol}$, we have that $\forall p \in P$ there exists a fixed site $s_w \in S_D^{(I)}$ such that $s_w \in \text{INTERNAL\_NODES}(p)$.*

**Proof.** Initially, we choose a cycle $\mathcal{C} \subset \mathcal{G}_{sol}$ containing al least two fixed sites. Iteratively, we build networks $H$ and the $H - paths$ of the following way:

1) We initialize $H = \mathcal{C}$.

2) If $S_D^{(I)} \setminus \text{FIXED\_SITES}(H) \neq \emptyset$, let $s_w$ be one of them. Let us consider two paths $p_1$ and $p_2$ from $s_w$ to $H$ so that:

   - $p_1, p_2 \subset \mathcal{G}_{sol}$, $p_1 \cap H = \{u\}$, $p_2 \cap H = \{v\}$ being $u$ and $v$ the endpoints of $p_1$ and $p_2$ respectively,
   - $p_1 \cap p_2 = \{s_w\}$.

   Necessarily these paths must exist, otherwise $\mathcal{G}_{sol}$ would not be 2-node-survivable.

3) Notice that the path $p = p_1 \cup p_2$ is an $H - path$ on $H$. The current network is updated by $H = H \cup p$ and the construction is resumed from $(2)$.

This process finalized once all the fixed sites of $S_D^{(I)}$ have been added to $H$. Let us suppose that the resulting network $H$ is a subgraph of $\mathcal{G}_{sol}$, i.e. $\mathcal{G}_{sol} \setminus H \neq \emptyset$. By Proposition 5.2.3 we have that $H$ is a 2-node-survivable network spanning the set $S_D^{(I)}$. This contradicts the minimality of $\mathcal{G}_{sol}$. Hence, $H = \mathcal{G}_{sol}$. To complete the proof, let us note that the sequence of paths computed in $(2)$ as well as the sequence of networks computed in $(3)$ satisfy $(\text{A})$ and $(\text{B})$.

**QED**

**Corollary 5.3.5** *Consider a minimal 2-node-survivable solution $\mathcal{G}_{sol}$ which is not a cycle. Let $P = (p_1, \ldots, p_k)$ and $\mathcal{H} = (H_1, \ldots, H_k)$ be a decomposition in $H - paths$ for $\mathcal{G}_{sol}$ satisfying points $(\text{A})$ and $(\text{B})$ of the previous theorem. Then, when adding a path $p_j$ to $H_{j-1}$, for all $j \in 2..K$, in the resulting network $H_j$ there does not exist any key-path connecting the endpoints of $p_j$.*

**Proof.** By contradiction, let us suppose that $\exists p_j \in P$ such that in $H_j = H_{j-1} \cup p_j$ there exists a key-path between the endpoints of $p_j$. Let us denote by $\hat{p}$ this key-path and $u, v$ its endpoints, i.e. the endpoints of $p_j$ on $H_j$. Firstly, let us notice that:

1) If $\hat{p}$ is also a key-path in $\mathcal{G}_{sol}$ then, by construction, $\mathcal{G}_{sol} \setminus \hat{p}$ is 2-node-survivable, which contradicts the minimality of $\mathcal{G}_{sol}$.

2) By removing a connection from $\hat{p}$ the resulting network preserves the node-survivability levels between fixed sites whose associated $H-paths$ have an endpoint non-belonging to $\hat{p}$ or their endpoints are $u$ and $v$.

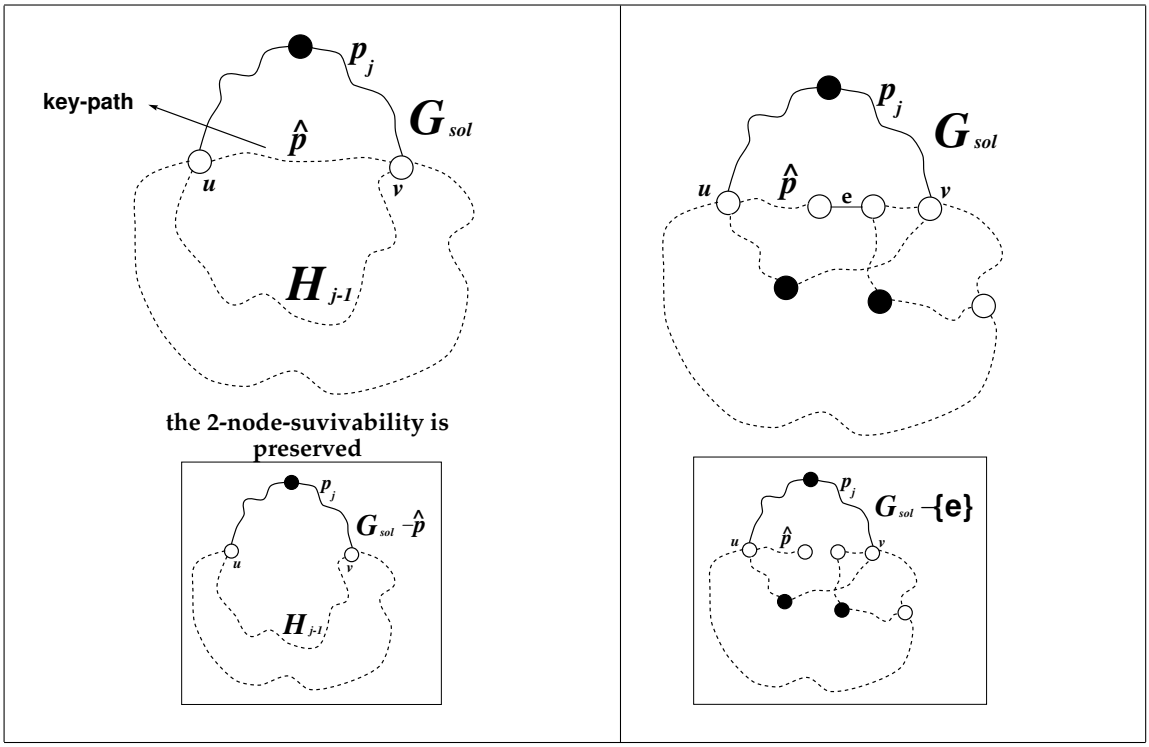



Figure 5.8: Situations $(1)$ and $(2)$.

Figure 5.8 shows these situations. Now, let us analyze the $H-paths$ whose endpoints are in $\hat{p}$ and one of them belongs to INTERNAL_NODES$(\hat{p})$. In this way, we define the set:

$$X_{\hat{p}} = \{p \in P | p \text{ has both endpoints in } \hat{p} \text{ and one of them belongs to } \text{INTERNAL\_NODES}(\hat{p})\}\,.$$

Let us denote by $\hat{\mathcal{G}}$ the subnetwork integrated by the set of paths $X_{\hat{p}} \cup \{\hat{p}\}$. Let us define the set:

$$Y = \left\{\hat{p}_{(s_1,s_2)} \subset \hat{p} | \forall p_i \in X_{\hat{p}} \text{ with endpoints } s_1, s_2\right\}.$$

We chose a path $q \in Y$ such that does not exist $p \in Y$ such that $p \subset q$. Let $\mathcal{C} \subset H_{j-1}$ be a cycle such that $\hat{p} \subset \mathcal{C}$. In Figure 5.9, we illustrate this case. Let $e \in q$ be an edge. It is easy to see that the subnetwork $\mathcal{H} = \mathcal{C} \cup (\hat{\mathcal{G}} \setminus \{e\})$ is 2-node-connected. Therefore, by Lema 5.3.2, by replacing in $\mathcal{G}_{sol}$ the subnetwork $\mathcal{C} \cup \hat{\mathcal{G}}$ by $\mathcal{H}$ the resulting network is 2-node-survivable. This contradicts the minimality of $\mathcal{G}_{sol}$, implying thus the nonexistence of a key-path connecting $u$ and $v$ in $H_j$, as required, and completing the proof.

**QED**

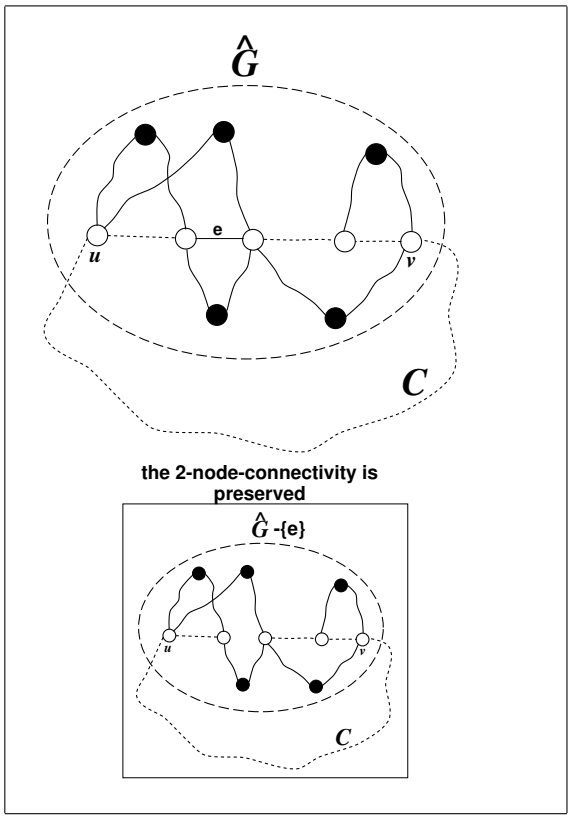



Figure 5.9: Networks: $\mathcal{C} \cup \hat{\mathcal{G}}$ and $\mathcal{C} \cup (\hat{\mathcal{G}} \setminus \{e\})$.

### 5.3.2 Algorithm ConstPhase2_2NS

We also propose another alternative algorithm for the BNDP2NS construction phase which we call $\mathrm{ConstPhase2\_2NS}$. In particular, we can see it as a variant of $\mathrm{ConstPhase}$ (introduced in 4.3) customized for designing 2-node-survivable networks. The proposed algorithm builds iteratively feasible solutions by reusing suitably the connections already present in the current solution, with the aim of minimizing the cost of adding new connections to satisfy (in each iteration) certain connection requirement between two fixed switch sites. It differs from $\mathrm{ConstPhase1\_2NS}$ by the fact that the 2-node-survivability (with respect to the already added fixed sites) is not guaranteed once finalized each iteration (except for the last iteration, where the constructed solution will be necessarily 2-node-survivable). Anyway, as we will show in Section 5.6, when using this algorithm in the GRASP construction phase we obtain good feasible topologies which are improved by the local search algorithms eliminating in certain cases redundant edges. Next, we introduce a detailed description for this algorithm.

The algorithm builds iteratively a 2-node-survivable network spanning the fixed switch sites of $S_D^{(I)}$. In more detail, the algorithm (shown in Figure 5.10) takes as inputs the graph $G_B$ of feasible connections on the backbone network, the matrix of connections cost $C$, and the GRASP parameter $k$. In line 1 we

**Procedure ConstPhase2_2NS($G_B$,$C$,$k$);**

1 $\mathcal{G}_{sol} \leftarrow (S_D^{(I)}, \emptyset)$; $m_{ij} \leftarrow 2\ \forall i,j \in S_D^{(I)}$; $\mathcal{P}_{ij} \leftarrow \emptyset\ \forall i,j \in S_D^{(I)}$; $A_{ij} \leftarrow 0\ \forall i,j \in S_D^{(I)}$;
2 **while** $\exists m_{ij} > 0$ **such that** $A_{ij} < \text{MAX\_ATTEMPT}$ **do**
3 **Let** $i,j \in S_D^{(I)}$ **be a randomly chosen pair of fixed switch sites such that** $m_{ij} > 0$ ;
4 $\bar{\mathcal{G}} \leftarrow (G_B \setminus \mathcal{P}_{ij})$;
5 **Let** $\bar{C}$ **be the matrix given by:** $\bar{c}_{uv} \leftarrow \begin{cases} 0 & \text{if } (u,v) \in \mathcal{G}_{sol}, \\ c_{uv} & \text{if } (u,v) \in (\bar{\mathcal{G}} \setminus \mathcal{G}_{sol}). \end{cases}$ ;
6 $\mathcal{L}_p \leftarrow$ **the** $k$ **shortest paths from** $i$ **to** $j$ **on** $\bar{\mathcal{G}}$**, considering the matrix** $\bar{C}$;
7 **if** $\mathcal{L}_p = \emptyset$ **then** $A_{ij} \leftarrow A_{ij} + 1$; $\mathcal{P}_{ij} \leftarrow \emptyset$; $m_{ij} \leftarrow 2$;
8 **else**
9 **if** $\exists \hat{p} \in \mathcal{L}_p$ **such that** $\text{COST}_{|\bar{C}}(\hat{p}) = 0$ **then** $p \leftarrow \hat{p}$;
10 **else** $p \leftarrow \text{Select\_Random}(\mathcal{L}_p)$; $\mathcal{G}_{sol} \leftarrow \mathcal{G}_{sol} \cup \{p\}$;
11 **if** $m_{ij} = 2$ **then** $\mathcal{P}_{ij} \leftarrow p$;
12 $m_{ij} \leftarrow m_{ij} - 1$;
13 $[\mathcal{P}, M] \leftarrow \text{Update\_Matrix}(\mathcal{G}_{sol}, \mathcal{P}, M, p, i, j)$;
14 **end_if;**
15 **end_while;**
16 **return** $\mathcal{G}_{sol}$;
**end ConstPhase2_2NS;**

Figure 5.10: ConstPhase2_2NS pseudo-code.

initialize:

- the current solution $\mathcal{G}_{sol}$ with the sites of $S_D^{(I)}$ without connections among them,
- the matrix $M = \{m_{ij}\}_{i,j \in S_D^{(I)}}$ (indicating the connection requirements not yet satisfied between fixed sites) with $m_{ij} = 2$, $\forall i,j \in S_D^{(I)}$,
- the auxiliary matrix $\mathcal{P} = \{\mathcal{P}_{ij}\}_{i,j \in S_D^{(I)}}$ (used to store paths between fixed sites) with $\mathcal{P}_{ij} = \emptyset$, $\forall i,j \in S_D^{(I)}$,
- and the auxiliary matrix $A = \{A_{ij}\}_{i,j \in S_D^{(I)}}$ (used to record when has not been found a path between two fixed sites) with $A_{ij} = 0$, $\forall i,j \in S_D^{(I)}$.

Loop 2-15 is repeated until all the fixed sites have satisfied their connection requirements (i.e. the resulting network is 2-node-survivable) or for certain pair of fixed sites have not been found two node-disjoint paths connecting them after $\text{MAX\_ATTEMT}$ attempts.

Each iteration works in the following way. Line 3 selects randomly (and uniformly) a pair $i,j \in S_D^{(I)}$ such that $m_{ij} > 0$ (i.e. there exists at least one requirement not yet fulfilled among them). Line 4 computes the network $\bar{\mathcal{G}} = (G_B \setminus \mathcal{P}_{ij})$. Note that this network does not contain any edge and node of $\mathcal{P}_{ij}$ excepting $i$ and $j$; therefore, all path communicating $i$ with $j$ in $\bar{\mathcal{G}}$ will be node-disjoint with respect to $\mathcal{P}_{ij}$. Line 5 computes an auxiliary matrix $\bar{C}$ of connections cost such that all connections $(u,v) \in \mathcal{G}_{sol}$ have cost

zero. This will allow to reuse already existing edges in $\mathcal{G}_{sol}$ (without considering their costs) when new node-disjoint paths are computed. Line 6 computes the $k$ shortest paths from $i$ to $j$ on $\bar{\mathcal{G}}$ using the matrix $\bar{C}$. These paths are stored in the restricted candidate list $\mathcal{L}_p$. Line 7 checks if $\mathcal{L}_p$ is empty. If this is the case (assuming that $G_B$ is 2-node survivable), we re-initialize $\mathcal{P}_{ij}$ and $m_{ij}$ since $\mathcal{P}_{ij}$ contains a separating set between $i$ and $j$ on $G_B$ and therefore does not exist a path from $i$ to $j$ in $\bar{\mathcal{G}}$, they are in different connected components. Otherwise, if $i$ and $j$ are in the same connected component in $\bar{\mathcal{G}}$, in order to not increase the cost of $\mathcal{G}_{sol}$, in line 9 we search a path $\hat{p} \in \mathcal{L}_p$ such that $\text{COST}_{|\bar{C}}(\hat{p}) = 0$ (its cost with respect to $\bar{C}$). If this is successful, we assign to $p$ the found path. Otherwise, line 10 selects randomly (and uniformly) a path $p$ from $\mathcal{L}_p$ which is added to $\mathcal{G}_{sol}$ in the same line. Line 11 checks if for the pair $i, j$ there is no computed path among them. In such a case, we assign to $\mathcal{P}_{ij}$ the path $p$ computed in lines 9-10. In this way, the next computed path for the pair $i, j$ will be node-disjoint to the path $\mathcal{P}_{ij}$. Line 13 calls the auxiliary procedure $\text{Update\_Matrix}$ to update the matrix $\mathcal{P}$ and $M$. The description of this procedure explains in detail the introduced updates.

Once the loop 2-15 finalized the built feasible solution $\mathcal{G}_{sol}$ is returned in line 16. Figure 5.11 shows an example of a new node-disjoint path being added between two fixed switch sites.

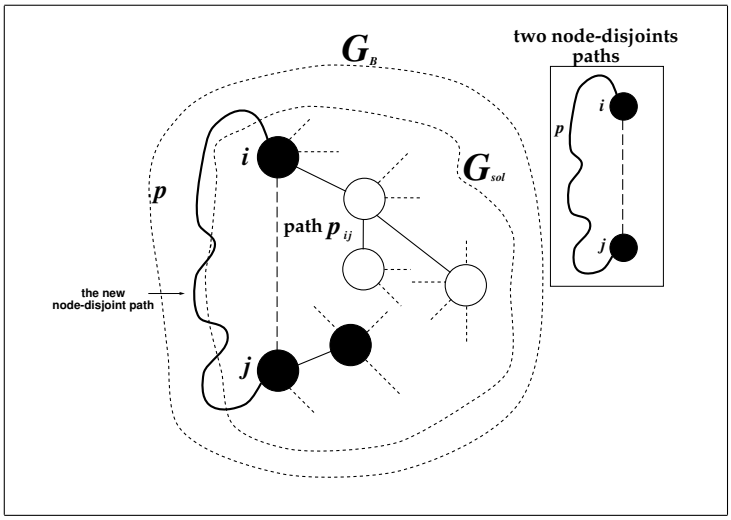



Figure 5.11: Computation of a new node-disjoint path between the fixed sites $i$ and $j$.

**Update_Matrix description**

The algorithm (shown in Figure 5.12) receives as inputs the current solution in construction $\mathcal{G}_{sol}$, the matrix $\mathcal{P}$ of computed paths between fixed sites, the matrix $M$ indicating the requirements not yet satisfied between fixed sites, two fixed sites $i$ and $j$, and the path $p$ computed among them. The loop 1-18 analyzes each fixed

site belonging to the path $p$ in order to update certain connection requirements with other fixed sites.

Each iteration woks of the following way. Let $t \in p, t \neq i, j$ be a fixed site. In line 2, we check if $m_{tj} = 2$. If this is the case, since the algorithm $\mathrm{ConstPhase}$ preserves the condition: $\mathcal{P}_{uv} = \emptyset$ iff $m_{uv} = 2$, $\forall u, v \in S_D^{(I)}$, we have then $\mathcal{P}_{tj} = \emptyset$. Line 3 assigns to $\mathcal{P}_{tj}$ the sub-path $p_{(t,j)}$ connecting $t$ with $j$ in $p$. The update of $m_{tj}$ depends on the number of paths already computed between the sites $i$ and $j$. If $m_{ij} = 1$ then $m_{tj}$ is decremented by one (line 4). Otherwise, if $m_{ij} = 2$ this implies the existence of a cycle included in $\mathcal{G}_{sol}$ containing $t$ and $j$, i.e. there exist two node-disjoint paths from $t$ to $j$ in $\mathcal{G}_{sol}$, and therefore $m_{tj}$ is decremented by two (line 5). Similarly, we update $m_{it}$ in lines 7-11. The internal loop 12-17 analyzes each fixed site $\bar{t} \in S_D^{(I)}, \bar{t} \notin p$ such that $m_{\bar{t}t} = 2$. Line 13 checks if a path from $i$ or $j$ to $\bar{t}$ has been computed in a previous iteration. If this is the case, there exists a path from $\bar{t}$ to $t$ in $\mathcal{G}_{sol}$ and then $m_{\bar{t}t}$ is decremented by one in line 14. In addition to satisfy the condition exposed above, as now $m_{\bar{t}t} = 1$, in line 15 we assign to $\mathcal{P}_{\bar{t}t}$ the shortest path from $\bar{t}$ to $t$ on $\mathcal{G}_{sol}$.

Once finalized the loop 1-18 the matrix $\mathcal{P}$ and $M$ are returned in line 19.

```
Procedure Update_Matrix(G_sol,P,M,p,i,j);

1  for each t ∈ p/t ∈ S_D^(I) do
2    if t ≠ i,j and m_tj = 2 then
3      P_tj ← p_(t,j);
4      if m_ij = 1 then m_tj ← m_tj − 1;
5      else m_tj ← m_tj − 2;
6    end_if;
7    if t ≠ i,j and m_it = 2 then
8      P_it ← p_(i,t);
9      if m_ij = 1 then m_it ← m_it − 1;
10     else m_it ← m_it − 2;
11   end_if;
12   for each t̄ ∈ S_D^(I)/t̄ ∉ p, t̄ ≠ i,j such that m_tt̄ = 2 do
13     if m_it̄ < 2 or m_jt̄ < 2 then
14       m_tt̄ ← m_tt̄ − 1;
15       P_tt̄ ← shortest path from t to t̄ on G_sol;
16     end_if;
17   end_for_each;
18 end_for_each;
19 return P,M;
end Update_Matrix;
```

Figure 5.12: Update_Matrix pseudo-code.

The updates on $M$ and $\mathcal{P}$ must preserve the data coherence with respect to the solution in construction $\mathcal{G}_{sol}$ and furthermore to guarantee that this one will be feasible once finalized $\mathrm{ConstPhase2\_2NS}$. For example, the $m_{ij}$ value must be decremented by one only if we found a new node-disjoint path between

the fixed sites $i$ and $j$, and moreover $\mathcal{P}_{ij} \neq 0$ only if it is a path connecting $i$ and $j$ in the current solution $\mathcal{G}_{sol}$. Thus, we designed $\mathrm{Update\_Matrix}$ so that it satisfies certain properties which guarantee in each $\mathrm{ConstPhase2\_2NS}$ iteration, the data consistency inherent to the current network $\mathcal{G}_{sol}$, the matrix of paths $\mathcal{P}$ and the matrix $M$. We introduce these properties in the following proposition.

**Proposition 5.3.6** *Once the algorithm* $\mathrm{Update\_Matrix}$ *finalizes the following points are satisfied* $\forall i, j \in S_D^{(I)}$*:*

*i)* $\mathcal{P}_{ij} = \emptyset$ *iff* $m_{ij} = 2$.

*ii) If* $m_{ij} = 0$ *then there exist two node-disjoint paths from* $i$ *to* $j$ *in* $\mathcal{G}_{sol}$.

*iii) If* $m_{ij} = 1$ *then there exists a path from* $i$ *to* $j$ *in* $\mathcal{G}_{sol}$.

**Proof.** Firstly, let us assume that when $\mathrm{Update\_Matrix}$ is called in line 11 of $\mathrm{ConstPhase2\_2NS}$, $\mathcal{P}$ and $M$ satisfy these conditions.
Let $i, j \in S_D^{(I)}$ be the input fixed switch sites and $p$ the path connecting $i$ with $j$ computed by $\mathrm{ConstPhase2\_2NS}$. Loop 1-18 analyze $\forall t \in S_D^{(I)}, t \in p, t \neq i, j$ the following cases.
*Case 1: Lines 2-6.* If $m_{tj} = 2$ then we know that $\mathcal{P}_{tj} = \emptyset$, therefore we assign to $\mathcal{P}_{tj}$ the sub-path $p_{(t,j)} \subset p$ connecting $t$ with $j$ in $p$ (line 4). If $m_{ij} = 1$, we decrement by one to $m_{tj}$ (line 4), but if $m_{ij} = 2$ then there exists a cycle in $\mathcal{G}_{sol}$ containing $t$ and therefore we can decrement $m_{tj}$ by two. Clearly, once finalized lines 2-6 $m_{tj}$ satisfies points $\mathrm{i} - \mathrm{iii}$.
*Case 2: Lines 7-11.* It is similar to the previous case but for the site $j$.
*Case 3: Lines 12-17.* For all $\bar{t} \in S_D^{(I)}$, $\bar{t} \notin$, $\bar{t} \neq i, j$, such that $m_{t\bar{t}} = 2$ we check if $m_{i\bar{t}} < 2$ or $m_{j\bar{t}} < 2$ in order to know if there already exists a path communicating $i$ or $j$ with $\bar{t}$ in $\mathcal{G}_{sol}$. If this condition is true, $m_{t\bar{t}}$ is decremented by one since there exists a path connecting $t$ with $\bar{t}$ in $\mathcal{G}_{sol}$. In addition, $\mathcal{P}_{t\bar{t}}$ is setting with the shortest path connecting them in $\mathcal{G}_{sol}$, guaranteeing therefore the fulfillment of conditions $\mathrm{i} - \mathrm{iii}$ for these pairs of fixed switch sites.

**QED**

Based on the previous proposition, the following proposition demonstrates the constructive correctness of the algorithm $\mathrm{ConstPhase2\_2NS}$.

**Proposition 5.3.7** *If* $A_{ij} < \mathrm{MAX\_ATTEMPT}$, $\forall i, j \in S_D^{(I)}$ *then the graph returned by* $\mathrm{ConstPhase2\_2NS}$ *is a 2-node-survivable feasible solution for the BNDP2NS.*

**Proof.** In line 1 the algorithm initializes:

- $\mathcal{G}_{sol}$ with the set of fixed switch sites $S_D^{(I)}$ without edges among them,
- the auxiliary matrix $M$ (indicating connections that we know satisfies network $\mathcal{G}_{sol}$) with $m_{ij} = 2$, $\forall i, j \in S_D^{(I)}$,
- the matrix $\mathcal{P}$ (which will store a path from $i$ to $j$ on $\mathcal{G}_{sol}$, $\forall i, j \in S_D^{(I)}$) with empty sets,
- the auxiliary matrix $A$ (indicating the unsuccessful attempts to find a new path between two nodes from $S_D^{(I)}$) with $A_{ij} = 0$, $\forall i, j \in S_D^{(I)}$.

Suppose that for a certain iteration the condition in line 2 is TRUE. In line 3 we choose randomly a pair $i, j \in S_D^{(I)}$ of fixed switch sites such that $m_{ij} > 0$. Line 4 computes the network $\bar{\mathcal{G}} = (G_B \setminus \mathcal{P}_{ij})$. Depending on $\mathrm{Update\_Matrix}$, assuming that it preserves the condition: $\mathcal{P}_{ij} = \emptyset$ iff $m_{ij} = 2$, we have the following possible cases:

A) There does not exist a path from $i$ to $j$ in $\bar{\mathcal{G}}$ since $\mathcal{P}_{ij}$ contains a separating set between $i$ and $j$ on $G_B$.

B) There exists a path from $i$ to $j$ in $\bar{\mathcal{G}}$. In this case, we have the following possibilities:

   i) $m_{ij} = 2$ and there exist already two node-disjoint paths from $i$ to $j$ in $\mathcal{G}_{sol}$,

   ii) $m_{ij} = 2$, there exists a path from $i$ to $j$ in $\mathcal{G}_{sol}$ but there do not exist two node-disjoint paths from $i$ to $j$ in $\mathcal{G}_{sol}$,

   iii) $m_{ij} = 2$ and there does not exist a path from $i$ to $j$ in $\mathcal{G}_{sol}$,

   iv) $m_{ij} = 1$ and there exists a path from $i$ to $j$ in $(\mathcal{G}_{sol} \setminus \mathcal{P}_{ij})$,

   v) $m_{ij} = 1$ and there does not exist a path from $i$ to $j$ in $(\mathcal{G}_{sol} \setminus \mathcal{P}_{ij})$.

Let us analyze each case.

*Case A.* In this case, line 7 re-initializes $\mathcal{P}_{ij}$ and $m_{ij}$, and the construction is resumed from line 2. This line is executed at the most $\mathrm{MAX\_ATTEMPT}$ times for any pair of fixed switch site, after which the search is finalized.

*Case B.i.* In this case $\mathcal{P}_{ij} = \emptyset$ and by definition of $\bar{C}$, after line 6 is computed, in line 7 we found a path $\hat{p} \in \mathcal{L}_p$ such that $\mathrm{cost}(\hat{p}) = 0$. Line 9 adds $\hat{p}$ to $\mathcal{P}_{ij}$ and $m_{ij}$ is decremented in line 10.

*Case B.ii.* Similar to case $i$.

*Case B.iii.* We have $\mathcal{P}_{ij} = \emptyset$ and by definition of $\bar{C}$, after line 6 is computed, in line 8 we found a path $p \in \mathcal{L}_p$ such that $\mathrm{cost}(p) > 0$. Since $p$ contains some edges that are not in $\mathcal{G}_{sol}$, the current solution is updated in the same line. Line 9 adds $p$ to $\mathcal{P}_{ij}$ and $m_{ij}$ is decremented in line 10.

*Case B.iv.* In this case $\mathcal{P}_{ij} \neq \emptyset$ and by definition of $\bar{C}$, after line 6 is computed, in line 7 we found a path $\hat{p} \in \mathcal{L}_p$ such that $\mathrm{cost}(\hat{p}) = 0$. Line 10 decrements $m_{ij}$.

*Case B.v.* We have $\mathcal{P}_{ij} \neq \emptyset$ and by definition of $\bar{C}$, after line 6 is computed, in line 8 we find a path $p \in \mathcal{L}_p$ such that $\text{COST}(p) > 0$. Since $p$ contains some edges that are not in $\mathcal{G}_{sol}$, $\mathcal{G}_{sol}$ is updated in the same line. Line 10 decrements $m_{ij}$.
Notice that:

- new edges are added only when in $\mathcal{G}_{sol}$ there are not two node-disjoint paths between $i$ and $j$,
- if $m_{ij} = 1$ then $\mathcal{P}_{ij} \cap p = \{i, j\}$.

Line 11 calls $\text{Update\_Matrix}$ in order to update $M$. As loop $2 - 12$ is repeated until $m_{ij} = 0, \forall i, j \in S_D^{(I)}$, by construction, the network $\mathcal{G}_{sol}$ returned in line 13 is 2-node-survivable.

**QED**

### 5.3.3 Algorithm ConstPhase3_2NS

Next, we introduce a third algorithm to build feasible solutions for a BNDP2NS instance. We can see it as a variant of the algorithm exposed previously but with the difference that this one satisfies the 2-node-survivability requirement for a different pair of fixed switch sites at each iteration.

Given a BNDP2NS instance, iteratively the algorithm selects a pair of fixed switch sites not yet analyzed and builds two node-disjoint paths connecting them, i.e. a cycle. In addition, when adding these paths to the current partial solution, the algorithm updates the 2-node-survivability requirements between the fixed switch sites belonging to the built cycle. Once all the pairs of fixed switch sites are analyzed, the resulting network is 2-node-survivable finalizing thus the algorithm.

In more detail, the algorithm (shown in Figure 5.13) takes as inputs the graph $G_B$ of feasible connections on the backbone network, the matrix of connections cost $C$, and the GRASP parameter $k$. In line 1 we initialize:

- the current solution $\mathcal{G}_{sol}$ with the sites of $S_D^{(I)}$ without connections among them,
- the matrix $M = \{m_{ij}\}_{i,j \in S_D^{(I)}}$ (indicating the node-survivability requirements not yet satisfied between fixed sites) with $m_{ij} = \text{FALSE}, \forall i, j \in S_D^{(I)}$.

Loop 2-16 is repeated until all the fixed sites have satisfied their connection requirements (i.e. the resulting network is 2-node-survivable).

Each iteration works in the following way. Line 3 selects randomly (and uniformly) a pair $i, j \in S_D^{(I)}$ such that $m_{ij} = \text{FALSE}$ (i.e. a pair not yet analyzed). Line 4 computes an auxiliary matrix $\bar{C}$ of connections cost such that every connection $(u, v) \in \mathcal{G}_{sol}$ has cost zero. This will allow the algorithm to reuse already existing edges in $\mathcal{G}_{sol}$ (without considering their costs), when computing two node-disjoint paths connecting

**Procedure ConstPhase3_2NS($G_B$,$C$,$k$);**

1 $\mathcal{G}_{sol} \leftarrow (S_D^{(I)}, \emptyset)$; $m_{ij} \leftarrow \text{FALSE}\ \forall i,j \in S_D^{(I)}$;
2 **while** $\exists i,j \in S_D^{(I)}$ **such that** $m_{ij} = \text{FALSE}$ **do**
3 **Let** $i,j \in S_D^{(I)}$ **be a randomly chosen pair of fixed switch sites such that** $m_{ij} = \text{FALSE}$;
4 **Let** $\bar{C}$ **be the matrix given by:** $\bar{c}_{uv} \leftarrow \begin{cases} 0 & \textbf{if } (u,v) \in \mathcal{G}_{sol}, \\ c_{uv} & \textbf{if } (u,v) \in (G_B \setminus \mathcal{G}_{sol}). \end{cases}$;
5 $\mathcal{L}_p^1 \leftarrow$ **the** $k$ **shortest paths from** $i$ **to** $j$ **on** $G_B$**, considering the matrix** $\bar{C}$;
6 **if** $\exists \hat{p} \in \mathcal{L}_p^1$ **such that** $\text{COST}_{|\bar{C}}(\hat{p}) = 0$ **then** $p_1 \leftarrow \hat{p}$;
7 **else** $p_1 \leftarrow \text{Select\_Random}(\mathcal{L}_p^1)$; $\mathcal{G}_{sol} \leftarrow \mathcal{G}_{sol} \cup \{p_1\}$;
8 $\bar{\mathcal{G}} \leftarrow (G_B \setminus p_1)$;
9 **Let** $\bar{C}_2$ **be the matrix given by:** $\bar{c}_{uv} \leftarrow \begin{cases} 0 & \textbf{if } (u,v) \in \mathcal{G}_{sol}, \\ c_{uv} & \textbf{if } (u,v) \in (\bar{\mathcal{G}} \setminus \mathcal{G}_{sol}). \end{cases}$;
10 $\mathcal{L}_p^2 \leftarrow$ **the** $k$ **shortest paths from** $i$ **to** $j$ **on** $\bar{\mathcal{G}}$**, considering the matrix** $\bar{C}_2$;
11 **if** $\exists \hat{p} \in \mathcal{L}_p^2$ **such that** $\text{COST}_{|\bar{C}_2}(\hat{p}) = 0$ **then** $p_2 \leftarrow \hat{p}$;
12 **else** $p_2 \leftarrow \text{Select\_Random}(\mathcal{L}_p^2)$; $\mathcal{G}_{sol} \leftarrow \mathcal{G}_{sol} \cup \{p_2\}$;
13 $m_{ij} \leftarrow \text{TRUE}$;
14 $M \leftarrow \text{Update\_Matrix\_2}(\mathcal{G}_{sol}, M, p_1, p_2)$;
15 $\mathcal{G}_{sol} \leftarrow \text{Remove\_Internal\_KeyPaths}(p_1, p_2, \mathcal{G}_{sol})$;
16 **end_while;**
17 **return** $\mathcal{G}_{sol}$;
**end ConstPhase3_2NS;**

Figure 5.13: ConstPhase3_2NS pseudo-code.

$i$ and $j$. Line 5 computes the $k$ shortest paths from $i$ to $j$ on $G_B$ using the matrix $\bar{C}$. These paths are stored in the restricted candidate list $\mathcal{L}_p^1$. In line 6 we search a path $\hat{p} \in \mathcal{L}_p^1$ such that $\text{COST}_{|\bar{C}}(\hat{p}) = 0$ (its cost with respect to $\bar{C}$). If this is successful, we assign to $p_1$ the found path. Notice that, in this case, the cost of the current partial solution is not increased. Otherwise, line 7 selects randomly (and uniformly) a path $p_1$ from $\mathcal{L}_p^1$ and it is added to $\mathcal{G}_{sol}$ in the same line. Line 8 computes the auxiliary network $\bar{\mathcal{G}} = (G_B \setminus p_1)$. Let us note that any path connecting $i$ with $j$ in $\bar{\mathcal{G}}$ will be node-disjoint with respect to $p_1$. In line 9, we compute another auxiliary matrix $\bar{C}_2$ so that every connection $(u,v) \in \mathcal{G}_{sol}$ has cost zero. Thus, when computing a new path from $i$ to $j$ on $\bar{\mathcal{G}}$, we can reuse the connections already present in $\mathcal{G}_{sol}$ without considering their costs. Line 10 computes the $k$ shortest paths from $i$ to $j$ on $\bar{\mathcal{G}}$ using the matrix $\bar{C}_2$. The computed paths are stored in the restricted candidate list $\mathcal{L}_p^2$. Line 11 searches for a path $\hat{p} \in \mathcal{L}_p^2$ such that $\text{COST}_{|\bar{C}_2}(\hat{p}) = 0$ (its cost with respect to $\bar{C}_2$). If such paths exist, line 11 selects one of them. Otherwise, line 12 selects randomly (and uniformly) a path $p_2$ from $\mathcal{L}_p^2$ which is added to $\mathcal{G}_{sol}$ in the same line. Since now in $\mathcal{G}_{sol}$ there exist two node-disjoint paths communicating $i$ with $j$, the indicator $m_{ij}$ is set to $\text{TRUE}$ in line 13. Line 14 calls the auxiliary procedure $\text{Update\_Matrix\_2}$ to update the matrix $M$. The description of this procedure explains in detail the introduced updates. In line 15, we remove from $\mathcal{G}_{sol}$ every key-path $p_k \subset \mathcal{G}_{sol}$ whose endpoints are in $p_1 \cup p_2$ and moreover:

$$\text{EDGES}(p_k) \cap \text{EDGES}(p_1 \cup p_2) = \emptyset.$$

In this way, when deleting these key-paths, the local 2-node-survivability with respect to the pairs of fixed switch sites already added is preserved. Below, we will prove it formally.

Once the loop 2-16 is finalized the built feasible solution $\mathcal{G}_{sol}$ is returned in line 17. Figure 5.14 shows when the algorithm computes two node-disjoint paths connecting two fixed switch sites not yet analyzed.

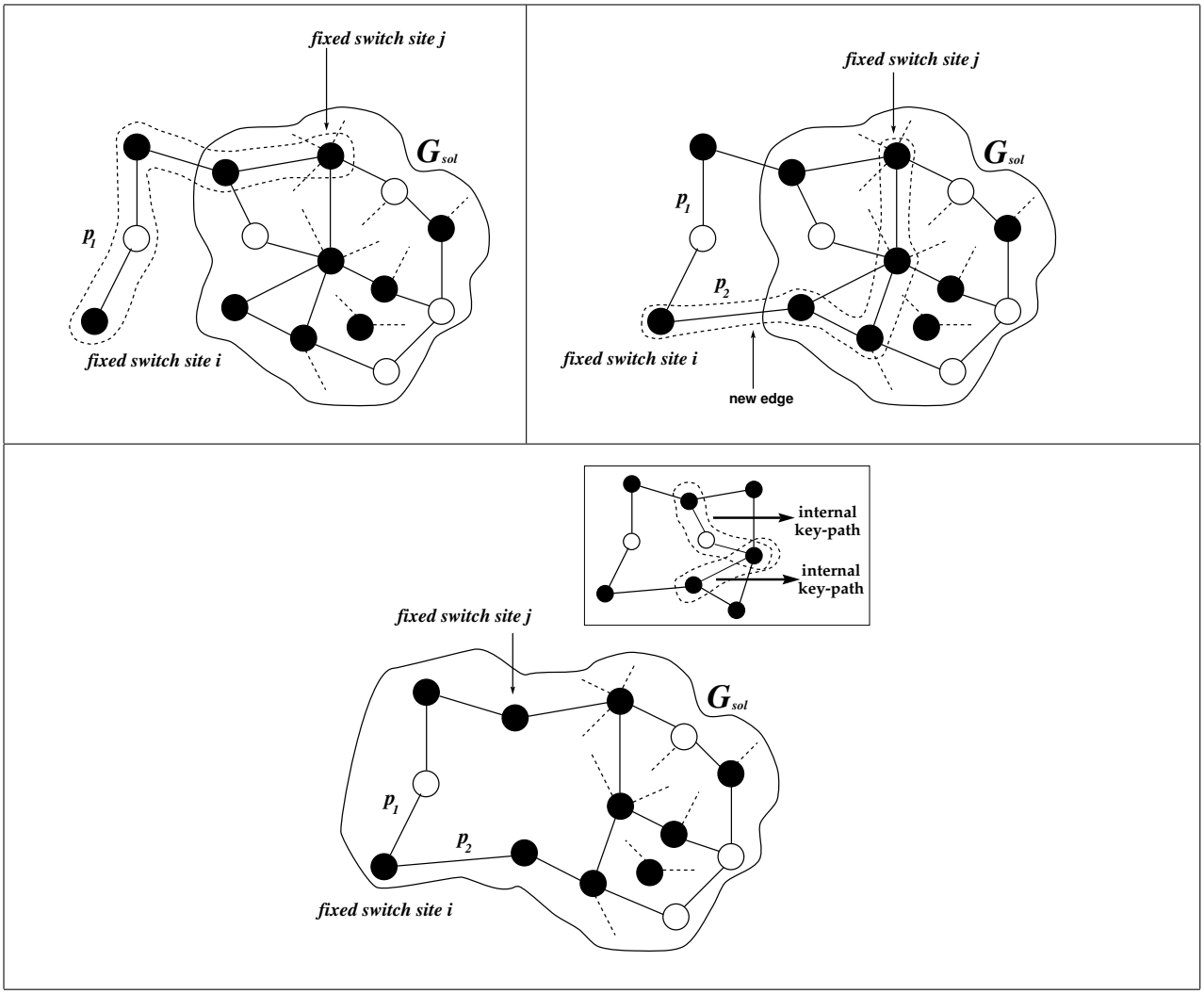



Figure 5.14: A typical ConstPhase3_2NS iteration.

We remark that in lines 6 and 10, we check if $\mathcal{L}_p^1$ or $\mathcal{L}_p^2$ are empty, in which case the algorithm finalizes since we will not be able to construct a feasible solution.

The procedure $\mathrm{Update\_Matrix\_2}$ updates the matrix $M$ by setting to $\mathrm{TRUE}$ the 2-node-survivability requirements between the fixed switch sites belonging to the connected component that contains the cycle computed by $\mathrm{ConstPhase3\_2NS}$ in each iteration.

Given a cycle $H \subseteq \mathcal{G}_{sol}$ conformed by two node-disjoint paths with the same endpoints, the auxil-

**Procedure Update_Matrix_2(**$\mathcal{G}_{sol}$**,**$M$**,**$p_1$**,**$p_2$**);**

1 **Let** $\hat{\mathcal{G}} \subseteq \mathcal{G}_{sol}$ **be the connected component such that** $(p_1 \cup p_2) \subseteq \hat{\mathcal{G}}$**;**
2 **for each** $u, v \in S_D^{(I)}$ **such that** $u, v \in \hat{\mathcal{G}}$ **do**
3 $m_{uv} \leftarrow \mathrm{TRUE}$**;**
4 **end_for_each;**
5 **return** $M$**;**
**end Update_Matrix_2;**

Figure 5.15: Update_Matrix_2 pseudo-code.

**Procedure Remove_Internal_KeyPaths(**$p_1$**,**$p_2$**,**$\mathcal{G}_{sol}$**);**

1 $H \leftarrow p_1 \cup p_2$**;**
2 **while** $\exists\ \mathrm{Internal\_KeyPaths}(H, \mathcal{G}_{sol})$ **do**
3 $p \leftarrow$ **a key-path not included in** $H$**, with endpoints in** $H$**;**
4 $\mathcal{G}_{sol} \leftarrow \mathcal{G}_{sol} \setminus p$**;**
5 **end_while;**
6 **return** $M$**;**
**end Remove_Internal_KeyPaths;**

Figure 5.16: Remove_Internal_KeyPaths pseudo-code.

iary procedure $\mathrm{Remove\_Internal\_KeyPaths}$ deletes from $\mathcal{G}_{sol}$ all the key-paths not included in $H$ whose endpoints belong to $H$ (denominated internal key-paths); since they are redundant in the current partial solution.

The following proposition demonstrates the constructive correctness of the algorithm $\mathrm{ConstPhase3\_2NS}$.

**Proposition 5.3.8** *If the algorithm* $\mathrm{ConstPhase3\_2NS}$ *returns a graph this will be a 2-node-survivable solution for the BNDP2NS.*

**Proof.** By induction on the number of iterations already computed, we will prove that the connected components that integrate the current partial solution $\mathcal{G}_{sol}$ preserve the 2-connectedness.
In line 1 the algorithm initializes:

- $\mathcal{G}_{sol}$ with the set of fixed switch sites $S_D^{(I)}$ without edges among them,
- the auxiliary matrix $M$ (indicating the pairs of fixed switch sites already analyzed, i.e. for which we known that the local 2-node-survivability is satisfied) with $m_{ij} = \mathrm{FALSE}$, $\forall i, j \in S_D^{(I)}$.

Let us suppose that for certain iteration the condition in line 2 is $\mathrm{TRUE}$. In line 3 we choose randomly a pair $i, j \in S_D^{(I)}$ of fixed switch sites such that $m_{ij} = \mathrm{FALSE}$. By construction, we have that:

i) lines 4-7 compute the $k$ shortest paths from $i$ to $j$ on $G_B$ so that the costs of the edges belonging to the partial solution $\mathcal{G}_{sol}$ are not considered, and one of them is chosen randomly. Let $p_1$ be the selected path, $\mathcal{G}_{sol}$ is updated by adding $p_1$ to it.

ii) lines 8-12 compute the $k$ shortest paths from $i$ to $j$ on $G_B \setminus p_1$ so that the costs of the edges belonging to the partial solution $\mathcal{G}_{sol}$ are not considered, and one of them is chosen randomly. Let $p_2$ be the selected path, $\mathcal{G}_{sol}$ is updated by adding $p_2$ to it.

Line 13 sets $m_{ij}$ to $\mathrm{TRUE}$, since $i$ and $j$ are locally 2-node-connected. It is easy to see that the computed paths $p_1$ and $p_2$ are node-disjoint. Let us denote by $H$ to the cycle conformed by $p_1 \cup p_2$. Let $\hat{\mathcal{G}} \subseteq \mathcal{G}_{sol}$ be the connected component containing $H$. By inductive hypothesis, $\hat{\mathcal{G}}$ is 2-node-connected. Consider $\hat{H} \subseteq \hat{\mathcal{G}}$ the subgraph conformed by $H$ union all the key-paths whose endpoints belong to $H$. By Lema 5.3.2 (modelling these key-paths as simple edges), the network $(\hat{\mathcal{G}} \setminus \hat{H}) \cup H$ is 2-node-connected. Thus, when removing from $\mathcal{G}_{sol}$ the internal key-paths of $H$ (line 12), in the resulting network all the connected component will preserve the 2-connectedness. In addition, in line 11, we update the indicator matrix $M$ by using the procedure $\mathrm{Update\_Matrix\_2}$.
Inductively, the algorithm finalizes once there are no pair $i, j \in S_D^{(I)}/m_{ij} = \mathrm{FALSE}$, i.e. the built network $\mathcal{G}_{sol}$ is 2-node-survivable, as required and completing the proof.

**QED**

## 5.4 BNDP2NS Local Search Phase Algorithms

As mentioned in Chapter 2, since the solution produced by the construction phase is not necessarily a local optimum, local search can be applied to improve it. In this section we propose two local search strategies for the BNDP2NS, one based in key-path replacements and the other in key-tree replacements which can work in complementary form by running in combined way.

The first step towards the implementation of a local search algorithm consists in identifying an appropriate neighborhood definition. In this way, before describing each local search algorithm, we will introduce the neighborhood structure on which the algorithm is based. Next, we present the local search based on key-path replacements.

### 5.4.1 Algorithm LocalSearch1_2NS

**Definition 5.4.1 (key-path based Neighborhood Structure)** *Let $\mathcal{G}_{sol}$ be a 2-node-survivable feasible solution. Given a key-path $p \subset \mathcal{G}_{sol}$, we define a neighbor solution of $\mathcal{G}_{sol}$ as: $\hat{\mathcal{G}}_{sol} = (\mathcal{G}_{sol} \setminus p) \cup \hat{p}$, where $\hat{p}$ is another key-path connecting the endpoints of $p$ and maintaining the feasibility in the new network $\hat{\mathcal{G}}_{sol}$.*

*The Key-Path Neighborhood of $\mathcal{G}_{sol}$ is composed of the neighbor solutions obtained by applying the previous operation to each of the different key-paths in $\mathcal{K}(\mathcal{G}_{sol}) = (p_1, \ldots, p_h)$.*

Notice that it is structurally equal to the key-path Neighborhood Structure defined in Chapter 4, but here it is defined on the BNDP2NS.

We propose a local search algorithm for the BNDP2NS which is based on the key-path Neighborhood Structure. We called it $\mathrm{LocalSearch1\_2NS}$. Next, we introduce a detailed description of $\mathrm{LocalSearch1\_2NS}$ and some topological properties satisfied by the constructed neighbor solutions.

The algorithm builds iteratively neighbor solutions by replacing each key-path from the current solution by another key-path which preserves the feasibility and has smaller cost.

**Procedure LocalSearch1_2NS($G_B$,$C$,$\mathcal{G}_{sol}$);**

1 $\mathcal{K}(\mathcal{G}_{sol}) \leftarrow$ **the decomposition in key-paths of** $\mathcal{G}_{sol}$;
2 $\mathcal{H}_{sol} \leftarrow \mathcal{G}_{sol}$;
3 **for each key-path** $p \in \mathcal{K}(\mathcal{G}_{sol})$ **with ends** $u, v$ **do**
4 $\quad\hat{\mathcal{H}} \leftarrow$ **the subgraph induced by** $\textsc{nodes}(p) \cup (S_D \setminus \textsc{nodes}(\mathcal{H}_{sol}))$;
5 $\quad\hat{p}$ **the shortest path from** $u$ **to** $v$ **on** $\hat{\mathcal{H}}$;
6 $\quad\mathcal{H}_{sol} \leftarrow (\mathcal{H}_{sol} \setminus p) \cup \hat{p}$;
7 **end_for_each;**
8 **return** $\mathcal{H}_{sol}$;
**end LocalSearch1_2NS;**

Figure 5.17: LocalSearch1_2NS pseudo-code.

In more detail, the algorithm (shown in Figure 5.17) takes as inputs the graph $G_B$ of feasible connections on the backbone network, the matrix of connections cost $C$, and the current feasible solution $\mathcal{G}_{sol}$. In line 1 we compute the decomposition in key-paths of $\mathcal{G}_{sol}$, denoted by $\mathcal{K}(\mathcal{G}_{sol})$. Line 2 initialize the network $\mathcal{H}_{sol}$ with the current solution. Loop 3-7 is repeated exactly $|\mathcal{K}(\mathcal{G}_{sol})|$ times. In each iteration a key-path $p \in \mathcal{K}(\mathcal{G}_{sol})$ not yet analyzed is selected randomly. Let $u, v$ be the ends of the current key-path $p$. In line 4 we compute the subgraph $\hat{\mathcal{H}}$ induced by $\textsc{nodes}(p) \cup (S_D \setminus \textsc{nodes}(\mathcal{H}_{sol}))$. Clearly, any path connecting $u$ with $v$ in $\hat{\mathcal{H}}$ can replace $p$ in $\mathcal{H}_{sol}$ and to preserve the feasibility. Accordingly, in line 5 we compute the shortest path from $u$ to $v$ on $\hat{\mathcal{H}}$, which is denoted by $\hat{p}$. Line 6 updates $\mathcal{H}_{sol}$ by replacing $p$ by $\hat{p}$. Once all the key-paths from $\mathcal{K}(\mathcal{G}_{sol})$ have been analyzed the feasible solution $\mathcal{H}_{sol}$ is returned in line 8.

We illustrate in Figure 5.18 the replacement of a key-path on the current neighbor solution $\mathcal{H}_{sol}$.

- The first graph shows $\mathcal{H}_{sol}$ and the subgraph induced by the set of sites $S_D \setminus \textsc{nodes}(\mathcal{H}_{sol})$. The substitute key-path $\hat{p}$ will have nodes of $\textsc{nodes}(p) \cup (S_D \setminus \textsc{nodes}(\mathcal{H}_{sol}))$ preserving thus the 2-node-survivability, i.e. the feasibility.

- The second graph shows network $\mathcal{H}_{sol}$ already updated.

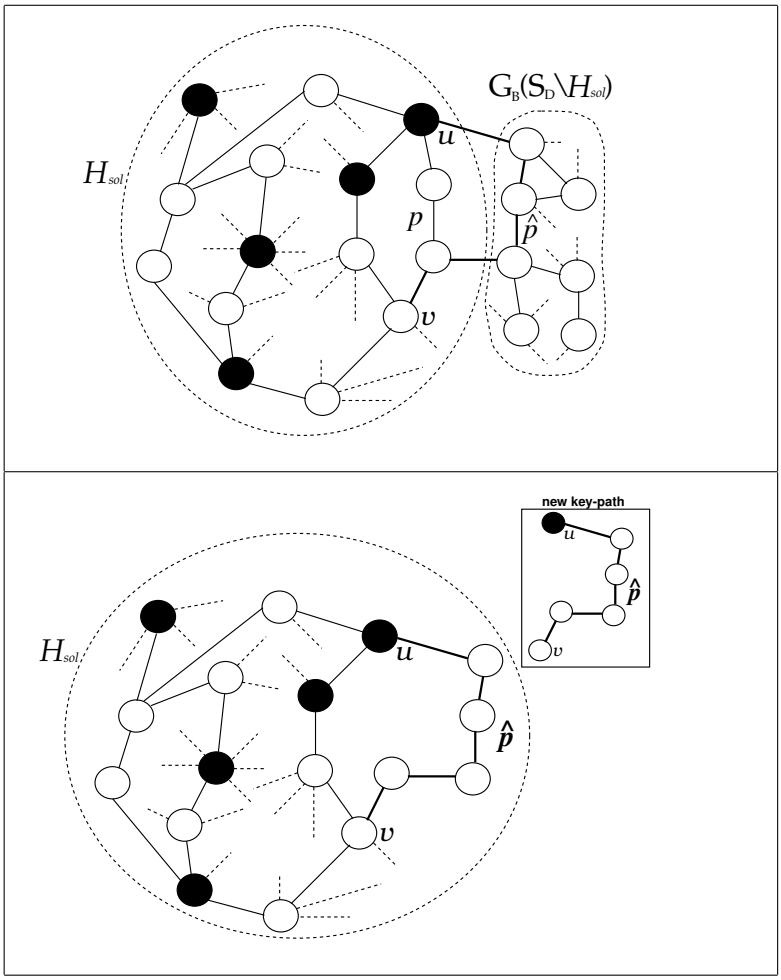



Figure 5.18: A key-path replacement.

When the construction phase delivers a minimal feasible solution, depending on the neighborhood computation complexity, the preservation of the minimality is a good property for the local search phase since the search of the global optimum is focused only on a subspace of feasible solutions that contains it. The following proposition demonstrates the minimality preservation in each $\mathrm{LocalSearch1\_2NS}$ iteration.

**Proposition 5.4.2** *If* $\mathrm{LocalSearch1\_2NS}$ *receives as input a minimal feasible solution* $\mathcal{G}_{sol}$*, the local search preserves the minimality at any time.*

**Proof.** By backward induction in the number of key-paths not yet analyzed (denoted by $n_k$), we will demonstrate that the network $\mathcal{H}_{sol}$ is minimal at any time.
*Basic Step:* $n_k = |\mathcal{K}(\mathcal{G}_{sol})|$. In line 2 the network $\mathcal{H}_{sol}$ is initialized with the network $\mathcal{G}_{sol}$ which is minimal.

*Inductive Step:* $n_k < |\mathcal{K}(\mathcal{G}_{sol})|$. The inductive step is presented as the following way.
As inductive thesis we have that if $0 < n_k = m \leq |\mathcal{K}(\mathcal{G}_{sol})|$ the network $\mathcal{H}_{sol}$ is feasible and minimal. As inductive thesis the property is fulfilled when $n_k = m - 1$.
Suppose that in certain local search iteration we have $n_k = m$. Let us analyze the following cases.
*Case 1.* $m > 0$. By I.H. the network $\mathcal{H}_{sol}$ resulting of the previous iteration is 2-node-survivable and minimal. As $m > 0$ the algorithm will execute loop $3 - 7$. Let $p \in \mathcal{K}(\mathcal{G}_{sol})$ be a key-path not yet analyzed with ends $u, v$. Line 5 computes a path $\hat{p}$ which is the shortest path from $u$ to $v$ on the network $\hat{\mathcal{H}} = G_B(\text{NODES}(p) \cup (S_D \setminus \text{NODES}(\mathcal{H}_{sol}))$. This path satisfies:

i) $\hat{p} \cap (\mathcal{H}_{sol} \setminus I) = \{u, v\}$, with $I$ the internal nodes of $p$,

ii) $\text{COST}(\hat{p}) \leq \text{COST}(p)$,

therefore the network $\hat{\mathcal{H}} = (\mathcal{H}_{sol} \setminus p) \cup \hat{p}$ is a minimal feasible solution satisfying $\text{COST}(\hat{\mathcal{H}}) \leq \text{COST}(\mathcal{H}_{sol})$. Line 6 updates the current solution $\mathcal{H}_{sol}$ with $\hat{\mathcal{H}}$.
*Case 2.* $m = 0$. By I.H. the network $\mathcal{H}_{sol}$ resulting of the previous iteration is 2-node-survivable and minimal. In the current iteration the loop $3 - 6$ is not executed since all the key-paths from $\mathcal{K}(\mathcal{G}_{sol})$ have already been analyzed, finalizing thus the local search.

**QED**

### 5.4.2 Algorithm LocalSearch2_2NS

Before introducing another local search algorithm for the BNDP2NS, we define a new Neighborhood Structure based on the substitution of key-trees by other trees (not necessarily key-trees) which preserve the feasibility.

**Definition 5.4.3 (tree based Neighborhood Structure)** *Let $\mathcal{G}_{sol}$ be a 2-node-survivable feasible solution. Given a key-node $v \in \mathcal{G}_{sol}$ and its associated key-tree $\mathcal{T}_v \subset \mathcal{G}_{sol}$, we define a neighbor solution of $\mathcal{G}_{sol}$ as: $\hat{\mathcal{G}}_{sol} = (\mathcal{G}_{sol} \setminus \mathcal{T}_v) \cup \mathcal{T}$, where $\mathcal{T}$ is a tree spanning the endpoints of $\mathcal{T}_v$ and maintaining the feasibility in the new network $\hat{\mathcal{G}}_{sol}$.*

*The Tree Neighborhood of $\mathcal{G}_{sol}$ is composed of the neighbor solutions obtained by applying iteratively the previous operation to each of the different key-trees in $\mathcal{G}_{sol}$.*

We propose another local search algorithm for the BNDP2NS which is based on the tree Neighborhood Structure. We called it $\text{LocalSearch2\_2NS}$. Next, we introduce a detailed description of $\text{LocalSearch2\_2NS}$ and some topological properties satisfied by the constructed neighbor solutions.

The algorithm builds iteratively neighbor solutions by replacing key-trees from the current solution by other trees which are suitably designed so that the 2-node-survivability (i.e. the feasibility) is preserved. This process is repeated until the key-tree replacements do not induce a better feasible solution.

```
Procedure LocalSearch2_2NS(G_B,C,G_sol);

1  improve ← TRUE;
2  while improve do
3     improve ← FALSE;
4     Let X be the set of key-nodes in G_sol;
5     S̄ ← S_D \ NODES(G_sol);
6     while not(improve) and ∃ key-nodes not yet analyzed do
7        Let v ∈ X be not yet analyzed;
8        [G_sol, improve] ← RecConnect(G_B, C, G_sol, v, S̄);
9     end_while;
10 end_while;
11 return G_sol;
end LocalSearch2_2NS;
```

Figure 5.19: LocalSearch2_2NS pseudo-code.

The algorithm (shown in Figure 5.19) takes as inputs the graph $G_B$ of feasible connections on the backbone network, the matrix of connections cost $C$, and the current feasible solution $\mathcal{G}_{sol}$. In line 1 we initialize with $\mathrm{FALSE}$ the indicator variable $improve$ used to indicate improvements obtained by the key-tree replacements. Loop 2-10 searches for neighbor solutions analyzing each key-node in the current solution $\mathcal{G}_{sol}$ and replacing their respective key-trees by trees in order to improve its cost without losing the feasibility.

Each iteration works of the following way. In line 3 $improve$ is set to $\mathrm{FALSE}$. Line 4 computes the set $X$ of key-nodes of $\mathcal{G}_{sol}$. Line 5 computes the set $\bar{S}$ of non-fixed switch sites non-belonging to $\mathcal{G}_{sol}$. The internal loop 6-9 analyzes the sites from $X$ one at a time with the aim of finding a suitable tree of smaller cost to replace the corresponding key-tree. Line 7 selects a site $v \in X$ randomly (and uniformly). In line 8 we execute the algorithm called $\mathrm{RecConnect}$ in order to replace the key-tree associated with $v$ for a substitute tree which has a smaller cost and maintains the 2-node-survivability (we give in 5.4.3 a detailed description for this algorithm and Proposition 5.10 proves that it preserves the feasibility). If this search is successful, the $\mathrm{RecConnect}$ delivers a better neighbor solution and the current solution $\mathcal{G}_{sol}$ is updated with it in the same line. In addition, $improve$ is set $\mathrm{TRUE}$, to restart the local search. Otherwise, if $\mathrm{RecConnect}$ does not find a substitute tree, another key-node not yet analyzed will be considered.

Once there are no more improvements by key-tree replacements the current solution $\mathcal{G}_{sol}$ is returned in line 11.

### 5.4.3 RecConnect description

The algorithm $\mathrm{RecConnect}$ is an auxiliary procedure used by the algorithm $\mathrm{LocalSearch2\_2NS}$. Given the current solution $\mathcal{G}_{sol}$ and a key-node $v \in \mathcal{G}_{sol}$, $\mathrm{RecConnect}$ tries to build a tree $\mathcal{T}$ spanning the endpoints of $\mathcal{T}_v$ (being $\mathcal{T}_v$ the key-tree associated a $v$). To preserve the feasibility, the substitute tree $\mathcal{T}$ is built using only the sites of $\mathcal{T}_v$ and the non-fixed switch sites non-belonging to $\mathcal{G}_{sol}$.

**Procedure RecConnect(**$G_B$,$C$,$\mathcal{G}_{sol}$,$v$,$\bar{S}$**);**

**1** $Y \leftarrow \mathrm{Nodes\_Key\_Tree}(v, \mathcal{G}_{sol})$**;**
**2** $Z \leftarrow \mathrm{Ends\_Key\_Tree}(v, \mathcal{G}_{sol})$**;**
**3** $cost \leftarrow \mathrm{Cost\_Key\_Tree}(v, \mathcal{G}_{sol})$**;**
**4** $\mathcal{H} \leftarrow$ **subgraph induced by** $(Y \cup \bar{S})$ **in** $G_B$**;** $\mathcal{H} \leftarrow \mathcal{H} \setminus \mathrm{EDGES}(\mathcal{G}_{sol}(Z))$**;**
**5** $\mathcal{T} \leftarrow Z$**;** $value \leftarrow 0$**;**
**6** $m_{ij} \leftarrow \mathrm{FALSE}, \forall i, j \in Z$**;**
**7** **while** $(\exists i, j \in Z$ **such that** $m_{ij} = \mathrm{FALSE})$ **and** $(value < cost)$ **do**
**8** **Let** $i, j \in Z$ **be a randomly chosen pair of nodes such that** $m_{ij} = \mathrm{FALSE}$**;**
**9** **Compute** $\bar{C}$ **where** $\bar{c}_{uk} = \begin{cases} 0 & \text{if } (u,k) \in \mathcal{T}, \\ c_{uk} & \text{otherwise}, \end{cases}$
**10** $[val, p_{ij}] \leftarrow$ **the shortest path from** $i$ **to** $j$ **in** $\mathcal{H}$ **using** $\bar{C}$**;**
**11** $\mathcal{T} \leftarrow \mathcal{T} \cup p_{ij}$**;**
**12** $value \leftarrow value + val$**;**
**13** **Let** $\hat{\mathcal{H}} \subseteq \mathcal{T}$ **be the connected component containing** $i$**;**
**14** $m_{uk} \leftarrow \mathrm{TRUE}, \forall u, k \in (\hat{\mathcal{H}} \cap Z)$**;**
**15** **end_while;**
**16** **if** $(value < cost)$ **then**
**17** $\mathcal{G}_{sol} \leftarrow (\mathcal{G}_{sol} \setminus (Y \setminus Z)) \cup \mathcal{T}$**;**
**18** $improve \leftarrow \mathrm{TRUE}$**;**
**19** **else** $improve \leftarrow \mathrm{FALSE}$**;**
**20** **return** $\mathcal{G}_{sol}$**,** $improve$**;**
**end RecConnect;**

Figure 5.20: RecConnect pseudo-code.

The algorithm (shown in Figure 5.20) takes as inputs the graph $G_B$ of feasible connections on the backbone network, the matrix of connections cost $C$, the current feasible solution $\mathcal{G}_{sol}$, the current key-node $v$, and the set $\bar{S}$ of non-fixed switch sites non-belonging to $\mathcal{G}_{sol}$. Let us denote by $\mathcal{T}_v$ the key-tree associated with $v$. Line 1 computes the set $Y$ of sites belonging to $\mathcal{T}_v$. Line 2 computes the set $Z$ of endpoints of $\mathcal{T}_v$. Line 3 computes the cost of $\mathcal{T}_v$. Line 4 computes the subnetwork $\mathcal{H}$ induced by the sites $Y \cup \bar{S}$. Furthermore, in the resulting network the edges belonging to $\mathrm{EDGES}(\mathcal{G}_{sol}(Z))$ are removed. In line 5 we initialize the substitute tree $\mathcal{T}$ with the set of sites $Z$ and its cost with zero. Line 6 initialize an indicator matrix $M = \{m_{ij}\}_{i,j \in Z}$ used to indicate the existence of a path from $i$ to $j$ in $\mathcal{T}$. Loop 7-15 compute shortest paths between pairs of sites of $Z$ not yet connected in $\mathcal{T}$, which are successively added to $\mathcal{T}$. The second condition in line 7 controls that at any moment the cost of $\mathcal{T}$ does not exceed the cost of $\mathcal{T}_v$.

Each iteration works in the following way. Line 8 selects randomly (and uniformly) a pair $i, j \in Z$ such that $m_{ij} = \text{FALSE}$, i.e. they are not connected in $\mathcal{T}$. Line 9 computes an auxiliary matrix $\bar{C}$ of connections cost such that all connection $(u, k) \in \mathcal{T}$ has cost zero. The objective is to reuse (if it is possible) the already existing connections in $\mathcal{T}$ whenever new paths between pairs of sites of $Z$ are computed. Line 10 computes the shortest path from $i$ to $j$ on $\mathcal{H}$ using the matrix $\bar{C}$. Let $p_{ij}$ be this path, the line 11 updates the tree $\mathcal{T}$ adding $p_{ij}$ to it. The cost of $\mathcal{T}$ is updated in line 12. Sine $i$ and $j$ could have been in different connected components the matrix $M$ must be updated with respect to the fixed sites of these; therefore in lines 13-14 $m_{uk}$ is set to $\text{TRUE}$ for all pair of fixed sites belonging to the connected component of $\mathcal{T}$ containing $i$ and $j$. The loop 7-15 finalizes once all the nodes of $Z$ are connected (i.e. $\mathcal{T}$ is a tree spanning $Z$) or when the cost of $\mathcal{T}$ is greater to the cost of $\mathcal{T}_v$. In lines 16-19 we check whether the tree $\mathcal{T}$ improves the cost of the current key-path $\mathcal{T}_v$. If this is the case, the current solution $\mathcal{G}_{sol}$ is updated replacing $\mathcal{T}_v$ by $\mathcal{T}$ and the indicator $improve$ is set to $\text{TRUE}$ in line 18. Both are returned in line 20. If no improving tree is found the indicator $improve$ is set to $\text{FALSE}$ in line 19 and returned in line 20.

In order to clarify how $\text{RecConnect}$ works, we exemplify in Figure 5.21 the replacement of a key-tree on the current neighbor solution $\mathcal{G}_{sol}$.

- In the first graph, we can see a key-node $v \in \mathcal{G}_{sol}$ and its corresponding key-tree $\mathcal{T}_v$.
- The second graph shows one of the possible substitute trees for the key-tree $\mathcal{T}_v$. Observe that in this case a new key-node has been introduced to the solution and furthermore the number of endpoints for the corresponding key-tree is smaller (eventually, $v$ is again a key-node).
- The third graph shows another substitute tree. In this case the substitute tree does not have a key-node. In addition $\mathcal{G}_{sol}$ has a key-node (and a key-tree) less than before.
- The fourth graph is a substitute tree with a simple path topology connecting the endpoints of $\mathcal{T}_v$. As above, $\mathcal{G}_{sol}$ has a key-node (and a key-tree) less than before.

In all cases, the solution obtained after the key-tree replacement preserves the 2-node-survivability.

Next, we present two auxiliary propositions relative to some topological properties satisfied by a key-tree belonging to certain feasible solution.

**Proposition 5.4.4** *Given a minimal 2-node-survivable network $\mathcal{G}_{sol}$, let $s \in \mathcal{G}_{sol}$ be a key-node and $\mathcal{T}_s$ the key-tree associated with $s$ with set of endpoints $Z$. Then, no pair of nodes of Z are adjacent in $\mathcal{G}_{sol}$.*

**Proof.** Let us suppose that $\exists u, v \in Z$ such that $(u, v) \in \mathcal{G}_{sol}$. Let $k \in Z, k \neq u, v$ be another endpoint and $p_u, p_v, p_k \subseteq \mathcal{T}_s$ the key-paths connecting $s$ with $u$, $v$, and $k$ respectively. As $\mathcal{G}_{sol}$ is 2-node-survivable, there

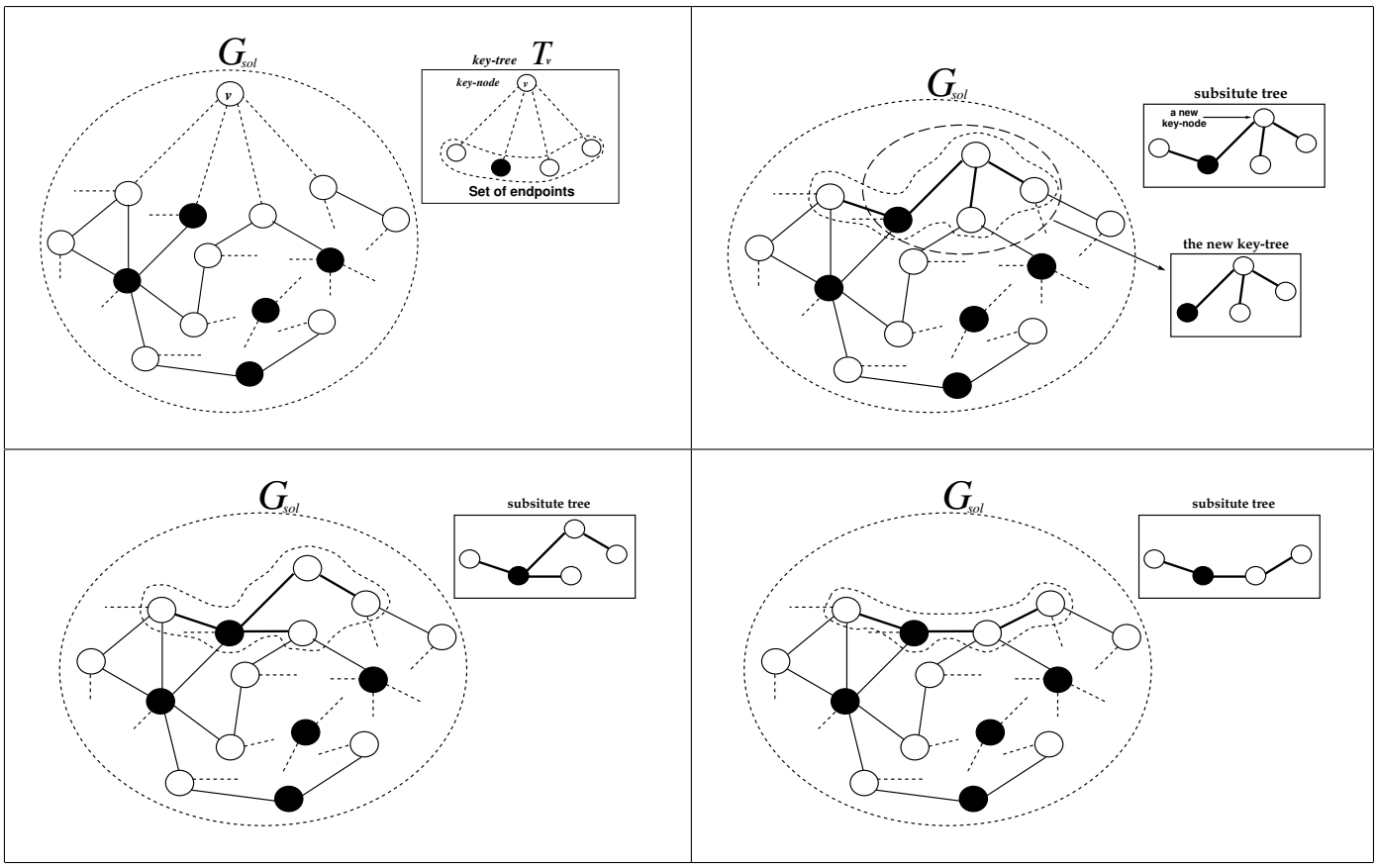



Figure 5.21: A key-tree replacement.

exists a path $\bar{p}$ from $k$ to $v$ (or $u$) and not containing $s$. Without loss of generality, we assume that $u \notin \bar{p}$ (if $u \in \bar{p}$, consider the sub-path $\bar{p}_{(k,u)} \subset \bar{p}$). Let us define the sub-network $\mathcal{H} = p_u \cup p_v \cup p_k \cup \bar{p} \cup \{(u, v)\}$. Notice that this network is 2-node-connected but non-minimal. Let $I$ be the nodes from $p_v$ with degree 2. Consider the network $\bar{\mathcal{H}} = \mathcal{H} \setminus I$, this is 2-node-connected and minimal. (If $p_v$ is a simple edge, consider $\bar{\mathcal{H}} = \mathcal{H} \setminus p_v$). Moreover, if we see the key-paths $p_u$, $p_v$, and $p_k$ as "super" edges, applying Lema 5.3.2, we have that: $\bar{\mathcal{G}}_{sol} = (\mathcal{G}_{sol} \setminus \text{EDGES}(\mathcal{H})) \cup \text{EDGES}(\bar{\mathcal{H}})$ is 2-node-connected (i.e. 2-node-survivable); Figure 5.22 illustrates this situation. This contradicts the minimality of $\mathcal{G}_{sol}$. Hence, $\nexists u, v \in Z$ such that $(u, v) \in \mathcal{G}_{sol}$.

**QED**

**Proposition 5.4.5** *Given a minimal 2-node-survivable network $\mathcal{G}_{sol}$, let $s \in \mathcal{G}_{sol}$ be a key-node, $\mathcal{T}_s$ the key-tree associated with $s$ with set of endpoints $Z$, and $I$ the set of internal nodes of $\mathcal{T}_s$. Then, by replacing in $\mathcal{G}_{sol}$ the key-tree $\mathcal{T}_s$ by a tree $\mathcal{T}$ verifying:*

*i)* $Z \subset \mathcal{T}$ *and the endpoints of* $\mathcal{T}$ *are a subset of nodes* $X \subseteq Z$,

*ii)* $\text{NODES}(\mathcal{T}) \cap \text{NODES}(\mathcal{G}_{sol}) = Z \cup J$, *with* $J \subseteq I$ *(eventually* $J = \emptyset$*),*

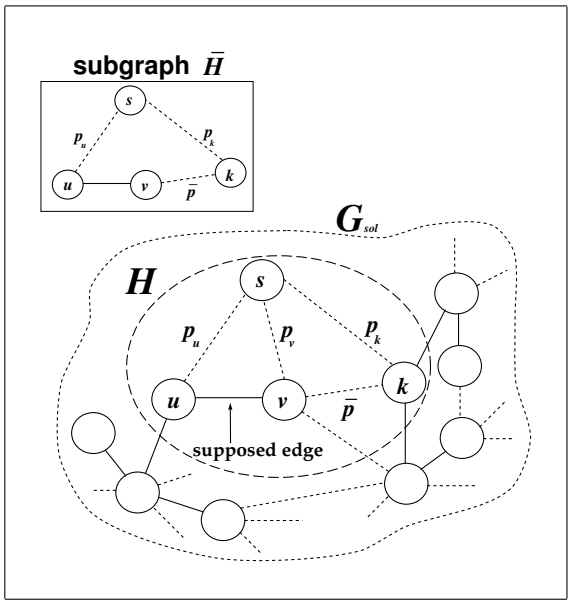



Figure 5.22: Example associated with Proposition 5.4.4.

*we obtain a network $\bar{\mathcal{G}}_{sol}$ which is 2-node-survivable.*

**Proof.** Let $\bar{\mathcal{G}}_{sol}$ be the network obtained by replacing in $\mathcal{G}_{sol}$ the key-tree $\mathcal{T}_s$ by $\mathcal{T}$. By the 2-node-survivability properties, it is easy to see that $\forall u, v \in Z$ there exists a path $p$ from $u$ to $v$ such that $p \subset (\mathcal{G}_{sol} \backslash I)$. In addition, points $\mathrm{i} - \mathrm{ii}$ guarantee the 2-node-survivability of network $\bar{\mathcal{G}}_{sol}$ since the lost node-connectivity requirements in $(\mathcal{G}_{sol} \setminus \mathcal{T}_s)$ will be satisfied again when adding $\mathcal{T}$.

**QED**

This last proposition in not true when the solution $\mathcal{G}_{sol}$ is non-minimal. Thus, if $\mathcal{G}_{sol}$ is not minimal, the following Proposition introduces an additional restriction to preserve the 2-node-survivability.

**Proposition 5.4.6** *Given a 2-node-survivable network $\mathcal{G}_{sol}$ fulfilling points $i - ii$ of Proposition 5.4.5 and moreover:*

$$\text{EDGES}(\mathcal{G}_{sol}) \cap \text{EDGES}(\mathcal{T}) = \emptyset,$$

*then, when replacing the key-tree $\mathcal{T}_s$ by the tree $\mathcal{T}$ we obtain a 2-node-survivable network $\bar{\mathcal{G}}_{sol}$.*

**Proof.** If $\mathcal{G}_{sol}$ is minimal, by Proposition 5.4.5 the resulting network is minimal.
If $\mathcal{G}_{sol}$ is not minimal, since the substitute tree $\mathcal{T}$ does not have any edge belonging to $\mathcal{G}_{sol}$, when removing from $\bar{\mathcal{G}}_{sol}$ the internal nodes of $\mathcal{T}_s$ the lost node-connectivity levels are reestablished by adding $\mathcal{T}$.

**QED**

Based on the previous proposition, the following proposition demonstrates the constructive correctness of the algorithm $\mathrm{RecConnect}$.

**Proposition 5.4.7** *Given a 2-node-survivable network $\mathcal{G}_{sol}$, the set $\bar{S}$ of non-fixed switch sites not included in $\mathcal{G}_{sol}$, and a key-node $v \in \mathcal{G}_{sol}$, the algorithm* $\mathrm{RecConnect}$ *builds a better neighbor solution by replacing the key-tree associated with $v$ by another tree which preserves the feasibility.*

**Proof.** Let $\mathcal{T}_v$ be the key-tree associated with $v$. Lines 1-6 computes: the set $Y$ of nodes in $\mathcal{T}_v$, the set $Z \subset \mathcal{T}_v$ of endpoints, and the subgraph $\mathcal{H} = G_B(Y \cup \bar{S})$. Line 5 initializes $\mathcal{T}$ with the nodes of $Z$ without edges among them. Line 6 initializes an auxiliary matrix which indicates at any time the pairs of nodes of $Z$ not yet connected in $\mathcal{T}$. It is easy to see that, by construction, once finalized loop $7-15$ and supposing that at any time $\mathrm{COST}(\mathcal{T}) < \mathrm{COST}(\mathcal{T}_v)$, the network $\mathcal{T}$ has tree topology and furthermore:

- $Z \subset \mathcal{T}$,
- the endpoints of $\mathcal{T}$ are a subset $X \subseteq Z$,
- $\mathrm{NODES}(\mathcal{T}) \cap \mathrm{NODES}(\mathcal{G}_{sol}) = Z \cup J$, with $J \subseteq (Y \setminus Z)$ (eventually $J = \emptyset$),
- $\mathrm{EDGES}(\mathcal{G}_{sol}) \cap \mathrm{EDGES}(\mathcal{T}) = \emptyset$.

By Proposition 5.4.6, the resulting network is 2-node-survivable. The network $\mathcal{G}_{sol}$ is updated in line 17 and returned in line 20. Since this is a better solution, the indicator variable $improve$ is set to $\mathrm{TRUE}$ in line 18 and also returned in line 20.
Notice that if in a given iteration $\mathrm{COST}(\mathcal{T}) > \mathrm{COST}(\mathcal{T}_v)$, then the loop finalizes, $improve$ is set to $\mathrm{FALSE}$ in line 19 and it is returned in line 20.

**QED**

Now, we introduce a small example which shows the application of $\mathrm{RecConnect}$ when replacing a key-tree by a suitable tree to obtain a neighbor feasible solution. Figure 5.23 includes the following graphs.

- The first graph corresponds to the graph of feasible connections on the backbone network $G_B$. The black nodes represent the fixed switch sites whereas the white nodes represent the non-fixed switch sites.
- The second graph corresponds to a minimal 2-node-survivable topology spanning the fixed sites. This feasible solution has two key-nodes, one of them labeled with $v$. The associated key-tree $\mathcal{T}_v$ has three endpoints: one non-fixed site and two fixed sites.

- The other three graphs are all the possible neighbor feasible solutions that we can obtain by replacing the key-tree $\mathcal{T}_v$ by another tree spanning their endpoints. In each case, the broken lines represent the edges of the substitute tree. Depending on the connection costs, any of them can be computed by the algorithm $\mathrm{RecConnect}$. Notice that two of them maintain $v$ as a key-node whereas the other consists of a simple path connecting the endpoints of $\mathcal{T}_v$. In addition the three topologies are minimal (when removing an edge the feasibility is lost).

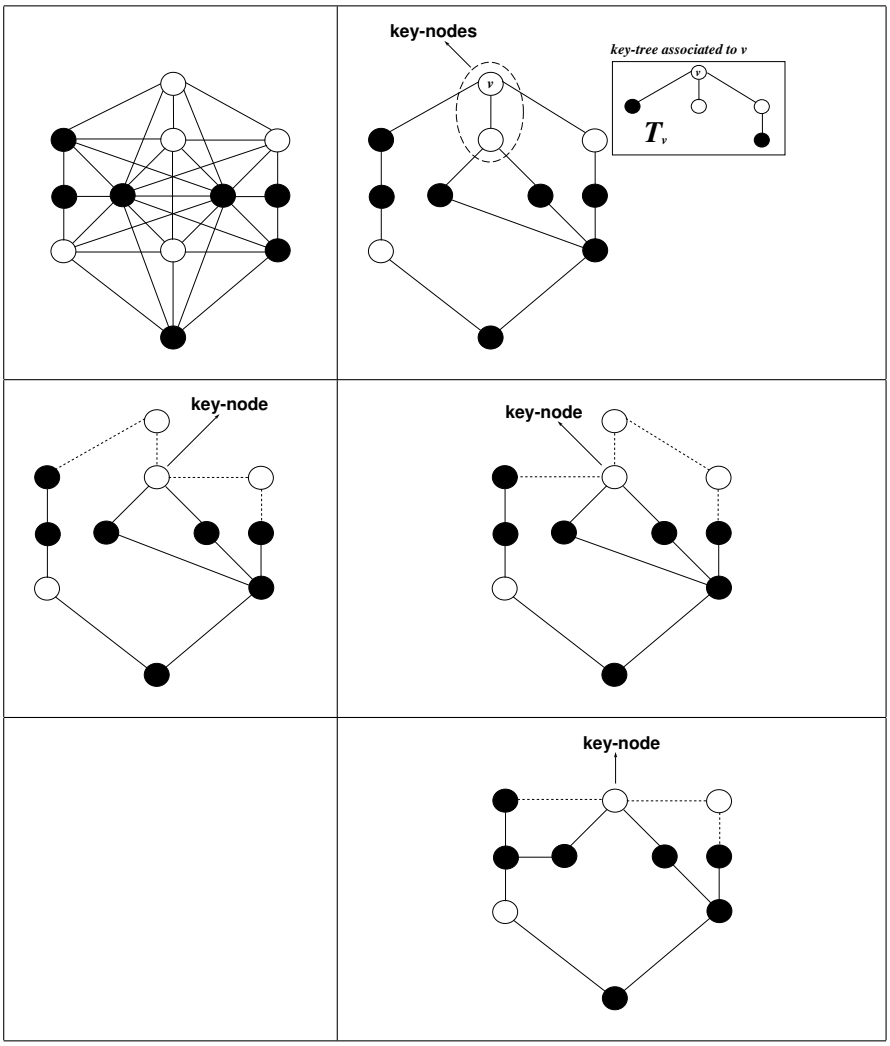



Figure 5.23: A feasible solution and its neighbor solutions built by replacing $\mathcal{T}_v$.

In this example all possible neighbor solutions obtained by applying $\mathrm{RecConnect}$ are minimal, but it could happen that the network resulting after the key-tree replacement loses the minimality. In addition, as in the fifth graph, a new neighbor solution can have a key-node less than before. We introduce the following proposition to prove formally these properties.

**Proposition 5.4.8** *A neighbor feasible solution constructed by* $\mathrm{RecConnect}$ *can be non-minimal and moreover it can have a key-node less than the original solution.*

**Proof.** Let us consider the networks shown in Figure 5.24. The black nodes model fixed switch sites and

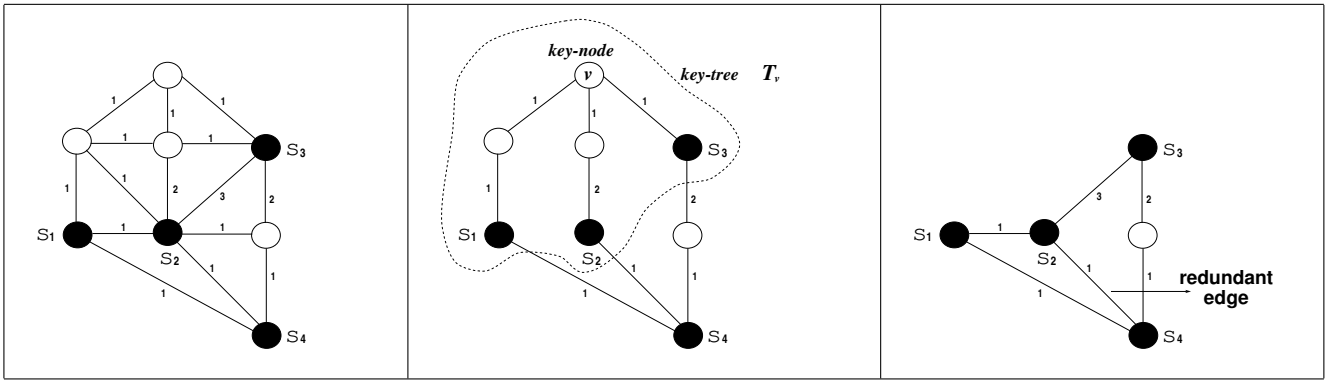



Figure 5.24: Networks: $G_B$, $\mathcal{G}_{sol}$ and $\hat{\mathcal{G}}_{sol}$ respectively.

the white nodes model non-fixed switch sites. The graphs from left to right are: the graph $G_B$ of feasible connections on the backbone network, the current solution $\mathcal{G}_{sol}$, and the graph $\hat{\mathcal{G}}_{sol}$ delivered by $\mathrm{RecConnect}$ when running with these inputs. Network $\hat{\mathcal{G}}_{sol}$ was built of the following way. Firstly, according to the $\mathrm{RecConnect}$ pseudo-code, it easy to see that $Z = \{s_1, s_2, s_3\}$. When executing lines 7-15, the pairs $(s_1, s_2)$ and $(s_2, s_3)$ were selected (in that order), obtaining as result the feasible solution $\hat{\mathcal{G}}_{sol}$. Clearly, $\hat{\mathcal{G}}_{sol}$ is non-minimal since by deleting the edge $(s_2, s_4) \in \hat{\mathcal{G}}_{sol}$ the resulting graph is 2-node-survivable. In addition, in $\hat{\mathcal{G}}_{sol}$ there are no key-nodes, completing thus the proof.

**QED**

In the example introduced in the previous proposition a redundant edge appears when adding connections between the endpoints of $\mathcal{T}_v$. The sites $s_2$ and $s_4$ are adjacent having degree three in the new solution. As we saw above, by deleting the connection between both sites the 2-node-survivability is preserved. For other BNDP2NS instances this action could induce the loss of the feasibility depending on the solution topology. To prove it, we introduce the following proposition.

**Proposition 5.4.9** *Let $\mathcal{G}$ be a 2-node-connected network such that there exist two adjacent nodes $u, v \in \mathcal{G}$ with $degree(u) \geq 3$ and $degree(v) \geq 3$, and moreover when removing the edge $(u, v)$ from $\mathcal{G}$ at most one articulation node is introduced. Then, the network $\bar{\mathcal{G}} = \mathcal{G} \setminus \{(u, v)\}$ is 2-edge-connected but not necessarily 2-node-connected.*

**Proof.** In order to demonstrate that $\bar{\mathcal{G}} = \mathcal{G} \setminus \{(u, v)\}$ could be non 2-node-connected, we introduce in Figure 5.25 a 2-node-connected network so that by removing an edge between nodes with degree greater to

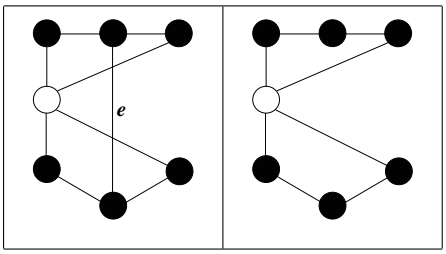



Figure 5.25: By deleting $e$ the network loses the 2-node-connectivity.

two the 2-node-connectivity is lost.
Now, we will prove that $\bar{\mathcal{G}}$ is 2-edge-connected. By the 2-node-connectivity, there exists a path $p \subset \mathcal{G}$ from $u$ to $v$ such that the edge $(u, v) \notin p$. Let $x_1, x_2 \in p$ be the adjacent sites to $u$ and $v$ respectively. Since

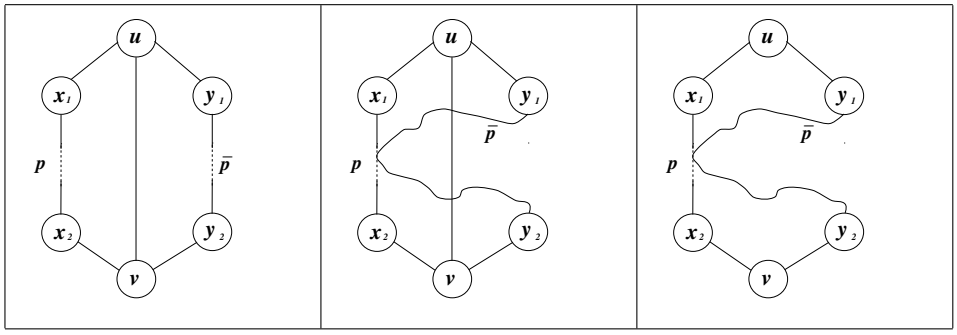



Figure 5.26: Cases: $p \cap \bar{p} = \emptyset$, $p \cap \bar{p} \neq \emptyset$ and subgraph $\bar{\mathcal{H}}$.

$degree(u) \geq 3$ and $degree(v) \geq 3$ there exist two sites $y_1, y_2 \in \mathcal{G}$, $y_1 \neq x_1, x_2$, $y_2 \neq x_1, x_2$ adjacent to $u$ and $v$ respectively. Again, by the 2-node-connectivity there exists a path $\bar{p} \subset \mathcal{G}$ connecting $y_1$ and $y_2$ such that $(u, v) \notin \bar{p}$. We have the following cases: $p \cap \bar{p} = \emptyset$ or $p \cap \bar{p} \neq \emptyset$, Figure 5.26 illustrates these situations. Let us define the subgraphs $\bar{\mathcal{H}} = p \cup \bar{p} \cup \{(u, y_1), (v, y_2)\}$ and $\mathcal{H} = \bar{\mathcal{H}} \cup \{(u, v)\}$. It easy to see that $\bar{\mathcal{H}}$ is 2-edge-connected; therefore by replacing $\mathcal{H}$ by $\bar{\mathcal{H}}$ in $\mathcal{G}$ and applying Lemma 5.3.2, we have that $\bar{\mathcal{G}}$ is also 2-edge-connected.

**QED**

Proposition 5.4.9 is also valid when $\mathcal{G}$ is 2-node-connected, $u$ and $v$ have degree greater than three and in addition they are connected by means of a key-path. In both cases, these results are very useful to eliminate redundant edges and key-paths from a solution, as long as the feasibility is preserved. In particular, given a 2-node-survivable feasible solution we can remove key-paths and edges whose endpoints have degree greater to three and to apply suitably the well-known DFS algorithm (Depth First Search) to determine the presence of articulation nodes [73]. That is, under these hypothesis, we can iteratively eliminate edges

and key-paths and simultaneously control the feasibility of the resulting solution, obtaining thus a feasible solution with smaller cost.

The following proposition demonstrates the feasibility preservation in each $\mathrm{LocalSearch2\_2NS}$ iteration.

**Proposition 5.4.10** *If* $\mathrm{LocalSearch2\_2NS}$ *receives as input a 2-node-survivable feasible solution* $\mathcal{G}_{sol}$*, the local search preserves the feasibility at any time.*

**Proof.** By contradiction, for certain iteration $\mathcal{G}_{sol}$ is 2-node-survivable with set of key-nodes $X$ and there exists $u \in X$ such that $\mathrm{RecConnect}$ delivers a non-feasible solution. This contradicts Proposition 5.4.7. Hence, the algorithm preserves the feasibility on each iteration.

**QED**

## 5.5 The GRASP algorithms for the BNDP2NS

We now describe the general GRASP algorithm for approximately solving the BNDP2NS. Figure 5.27 shows the corresponding pseudo-code. The algorithm $\mathrm{GRASP\_BNDP2NS}$ has two generic procedures which can be instanced by different designed algorithms for the construction phase and local search phase. More precisely, the procedures $\mathrm{Construction\_Phase}$ and $\mathrm{Local\_Search}$ can be instantiated of the following way:

- $\mathrm{Construction\_Phase}$: by $\mathrm{ConstPhase1\_2NS}$, $\mathrm{ConstPhase2\_2NS}$, or $\mathrm{ConstPhase3\_2NS}$.
- $\mathrm{Local\_Search}$: by $\mathrm{LocalSearch1\_2NS}$ or $\mathrm{LocalSearch1}$ (used by $\mathrm{GRASP\_BNDP}$).

Let us notice that, given a feasible solution $\mathcal{G}_{sol}$ the algorithm $\mathrm{LocalSearch1\_2NS}$ replaces each key-path from $\mathcal{G}_{sol}$ exactly one time, whereas the algorithm $\mathrm{LocalSearch1}$ resumes the key-path replacement process whenever it finds a better neighbor feasible solution when replacing a certain key-path by other key-path. Therefore, we can see $\mathrm{LocalSearch1}$ as a generalization of $\mathrm{LocalSearch1\_2NS}$. Each of these algorithms are used in combined way with the algorithm $\mathrm{LocalSearch2\_2NS}$ which works based on key-tree replacement moves.

The local search phase applies first key-path replacement moves (by running $\mathrm{LocalSearch1\_2NS}$ or $\mathrm{LocalSearch1}$ for key-paths replacements) and the evaluation of key-tree replacement moves is performed only if there are no improving key-path replacement moves. In this way, we can explore structurally different neighborhoods in combined form and the search is resumed from the beginning whenever we find a

better neighbor feasible solution. The local search phase finalizes once no better neighbors are found when exploring the neighborhoods.

In order to present the different versions of the GRASP for solving the BNDP2NS, we will reference by $\mathrm{Construction\_Phase}$ indifferently to the algorithms $\mathrm{ConstPhase1\_2NS}$, $\mathrm{ConstPhase2\_2NS}$ and $\mathrm{ConstPhase3\_2NS}$, and similarly by $\mathrm{Local\_Search}$ to the algorithms $\mathrm{LocalSearch1\_2NS}$ and $\mathrm{LocalSearch1}$. Next, we give a detailed description of the algorithm $\mathrm{GRASP\_BNDP2NS}$.

```
Procedure GRASP_BNDP2NS;
Input: G_B, C, R, k, seed, MaxIter;

1   min_cost ← ∞;
2   for i = 1, ..., MaxIter do
3      [G_sol] ← Construction_Phase(G_B, C, k);
4      cost_sol ← COST(G_sol);
5      G_sol ← Local_Search(G_B, C, G_sol);
6      best ← COST(G_sol);
7      if (best < cost_sol) then goto line 4;
8      G_sol ← LocalSearch2_2NS(G_B, C, G_sol);
9      best ← COST(G_sol);
10     if (best < cost_sol) then goto line 4;
11     if (cost_sol < min_cost) then
12        G^(opt) ← G_sol; min_cost ← cost_sol;
13     end_if;
14  end_for;
15  return G^(opt);
end GRASP_BNDP2NS;
```

Figure 5.27: General Version of the algorithm $\mathrm{GRASP\_BNDP2NS}$.

The algorithm takes as inputs the graph $G_B$ of feasible connections on the backbone network, the matrix of connection costs $C$, the GRASP parameters $k$ (used in the construction phase), a seed for the pseudo random number generator $seed$ and the number of iterations $MaxIter$ to be performed. The cost of the best found feasible solution is initialized with the value infinity ($\infty$) in line 1. The algorithm is repeated $MaxIter$ times exploring the space of feasible solutions and searching for the optimal feasible solution for the BNDP2NS. Each iteration works in the following way.

In line 3, a greedy randomized feasible solution $\mathcal{G}_{sol}$ is built using the algorithm $\mathrm{Construction\_Phase}$ (i.e. $\mathrm{ConstPhase1\_2NS}$, $\mathrm{ConstPhase2\_2NS}$ or $\mathrm{ConstPhase3\_2NS}$). In line 4 the cost of $\mathcal{G}_{sol}$ is assigned to variable $cost\_sol$. In line 5 we call $\mathrm{Local\_Search}$ (i.e. $\mathrm{LocalSearch1\_2NS}$ or $\mathrm{LocalSearch1}$) in order to find neighbor feasible solutions with smaller cost. In any case the local search algorithm builds better neighbor feasible solutions by means of key-path replacement moves. Line 6 computes the cost of the neighbor solution $\mathcal{G}_{sol}$ built in line 5. Line 7 compares the cost of the current solution with the one returned by $\mathrm{Local\_Search}$. If a neighbor solution with smaller cost has been computed by $\mathrm{Local\_Search}$, then, the local

search resumes from this new current solution executing from line 4. Otherwise, if no neighbor solution of better cost is found by $\mathrm{Local\_Search}$, then, in line 8 we call the algorithm $\mathrm{LocalSearch2\_2NS}$, which searches for neighbor solutions with smaller cost by applying key-tree replacement moves. As mentioned previously, the $\mathrm{LocalSearch2\_2NS}$ computes neighbor solutions by replacing key-trees by other trees which are not necessarily key-trees. Line 9 computes the cost of the solution delivered in line 8. Again, if a neighbor feasible solution with smaller cost has been found by $\mathrm{LocalSearch2\_2NS}$, then the local search resumes from this new current solution executing from line 4. Otherwise, if no neighbor solution with better cost is found by $\mathrm{LocalSearch2\_2NS}$, then, if the solution found at the end of the local search phase is better than the best solution so far (line 11), we update in line 12 the best found feasible solution and the minimum cost. Let us remark that the local search phase is conformed by the key-path replacement moves as well as the key-tree replacement moves; this corresponds to lines $5-10$ in the pseudo-code. Once finalized the loop 2-14, the best found feasible solution $\mathcal{G}^{(\mathrm{opt})}$ is returned in line 15.

## 5.6 Performance Tests

In this section we introduce the experimental results obtained when applying the different combinations of algorithms for the construction phase and the local search phase. All the algorithms were implemented in ANSI C. The experiments were obtained on a Pentium IV with 1.7 GHz, and 1 Gbytes of RAM, running under Windows XP. In the performance testing phase all instances were solved with the same GRASP parameter settings. In a previous tuning phase the candidate list size $k$ was chosen in the set $\{5, 10, 15, 20, 30\}$ and the maximum number of iterations $MaxIter$ in the set $\{50, 100, 300, 500\}$. We tuned the value for the candidate list size by considering a reduced group of BNDP2NS instances. As result of the tuning phase, we selected $k = 20$ and $k = 30$ as the values with better results since they obtained at least the same solution costs that the other parameter combinations, and better ones in many cases. Thus, we fixed $k = 20$ and $MaxIter = 300$ when running all the performance testing problems. Next, we will describe the BNDP2NS instances used in the testing phase.

### 5.6.1 BNDP2NS test-set description

As far as we known, there is no benchmark library neither for the STNSNP nor for the STESNP. Usually, in other works related to the BNDP2NS the authors generate random graphs to perform the computational testing. For instance in [6], cases from the Travelling Salesman Problem (TSP) extracted from the TSPLIB library [115] are used (without adding them Steiner nodes) with the aim of studying the efficiency of a polyhedral algorithm specially designed for the STESNP.

Let us notice that, if we use TSP instances as BNDP2NS instances, the feasible solutions of the TSP

will also be feasible in the BNDP2NS, but the optimal solutions of the TSP will not necessarily be optimal for the BNDP2NS. A particularity of these cases is that the optional nodes presence is not considered to reduce the network designing costs, what considerably restricts the performance analysis of our GRASP algorithms. In particular, we would not be able to study feasible solutions containing key-trees and key-nodes as topological components. For these reasons, we focus on the generation of test cases based on TSP instances, containing besides non-fixed switch sites. In what follows, we will indifferently refer to Steiner nodes as non-fixed switch sites, owing to the analogy between the BNDP2NS and the STNSNP problem cited in [6, 102, 126].

We generated a test-set using Traveling Salesman Problem (TSP) instances extracted from the well-known TSPLIB library [115] and customized (by adding to them certain amount of Steiner nodes) to our BNDP2NS. The TSPLIB contains many problem classes related to the TSP. In particular, we are interested only in the Symmetric Traveling Salesman Problems (denoted simply by TSP) since the other classes including in the TSPLIB are not applicable or directly customizable to our BNDP2NS. Below, we describe the transformation process used to customize the TSP instances to our BNDP2NS.

Given a TSP instance from the TSPLIB and a non-negative integer $m$ (the number of Steiner nodes to be added), we generate a BNDP2NS instance as follows:

i) We will denote by $\mathcal{G}$ and $\hat{\mathcal{G}}$ the graphs associated with the original TSP instance and its corresponding BNDP2NS instance (to be constructed) respectively.

ii) All the nodes of $\mathcal{G}$ will be considered fixed switch sites in $\hat{\mathcal{G}}$. In this way, initially we have: $\hat{\mathcal{G}} = \mathcal{G}$.

iii) We distinguish the following cases:

   a) If $\mathcal{G}$ is an implicit Euclidian graph (i.e. in the input TSP file each node has associated a position with respect to a system of geographic coordinates in $\mathbb{R}^2$), then, iteratively we added $m$ Steiner nodes (one node at a time) by selecting randomly their positions in the range to which belongs the nodes of $\mathcal{G}$. The resulting BNDP2NS instance is modelling by a complete graph $\hat{\mathcal{G}}$ having all the original nodes of the TSP instance and the $m$ added Steiner nodes. The connection costs are given by the Euclidian distances.

   b) If $\mathcal{G}$ is an explicit Euclidian graph (i.e. the input TSP file provides explicitly the matrix of geographic distances between the nodes), then, iteratively we added to $\hat{\mathcal{G}}$ $m$ Steiner nodes (one node at a time) so that at each iteration we generate a connection between the new Steiner node and each node already present in $\hat{\mathcal{G}}$. These connections have associated costs which are randomly chosen in the interval $\varrho \cdot [c_{min}, c_{max}]$, where $c_{min}$ and $c_{max}$ are the minimum and maximum euclidian distances between two nodes in $\mathcal{G}$, and $\varrho \in (0, 1)$ is a prefixed parameter.

Clearly, the final topology will be a complete graph but their costs not necessarily satisfy the triangular inequality.

Once finalized this process, the resulting graph $\hat{\mathcal{G}}$ models a new BNDP2NS instance.

The TSP instances used to build BNDP2NS instances were: att48, berlin52, brazil58, dantzing42, eil51, eil76, gr48, gr96, hk48, kroA100, kroB100, kroC100, kroD100, kroE100, pr76, rat99, rd100, st70, and swiss42. The numbers that appear in their names indicate the number of nodes of the problem. Table 5.1 indicates for each of these TSP instances the main characteristics of the BNDP2NS instances generated by applying the process exposed above. The first column contains the names of the original TSP instances and the entries from left to right are:

- the format type of the input TSP file depending on if this one contains or not in explicit form the matrix of geographic distances between the nodes (EXP or IMP denoting Explicit or Implicit form respectively),
- the number of Steiner nodes added to the original graph (we used the values: $per$=25%, 45%, and 65% percent of the number of nodes in the TSP instance),
- the total number of nodes in the resulting BNDP2NS instance (TNODES),
- the type of graph associated with the generated BNDP2NS instances (EUC or G denoting an euclidian graph or a general graph respectively),
- and the number of generated instances (NI).

Most of the BNDP2NS instances generated by us, satisfy the triangular inequality between its nodes (45 out of 57 instances). This property is particularly important since, as we will see, we will be able to compute (by means of the application of theoretical results present in the literature) lower and upper bonds for the optimal costs of such instances; and in this way bound the relative distance of GRASP solution costs to the optimal costs. Obviously, low values with respect to the lower bounds will involve good quality sub-optimal solutions or reaching the optimality. Lower bounds will be useful as long as they be relatively close to the optimal values (tight lower bounds), otherwise, relatively high gaps could be obtained and, despite this, being very close to the optimal global cost or achieving the optimality.

On the other hand, based on four explicit format of TSP instances, we generated twelve BNDP2NS instances with Steiner nodes whose connections to other nodes not necessarily satisfy the triangular inequality among costs. More precisely, for the cases where the input TSP file is given in explicit format, when adding new connections we choose $\varrho = 1/6$ with the aim of generating Steiner nodes with connection costs towards its adjacent nodes much lower when comparing them to the connection costs between fixed switch

| TSP problem | Input TSP file | $per$ | TNODES | Graphs | NI |
|---|---|---|---|---|---|
| att48 | IMP | 25%, 45%, 65% | 60, 69, 79 | EUC | 3 |
| berlin52 | IMP | 25%, 45%, 65% | 65, 75, 85 | EUC | 3 |
| brazil58 | EXP | 25%, 45%, 65% | 72, 84, 95 | G | 3 |
| dantzing42 | EXP/IMP | 25%, 45%, 65% | 52, 60, 69 | EUC | 3 |
| eil51 | IMP | 25%, 45%, 65% | 63, 73, 84 | EUC | 3 |
| eil76 | IMP | 25%, 45%, 65% | 95, 110, 125 | EUC | 3 |
| gr48 | EXP | 25%, 45%, 65% | 60, 69, 79 | G | 3 |
| gr96 | IMP | 25%, 45%, 65% | 120, 139, 158 | EUC | 3 |
| hk48 | EXP | 25%, 45%, 65% | 60, 69, 79 | G | 3 |
| kroA100 | IMP | 25%, 45%, 65% | 125, 145, 165 | EUC | 3 |
| kroB100 | IMP | 25%, 45%, 65% | 125, 145, 165 | EUC | 3 |
| kroC100 | IMP | 25%, 45%, 65% | 125, 145, 165 | EUC | 3 |
| kroD100 | IMP | 25%, 45%, 65% | 125, 145, 165 | EUC | 3 |
| kroE100 | IMP | 25%, 45%, 65% | 125, 145, 165 | EUC | 3 |
| pr76 | IMP | 25%, 45%, 65% | 95, 110, 125 | EUC | 3 |
| rat99 | IMP | 25%, 45%, 65% | 123, 143, 163 | EUC | 3 |
| rd100 | IMP | 25%, 45%, 65% | 125, 145, 165 | EUC | 3 |
| st70 | IMP | 25%, 45%, 65% | 87, 101, 115 | EUC | 3 |
| swiss42 | EXP | 25%, 45%, 65% | 52, 60, 69 | G | 3 |

Table 5.1: Test-set for the BNDP2NS.

sites. In this way, when integrating new optional nodes, we will increase the chances of these of being potential improvers of 2-node-survivable feasible solutions. Intuitively, the lower the value of $\varrho$, the larger the probability that the global optimal solutions of a generated BNDP2NS instance has Steiner nodes in its topologies.

Moreover, if we observe the design of our local search algorithms, we will be able to notice that these are strongly related to the analysis of feasible solutions which have Steiner nodes as network components (for example key-nodes, or Steiner nodes belonging to key-paths or to key-trees). As a result of this we are interested in having test instances available with considerably large amounts of Steiner nodes (more than $29\%$ out of the totality of fixed switch nodes), and therefore in analyzing our algorithms performance and studying the impact on the optimal cost reduction when we increasingly add more Steiner nodes to the TSP instance taken as basis.

### 5.6.2 Auxiliary topological properties

Before introducing the results obtained in the testing phase, we will give two theorems which can be used in order to compute a lower bound for the optimum cost of an Euclidian BNDP2NS instance. This lower bound depends on the optimum value of the original TSP instance.

Let us place in the following context. Consider a set of nodes $V$ with a nonnegative, symmetric *distance function* (or *metric*) $d(\cdot)$ defined on $V \times V$ which satisfies the triangle inequality. Let us call $d(u, v)$ the *cost*

or *length* of the edge $(u, v)$. A subset of edges $U \subseteq V \times V$ defines a graph $H = (V, U)$ whose cost is given by $d(U) = \sum_{(u,v)\in U} d(u, v)$. Given a subset of special nodes $D \subseteq V$, we let $C_{opt}(D)$ denote an optimal cycle spanning $D$ (without using nodes of $V \setminus D$) and $TC_{opt}(D)$ denote an optimal 2-node-connected solution spanning $D$ (without using nodes of $V \setminus D$) of cost $d(C_{opt}(D))$ and $d(TC_{opt}(D))$ respectively. In addition, let us denote an optimal Steiner 2-node-connected solution spanning $D$ by $STC_{opt}(D, V)$ with cost $d(STC_{opt}(D, V))$ (in this case the nodes of $V \setminus D$ may be used if they help reduce the overall cost). In this context, Monma, Munson and Pulleyblank [102] proved that a minimum-cost traveling salesman tour may be a good approximation to a minimum-cost 2-node-connected graph, more exactly they establish that:

**Theorem 5.6.1 (Monma, Munson and Pulleyblank)** *For any set of nodes $V$, $D \subseteq V$ and a distance function $d(\cdot)$,*

$$\frac{3}{4}d(C_{opt}(D)) \leq d(TC_{opt}(D)) \leq d(C_{opt}(D)).$$

Furthermore, they provide [102] a relation between an optimal 2-node-connected solution spanning $D$ and a Steiner 2-node-connected solution spanning $D$.

**Theorem 5.6.2 (Monma, Munson and Pulleyblank)** *For any set of nodes $V$, $D \subseteq V$ and a distance function $d(\cdot)$,*

$$\frac{3}{4}d(TC_{opt}(D)) \leq d(STC_{opt}(D, V)) \leq d(TC_{opt}(D)).$$

In this way, by combining both results, we have the relation:

$$\frac{9}{16}d(C_{opt}(D)) \leq d(STC_{opt}(D, V)) \leq d(C_{opt}(D)),$$

obtaining thus a lower bound and an upper bound for the $d(STC_{opt}(D, V))$, which only depend on $d(C_{opt}(D))$. Let us note that, since in most cases our BNDP2NS instances are Euclidian graphs, these bounds are useful in practice to approximate the gap between the solution found by the algorithm GRASP_BNDP2NS and the corresponding optimal solution. In the same context, Monma, Munson and Pulleyblank proved [102] an important structural theorem related to optimal two-node-connected solutions, whose wording is the following.

**Theorem 5.6.3 (Monma, Munson and Pulleyblank)** *For any set of nodes $V$ with distance function $d(\cdot)$ on $V \times V$, there exists a minimum-weight two-connected network $H = (V, U)$ satisfying the following conditions:*

*a) every node of $H$ has degree 2 or 3.*

*b) deleting any edge or pair of edges in $H$ leaves a bridge in one of the resulting connected components of $H$.*

Later, we will use this theorem to analyze comparatively certain solutions delivered by our GRASP heuristics.

### 5.6.3 Numerical Results

Let us turn now to the study of the computational results.
Table 5.2 shows for each original TSP instance:

- the optimum value of the original TSP instance (denoted by $\mathrm{COPT\_TSP}$),
- the value $\mathrm{LB1} = \frac{3}{4}\mathrm{COPT\_TSP}$ which is a lower bound for the cost of a feasible solution which does not have Steiner nodes,
- the value $\mathrm{LB2} = \frac{9}{16}\mathrm{COPT\_TSP}$ which is a lower bound for the optimal BNDP2NS solution. It is easy to see that if all the optimal solutions of a BNDP2NS instance have Steiner nodes, the value $\mathrm{LB1}$ cannot be used as lower bound for the solutions delivered by the algorithm $\mathrm{GRASP\_BNDP2NS}$. In this sense the value $\mathrm{LB2}$ provides us a lower bound when the best solution found by $\mathrm{GRASP\_BNDP2NS}$ has Steiner nodes.

| **TSP problem** | COPT_TSP | LB1 | LB2 |
|---|---|---|---|
| att48 | 10628 | 7971 | 5978.25 |
| berlin52 | 7542 | 5656.5 | 4242.4 |
| brazil58 | 25395 | - | - |
| dantzing42 | 699 | 524.25 | 393.18 |
| eil51 | 426 | 319.5 | 239.62 |
| eil76 | 538 | 403.5 | 302.62 |
| gr48 | 5046 | - | - |
| gr96 | 55209 | 41406.75 | 31055.06 |
| hk48 | 11461 | - | - |
| kroA100 | 21282 | 15961.5 | 11971.12 |
| kroB100 | 22141 | 16605.75 | 12454.31 |
| kroC100 | 20749 | 15561.75 | 11696.62 |
| kroD100 | 21294 | 15970.5 | 11671.31 |
| kroE100 | 22068 | 16551 | 12413.25 |
| pr76 | 108159 | 81119.25 | 60839.43 |
| rat99 | 1211 | 908.25 | 681.18 |
| rd100 | 7910 | 5932.5 | 4449.37 |
| st70 | 675 | 506.25 | 379.68 |
| swiss42 | 1273 | - | - |

Table 5.2: Comparative values for the generated BNDP2NS instances.

Let us notice that even though $\mathrm{LB2}$ was deduced by the combination of inequalities derived from the worst case ratios inherent to structurally different feasible topologies, we cannot a-priori know the approximation degree which provides us such bound with respect to the optimal values of the BNDP2NS. In relation to this latter, if we analyze the bounding interval present in theorem 5.6.1 and the bounding interval

resultant of relating theorems 5.6.1 and 5.6.2, we can at once infer that this last one is $1.75$ times higher than the first one (that is $1.75 = \frac{1-9/16}{1-3/4}$).

In a first stage, we tested all the combinations of construction phase algorithms with local search search algorithms. Since in this testing phase the experimental results obtained when using the local search $\mathrm{LocalSearch1}$ surpassed qualitatively or at least obtained solutions of the same quality than the ones found when using the local search $\mathrm{LocalSearch1\_2NS}$ (probably, this could be explained by the fact that the local search $\mathrm{LocalSearch1}$ may be seen as a generalization of $\mathrm{LocalSearch1\_2NS}$ owing to the way in which the key-paths present in the solution to improve are replaced), from here on we will summarize the computational results obtained by the GRASP algorithms that used to $\mathrm{LocalSearch1}$ in the local search phase. In order to present the results, we will introduce the following notation.

**Heuristic $\mathcal{H}_1$:** $\mathrm{GRASP\_BNDP2NS}$ instantiated with $\mathrm{ConstPhase1\_2NS}$, $\mathrm{LocalSearch1}$ and $\mathrm{LocalSearch2\_2NS}$,

**Heuristic $\mathcal{H}_2$:** $\mathrm{GRASP\_BNDP2NS}$ instantiated with $\mathrm{ConstPhase2\_2NS}$, $\mathrm{LocalSearch1}$ and $\mathrm{LocalSearch2\_2NS}$,

**Heuristic $\mathcal{H}_3$:** $\mathrm{GRASP\_BNDP2NS}$ instantiated with $\mathrm{ConstPhase3\_2NS}$, $\mathrm{LocalSearch1}$ and $\mathrm{LocalSearch2\_2NS}$,

In Tables 5.3 to 5.5 we show a summary of computational results obtained by applying the algorithm $\mathrm{GRASP\_BNDP2NS}$ to the test-set presented in Table 5.1. More precisely, they correspond to the performance tests of the heuristics $\mathcal{H}_1$, $\mathcal{H}_2$ and $\mathcal{H}_3$.

In each table, the first column contains the names of the original TSP instances and the entries from left to right are:

- an indicator if the best solution found by our heuristic has Steiner nodes ($\mathrm{SNI}$),
- an indicator if the best solution found by our heuristic has cycle topology ($\mathrm{CTI}$),
- the cost of the best solution found by $\mathrm{GRASP\_BNDP2NS}$ (denoted by $\mathrm{BCF}$),
- the value $\mathrm{GAP\_1}$, where: $\mathrm{GAP\_1} \stackrel{\mathrm{def}}{=} 100 \times \frac{(\mathrm{BCF}-\mathrm{LB1})}{\mathrm{LB1}}$. This is the gap with respect to the lower bound provided by theorem 5.6.1 inherent to optimal 2-node-connected solutions (without using Steiner nodes),
- the value $\mathrm{GAP\_2}$, where: $\mathrm{GAP\_2} \stackrel{\mathrm{def}}{=} 100 \times \frac{(\mathrm{BCF}-\mathrm{LB2})}{\mathrm{LB2}}$. This is the gap with respect to the lower bound introduced by combining theorems 5.6.1 and 5.6.2, inherent to optimal Steiner 2-node-connected solutions. Let us notice that if $\mathrm{SNI} = \mathrm{FALSE}$, the value $\mathrm{LB1}$ is a lower bound for the solution delivered by the GRASP algorithm.
- the value $\mathrm{UB\_GAP} \stackrel{\mathrm{def}}{=} 100 \times \frac{|\mathrm{BCF}-\mathrm{COPT\_TSP}|}{\mathrm{COPT\_TSP}}$; this is the gap with respect to the optimum TSP value,

- the average of the improvement of the results of the local search phase over the construction phase (LSI),
- the running time per iteration (secs./itr).

For the instances derived from cases brazil58, gr48, hk48 and swiss42, we did not compute the values of GAP_1 and GAP_2 because their costs could not satisfy the triangular inequality and therefore we cannot apply the theorems 5.6.1 and 5.6.2. Table 5.6 shows certain topological characteristics of the best solutions found by the heuristics $\mathcal{H}_1$, $\mathcal{H}_2$ and $\mathcal{H}_3$; specifically, for each BNDP2NS instance we provide:

- the number of Steiner nodes of the best solution found (denoted by NS),
- the number of edges in the best solution found (Edges).

In addition, in Table 5.7, we introduce only the best cost found for each one of the generated instances as well as the heuristics that attained these values.

In what follows, we will discuss the computational results obtained by the GRASP heuristics. In Tables 5.3, 5.4, and 5.5 the costs corresponding to the best feasible solutions found by the heuristics $\mathcal{H}_1$, $\mathcal{H}_2$ and $\mathcal{H}_3$ are in bold letters. For each instance we indicate which of the three heuristics produced the lowest cost solution. Let us notice in many cases the same cost was reached by more than one heuristic.

The heuristic $\mathcal{H}_3$ was the one which found in a larger number of BNDP2NS instances the best feasible solutions, followed in order by $\mathcal{H}_2$ and $\mathcal{H}_1$. Specifically, out of 57 instances, when comparing the costs of the solutions returned by the three heuristics we have:

- $\mathcal{H}_3$ found 50 best solutions, being 41 of them not equalled in cost by the other heuristics and from the eight remaining, six were also reached by $\mathcal{H}_2$ and five by $\mathcal{H}_1$.
- $\mathcal{H}_2$ found 8 best solutions, being all of them equalled by $\mathcal{H}_1$ and/or $\mathcal{H}_3$. To be more precisely, four of them were equalled in cost by $\mathcal{H}_1$ and seven of them were equalled in cost by $\mathcal{H}_3$.
- $\mathcal{H}_1$ found 12 best solutions, 6 of them not equalled by the other heuristics. Of the remaining, four of them were equalled in quality by $\mathcal{H}_2$ and other five by $\mathcal{H}_3$.

As can be seen in Tables 5.3, 5.4, and 5.5, in no case the feasible topologies returned by the GRASP heuristics were cycles. In addition to this, in most cases the best solutions found had at least a Steiner node as part of its topology, excepting four instances for $\mathcal{H}_1$, three instances for $\mathcal{H}_2$ and an instance for $\mathcal{H}_3$ .We noticed that interestingly, two of these feasible solutions without Steiner nodes (corresponding to Euclidian instances), fulfilled points $(a)$ and $(b)$ from theorem 5.6.3. Another important point to emphasize is that the best GRASP solutions found were minimal (i.e. by removing an edge the feasibility is lost).

| **TSP problem** | SNI | CTI | BCF | GAP_1 | GAP_2 | UB_GAP | LSI | secs./itr |
|---|---|---|---|---|---|---|---|---|
| att48 | × ✓✓ | × × × | 9491, 8696, 6828 | 19.07%, 9.10%, -14.34% | 58.76%, 45.46%, 14.21% | 10.70%, 18.18%, 35.75% | 3.39%, 3.12%, 6.82% | 3.95, 4.92, 6.23 |
| berlin52 | ✓✓✓ | × × × | 6759, 6161, 4875 | 19.49%, 8.92%, -13.82% | 59.32%, 45.23%, 14.91% | 10.38%, 18.31%, 35.36% | 3.12%, 4.04%, 4.36% | 4.18, 5.05, 6.40 |
| brazil58 | ✓✓✓ | × × × | 22360, 20660, 16065 | - | - | 11.95%, 18.65%, 36.74% | 4.35%, 4.62%, 6.29% | 4.55, 5.42, 6.53 |
| dantzing42 | ✓✓✓ | × × × | 649, 586, **473** | 23.80%, 11.78%, -9.78% | 65.06%, 49.04%, 20.30% | 7.15%, 16.17%, 32.33% | 5.01%, 5.56%, 7.33% | 3.73, 4.36, 5.51 |
| eil51 | ✓✓✓ | × × × | **387**, 397, 357 | 21.13%, 16.08%, 4.39% | 61.50%, 54.78%, 39.18% | 9.15%, 12.94%, 21.71% | 5.69%, 6.96%, 7.68% | 3.97, 4.78, 5.95 |
| eil76 | ✓✓✓ | × × × | **470**, 448, **382** | 16.48%, 11.03%, -5.33% | 55.31%, 48.04%, 26.23% | 12.64%, 16.73%, 29.0% | 7.24%, 7.85%, 9.02% | 5.18, 6.51, 8.17 |
| gr48 | × ✓✓ | × × × | 4450, 4107, 3269 | - | - | 11.81%, 18.61%, 35.2% | 6.56%, 8.52%, 7.25% | 3.62, 4.27, 5.11 |
| gr96 | ✓✓✓ | × × × | 48314, 44858, 34974 | 16.68%, 8.33%, -15.54% | 55.58%, 44.45%, 12.62% | 12.49%, 18.75%, 36.65% | 4.12%, 4.56%, 6.12% | 6.30, 7.53, 9.28 |
| hk48 | × ✓✓ | × × × | 10182, 9346, 7291 | - | - | 11.16%, 18.45%, 36.38% | 5.22%, 6.83%, 7.96% | 3.81, 4.89, 6.06 |
| kroA100 | ✓✓✓ | × × × | 18746, 17295, 13488 | 17.45%, 8.35%, -15.50% | 56.59%, 44.47%, 12.67% | 11.92%, 18.73%, 35.42% | 4.52%, 5.13%, 7.23% | 6.95, 8.33, 9.78 |
| kroB100 | ✓✓✓ | × × × | 19393, 18003, 14324 | 16.78%, 8.41%, -13.74% | 55.71%, 44.55%, 15.01% | 12.41%, 18.69%, 34.31% | 4.23%, 6.93%, 6.01% | 7.11, 8.18, 9.66 |
| kroC100 | ✓✓✓ | × × × | 18347, 16938, 12717 | 17.90%, 8.84%, -18.28% | 57.20%, 45.13%, 8.96% | 11.58%, 18.37%, 38.71% | 3.97%, 5.02%, 7.85% | 6.65, 7.80, 9.36 |
| kroD100 | ✓✓✓ | × × × | 20214, 17347, 15454 | 26.57%, 8.62%, -3.23% | 68.76%, 44.83%, 29.02% | 5.07%, 18.54%, 27.43% | 4.10%, 4.07%, 8.23% | 7.23, 8.88, 10.52 |
| kroE100 | ✓✓✓ | × × × | **19470**, 17936, 17002 | 17.64%, 8.37%, 2.72% | 56.85%, 44.49%, 36.97% | 11.77%, 18.72%, 22.96% | 4.08%, 5.86%, 9.71% | 6.82, 8.41, 9.94 |
| pr76 | ✓✓✓ | × × × | 97413, **94472**, 68443 | 20.09%, 16.46%, -15.63% | 60.11%, 55.28%, 12.50% | 9.94%, 12.65%, 36.72% | 5.63%, 7.23%, 6.94% | 5.20, 6.14, 7.20 |
| rat99 | ✓✓✓ | × × × | 1235, **1173**, **782** | 35.98%, 29.15%, -13.90% | 81.30%, 72.20%, 14.80% | 1.98%, 3.14%, 35.43% | 2.92%, 6.14%, 7.50% | 6.99, 8.08, 9.79 |
| rd100 | ✓✓✓ | × × × | **7450**, 7019, 6224 | 27.10%, 18.31%, 4.91% | 69.46%, 57.75%, 39.88% | 4.68%, 11.26%, 21.31% | 4.07%, 5.96%, 8.01% | 7.18, 8.58, 10.12 |
| st70 | ✓✓✓ | × × × | 663, **637**, **520** | 30.96%, 25.83%, 2.72% | 74.62%, 67.77%, 36.95% | 1.78%, 5.63%, 22.96% | 2.23%, 4.79%, 7.23% | 5.41, 6.75, 8.21 |
| swiss42 | ✓✓✓ | × × × | 1254, 1237, **864** | - | - | 1.49%, 2.83%, 32.13% | 2.10%, 5.65%, 9.34% | 3.93, 4.63, 5.75 |
| Average | | | | 21.71%, 13.72%, -7.80% | 62.27%, 51.62%, 22.93% | 9.01%, 14.70%, 31.75% | 4.34%, 5.72%, 7.41% | 5.41, 6.50, 7.87 |

Table 5.3: Results with ConstPhase1_2NS, LocalSearch1 and LocalSearch2_2NS (heuristic $\mathcal{H}_1$).

| **TSP problem** | SNI | CTI | BCF | GAP_1 | GAP_2 | UB_GAP | LSI | secs./itr |
|---|---|---|---|---|---|---|---|---|
| att48 | ✓✓✓ | × × × | 9314, 8420, 6671 | 16.85%, 5.63%, -16.31% | 55.80%, 40.84%, 11.59% | 12.36%, 20,78%, 37,23% | 4.02%, 4.82%, 7.85% | 5.01, 7.12, 7.95 |
| berlin52 | ✓✓✓ | × × × | 6483, 6042, 4747 | 14.61%, 6.82%, -16.08% | 52.82%, 42.42%, 11.89% | 14.04%, 19.89%, 37.06% | 3.85%, 4.61%, 8.23% | 5.94, 7.49, 8.44 |
| brazil58 | ✓✓✓ | × × × | 21847, 20120, 17396 | - | - | 13.97%, 20.77%, 31.50% | 4.64%, 5.85%, 6.85% | 6.30, 7.74, 8.75 |
| dantzing42 | ✓✓✓ | × × × | **638**, 571, 532 | 21.70%, 8.92%, 1.48% | 62.26%, 45.22%, 35.30% | 8.73%, 18.31%, 23.89% | 5.02%, 5.23%, 8.63% | 5.64, 6.50, 7.08 |
| eil51 | ✓✓✓ | × × × | 394, 412, **319** | 23.32%, 28.95%, -0.16% | 64.42%, 71.94%, 33.12% | 7.51%, 3.29%, 25.12% | 5.20%, 3.28%, 7.32% | 5.75, 6.64, 7.53 |
| eil76 | ✓✓✓ | × × × | 508, 452, 412 | 25.90%, 12.02%, 2.11% | 67.86%, 49.36%, 36.14% | 5.58%, 15.99%, 23.42% | 5.17%, 6.52%, 8.92% | 8.08, 9.87, 11.18 |
| gr48 | ✓✓✓ | × × × | 4352, 3953, 3200 | - | - | 13.75%, 21.66%, 36.58% | 4.58%, 7.85%, 9.23% | 5.40, 7.02, 8.22 |
| gr96 | ✓✓✓ | × × × | 48490, 43742, 34325 | 17.11%, 5.64%, -17.10% | 56.14%, 40.85%, 10.53% | 12.17%, 20.77%, 37.83% | 6.69%, 7.12%, 9.12% | 8.89, 10.88, 11.92 |
| hk48 | ✓✓✓ | × × × | 10200, 9165, 7586 | - | - | 11.00%, 20.03%, 33.81% | 2.72%, 4.17%, 6.30% | 5.41, 7.12, 7.48 |
| kroA100 | ✓✓✓ | × × × | 18403, 16957, 14186 | 15.30%, 6.24%, -11.12% | 53.73%, 41.65%, 18.50% | 13.53%, 20.32%, 33.34% | 5.16%, 5.06%, 6.67% | 9.13, 11.32, 12.45 |
| kroB100 | ✓✓✓ | × × × | 20042, **17829**, 14232 | 20.69%, 7.37%, -14.29% | 60.92%, 43.16%, 14.27% | 9.48%, 19.48%, 35.72% | 5.12%, 4.53%, 5.87% | 9.32, 11.35, 12.95 |
| kroC100 | ✓✓✓ | × × × | 17912, 16535, 12318 | 14.85%, 6.02%, -21.02% | 53.14%, 41.37%, 5.31% | 13.86%, 20.48%, 40.76% | 6.35%, 5.21%, 4.86% | 9.48, 10.61, 12.22 |
| kroD100 | ✓✓✓ | × × × | 19753, 16902, **15002** | 23.68%, 5.83%, -6.06% | 64.91%, 41.11%, 25.25% | 7.24%, 20.63%, 29.55% | 4.04%, 5.19%, 7.12% | 9.70, 11.56, 12.00 |
| kroE100 | ✓✓✓ | × × × | **19470**, 17469, 16527 | 17.64%, 5.55%, -0.15% | 56.85%, 40.73%, 33.14% | 11.77%, 20.84%, 25.11% | 4.89%, 4.63%, 5.60% | 9.29, 11.42, 12.60 |
| pr76 | × ✓✓ | × × × | 95322, **94472**, 66938 | 17.51%, 16.46%, -17.48% | 56.68%, 55.28%, 10.02% | 11.87%, 12.65%, 38.11% | 2.56%, 3.92%, 3.75% | 8.20, 9.78, 10.35 |
| rat99 | ✓✓✓ | × × × | 1195, 1302, 867 | 31.57%, 43.35%, -4.54% | 75.43%, 91.14%, 27.28% | 1.32%, 7.51%, 28.41% | 6.31%, 4.87%, 5.06% | 8.88, 10.82, 12.09 |
| rd100 | ✓✓✓ | × × × | **7450**, 6862, 6054 | 25.58%, 15.67%, 2.05% | 67.44%, 54.22%, 36.06% | 5.82%, 13.25%, 23.46% | 4.16%, 5.67%, 5.13% | 8.95, 11.41, 12.47 |
| st70 | ✓✓✓ | × × × | 645, **637**, 534 | 27.41%, 25.83%, 5.48% | 69.88%, 67.77%, 40.64% | 4.44%, 5.63%, 20.89% | 3.96%, 4.07%, 6.23% | 7.42, 9.21, 10.02 |
| swiss42 | × ✓✓ | × × × | 1241, 1114, 891 | - | - | 2.51%, 12.49%, 30.01% | 5.16%, 6.12%, 7.97% | 4.58, 6.22, 7.00 |
| Average | | | | 20.93%, 13.37%, -7.54% | 61.24%, 51.16%, 23.29% | 9.51%, 16.56%, 31.14% | 4.71%, 5.19%, 6.87% | 7.44, 9.16, 10.14 |

Table 5.4: Results with ConstPhase2_2NS, LocalSearch1 and LocalSearch2_2NS (heuristic $\mathcal{H}_2$).

| **TSP problem** | SNI | CTI | BCF | GAP_1 | GAP_2 | UB_GAP | LSI | secs./itr |
|---|---|---|---|---|---|---|---|---|
| att48 | × ✓✓ | × × × | **9297, 8205, 6424** | 16.64%, 2.94%, -19.41% | 55.51%, 37.25%, 7.46% | 12.52%, 22.80%, 39.56% | 3.29%, 5.28%, 6.17% | 4.63, 6.08, 7.56 |
| berlin52 | ✓✓✓ | × × × | **6087, 5863, 4523** | 7.61%, 3.65%, -20.04% | 43.48%, 38.20%, 6.61% | 19.29%, 22.26%, 40.03% | 4.23%, 5.20%, 6.12% | 5.06, 6.54, 8.02 |
| brazil58 | ✓✓✓ | × × × | **20463, 19592, 15871** | - | - | 19.42%, 22.85%, 37.50% | 3.62%, 4.34%, 5.45% | 5.54, 7.12, 8.33 |
| dantzing42 | ✓✓✓ | × × × | **638, 557**, 507 | 21.70%, 6.25%, -3.29% | 62.26%, 41.66%, 28.95% | 8.73%, 20.31%, 27.47% | 4.29%, 4.49%, 6.47% | 4.61, 5.57, 6.72 |
| eil51 | ✓✓✓ | × × × | **387, 384, 319** | 21.13%, 20.19%, -0.16% | 61.50%, 60.25%, 33.12% | 9.15%, 9.86%, 25.12% | 4.32%, 3.40%, 7.01% | 4.93, 5.92, 7.15 |
| eil76 | ✓✓✓ | × × × | 472, **441**, 387 | 16.98%, 9.29%, -4.09% | 55.97%, 45.72%, 27.88% | 12.27%, 18.03%, 28.07% | 3.98%, 5.22%, 6.48% | 7.06, 8.85, 10.65 |
| gr48 | ✓✓✓ | × × × | **4280, 3880, 3121** | - | - | 15.18%, 23.11%, 38.15% | 3.91%, 6.96%, 7.39% | 4.62, 6.25, 7.81 |
| gr96 | ✓✓✓ | × × × | **46939, 42587, 32473** | 13.36%, 2.85%, -21.58% | 51.15%, 37.13%, 4.57% | 14.98%, 22.86%, 41.18% | 4.94%, 6.33%, 7.75% | 8.01, 9.76, 11.35 |
| hk48 | ✓✓✓ | × × × | **9982, 8892, 7328** | - | - | 12.90%, 22.42%, 36.06% | 3.20%, 4.05%, 4.02% | 4.88, 6.06, 7.11 |
| kroA100 | ✓✓✓ | × × × | **18214, 16845, 13117** | 14.11%, 5.54%, -17.82% | 52.15%, 40.71%, 9.57% | 14.42%, 20.85%, 38.37% | 4.33%, 5.57%, 5.08% | 8.52, 10.26, 11.85 |
| kroB100 | ✓✓✓ | × × × | **18343, 17829, 13467** | 10.46%, 7.37%, -18.90% | 47.28%, 43.16%, 8.13% | 17.15%, 19.48%, 39.18% | 4.37%, 5.95%, 4.21% | 8.50, 10.28, 11.56 |
| kroC100 | ✓✓✓ | × × × | **16767, 15712, 12211** | 7.74%, 0.97%, -21.53% | 43.66%, 34.62%, 4.62% | 19.19%, 24.28%, 41.15% | 3.86%, 4.69%, 4.01% | 8.24, 9.86, 11.63 |
| kroD100 | ✓✓✓ | × × × | **19514, 15886, 15002** | 22.19%, -0.53%, -6.06% | 62.92%, 32.63%, 25.25% | 8.36%, 25.40%, 29.55% | 3.67%, 6.85%, 6.83% | 8.42, 10.08, 11.42 |
| kroE100 | ✓✓✓ | × × × | **19470, 17019, 14245** | 17.64%, 2.83%, -13.93% | 56.85%, 37.10%, 14.76% | 11.77%, 22.88%, 35.45% | 4.43%, 4.77%, 4.95% | 8.01, 10.28, 12.00 |
| pr76 | ✓✓✓ | × × × | **88237, 94472, 63806** | 8.77%, 16.46%, -21.34% | 45.03%, 55.28%, 4.88% | 18.42%, 12.65%, 41.01% | 2.75%, 4.49%, 5.23% | 7.00, 8.47, 9.85 |
| rat99 | ✓✓✓ | × × × | **1071, 1173**, 818 | 17.92%, 29.15%, -9.94% | 57.23%, 72.20%, 20.08% | 11.56%, 3.14%, 32.45% | 4.64%, 5.36%, 4.44% | 7.80, 9.31, 11.52 |
| rd100 | ✓✓✓ | × × × | 7472, **6448, 5806** | 25.95%, 8.69%, -2.13% | 67.93%, 44.92%, 30.49% | 5.54%, 18.48%, 26.60% | 3.38%, 6.17%, 5.65% | 8.12, 10.02, 11.87 |
| st70 | ✓✓✓ | × × × | **593, 637**, 542 | 17.14%, 25.83%, 7.06% | 56.18%, 67.77%, 42.75% | 12.15%, 5.63%, 19.70% | 2.93%, 5.01%, 7.37% | 6.52, 8.16, 9.53 |
| swiss42 | ✓✓✓ | × × × | **1116, 932**, 870 | - | - | 12.33%, 26.79%, 31.66% | 4.07%, 5.07%, 7.02% | 4.12, 5.21, 6.65 |
| | | Average | | 15.96%, 9.43%, -11.54% | 54.61%, 45.91%, 17.94% | 13.44%, 19.16%, 34.12% | 3.91%, 5.22%, 5.88% | 6.56, 8.11, 9.61 |

Table 5.5: Results with ConstPhase3_2NS, LocalSearch1 and LocalSearch2_2NS (heuristic $\mathcal{H}_3$).

| | Solutions of the $\mathcal{H}_1$ heuristic | | Solutions of the $\mathcal{H}_2$ heuristic | | Solutions of the $\mathcal{H}_3$ heuristic | |
|---|---|---|---|---|---|---|
| **TSP problem** | NS | Edges | NS | Edges | NS | Edges |
| att48 | 0, 2, 5 | 51, 54, 58 | 4, 3, 4 | 55, 56, 57 | **4**, **4**, **5** | **55**, **57**, **59** |
| berlin52 | 3, 3, 6 | 58, 59, 62 | 4, 3, 5 | 59, 58, 61 | **5**, **4**, **5** | **61**, **60**, **60** |
| brazil58 | 4, 7, 9 | 65, 69, 72 | 4, 6, 8 | 65, 69, 72 | **3**, **4**, **7** | **64**, **66**, **69** |
| dantzing42 | 3, 4, **6** | 49, 50, **53** | **3**, 4, 5 | **48**, 52, 54 | **3**, **4**, 5 | **48**, **49**, 51 |
| eil51 | **5**, 4, 8 | **59**, 59, 65 | 2, 3, **4** | 55, 57, **59** | **5**, **3**, **5** | **59**, **58**, **61** |
| eil76 | **6**, 6, **5** | **86**, 87, **84** | 5, 4, 5 | 84, 83, 85 | 6, **6**, 7 | 86, **88**, 87 |
| gr48 | 0, 3, 7 | 51, 55, 59 | 4, 3, 5 | 55, 54, 57 | **4**, **4**, **5** | **58**, **57**, **58** |
| gr96 | 4, 5, 5 | 105, 106, 104 | 4, 5, 5 | 103, 105, 105 | **4**, **6**, **5** | **105**, **108**, **106** |
| hk48 | 0, 5, 9 | 51, 56, 62 | 3, 4, 6 | 54, 55, 58 | **3**, **6**, **6** | **56**, **59**, **62** |
| kroA100 | 4, 7, 11 | 108, 111, 117 | 5, 6, 8 | 111, 111, 114 | **4**, **5**, **7** | **111**, **113**, **116** |
| kroB100 | 4, 5, 10 | 108, 110, 116 | 5, **6**, 5 | 109, **113**, 112 | **5**, **6**, **9** | **113**, **113**, **116** |
| kroC100 | 3, 6, 11 | 107, 110, 118 | 5, 4, 7 | 110, 109, 113 | **4**, **6**, **8** | **112**, **111**, **114** |
| kroD100 | 2, 6, 8 | 106, 109, 112 | 5, 6, 8 | 111, 112, 114 | **5**, **6**, **8** | **111**, **112**, **114** |
| kroE100 | **4**, 8, 9 | **109**, 112, 114 | **4**, 7, 7 | **109**, 112, 113 | **4**, **7**, **9** | **109**, **112**, **119** |
| pr76 | 4, **6**, 7 | 84, **85**, 88 | 3, **6**, 5 | 86, **85**, 85 | **4**, **6**, **5** | **88**, **85**, **89** |
| rat99 | 3, **5**, **8** | 106, **107**, **112** | 4, 4, 6 | 108, 107, 110 | **6**, **5**, 6 | **109**, **107**, 111 |
| rd100 | **5**, 5, 10 | **110**, 109, 117 | **5**, 5, 7 | **110**, 113, 115 | 5, **6**, **7** | 110, **109**, **114** |
| st70 | 2, **4**, **6** | 76, **79**, **83** | 2, **4**, 5 | 76, **79**, 79 | **3**, **4**, 6 | **79**, **79**, 79 |
| swiss42 | 1, 4, **9** | 45, 49, **57** | 1, 3, 7 | 47, 51, 55 | **2**, **6**, 6 | **47**, **51**, 54 |
| Amount of best solutions | 12 (6 of them not overcome) | | 8 (also all computed by $\mathcal{H}_1$ and/or $\mathcal{H}_3$) | | 50 (41 of them not overcome) | |

Table 5.6: Features of the solutions found by the GRASP algorithms.

When analyzing the topological structure of the best GRASP solutions for the 57 instances (the best comparing the three heuristics), we noticed that the best solutions corresponding to the 45 Eucledian instances did not have a Steiner node of degree 2 as network component. This is particularly important since in those instances where the triangular inequality among costs is satisfied, the existence of degree 2 Steiner nodes in a feasible solution implies that this is not globally optimal. More precisely, for the Eucledian instances, we noticed that all the Steiner nodes of the best solutions were of degree three. With this, for the BNDP2NS Eucledian instances, our best GRASP solutions satisfied this necessary optimality condition [102]. According to this, from Table 5.6, we can state that the amount of key-nodes (and therefore key-trees) present on each of the 45 best GRASP solutions (indicated in bold letters) corresponding to the Eucledian instances, goes from 3 up to 9 Steiner nodes of degree higher than 2. With respect to the non-Eucledian BNDP2NS instances, their best solutions had in several cases key-paths with Steiner nodes of degree 2, what do not discard its potential optimality.

Let us analyze now the values of the gaps $\mathrm{GAP\_1}$ and $\mathrm{GAP\_2}$. Notice that for the three heuristics (as Tables 5.3, 5.4, and 5.5 show), given a TSP instance, when increasing the number of Steiner nodes present in the generated BNDP2NS instance, the relative distance between the best feasible solution cost and the value $\frac{3}{4}\mathrm{COPT\_TSP}$ significantly diminishes until the point in which (when we add the $65\%$ of Steiner

| **TSP problem** | COPT_TSP | $per = 25\%$ | Heuristics | $per = 45\%$ | Heuristics | $per = 65\%$ | Heuristics |
|---|---|---|---|---|---|---|---|
| att48 | 10628 | 9297 | $\mathcal{H}_3$ | 8205 | $\mathcal{H}_3$ | 6424 | $\mathcal{H}_3$ |
| berlin52 | 7542 | 6087 | $\mathcal{H}_3$ | 5863 | $\mathcal{H}_3$ | 4523 | $\mathcal{H}_3$ |
| brazil58 | 25395 | 20463 | $\mathcal{H}_3$ | 19592 | $\mathcal{H}_3$ | 15871 | $\mathcal{H}_3$ |
| dantzing42 | 699 | 638 | $\mathcal{H}_2$ $\mathcal{H}_3$ | 557 | $\mathcal{H}_3$ | 473 | $\mathcal{H}_1$ |
| eil51 | 426 | 387 | $\mathcal{H}_1$ $\mathcal{H}_3$ | 384 | $\mathcal{H}_3$ | 319 | $\mathcal{H}_2$ $\mathcal{H}_3$ |
| eil76 | 538 | 470 | $\mathcal{H}_1$ | 441 | $\mathcal{H}_3$ | 382 | $\mathcal{H}_1$ |
| gr48 | 5046 | 4280 | $\mathcal{H}_3$ | 3880 | $\mathcal{H}_3$ | 3121 | $\mathcal{H}_3$ |
| gr96 | 55209 | 46939 | $\mathcal{H}_3$ | 42587 | $\mathcal{H}_3$ | 32473 | $\mathcal{H}_3$ |
| hk48 | 11461 | 9982 | $\mathcal{H}_3$ | 8892 | $\mathcal{H}_3$ | 7328 | $\mathcal{H}_3$ |
| kroA100 | 21282 | 18214 | $\mathcal{H}_3$ | 16845 | $\mathcal{H}_3$ | 13117 | $\mathcal{H}_3$ |
| kroB100 | 22141 | 18343 | $\mathcal{H}_3$ | 17829 | $\mathcal{H}_2$ $\mathcal{H}_3$ | 13467 | $\mathcal{H}_3$ |
| kroC100 | 20749 | 16767 | $\mathcal{H}_3$ | 15712 | $\mathcal{H}_3$ | 12211 | $\mathcal{H}_3$ |
| kroD100 | 21294 | 19514 | $\mathcal{H}_3$ | 15886 | $\mathcal{H}_3$ | 15002 | $\mathcal{H}_2$ $\mathcal{H}_3$ |
| kroE100 | 22068 | 19470 | $\mathcal{H}_1$ $\mathcal{H}_2$ $\mathcal{H}_3$ | 17019 | $\mathcal{H}_3$ | 14245 | $\mathcal{H}_3$ |
| pr76 | 108159 | 88237 | $\mathcal{H}_3$ | 94472 | $\mathcal{H}_1$ $\mathcal{H}_2$ | 63806 | $\mathcal{H}_3$ |
| rat99 | 1211 | 1071 | $\mathcal{H}_3$ | 1173 | $\mathcal{H}_1$ $\mathcal{H}_3$ | 782 | $\mathcal{H}_1$ |
| rd100 | 7910 | 7450 | $\mathcal{H}_1$ $\mathcal{H}_2$ | 6448 | $\mathcal{H}_3$ | 5806 | $\mathcal{H}_3$ |
| st70 | 675 | 593 | $\mathcal{H}_3$ | 637 | $\mathcal{H}_1$ $\mathcal{H}_2$ $\mathcal{H}_3$ | 542 | $\mathcal{H}_1$ |
| swiss42 | 1273 | 1116 | $\mathcal{H}_3$ | 932 | $\mathcal{H}_3$ | 864 | $\mathcal{H}_1$ |

Table 5.7: Best costs found for each BNDP2NS instance.

nodes) the value of the best GRASP solution found is under the value of $\mathrm{LB1}$, except for a few cases. This is a completely expected result since when increasing the amount of optional nodes, the space of feasible solutions is expanded, and the probability of finding Steiner 2-node-survivable feasible solutions whose costs improve the cost of the best 2-node-connected solution (which does not use Steiner nodes) grows. Let us observe that in the three heuristics, the average values of $\mathrm{GAP\_1}$ corroborate these tendency as well. Computing the average (over the heuristics) of the value of $\mathrm{GAP\_1}$ with respect to the instances generated with $45\%$ of Steiner nodes, we obtain $10.17\%$ of average relative distance between the GRASP solution found and the lower bound supplied for the best feasible solution not containing Steiner nodes ($\mathrm{LB1}$).

In accordance with the mentioned above, when analyzing the values of $\mathrm{GAP\_2}$, we notice that in most cases as the number of Steiner nodes is increased in a generated BNDP2NS instance, the relative distance between the best solution found and the lower bound given by $\frac{9}{16}\mathrm{COPT\_TSP}$ considerably diminishes, what also was supposed to happen since the sub-space of feasible solutions that contain Steiner nodes exponentially expands, and consequently the possibility of finding every time better feasible solutions grows.

Note that the $\mathrm{LB2} = \frac{9}{16}\mathrm{UB\_GAP}$, this means that the gap between $\mathrm{LB2}$ and $\mathrm{UB\_GAP}$ is of the order of $78\%$ therefore a value of $\mathrm{GAP\_2}$ which is relatively high does not necessarily implies a great distance with respect to the global optimal value. This is to say, eventually, the case in which the cost of our best feasible solution is very close (or is the same) to the optimal and at the same time is relatively far of the lower bound, might happen.

Considering the BNDP2NS Eucledian instances generated by adding a $65\%$ of Steiner nodes, when averaging the values of $\mathrm{GAP\_2}$ over the three heuristics, we obtain an average relative distance with respect to the lower bound $\mathrm{LB2}$ equal to $21.38\%$. Nevertheless, considering only the best GRASP solutions found and calculating the average of $\mathrm{GAP\_2}$ for these solutions, we have an average relative distance with respect to the lower bound $\mathrm{LB2}$ equal to $12.73\%$, which is not necessarily a "bad value" since we do not know how tight the inferred bound is. Practical evidence suggests that testing applied to other BNDP2NS instances with grater number of Steiner nodes would lead to smaller gaps.

Let us analyze now the values of $\mathrm{UB\_GAP}$. As expected, when increasing the number of Steiner nodes added to the TSP original instances, almost in every case, the values of $\mathrm{UB\_GAP}$ were higher (the only three exceptions happened when applying the heuristic $\mathcal{H}_3$ on the instances generated from the problems pr76, rat99 and st70, in particular when passing from the $25\%$ to the $45\%$ of Steiner nodes to be added). The reason for this is that in presence of more Steiner nodes, the space of BNDP2NS feasible solutions significantly grows. In particular, there exists more feasible solutions having Steiner nodes as topological components. Then the probability of finding any Steiner 2-node-survivable solution beating the optimal TSP solution grows. Let us notice that, for the three heuristics, the average values of $\mathrm{UB\_GAP}$ corroborate the mentioned observation.

Let us see now the behaviour of the values of $\mathrm{LSI}$. In every case, the value of $\mathrm{LSI}$ was higher than

$2\%$ and lower than $10\%$. For the three GRASP heuristics, in most of cases, as we increased the number of Steiner nodes added to the original TSP instance, the values of $\mathrm{LSI}$ were higher (there existed some exceptions for the three heuristics which can be seen in Tables 5.3, 5.4, and 5.5). Let us recall that the design of our strategies of local search is strongly linked to the presence of Steiner nodes as components of the starting feasible solution, and therefore its performance will be influenced in a certain way by the amount of Steiner nodes existing in the solution delivered by the construction phase. When analyzing each of the BNDP2NS instances generated, as we compare them we find that there did not exist any relevant differences in the values of $\mathrm{LSI}$ corresponding to the three heuristics (these differences are always lower than $4$ percentage points and in $93\%$ of cases lower than $2.5$ percentage points). The average values of $\mathrm{LSI}$ for each of the heuristics were higher than $3.5\%$, $5\%$ y $5.5\%$ when $per = 25\%$, $per = 45\%$ and $per = 65\%$ respectively. Besides, they were comparatively very similar, being at less than $1$ percentage point of distance between them.

Finally, when analyzing the execution times of every one of the heuristics, we conclude that on average the fastest per iteration was the heuristic $\mathcal{H}_1$ followed in order by $\mathcal{H}_3$ and $\mathcal{H}_2$.

Making a global balance of the obtained results, we have that even when the heuristic $\mathcal{H}_3$ was the one which achieved the best solution in a higher number of instances, it was beat by $\mathcal{H}_1$ in six occasions. Furthermore, over the 57 instances, $\mathcal{H}_3$ achieved the $87.7\%$ of the best solutions found, while $\mathcal{H}_1$ achieved the $21.1\%$ of the best solutions found. Often, in the literature, in these kinds of situations (where an algorithm beats the other and inversely) it is said that both algorithms are incomparable. On the other hand, the execution times of $\mathcal{H}_1$ were lower than the ones of $\mathcal{H}_3$ in every case, having an average difference of $1.5$ seconds by GRASP iteration and in percentage $\mathcal{H}_1$ was $18.17\%$ faster than $\mathcal{H}_3$. The heuristic $\mathcal{H}_2$ achieved the $14.03\%$ percent over the total of best solutions found, however, the topologies found in these cases were equalled in quality (i.e. in cost) either $\mathcal{H}_1$ or $\mathcal{H}_3$, but they were not beaten by $\mathcal{H}_1$ or by $\mathcal{H}_3$ if we compare separately $\mathcal{H}_2$ with $\mathcal{H}_1$ and $\mathcal{H}_2$ with $\mathcal{H}_3$. When comparing the average execution times, $\mathcal{H}_2$ had $2.32$ more seconds per iteration compared to $\mathcal{H}_1$ and $0.82$ more seconds per iteration compared to $\mathcal{H}_3$. In percentage, $\mathcal{H}_1$ and $\mathcal{H}_3$ were $26.03\%$ and $9.2\%$ faster than $\mathcal{H}_2$ respectively.

Let us notice that the number of Steiner nodes added to a BNDP2NS instance has a great impact on the cost of the feasible solutions found. We see that a large number of Steiner nodes lead to smaller costs, this can be explained in part by the fact that the local search algorithms designed take great advantage of the presence of Steiner nodes as potential enhancers of the current solution.

## 5.7 Conclusions

We investigated the design of 2-node-survivable backbone network topologies from the heuristic point of view, with the aim of developing GRASP algorithms to solve problems which arise in practice. Such problem is a particular case of the BNDP introduced in Chapter 4, and we denominate it BNDP2NS. Our main motivation for studying this problem comes from the great applicability of the 2-node-connected models in real problems of design of High Speed Optical Data Transmission Networks (HSODTN) cores. We were able to develop several GRASP algorithms which can give a good quality 2-node-survivable solution. We designed three construction phase algorithms and two local search algorithms, and by combining them suitably with one of the local search algorithm used in the BNDP, we yielded six GRASP heuristics for the BNDP2NS. The implementation of the algorithms was tested on a number of different problems with heterogeneous characteristics. In particular, we built a set of 57 BNDP2NS instances by transforming 19 TSP instances (extracted from TSPLIB). By means of the application of certain theoretical results introduced in [102], we deduce lower bounds for 45 of the BNDP2NS instances generated.

Considering the best solutions found for the 57 instances (over the set of better solutions achieved by $\mathcal{H}_1$, $\mathcal{H}_2$, and $\mathcal{H}_3$), we notice that:

- For the instances generated with $per = 25\%$, the best solutions found were of better quality than the feasible solution of the TSP. In particular, the relative average gap with respect to the TSP optimal values was of $-13.47\%$, what indicates a significant improvement compared with the optimal TSP solutions. Considering only the BNDP2NS Eucledian instances, the average value of $\mathrm{GAP\_1}$ was of $10.55\%$, which measures the relative distance to the lower bound of the best feasible solution that does not contain Steiner nodes.

- In the same way, for the instances generated with $per = 45\%$ and $per = 65\%$ the relative average gaps with respect to the TSP optimal values were of $-19.16\%$ and $-34.60\%$ respectively. Furthermore, in all these cases the cost of the feasible solution found was considerably lower than the cost of the TSP optimal solution. The average values of $\mathrm{GAP\_1}$ were of $7.45\%$ and $-9.73\%$ for the values $per = 45\%$ and $per = 65\%$ respectively. With $per = 45\%$, for the corresponding instance for the case kroD100, we achieved a solution with lower cost than the value of LB1, and with $per = 65\%$ almost all the solutions had costs sensibly lower than this bound (less than $-17\%$ gap in 7 over 15 instances).

- When analyzing the non-Eucledian instances, even though we did not have lower bounds, we observe that the quality (i.e. the cost) of the achieved solutions was greatly superior than the TSP optimal solutions. In particular, the relative distance between the best solutions found for these instances and the TSP optimal solutions was on average of $-24.9\%$ percent (and always lower than $-14.0\%$).

Even if we have lower bounds for the optima of the BNDP2NS Eucledian cases, the real interval $[\frac{9}{16}\mathrm{COPT\_TSP}, \mathrm{COPT\_TSP}]$ generated between the lower and upper bound for the optimal value is $1.75$ times more greater than the interval introduced by theorem 5.6.2, in a way, if the cost of our best solution is in that interval, we will not be able to estimate with more precision its location unless it be very close to the lower bound.

We noticed that, as expected, the execution times of the proposed algorithms are strongly dependant on the number of fixed sites and Steiner nodes (non-fixed sites).

To summarize, we think that the results obtained by means of the application of the GRASP meta-heuristic to solve the BNDP2NS are good since we obtained minimal feasible topologies of low cost if we compare it to the best TSP solutions and to the value of $\mathrm{LB1}$, reaching in many cases solutions which significantly beat the quality of the best 2-node-connected feasible solution that does not contain Steiner nodes.

As future work, it is possible to search for new methods which improve either the initial construction or the local search phases of the GRASP. Moreover, we are focusing on getting BNDP2NS instances with known optimal costs or more tight lower bounds in order to compare them to the solutions found by our algorithms.

# Chapter 6

# Conclusions

In this thesis we have studied the topological design of a WAN (Wide Area Network) considering only the construction costs, for instance the costs of digging trenches and placing a fiber cable into service [126]. The reason for this following approach is that construction costs have the largest share in the overall cost of a WAN planning and design stage. Let us point out that even a very small reduction in this cost may represent many million dollars of savings for, say, telephone companies.

We tackled the problem of designing a WAN by breaking it down into two inter-related sub-problems: the Access Network Design Problem (ANDP) and the Backbone Network Design Problem (BNDP).

We modeled the ANDP as a variant of the *Steiner Problem in Graphs* (SPG), and the BNDP on the basis of the *Generalized Steiner Problem with Node-Connectivity Constraints* (GSP-NC) [1, 126]. Furthermore, we studied the specific case of BNDP when there exist 2-node-survivability requirements between pairs of backbone fixed switch nodes (which we called BNDP2NS). BNDP2NS is equivalent to the *Steiner 2-node-survivable network problem* (STNSNP) [6, 102, 126]. The reason for focusing on BNDP2NS is that it can be widely applied to the design of high-speed optic fiber networks, where the network is usually required to remain in operation in case of a single link or node failure. Moreover, the topological features of the 2-node-connected networks enable us to design customized algorithms taking advantage of their structure.

ANDP, BNDP and BNDP2NS are all NP-Hard problems. This means that applying exact algorithms in order to solve them calls for prohibitive (that is to say, exponential) computational time even for small or medium-sized networks.

Hence, we studied the ANDP, BNDP and BNDP2NS problems heuristically, opting for the Greedy Randomized Adaptive Search Procedure (GRASP) for solving them. The first reason for this approach is that the GRASP methodology has proved both powerful and efficient in other combinatorial optimization problems [47]. The second one is that GRASP appears as very flexible and potentially adaptable to the specific problem to solve. It offers a general framework where the analyst can carefully tune the global scheme for the problem at hand. Saying this differently, GRASP naturally forces the user to take advantage

of the specificities of the problem, and this can be extremely efficient. Observe that we got good results for both families of problemas, for the ANDP and for the BNDP classes.

We now provide a summary of the experimental results obtained for each one of the problems referred to above.

For ANDP we designed two algorithms for the feasible solution construction phase and two algorithms for the local search phase, the two components of a GRASP procedure. The two construction algorithms work by connecting one terminal at a time to a partial solution; one of them selects randomly a terminal and chooses among the $k$-shortest paths to connect it, while the other chooses among the $k$ nearest terminals, and always uses the shortest path for the connection. Both local search algorithms use Steiner node insertion and deletion moves and Minimum Spanning Tree algorithms, but while one uses a traditional neighborhood, the other one is based on a Random Neural Network model (RNN) [58, 59], which makes it radically different from the usual local searches applied to similar problems. RNN models have also proved remarkably effective when applied to other NP-Hard optimization problems [61, 62, 65, 63]. The numerical experiments were done on a testing set containing 210 SPG instances extracted from the SteinLib repository and customized for ANDP. The optimal values of the SPG instances provided lower bounds for the optimal values of ANDP. The experimental results obtained for the four combinations were successful. While in many cases they reached the value of the lower bound, that is, optimality, in many others there were relatively small gaps with respect to the lower bounds. Although the results of all the GRASP variants were very close, the construction method which only uses pre-computed shortest paths and the RNN based local search obtained in average the best results. Considering that in the ANDP generation process all the connections with terminal nodes were eliminated, and further that ANDP's feasible solution space is more restrictive than that of SPG, the fact that we obtained small gaps in average shows the potential of the GRASP methodology for finding good-quality solutions.

As to BNDP, we designed an algorithm for the construction of feasible solutions and three neighborhood definitions, two based on substituting $k$-paths by $k$-paths or general paths respectively, and one based on substituting $k$-trees. We tried out two local phase algorithms, combining each of the $k$-paths neighborhoods with the $k$-trees neighborhood. As testing set for this part of the thesis we used instances extracted from specialized literature, instances generated constructively and with known optimums, and instances generated by transforming TSP problems taken from the TSPLIB repository (adding to them a certain number of Steiner nodes which model non fixed switch sites). The full testing set consisted of a total of 29 BNDP instances. Connectivity requirements varied widely according to each specific problem. The results obtained were extremely promising. The results show that the variant using the second local search obtains better results. This can be explained as the paths neighborhood includes the $k$-paths neighborhood, leading then to a more flexible search. With this variant, we achieved optimality in almost all instances with known optima (except for only one case where there was a gap of less than 0.7 per cent with respect to

the global optimum) and found good-quality minimal feasible solutions in those cases where we did not know the global optimum. The latter were compared to the optimal values of the TSP and with lower bounds provided for the minimum-cost 2-node-connected network spanning problem [102]. Again, our GRASP algorithms proved highly efficient in constructing minimal feasible solutions, taking advantage of the presence of Steiner nodes as potential enhancers of the solutions.

With regard to BNDP2NS we tested three feasible solution construction algorithms and three neighborhood definitions, which were combined like in BNDP to obtained two local search methods. One of the construction algorithms was similar to the one used for BNDP, the other two were designed using properties of the BNDP2NS (one uses a characterization of its minimal solutions, and the other employs substitutions of subgraphs preserving 2-node connectedness). As testing set for the performance test phase we generated BNDP2NS instances by transforming TSP problems extracted from the TSPLIB repository (adding different numbers of Steiner nodes). Specifically, for each TSP problem selected we generated three BNDP2NS problems by adding 25, 45, and 65 per cent of Steiner nodes. The complete test set consisted of a total of 57 BNDP2NS instances. The results showed that among the local search algorithms (based on using the key-paths and key-tree substitution neighborhoods), one produced much better results than the rest. All the experiments using the three construction algorithms with this local search had very good results; if we compare the construction algorithms by pairs, there are cases in which each one beats the other. In fact, the best solutions found were good-quality minimal topologies which in many cases outperformed significantly the optimal 2-node-connected topology spanning the set of fixed nodes without using Steiner nodes (this is inferred when attaining solutions with costs smaller than the lower bound for these ones). As is the case of the general BNDP problem, the local search algorithms of BNDP2NS take advantage of the presence of Steiner nodes as enhancers of the starting solution. We therefore observed that as the number of Steiner nodes added to the original TSP problem increased, the best feasible solutions found are significantly less costly.

This thesis work divides the WAN network topological design problem into two separate parts: backbone (Chapters 4 and 5) and access network (Chapter 3).

Future research, based on this line of work, could investigate means for combining adequately the proposed optimization methodologies. In this sense, we propose a possible scheme for designing the overall topological architecture of a WAN by means of the combined use of the algorithms designed for the ANDP and BNDP sub-problems. Figure 6.1 shows a pseudocode of the iterative algorithm proposed. The algorithm would work as follows. In Phase 1 a feasible solution for ANDP is constructed by applying one of the construction algorithms proposed (that is, no local search is applied here, just a feasible solution is built). The resulting access network induces a set of fixed switch sites on the backbone network. In Phase 2, assuming that those backbone sites connected with access sub-networks are fixed, we apply one of the GRASP algorithms proposed for BNDP -or BNDP2NS- optimizing the backbone topology as much as pos-

sible. In Phase 3, considering only the switch sites that integrates the backbone network delivered by Phase 2, we suitably apply one of the local search algorithms proposed for ANDP in order to re-optimize the overall access network. Once the main loop has been finalized, the best WAN topology found is returned.

**Procedure WAN_DESIGN;**

**0** **for** $i = 1$ **to** $MaxIter$ **do**
**1** $AccessNetwork \leftarrow$ ANDP_Construction**;**
**2** $BackboneNetwork \leftarrow$ GRASP_BNDP($AccessNetwork$)**;**
**3** $AccessNetwork \leftarrow$ ANDP_Improver($BackboneNetwork, AccessNetwork$)**;**
**4** **end_for;**
**5** **return** $AccessNetwork$**,** $BackboneNetwork$**;**

Figure 6.1: A model for designing a WAN topology.

Regarding the problem definition itself extensions can consider both topological restrictions to current problem definition and considering new variables.

New topological restrictions can be introduced in the problem, like maximum number of incident links at concentrator nodes. It is also worth minimizing the impact of failures on the access network, as there is no redundancy in its topology. It might be relevant limiting the depth of cascaded concentrators connected to a single switch or balancing the number of subscribers per switch.

Other analysis variables could be introduced in the model. State of-the-art imposes limitations in the length of links, both in the trunks (backbone) and in the access network. Bandwidth constraints should be considered mainly while designing the access network.

# Appendix A

# Equivalent formulations for ANDP

In this chapter we will demonstrate the equivalence between the general formulation of the access network design problem and the problem of designing the global access topology when modelling the backbone as a single fixed node. As introduced in Section 1.3, we use the following notation:

- $V = S_T \cup S_C \cup S_D$,
- $A = \{a_{ij}\}_{i,j \in S}$ is the matrix which gives for any pair of sites of $V$, the cost of laying a line between them. When the direct connection between $i$ and $j$ is not possible, we take $a_{ij} = \infty$,
- $U = \{(i,j); \forall i,j \in V$ such that $a_{ij} < \infty\}$, is the set of feasible connections between sites of $V$,
- $H = (V,U)$ is the graph of feasible connections.

**Definition A.0.1 (General Access Network Design Problem - GANDP)** *We define the General Access Network Design Problem $GANDP(V,U,A)$ as the problem of finding a subgraph $\mathcal{T} \subset H$ of minimum cost such that $\forall s_t \in S_T$ there exists a unique path from $s_t$ to some fixed switch site $s_w \in S_D$ and such that terminal sites can not be used as intermediate nodes (they must have degree 1 in the solution).*

The problem ANDP defined in Chapter 3 is derived from GANDP model by applying the following points:

A) The set of switch sites $S_D$ is modelled by a single fixed node $z$. The total set of nodes considered in the ANDP is $S = S_T \cup S_C \cup \{z\}$.

B) The set of edges $E$ used in the problem ANDP satisfies the following points:

   i) Given a site $s \in S_T \cup S_C$, there exists the edge $(s,z) \in E$ iff there exists an edge $(s,s_w) \in U$ such that $s_w \in S_D$. Moreover, we define:

$$C(s,z) = \min \left\{ a_{(s,s_w)} | s_w \in S_D \right\}.$$

ii) The feasible connections between pairs of sites of $S_T \cup S_C$ are the same in both problems and their costs are equal.

**Theorem A.0.2 (GANDP-ANDP relation)** *Given an instance $GANDP(V, U, A)$ and its respective instance $ANDP(S, E, C)$, the optimal solutions for both problems have the same cost.*

**Proof.** Let $\mathcal{H}_{opt}$ and $\mathcal{T}_{opt}$ be global optimal solutions for the GANDP and ANDP respectively. It is easy to see that the network $\mathcal{H}_{opt}$ has a forest topology where each tree that composes it has a unique switch site like root. Let $\hat{\mathcal{T}}$ be the resultant network of modelling the set of switch sites of $\mathcal{H}_{opt}$ as a single node. Clearly $\text{COST}(\hat{\mathcal{T}}) = \text{COST}(\mathcal{H}_{opt})$ and moreover $\hat{\mathcal{T}}$ is feasible for the ANDP. Hence, we have the inequality:

$$\text{COST}(\mathcal{H}_{opt}) = \text{COST}(\hat{\mathcal{T}}) \overset{\text{optimality of } \mathcal{T}_{opt}}{\uparrow \geq} \text{COST}(\mathcal{T}_{opt}).$$

On the other hand, let us consider a network $\hat{\mathcal{H}}$ so that $\forall (s, z) \in \mathcal{T}_{opt}$, $\hat{\mathcal{H}}$ includes an edge $(s, s_w)$ fulfilling:

$$(s, s_w) = \arg\min \left\{ a_{(s,v)} | \forall v \in S_D \right\},$$

and the other edges of $\hat{\mathcal{H}}$ are those edges of $\mathcal{T}_{opt}$ whose two endpoints belong to $S_T \cup S_C$. In this way, by construction, we have that the network $\hat{\mathcal{H}}$ is feasible for the GANDP and furthermore satisfies the inequality:

$$\text{COST}(\mathcal{T}_{opt}) \overset{\text{by def. of } C(\cdot)}{\uparrow =} \text{COST}(\hat{\mathcal{H}}) \overset{\text{optimality of } \mathcal{H}_{opt}}{\uparrow \geq} \text{COST}(\mathcal{H}_{opt}),$$

But this implies that $\text{COST}(\mathcal{T}_{opt}) = \text{COST}(\mathcal{H}_{opt})$, as required, and completing the proof.

**QED**

# Appendix B

# ANDP test cases

We include here the information on the 210 ANDP instances generated from SteinLib library by the method discussed in Chapter 3. Tables B.1 to B.7 show for each generated ANDP instance, the cost of the best feasible solution found and its respective gap with respect to the lower bound. In the first column we have the names of the original SPG instances and in the second column their topological characteristics (number of nodes, number of edges, and number of terminals). We remark that in most cases the best solutions were reached by more than two heuristics. The cases marked with "$^{*}$" indicate ANDP instances where the lower bound was only reached with two different heuristics whereas the cases marked with "$^{+}$" indicate ANDP instances where the lower bound was reached by only one heuristic.

| **SPG problem** | $\|V\|$ $\|E\|$ $\|T\|$ | ANDP_BCF | LB_GAP |
|---|---|---|---|
| Class C | | | |
| c03 | 500 625 83 | 756 | 0.27% |
| c04 | 500 1000 125 | 1082 | 0.28% |
| c05 | 500 1000 250 | 1584 | 0.32% |
| c08 | 500 1000 83 | 510 | 0.20% |
| c10 | 500 2500 250 | 1094 | 0.09% |
| Class MC | | | |
| mc13 | 150 11175 80 | 92 | OPT |
| mc2 | 120 71140 60 | 73 | 2.82% |
| mc3 | 97 4656 45 | 49 | 4.26% |
| Class X | | | |
| berlin52 | 52 1326 16 | 1379 | 32.09% |
| brazil58 | 58 1653 25 | 18093 | 32.50% |
| Class PUC | | | |
| cc3-4p | 64 288 8 | 2340 | 0.09% |
| cc3-5p | 125 750 13 | 3666 | 0.14% |
| cc3-5u | 125 750 13 | 36 | OPT |
| hc7p | 128 448 64 | 7905 | OPT |
| Class P6E | | | |
| P6E1 | 100 180 5 | 8810 | 17.70% |
| P6E2 | 100 180 5 | 10065 | 15.08% |
| P6E3 | 100 180 5 | 10251 | 17.99% |
| P6E4 | 100 180 10 | 15972 | OPT |
| P6E5 | 100 180 10 | 22990 | 17.92% |
| P6E6 | 100 180 20 | 23870 | 17.90% |
| P6E7 | 100 180 20 | 27220 | 17.95% |
| P6E8 | 100 180 20 | 26360 | 17.96% |
| P6E12 | 200 370 10 | 26125 | OPT |
| P6E13 | 200 370 20 | 46019 | 17.80% |
| Class P6Z | | | |
| P6Z1 | 100 180 5 | 9618 | 18.99% |
| P6Z2 | 100 180 5 | 5976 | 19.00% |
| P6Z4 | 100 180 10 | 12423 | 19.97% |
| P6Z12 | 200 370 10 | 18429 | OPT |
| P6Z13 | 200 370 20 | 32458 | 19.00% |

Table B.1: Best found solutions for instances derived from classes C, MC, X, PUC, P6E, and P6Z.

| **SPG problem** | $\|V\|$ $\|E\|$ $\|T\|$ | ANDP_BCF | LB_GAP |
|---|---|---|---|
| Class I080 | | | |
| I080-001 | 80 120 6 | 1843 | 3.13% |
| I080-002 | 80 120 6 | 1666 | 3.67% |
| I080-003 | 80 120 6 | 1829 | 6.77% |
| I080-004 | 80 120 6 | 1987 | 6.48% |
| I080-005 | 80 120 6 | 1918 | 7.15% |
| I080-011 | 80 350 6 | 1579 | 6.76% |
| I080-012 | 80 350 6 | 1484 | OPT |
| I080-013 | 80 350 6 | 1470 | 6.44% |
| I080-014 | 80 350 6 | 1412 | 1.07% |
| I080-015 | 80 350 6 | 1503 | 0.54% |
| I080-021 | 80 3160 6 | 1258 | 7.06% |
| I080-022 | 80 3160 6 | 1244 | 5.60% |
| I080-023 | 80 3160 6 | 1174 | OPT* ($\mathcal{H}_2$ and $\mathcal{H}_4$) |
| I080-024 | 80 3160 6 | 1161 | OPT |
| I080-025 | 80 3160 6 | 1247 | 7.31% |
| I080-031 | 80 160 6 | 1613 | 2.74% |
| I080-032 | 80 160 6 | 2214 | 6.03% |
| I080-033 | 80 160 6 | 1794 | OPT |
| I080-034 | 80 160 6 | 1812 | 7.35% |
| I080-035 | 80 160 6 | 1903 | 2.20% |
| I080-041 | 80 632 6 | 1276 | OPT |
| I080-042 | 80 632 6 | 1302 | 1.17% |
| I080-043 | 80 632 6 | 1383 | 6.80% |
| I080-044 | 80 632 6 | 1463 | 7.10% |
| I080-045 | 80 632 6 | 1384 | 5.65% |
| I080-101 | 80 120 8 | 2775 | 6.40% |
| I080-102 | 80 120 8 | 2566 | 6.78% |
| I080-103 | 80 120 8 | 2684 | 3.11% |
| I080-104 | 80 120 8 | 2651 | 6.64% |
| I080-105 | 80 120 8 | 2349 | 6.63% |
| I080-111 | 80 350 8 | 2051 | OPT |
| I080-112 | 80 350 8 | 2018 | 7.06% |
| I080-113 | 80 350 8 | 2022 | 7.32% |
| I080-114 | 80 350 8 | 1895 | OPT* ($H_2$ and $H_4$) |

Table B.2: Best found solutions for instances derived from class I080.

| **SPG problem** | $\|V\|$ $\|E\|$ $\|T\|$ | ANDP_BCF | LB_GAP |
|---|---|---|---|
| Class I080 | | | |
| I080-115 | 80 350 8 | 1997 | 6.91% |
| I080-121 | 80 3160 8 | 1643 | 5.25% |
| I080-122 | 80 3160 8 | 1604 | 2.75% |
| I080-123 | 80 3160 8 | 1569 | OPT |
| I080-124 | 80 3160 8 | 1667 | 7.20% |
| I080-125 | 80 3160 8 | 1572 | OPT |
| I080-131 | 80 160 8 | 2377 | 4.07% |
| I080-132 | 80 160 8 | 2328 | 6.79% |
| I080-133 | 80 160 8 | 2388 | 5.62% |
| I080-134 | 80 160 8 | 2207 | 6.62% |
| I080-135 | 80 160 8 | 2184 | 3.90% |
| I080-141 | 80 632 8 | 1788 | OPT |
| I080-142 | 80 632 8 | 1788 | 4.68% |
| I080-143 | 80 632 8 | 1889 | 6.90% |
| I080-144 | 80 632 8 | 1843 | 4.01% |
| I080-145 | 80 632 8 | 1884 | 6.92% |
| I080-211 | 80 350 16 | 3631 | OPT |
| I080-212 | 80 350 16 | 3677 | OPT |
| I080-213 | 80 350 16 | 3912 | 6.36% |
| I080-214 | 80 350 16 | 3847 | 3.03% |
| I080-215 | 80 350 16 | 3784 | 2.80% |
| I080-221 | 80 3160 16 | 3158 | OPT |
| I080-222 | 80 3160 16 | 3141 | OPT |
| I080-223 | 80 3160 16 | 3270 | 3.61% |
| I080-224 | 80 3160 16 | 3187 | 0.89% |
| I080-225 | 80 3160 16 | 3150 | OPT |
| I080-241 | 80 632 16 | 3778 | 6.78% |
| I080-321 | 80 3160 20 | 4123 | 4.86% |
| I080-322 | 80 3160 20 | 4019 | 2.08% |
| I080-323 | 80 3160 20 | 3946 | OPT |
| I080-324 | 80 3160 20 | 4199 | 6.79% |
| I080-325 | 80 3160 20 | 4190 | 6.78% |
| I080-342 | 80 632 20 | 4337 | OPT |
| I080-343 | 80 632 20 | 4533 | 6.76% |
| I080-344 | 80 632 20 | 4480 | 3.94% |
| I080-345 | 80 632 20 | 4637 | 6.82% |

Table B.3: Best found solutions for instances derived from class I080.

| **SPG problem** | $\|V\|$ $\|E\|$ $\|T\|$ | ANDP_BCF | LB_GAP |
|---|---|---|---|
| Class I160 | | | |
| I160-011 | 160 812 7 | 1743 | 3.94% |
| I160-012 | 160 812 7 | 1827 | OPT |
| I160-013 | 160 812 7 | 1722 | 3.67% |
| I160-014 | 160 812 7 | 1848 | 3.93% |
| I160-015 | 160 812 7 | 1845 | 4.36% |
| I160-031 | 160 320 7 | 2240 | 3.23% |
| I160-032 | 160 320 7 | 2432 | OPT |
| I160-033 | 160 320 7 | 2183 | 3.91% |
| I160-034 | 160 320 7 | 2174 | OPT |
| I160-035 | 160 320 7 | 2195 | 4.37% |
| I160-041 | 160 2544 7 | 1552 | 3.88% |
| I160-042 | 160 2544 7 | 1551 | 4.37% |
| I160-043 | 160 2544 7 | 1617 | 4.39% |
| I160-044 | 160 2544 7 | 1543 | OPT |
| I160-045 | 160 2544 7 | 1622 | 4.38% |
| I160-111 | 160 812 12 | 2985 | 4.04% |
| I160-112 | 160 812 12 | 3052 | OPT |
| I160-113 | 160 812 12 | 2965 | 3.45% |
| I160-114 | 160 812 12 | 3120 | 4.38% |
| I160-115 | 160 812 12 | 3061 | 4.22% |
| I160-141 | 160 2544 12 | 2661 | OPT |
| I160-142 | 160 2544 12 | 2674 | OPT |
| Class I320 | | | |
| I320-003 | 320 480 8 | 3042 | 2.36% |
| I320-004 | 320 480 8 | 2986 | 2.79% |
| I320-005 | 320 480 8 | 3074 | 2.77% |
| I320-011 | 320 1845 8 | 2053 | OPT |
| I320-033 | 320 640 8 | 2769 | OPT |
| I320-042 | 320 10208 8 | 1729 | 2.79% |
| I320-043 | 320 10208 8 | 1771 | 2.79% |
| I320-111 | 320 1845 17 | 4392 | 2.78% |
| I320-112 | 320 1845 17 | 4330 | 2.78% |
| I320-113 | 320 1845 17 | 4322 | 2.78% |
| I320-142 | 320 10208 17 | 3654 | 2.44% |
| I320-143 | 320 10208 17 | 3660 | 2.78% |
| I320-144 | 320 10208 17 | 3512 | OPT |
| I320-241 | 320 10208 34 | 7212 | 2.63% |
| I320-242 | 320 10208 34 | 7140 | 0.96% |

Table B.4: Best found solutions for instances derived from classes I160 and I320.

| **SPG problem** | $\|V\|$ $\|E\|$ $\|T\|$ | ANDP_BCF | LB_GAP |
|---|---|---|---|
| Class I640 | | | |
| I640-011 | 640 4135 9 | 2432 | 1.67% |
| I640-012 | 640 4135 9 | 2543 | 3.16% |
| I640-013 | 640 4135 9 | 2410 | 0.46% |
| I640-014 | 640 4135 9 | 2234 | 2.90% |
| I640-015 | 640 4135 9 | 2347 | OPT |
| I640-031 | 640 1280 9 | 3382 | 3.17% |
| I640-032 | 640 1280 9 | 3223 | 1.13% |
| I640-033 | 640 1280 9 | 3364 | 3.19% |
| I640-034 | 640 1280 9 | 3047 | 3.18% |
| I640-035 | 640 1280 9 | 3298 | 0.18% |
| I640-102 | 640 960 25 | 9420 | 3.41% |
| I640-103 | 640 960 25 | 9101 | 3.20% |
| I640-105 | 640 960 25 | 9623 | OPT |
| I640-111 | 640 4135 25 | 6364 | 3.19% |
| I640-113 | 640 4135 25 | 6432 | 2.93% |

Table B.5: Best found solutions for instances derived from class I640.

| **SPG problem** | $\|V\|$ $\|E\|$ $\|T\|$ | ANDP_BCF | LB_GAP |
|---|---|---|---|
| Class WRP3 | | | |
| WRP3-11 | 128 227 11 | 1100365 | 0.000363% |
| WRP3-12 | 84 149 12 | 1200247 | 0.000833% |
| WRP3-13 | 311 613 13 | 1300500 | 0.000230% |
| WRP3-14 | 128 247 14 | 1400255 | 0.000357% |
| WRP3-15 | 138 257 15 | 1500428 | 0.000399% |
| WRP3-16 | 204 374 16 | 1600208 | OPT |
| WRP3-17 | 177 354 17 | 1700448 | 0.000352% |
| WRP3-20 | 245 454 20 | 2000271 | OPT |
| WRP3-21 | 237 444 21 | 2100530 | 0.000380% |
| WRP3-22 | 233 431 22 | 2200560 | 0.000136% |
| WRP3-25 | 246 468 25 | 2500540 | OPT |
| WRP3-26 | 402 780 26 | 2600494 | 0.000384% |
| WRP3-27 | 370 721 27 | 2700512 | 0.000370% |
| WRP3-28 | 307 559 28 | 2800379 | OPT |
| WRP3-30 | 467 896 30 | 3000581 | 0.000399% |
| WRP3-31 | 323 592 31 | 3100700 | 0.002096% |
| WRP3-33 | 437 838 33 | 3300515 | 0.000060% |
| WRP3-36 | 435 818 36 | 3600624 | 0.000388% |
| WRP3-38 | 603 1207 38 | 3800656 | OPT |
| WRP3-39 | 703 1616 39 | 3900450 | $\mathrm{OPT}^{+}$ ($\mathcal{H}_4$) |
| WRP3-42 | 705 1373 42 | 4200598 | OPT |
| WRP3-48 | 925 1738 48 | 4800571 | 0.000395% |
| WRP3-49 | 886 1800 49 | 4900901 | 0.000387% |
| WRP3-52 | 701 1352 52 | 5200845 | 0.000384% |
| WRP3-53 | 775 1471 53 | 5300868 | 0.000396% |
| WRP3-75 | 729 1395 75 | 7501020 | OPT |
| WRP3-88 | 743 1409 88 | 88001527 | 0.000399% |

Table B.6: Best found solutions for instances derived from class WRP3.

| **SPG problem** | $\|V\|$ $\|E\|$ $\|T\|$ | ANDP_BCF | LB_GAP |
|---|---|---|---|
| Class WRP4 | | | |
| WRP4-11 | 123 233 11 | 1100197 | 0.001636% |
| WRP4-13 | 110 188 13 | 1300816 | 0.001383% |
| WRP4-14 | 145 283 14 | 1400308 | 0.001285% |
| WRP4-15 | 193 369 15 | 1500408 | 0.000199% |
| WRP4-17 | 223 404 17 | 1700548 | 0.001352% |
| WRP4-18 | 211 380 18 | 1801501 | 0.002053% |
| WRP4-19 | 119 206 19 | 1901472 | 0.001367% |
| WRP4-21 | 529 1032 21 | 2103312 | 0.001378% |
| WRP4-22 | 294 568 22 | 2200394 | OPT |
| WRP4-23 | 257 515 23 | 2300376 | OPT |
| WRP4-24 | 493 963 24 | 2403365 | 0.001373% |
| WRP4-27 | 243 497 27 | 2700508 | 0.002481% |
| WRP4-28 | 272 545 28 | 2800507 | 0.001464% |
| WRP4-29 | 247 505 29 | 2900573 | 0.003068% |
| WRP4-31 | 290 786 31 | 3100603 | 0.002483% |
| WRP4-32 | 311 632 32 | 3200720 | 0.005186% |
| WRP4-33 | 304 571 33 | 3300704 | 0.001484% |
| WRP4-34 | 314 650 34 | 3400572 | 0.001382% |
| WRP4-35 | 471 954 35 | 3500650 | 0.001399% |
| WRP4-36 | 363 750 36 | 3600596 | OPT |
| WRP4-37 | 522 1054 37 | 3700663 | 0.000432% |
| WRP4-38 | 294 618 38 | 3800662 | 0.001473% |
| WRP4-39 | 802 1553 39 | 3903734 | OPT* ($\mathcal{H}_2$ and $\mathcal{H}_4$) |
| WRP4-42 | 552 1131 42 | 4200759 | 0.001380% |
| WRP4-44 | 398 788 44 | 4401565 | 0.001385% |
| WRP4-45 | 388 815 45 | 4500791 | 0.001399% |
| WRP4-46 | 632 1287 46 | 4600756 | OPT |
| WRP4-52 | 547 1115 52 | 5201161 | 0.001538% |
| WRP4-56 | 839 1617 56 | 5602377 | 0.001392% |
| WRP4-59 | 904 1806 59 | 5901674 | 0.001389% |
| WRP4-75 | 938 1869 75 | 7501817 | 0.001399% |

Table B.7: Best found solutions for instances derived from class WRP4.

# Appendix C

# Properties used for generating BNDP test cases

We introduce here some properties used to build the test cases for the BNDP problem denoted Networks 2 to 6 and introduced in Chapter 4. We remark that the properties will be proved in the context of the *Generalized Steiner Problem with node-connectivity constraints* (denoted by GSP-NC) which is the model on which the BNDP is based. Next, we give its definition, some notation and auxiliary definitions.

**Definition C.0.3** *We define the GSP-NC as follows. Given a non-directed simple graph $G = (V, E)$, a matrix $C = \{c_{ij}\}_{i,j \in V}$ of nonnegative edge-costs, a subset $T \subseteq V$ called "set of terminal nodes", a matrix $R = \{r_{ij}\}_{i,j \in T}$ of required local node-connectivities between any pair of different nodes in $T$ (for any $i, j \in T$, $r_{ij}$ is a non-negative integer number), the goal is to find a subgraph $G_T$ of $G$ with minimal cost so that for every pair of nodes $i, j \in T, i \neq j$, there are at least $r_{ij}$ node-disjoint paths connecting $i$ and $j$ in $G_T$. The nodes in $V \backslash T$ are usually called Steiner nodes. We will denote by $\Gamma_{GSPNC}$ the space of feasible solutions associated with the problem.*

**Notation C.0.4** *Given a non-directed simple graph $G$ and a node $v \in G$, we denote by $d_G(v)$ the degree of $v$ in $G$.*

**Notation C.0.5** *Given a GSP-NC instance we will denote by $\Gamma_{GSPNC}$ to the space of feasible solutions.*

**Notation C.0.6** *Given a path or a tree $H$, we call $u$ an endpoint of $H$ if $d_H(u) = 1$.*

**Definition C.0.7** *Given a non-directed simple graph $H$, we call $p_H$ an $H - path$ if $p_H$ is a (non-trivial) path, which meets $H$ exactly in its endpoints (i.e., the endpoints of $p_H$ are in $H$, and if $p_H$ is not a simple edge, the other nodes appearing in $p_H$ are not in $H$).*

**Definition C.0.8** *We define a $u - tree$ on a graph $G$ as a tree $\mathcal{T}$ rooted in $u$ which meets $G$ exactly in its endpoints (the endpoints of $\mathcal{T}$ are in $G$) and the other nodes appearing in $\mathcal{T}$ are not in $G$ and have degree 2, except $u$ which may have larger degree. We will call a pendant to a path $p_{(u,v)} \subset \mathcal{T}$ such that $v$ is an endpoint.*

The following property introduces "splitting" and "merging" operations.

**Proposition C.0.9**

1) **(Splitting)** *Let $(v_i, v_j)$ be any edge of $G$, with cost $c_{ij}$. Splitting this edge corresponds to adding a new (non-terminal) node $v_k$, and replacing edge $(v_i, v_j)$ by two edges $\{(v_i, v_k), (v_k, v_j)\}$. If the costs of these new edges verify that $c_{ik} > 0$, $c_{kj} > 0$ and $c_{ik} + c_{kj} = c_{ij}$; and if $\mathcal{G}_{opt}$ is an optimal solution of the original problem, then the graph $\bar{\mathcal{G}}_{opt}$ resulting of applying this splitting operation on $\mathcal{G}_{opt}$ is an optimal solution of the modified problem.*

2) **(Merging)** *Let $v_i, v_k, v_j$ be three nodes of $G$, such that $v_k \notin T$, there is an edge $(v_i, v_k)$ of cost $c_{ik}$, there is an edge $(v_k, v_j)$ of cost $c_{jk}$, there is no edge between $v_i$ and $v_j$, and $d_G(v_k) = 2$. Merging edges $(v_i, v_k)$ and $(v_k, v_j)$ corresponds to deleting node $v_k$, and replacing edges $\{(v_i, v_k), (v_k, v_j)\}$ by a new edge $(v_i, v_j)$. If the cost of the new edge verifies that $c_{ij} = c_{ik} + c_{kj}$, and $\mathcal{G}_{opt}$ is an optimal solution of the original problem, then the graph $\bar{\mathcal{G}}_{opt}$ resulting of applying this merging operation on $\mathcal{G}_{opt}$ is an optimal solution of the modified problem.*

**Proof.** 1) Trivial. The set of feasible solutions which do not contain edge $(v_i, v_j)$ is the same for the original problem and for the problem modified after the splitting operation. There is a bijection between the set of feasible solutions which contain edge $(v_i, v_j)$ in the original problem and the set of feasible solutions wich contain edges $(v_i, v_k)$ and $(v_k, v_j)$ after the splitting operations; and the costs of the corresponding solutions are identical. If there are new feasible solutions which only have edge $(v_i, v_k)$ or edge $(v_k, v_j)$, they are not minimal (the edge can be deleted preserving feasibility, as $k \notin T$). Then, both sets will have optimal solutions of the same value.

2) Similar to part 1.

**QED**

It is clear that splitting and merging operations can be applied as many times as needed. Then, it is possible to substitute an edge $(v_i, v_j)$ by an $H - path$ with endpoints $v_i$ and $v_j$, and fix the costs of the new edges so that the optimal solutions are preserved. The following property is useful to modify the costs of existing edges.

**Proposition C.0.10** *Let $\mathcal{G}_{opt}$ be an optimal solution of the original GSPNC instance, and let $(v_i, v_j)$ be an edge of $G$ with cost $c_{ij}$.*

1) *If $(v_i, v_j) \in \mathcal{G}_{opt}$, if we modify the problem by assigning this edge a new cost $\bar{c}_{ij} < c_{ij}$, then $\mathcal{G}_{opt}$ is still an optimal solution of the new problem instance.*

*2) If $(v_i, v_j) \notin \mathcal{G}_{opt}$, if we modify the problem by assigning this edge a new cost $\bar{c}_{ij} > c_{ij}$, then $\mathcal{G}_{opt}$ is still an optimal solution of the new problem instance.*

**Proof.** 1) Given an edge $e = (v_i, v_j) \in \mathcal{G}_{opt}$, let us consider the following notation.

- $\Gamma_{opt}^{(e)}$ is the subspace of optimal solutions containing the edge $e$,
- $\Gamma_{opt}^{(\setminus e)}$ is the subspace of optimal solutions not containing the edge $e$,
- $\Gamma_{nonopt}^{(e)}$ is the subspace of feasible solutions (non-optimal) containing the edge $e$.

If the cost of the edge $e \in \mathcal{G}_{opt}$ is diminished, then the cost of $\mathcal{G}_{opt}$ and of all solutions in $\Gamma_{opt}^{(e)}$ is also diminished and by the same amount $c_{ij} - \bar{c}_{ij}$.

The costs of the solutions in $\Gamma_{opt}^{(\setminus e)}$ do not change, so these solutions are no longer optimal in the modified instance.

We will prove (by contradiction) that in addition the feasible solutions in $\Gamma_{nonopt}^{(e)}$ are not optimal for the new instance. Let us suppose that a network $\mathcal{H} \in \Gamma_{nonopt}^{(e)}$ is optimal for the new instance. Then, the following relation is satisfied:

$$\text{COST}(\hat{\mathcal{H}}) \leq \text{COST}(\hat{\mathcal{G}}_{opt}),$$

where $\hat{\mathcal{H}}$ and $\hat{\mathcal{G}}$ are the networks $\mathcal{H}$ and $\mathcal{G}$ respectively but with the new cost for $e$. This inequality implies:

$$\text{COST}(\mathcal{H}) + \Delta e \leq \text{COST}(\mathcal{G}_{opt}) + \Delta e,$$

and therefore $\text{COST}(\mathcal{H}) \leq \text{COST}(\mathcal{G}_{opt})$, which is a contradiction since $\mathcal{H}$ is not optimal for the original instance.

Then, the optimal solutions for the new instance are those in $\Gamma_{opt}^{(e)}$.

2) Similar to part 1.

**QED**

We now look at some new operations, which we call $H - path$ insertions and $u - tree$ insertions.

**Proposition C.0.11 ($H - path$ insertions)** *Let $\mathcal{G}_{opt}$ be an optimal solution of the original GSPNC instance.*

*1) Assuming $r_{ij} \geq 2$, $\forall i, j \in T$; if on $G$ we add an $H - path$ $p_H$ with endpoints two nodes $v_i, v_j \in \mathcal{G}_{opt}$ so that $\text{COST}(p_H) \geq \text{COST}(p_{l(i,j)})$, where $p_{l(i,j)}$ is the longest path from $v_i$ to $v_j$ in $\mathcal{G}_{opt}$, then $\mathcal{G}_{opt}$ is an optimal solution for the new instance.*

2) *If on $G$ we add an $H-path\ p_H$ with endpoints two adjacent nodes $v_i, v_j \in \mathcal{G}_{opt}$, where at least one of them is a Steiner node and so that:*

$$\text{COST}(p_H) \geq c_{ij},$$

*then $\mathcal{G}_{opt}$ is an optimal solution for the new instance.*

3) *If on $G$ we add an $H-path\ p_H$ with endpoints two adjacent terminal nodes $v_i, v_j \in \mathcal{G}_{opt}$ satisfying $\text{COST}(p_H) \geq \text{COST}(p)$ for all path $p$ where:*

   i) *$p$ is a path from $v_i$ to $v_j$ on $G \setminus \text{EDGES}(\mathcal{G}_{opt})$ and $(\mathcal{G}_{opt} \setminus \{(v_i, v_j)\} \cup p)$ is feasible for the original instance (the existence of at least one path in these conditions is assumed),*

   ii) *or $p$ is a path from $v_i$ to $v_j$ on $G \setminus \text{NODES}(\mathcal{G}_{opt} \setminus T)$,*

*then $\mathcal{G}_{opt}$ is an optimal solution for the new instance.*

4) *Assuming $r_{ij} \geq 2, \forall i, j \in T$, $|T| > 2$; if on $G$ we add an $H-path\ p_H$ with endpoints two adjacent nodes $v_i, v_j \in G$ such that $\text{COST}(p_H) \geq c_{ij}$, then $\mathcal{G}_{opt}$ is an optimal solution for the new instance.*

5) *If on $G$ we add an $H-path\ p_H$ with endpoints two adjacent nodes $v_i, v_j \in \mathcal{G}_{opt}$ such that:*

$$\text{COST}(p_H) \geq \max\{c_{ij}, \text{COST}(p_{(i,j)})\},$$

*for all path $p_{(i,j)} \subset \mathcal{G}_{opt} \setminus \{(v_i, v_j)\}$ integrated by Steiner nodes of degree 2 communicating $v_i$ with $v_j$, then $\mathcal{G}_{opt}$ is an optimal solution for the new instance.*

**Proof.** We will prove in order each one of the previous points.
1) Let us suppose that $\mathcal{G}_{opt}$ is not optimal for the new instance. Necessarily, if $\mathcal{H}_{opt}$ is a global optimal solution for the new instance, then $p_H \subset \mathcal{H}_{opt}$ (otherwise $\mathcal{H}_{opt}$ would be a better solution than $\mathcal{G}_{opt}$ for the original instance). Since $\mathcal{G}_{opt}$ is a feasible solution for the new instance, we have $\text{COST}(\mathcal{H}_{opt}) < \text{COST}(\mathcal{G}_{opt})$. Let us consider the network $\bar{\mathcal{H}} = \mathcal{H}_{opt} \setminus p_H$. By minimality of $\mathcal{H}_{opt}$, $\bar{\mathcal{H}}$ is not feasible for the original instance. On the other hand, as $r_{ij} \geq 2, \forall i, j \in T$, there exists a path $p \subset \mathcal{G}_{opt}$ from $v_i$ to $v_j$ such that the network $\hat{\mathcal{H}} = \bar{\mathcal{H}} \cup p$ is a feasible solution for the original instance. As $\mathcal{G}_{opt}$ is optimal for the original instance, we know:

$$\text{COST}(\mathcal{G}_{opt}) \leq \text{COST}(\hat{\mathcal{H}}).$$

We define the following sets of edges:

- $A = \{e \in p | e \in \mathcal{H}_{opt}\} = \{e \in p | e \in \mathcal{G}_{opt} \cap \mathcal{H}_{opt}\}$,
- $B = \{e \in p | e \notin \mathcal{H}_{opt}\}$.

Clearly, $\text{EDGES}(p) = A \cup B$ and moreover:

$$\text{COST}(p_H) \overset{\text{by hyp.}}{\geq} \text{COST}(p) = \text{COST}(A) + \text{COST}(B) \geq \text{COST}(B).$$

Let us analyze the cost of $\hat{\mathcal{H}}$:

$$\text{COST}(\hat{\mathcal{H}}) = \text{COST}(\mathcal{H}_{opt}) - \text{COST}(p_H) + \text{COST}(B) \leq \text{COST}(\mathcal{H}_{opt}),$$

implying $\text{COST}(\mathcal{G}_{opt}) \leq \text{COST}(\mathcal{H}_{opt})$, which is a contradiction.

2) Again, by contradiction, let us suppose that $\mathcal{G}_{opt}$ is not optimal for the new instance. Let $\mathcal{H}_{opt}$ be an optimal solution for the new instance. As in the previous case, it is easy to see that $p_H$ must satisfy $p_H \subset \mathcal{H}_{opt}$. Now, let us suppose that $(v_i, v_j) \notin \mathcal{H}_{opt}$, then the network $\bar{\mathcal{H}} = (\mathcal{H}_{opt} \setminus p_H) \cup \{(v_i, v_j)\}$ would be feasible for the new instance and in addition:

$$\text{COST}(\bar{\mathcal{H}}) = \text{COST}(\mathcal{H}_{opt}) - \text{COST}(p_H) + c_{ij} \overset{\text{by hyp.}}{\leq} \text{COST}(\mathcal{H}_{opt}),$$

implying the optimality of $\bar{\mathcal{H}}$ with $p_H \not\subset \bar{\mathcal{H}}$, which is a contradiction. Hence, necessarily the edge $(v_i, v_j) \in \mathcal{H}_{opt}$. We define the network: $\hat{\mathcal{H}} = (\mathcal{H}_{opt} \setminus p_H)$. Since the requirements are of node connectivity, considering in $\mathcal{H}_{opt}$ the edge $(v_i, v_j)$ and the path $p_H$, only one of them can contribute to satisfy a connection requirement between a pair of terminal nodes (except for $\{v_i, v_j\}$ if both are terminals). Thus, $\hat{\mathcal{H}}$ is feasible for the original instance and besides:

$$\text{COST}(\hat{\mathcal{H}}) = \text{COST}(\mathcal{H}_{opt}) - \text{COST}(p_H) < \text{COST}(\mathcal{H}_{opt}) \overset{\text{by optimality}}{<} \text{COST}(\mathcal{G}_{opt}),$$

implying the optimality of $\hat{\mathcal{H}}$ for the original instance, which is a contradiction.

3.i) Since $v_i$ and $v_j$ are terminal nodes, $(v_i, v_j) \in \mathcal{G}_{opt}$ and the requirements are of node connectivity, the inclusion of the $H - path\ p_H$ would result in an improvement in the connectivity level only for the terminal nodes $v_i$ and $v_j$ (in the new instance, any other pair of terminal nodes would use in an excluding way the path $p_H$ or the edge $(v_i, v_j)$ to satisfy one of their connection requirements). Let us suppose that $\mathcal{G}_{opt}$ is not optimal for the new instance. It is easy to prove that there exists an optimal solution for the new instance whose topology is given by: $\mathcal{H}_{opt} = (\mathcal{G}_{opt} \cup p_H) \setminus p_{(i,j)}$, where $p_{(i,j)}$ is a path from $v_i$ to $v_j$ on $\mathcal{G}_{opt}$. Let $p$ be a path in the hypothesis of the Proposition. Let us denote $\bar{\mathcal{H}} = (\mathcal{G}_{opt} \setminus \{(v_i, v_j)\} \cup p)$. As $\bar{\mathcal{H}}$ is feasible for the original instance, it is easy to see that the network $\hat{\mathcal{H}} = (\mathcal{H}_{opt} \setminus p_H) \cup p$ is feasible for the original instance and therefore for the new instance. By optimality of $\mathcal{H}_{opt}$, we have:

$$\text{COST}(\mathcal{H}_{opt}) \leq \text{COST}(\hat{\mathcal{H}}).$$

If $\text{COST}(\mathcal{H}_{opt}) = \text{COST}(\hat{\mathcal{H}})$ then $\hat{\mathcal{H}}$ would be an optimal solution for the new instance and not containing $p_H$, which is a contradiction. If $\text{COST}(\mathcal{H}_{opt}) < \text{COST}(\hat{\mathcal{H}})$, then:

$$\text{COST}(\mathcal{H}_{opt}) < \text{COST}(\mathcal{H}_{opt}) - \text{COST}(p_H) + \text{COST}(p),$$

implying $\text{COST}(p_H) < \text{COST}(p)$, which also is a contradiction. Hence, $\mathcal{G}_{opt}$ is an optimal solution for the new instance.

3.ii) Let us suppose that $\mathcal{G}_{opt}$ is not optimal for the new instance. Like $(3.\text{i})$, necessarily there exists an optimal solution for the new instance given by a network: $\mathcal{H}_{opt} = (\mathcal{G}_{opt} \cup p_H) \setminus p_{(i,j)}$, where $p_{(i,j)}$ is a path from $v_i$ to $v_j$ on $\mathcal{G}_{opt}$. Let $p$ be a path in the hypothesis of the Proposition (we assume that it exists, otherwise $(3.\text{ii})$ is fulfilled emptily). Let us define the network $\hat{\mathcal{H}} = (\mathcal{H}_{opt} \setminus p_H) \cup p$; clearly this network is feasible for the new instance and the original instance. Then,

$$\text{COST}(\mathcal{H}_{opt}) \leq \text{COST}(\hat{\mathcal{H}}).$$

As above, the equality induces a contradiction and the strict inequality would imply $\text{COST}(p_H) < \text{COST}(p)$, which also is a contradiction. Therefore $\mathcal{G}_{opt}$ is globally optimal for the new instance.

4) Let us suppose that $\mathcal{G}_{opt}$ is not optimal for the new instance. Let $\mathcal{H}_{opt}$ be an optimal solution for the new instance. Necessarily $p_H \subset \mathcal{H}_{opt}$. Let us consider the network $\bar{\mathcal{H}} = (\mathcal{H}_{opt} \setminus p_H) \cup \{(v_i, v_j)\}$ (since $|T| > 2$ the edge $(v_i, v_j) \notin \mathcal{H}_{opt}$). The network $\bar{\mathcal{H}}$ is a feasible solution for the original instance and moreover:

$$\text{COST}(\bar{\mathcal{H}}) = \text{COST}(\mathcal{H}_{opt}) - \text{COST}(p_H) + c_{ij} \leq \text{COST}(\mathcal{H}_{opt}) \overset{\text{optimality of } \mathcal{H}_{opt}}{\uparrow} < \text{COST}(\mathcal{G}_{opt}),$$

and therefore: $\text{COST}(\bar{\mathcal{H}}) < \text{COST}(\mathcal{G}_{opt})$. This contradicts the optimality of $\mathcal{G}_{opt}$, hence $\mathcal{G}_{opt}$ is a global optimal solution for the new instance.

5) Let us suppose that $\mathcal{G}_{opt}$ is not optimal for the new instance. As in the previous case, if $\mathcal{H}_{opt}$ is an optimal solution for the new instance, this must satisfy $p_H \subset \mathcal{H}_{opt}$. Let $P$ be the set of paths including in $\mathcal{G}_{opt} \setminus \{(v_i, v_j)\}$ integrated by Steiner nodes of degree 2 communicating $v_i$ with $v_j$. If $(P \cup \{(v_i, v_j)\}) \subset \mathcal{H}_{opt}$ the network $\bar{\mathcal{H}} = \mathcal{H}_{opt} \setminus p_H$ is a feasible solution for the original instance and therefore:

$$\text{COST}(\bar{\mathcal{H}}) \geq \text{COST}(\mathcal{G}_{opt}).$$

Let us analyze its cost:

$$\text{COST}(\bar{\mathcal{H}}) = \text{COST}(\mathcal{H}_{opt}) - \text{COST}(p_H) \leq \text{COST}(\mathcal{H}_{opt}) \overset{\text{optimality of } \mathcal{H}_{opt}}{\uparrow} < \text{COST}(\mathcal{G}_{opt}),$$

which is a contradiction.

If there exists $p \in (P \cup \{(v_i, v_j)\})$ such that $p \notin \mathcal{H}_{opt}$, considering the network $\hat{\mathcal{H}} = (\mathcal{H}_{opt} \setminus p_H) \cup p$, this

is feasible for the new instance and moreover:

$$\text{COST}(\hat{\mathcal{H}}) = \text{COST}(\mathcal{H}_{opt}) - \text{COST}(p_H) + \text{COST}(p) \overset{\text{by hyp.}}{\underset{\uparrow}{\leq}} \text{COST}(\mathcal{H}_{opt}).$$

If the relation is satisfied by means of the equality: $\text{COST}(\hat{\mathcal{H}}) = \text{COST}(\mathcal{H}_{opt})$, then $\hat{\mathcal{H}}$ would be an optimal solution for the new instance and in addition not containing to the path $p_H$. If the relation is satisfied by means of the strict inequality, $\hat{\mathcal{H}}$ would be a better feasible solution than $\mathcal{H}_{opt}$ for the new instance, which is a contradiction.

**QED**

**Proposition C.0.12 (**$u - tree$ **insertion)** *Let $\mathcal{G}_{opt}$ be an optimal solution of the original instance. If on an edge $(v_i, v_j) \in G$ we apply consecutively $k$ splitting operations preserving the optimality of $\bar{\mathcal{G}}_{opt}$ (with $\mathcal{G}_{opt} = \bar{\mathcal{G}}_{opt}$ if $(v_i, v_j) \notin \mathcal{G}_{opt}$ or $\bar{\mathcal{G}}_{opt}$ is $\mathcal{G}_{opt}$ transformed by the splitting operations if $(v_i, v_j) \in \mathcal{G}_{opt}$), creating thus a set $K$ of new Steiner nodes, and posteriorly we add a $u-tree$ $\mathcal{T}$ connecting $u$ with a subset $W \subseteq K$ (the endpoints of $\mathcal{T}$) of nodes such that $\forall w_1, w_2 \in W$:*

$$\text{COST}(p_{(u,w_1)}) + \text{COST}(p_{(u,w_2)}) \geq \text{c}_{sp}^{(w_1,w_2)},$$

*where $\text{c}_{sp}^{(w_1,w_2)}$ is the cost from $w_1$ to $w_2$ on $(v_i, v_j)$ after splitting operations and $p_{(u,w_1)}, p_{(u,w_2)} \subset \mathcal{T}$ are the paths from $u$ to $w_1$ and $w_2$ respectively; then $\bar{\mathcal{G}}_{opt}$ (or $\mathcal{G}_{opt}$ if $(v_i, v_j) \notin \mathcal{G}_{opt}$) is an optimal solution for the new instance.*

**Proof.** Without loss of generality, we will analyze the case $(v_i, v_j) \in \mathcal{G}_{opt}$ (the other case is analogous). Let us suppose that $\bar{\mathcal{G}}_{opt}$ is not optimal for the new instance. Then, a subset of nodes $\{x_1, x_2\} \subseteq W$ will have to integrate an optimal solution $\mathcal{H}_{opt}$ for the new instance. Let $A$ be the set of new edges resulting of the splitting operations on $(v_i, v_j)$. Let us consider $B \subseteq A$ the set of edges belonging to $\mathcal{H}_{opt}$. We have the following relation:

$$\begin{aligned}\text{COST}(\mathcal{H}_{opt}) &= \text{COST}(\bar{\mathcal{G}}_{opt}) + \text{COST}(p_{(u,x_1)}) + \text{COST}(p_{(u,x_2)}) + \text{COST}(B) - \text{COST}(A \setminus B) \\ &= \text{COST}(\bar{\mathcal{G}}_{opt}) + \text{COST}(p_{(u,x_1)}) + \text{COST}(p_{(u,x_2)}) - \text{c}_{sp}^{(x_1,x_2)} \\ &\overset{\text{nonoptimality of } \bar{\mathcal{G}}_{opt}}{\underset{\uparrow}{<}} \text{COST}(\bar{\mathcal{G}}_{opt}),\end{aligned}$$

this implies,

$$\text{COST}(p_{(u,x_1)}) + \text{COST}(p_{(u,x_2)}) < \text{c}_{sp}^{(x_1,x_2)},$$

which is a contradiction; hence $\bar{\mathcal{G}}_{opt}$ is globally optimal for the new instance.

**QED**

**Proposition C.0.13** *Let $\mathcal{G}_{opt}$ be an optimal solution and let $p_H$ be an $H-path$ added to $G$ with one endpoint a node $u \notin \mathcal{G}_{opt}$. Let us denote by $\mathcal{T}_u$ a graph such that $p_H \subset \mathcal{T}_u$ and there exists a set of edges $U \subseteq \text{EDGES}(\mathcal{G}_{opt})$ satisfying: $\mathcal{T}_u \cup (\mathcal{G}_{opt} \setminus U)$ is a minimal network belonging to $\Gamma_{GSPNC}$. If we assign a cost to $p_H$ so that:*

$$\text{COST}(p_H) \geq \text{COST}(U) - \text{COST}(\mathcal{T}_u \setminus p_H),\ \forall \mathcal{T}_u, \forall U,$$

*then $\mathcal{G}_{opt}$ is optimal solution for the new instance.*

**Proof.** Let us suppose that $\mathcal{G}_{opt}$ is not optimal for the new instance. Necessarily, there exists an optimal solution $\mathcal{H}_{opt}$ for the new instance, a sub-graph $\mathcal{T}_u$ and a set of edges $U \subseteq \text{EDGES}(\mathcal{G}_{opt})$ so that $p_H \subset \mathcal{T}_u \subset \mathcal{H}_{opt}$ and $\mathcal{H}_{opt} = \mathcal{T}_u \cup (\mathcal{G}_{opt} \setminus U)$. Then, we have the following relation:

$$\text{COST}(\mathcal{H}_{opt}) < \text{COST}(\mathcal{G}_{opt}),$$

and moreover,

$$\begin{aligned}\text{COST}(\mathcal{H}_{opt}) &= \text{COST}(\mathcal{T}_u) + \text{COST}(\mathcal{G}_{opt}) - \text{COST}(U) \\ &= \text{COST}(\mathcal{T}_u \setminus p_H) + \text{COST}(p_H) + \text{COST}(\mathcal{G}_{opt}) - \text{COST}(U) < \text{COST}(\mathcal{G}_{opt}),\end{aligned}$$

implying: $\text{COST}(\mathcal{T}_u \setminus p_H) + \text{COST}(p_H) - \text{COST}(U) < 0$, which is a contradiction. Hence, $\mathcal{G}_{opt}$ is globally optimal for the new instance.

**QED**

**Proposition C.0.14** *Let $\mathcal{G}_{opt}$ be an optimal solution of the original instance. Let $\mathcal{T}$ be a $u-tree$ added to $G$. If we assign costs to $\mathcal{T}$ so that $\forall U \subseteq \text{EDGES}(\mathcal{G}_{opt})$ and $\forall \bar{\mathcal{T}} \subseteq \mathcal{T}$ such that $\bar{\mathcal{T}} \cup (\mathcal{G}_{opt} \setminus U) \in \Gamma_{GSPNC}$ and minimal, we have the inequality:*

$$\text{COST}(\bar{\mathcal{T}}) \geq \text{COST}(U),$$

*then $\mathcal{G}_{opt}$ is globally optimal for the new instance.*

**Proof.** Let us suppose that $\mathcal{G}_{opt}$ is not optimal for the new instance. Necessarily, the $u-tree$ $\mathcal{T}$ introduces new paths so that there exists an optimal solution $\mathcal{H}_{opt}$ for the new instance, a set $\bar{U} \subseteq \text{EDGES}(\mathcal{G}_{opt})$ and a graph $\bar{\mathcal{T}} \subseteq \mathcal{T}$ such that $\mathcal{H}_{opt} = (\mathcal{G}_{opt} \setminus \bar{U}) \cup \bar{\mathcal{T}}$. Since $\mathcal{G}_{opt}$ is feasible for the new instance but not optimal, we have:

$$\text{COST}(\mathcal{H}_{opt}) < \text{COST}(\mathcal{G}_{opt}).$$

Then,

$$\textsc{cost}(\mathcal{H}_{opt}) = \textsc{cost}(\mathcal{G}_{opt}) - \textsc{cost}(\bar{U}) + \textsc{cost}(\bar{\mathcal{T}}) < \textsc{cost}(\mathcal{G}_{opt}),$$

implying: $\textsc{cost}(\bar{\mathcal{T}}) - \textsc{cost}(\bar{U}) < 0$, which is a contradiction. Therefore $\mathcal{G}_{opt}$ is globally optimal for the new instance.

**QED**

The following Proposition is particularly useful to assign costs to the original graph preserving as global optimal solution a known feasible topology having minimum number of edges.

**Proposition C.0.15 (minimal topology)** *Let $\mathcal{G}$ be a feasible solution of the GSPNC instance such that for any other feasible solution $\mathcal{H}$ we have:*

*1)* $|\textsc{edges}(\mathcal{G})| \leq |\textsc{edges}(\mathcal{H})|$ *and moreover,*

*2) for any edge $e \in \mathcal{G}$ and for any edge $\bar{e} \in (G \setminus \textsc{edges}(\mathcal{G}))$, $c_e \leq c_{\bar{e}}$,*

*then $\mathcal{G}$ is a global optimal solution.*

**Proof.** Let us suppose that $\mathcal{G}$ is not optimal. Then, there exists a feasible solution $\bar{\mathcal{G}}$ such that $\textsc{cost}(\bar{\mathcal{G}}) < \textsc{cost}(\mathcal{G})$. We define the following sets of edges:

- $A = \{e \in \mathcal{G} | e \in \bar{\mathcal{G}}\}$ and $B = \{e \in \mathcal{G} | e \notin \bar{\mathcal{G}}\}$,
- $\bar{A} = \{e \in \bar{\mathcal{G}} | e \in \mathcal{G}\}$ and $\bar{B} = \{e \in \bar{\mathcal{G}} | e \notin \mathcal{G}\}$.

Clearly, $\textsc{edges}(\mathcal{G}) = A \cup B$, $\textsc{edges}(\bar{\mathcal{G}}) = \bar{A} \cup \bar{B}$ and $A = \bar{A}$. On the other hand, it is easy to see that:

$$\textsc{cost}(\mathcal{G}) \leq \textsc{cost}(A) + c^{(\mathcal{G})}_{max} \cdot |B|,$$

where $c^{(\mathcal{G})}_{max} = \max\{c_e | e \in \mathcal{G}\}$. If $|\bar{B}| \geq |B|$ we have:

$$\textsc{cost}(\mathcal{G}) \leq \textsc{cost}(A) + c^{(\mathcal{G})}_{max} \cdot |\bar{B}| \overset{\text{by hyp. 2)}}{\leq} \textsc{cost}(A) + \textsc{cost}(\bar{B}) = \textsc{cost}(\bar{\mathcal{G}}),$$

which is a contradiction. Otherwise, if $|\bar{B}| < |B|$ we would have the relation:

$$|\textsc{edges}(\bar{\mathcal{G}})| = |A \cup \bar{B}| = |A| + |\bar{B}| < |A| + |B| = |A \cup B| = |\textsc{edges}(\mathcal{G})|,$$

which contradicts hypothesis $(1)$. Hence, $\mathcal{G}$ is globally optimal.

**QED**

Now, let us place in the context of the BNDP. We describe below the main characteristics of the six first problem instances presented in Chapter 4. Figure 4.16 shows the topologies associated with these test cases. The black nodes represent the fixed nodes, while the white nodes represent the Steiner nodes, which may or may not be included in the solution.

- Network 1 has a double grid structure, with 33 fixed nodes, 87 Steiner nodes and 286 edges. The link costs were chosen so that a link between Steiner nodes has cost 1, a link between a Steiner node and a fixed node has cost 2 and a link between two fixed nodes has cost 4. The objective is to find a 2-node-survivable subnetwork with minimal cost (all connection requirement between fixed nodes are equal to 2). One optimal solution for this problem of cost 140 (shown in Figure 4.18) was found by an exact parallel-distributed backtracking algorithm [121].

- Network 2 has been constructed in the following way. Taking as basis a network 4-node-connected, we applied iteratively splitting operations followed by the substitution of four edges by four 2-octahedron topologies (this topology can be seen in [43]); and posteriorly we applied $H - path$ insertions until forming the topology associated to Network 2. We chose 22 nodes as fixed nodes. From the original topology, we deleted edges in order to have a minimal 4-node-survivable network spanning $S_D^{(I)}$ (we employ Menger's theorem, cited in [43], to guarantee the feasibility), and assigned costs of the edges obtaining a minimal feasible solution of cost 680. The edges that were deleted were re-inserted into the network and the costs were selected carefully by means of the application of properties like the exposed above and other properties related to edges satisfying the triangular inequality, so that the optimum cost is preserved. However, other optimal solutions may come up. The resulting instance has 22 fixed nodes, 61 Steiner nodes and 262 edges. The links have costs in the interval $[1, 200]$. The objective is to find a 4-node-survivable subnetwork spanning the fixed nodes set $S_D^{(I)}$ (all connection requirement between fixed nodes are equal to 4).

- Network 3 has 41 fixed nodes, 38 Steiner nodes and 364 edges. This network was designed taking as basis two dodecahedron topologies connected through a fixed node and other connections. Figure C.1 shows the initial network from which applying splitting operations and $H - path$ insertions we obtained our testing instance. Our aim is to find a minimum cost network which must be 2-node-survivable for all pair of fixed nodes and 3-node-survivable for the square fixed nodes. From the original network, we eliminated some of the edges to have a unique feasible solution, and fixed arbitrarily the edge values; obtaining a solution of cost 1848 which is shown in Figure C.1 and which we will call "primary optimal solution". Then the edges which were eliminated were re-inserted into the network, with costs chosen in order to preserve the optimality of the primary optimal solution. Afterwards new Steiner nodes and new connections (and the costs associated with these connections)

were added by splitting operations and $H - path$ insertions, also verifying the conditions of propositions C.0.9, C.0.10, C.0.11 and C.0.13 in order to preserve the optimality of the known optimal topology. Furthermore, we applied some $u - tree$ insertions verifying the conditions of Proposition C.0.12 and guaranteeing therefore the preservation of the optimality of the primary solution. In these operations we systematically apply suitable controls to the triangles and cycles formed each time we add an $H - path$, a $u - tree$ or two new edges are created by a splitting operation so that optimality in this solution is maintained (typically, using the propositions C.0.12, C.0.13 and C.0.14). The "primary optimal" solution is preserved, and new optimal solutions may also come up. All the edge costs were selected within the $[10, 200]$ range.

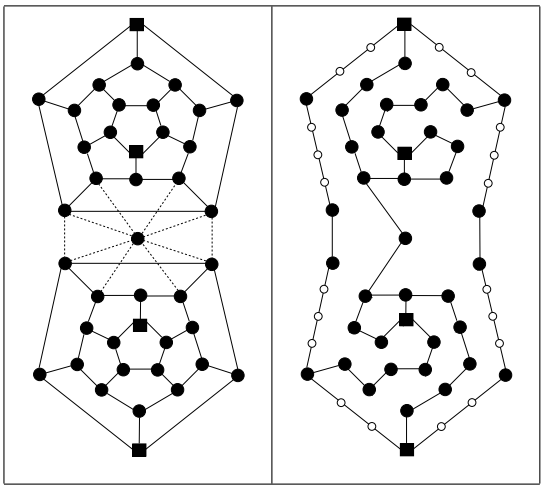


Figure C.1: Initial network for Network 3 and the primary optimal solution.

- Network 4 has 41 fixed nodes, 68 Steiner nodes and 383 edges. This network was designed taking as basis four Brinkman sub-graphs (Four Brinkman graphs with two edges less on each one) interconnected according to Figure 4.16. The Brinkman graph is 4-regular, 4-connected and of girth at least 5. In Figure 4.16 we show the built topology; the edges of the Brinkman sub-graphs are represented by continue lines and the other connections are represented by broken lines. This instance was formulated as a $\mathrm{NCON}(\cdot)$ problem [78, 126], which is a particular case of the GSP-NC, where each fixed node $i \in S_D^{(I)}$ is labeled with a positive integer number $r_i$, and the aim is to find a minimum cost sub-network so that for every pair of fixed nodes $i, j \in S_D^{(I)}$ there exists at least $r_{ij} = \min\{r_i, r_j\}$ node-disjoint paths. We have 10 fixed nodes with $r_i = 4$, 13 fixed nodes with $r_i = 3$ and 18 fixed nodes with $r_i = 2$. We know a global optimal solution of cost 3980 which is shown in Figure C.2 and we will call it the "primary optimal solution". Taking as basis the primary solution and applying $H - path$ and $u - tree$ insertions, we "rebuilt" the topology of the original instance, and like in the previous instance we assigned costs to the edges in order to keep the optimality of the primary

solution. For this, we used the propositions C.0.10, C.0.11, C.0.13 and C.0.14. In the assignment cost process of the edges, new optimal feasible solutions may eventually come up. The edge costs were selected within the $[1, 200]$ range.

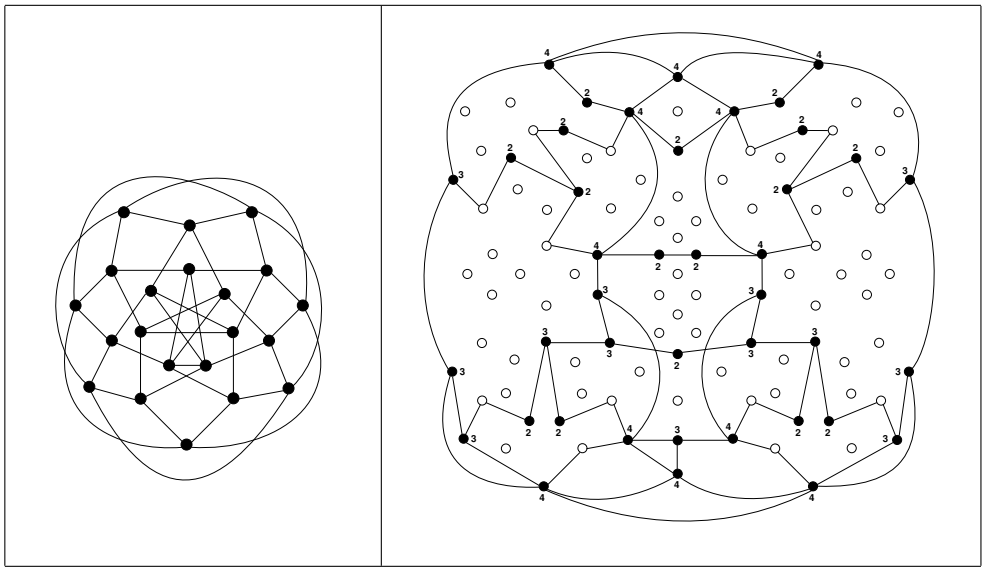


Figure C.2: Brinkman graph and the primary solution for Network 4.

- Network 5 has 27 fixed nodes, 94 Steiner nodes and 386 edges. The design of this instance was made from a great cycle having only the set of fixed nodes. Then, applying splitting operations on this cycle we obtained another cycle which contains some Steiner nodes (specifically 24 Steiner nodes), we call this network; as in the previous case; the "primary optimal solution". Figure C.3 shows its topology. On a second phase, we iteratively added different $H - paths$ and $u - trees$ on the network in construction; in this way new Steiner nodes and new edges were inserted. Our goal is to find a 2-node-survivable sub-network spanning the fixed nodes. The costs associated with the primary solution edges as well as the costs associated with the other edges were suitably selected (according to the properties enunciated above) in order to preserve the optimality of the "primary solution", which has cost 2393. In particular, for this instance, we applied the propositions C.0.9, C.0.11, C.0.12 and C.0.13. Once more, after the edge cost assignment process is finished, new optimal feasible solutions may eventually come up. The edge costs were selected within the $[1, 300]$ range.

- Network 6 has 38 fixed nodes, 33 Steiner nodes and 301 edges. We built this problem from the topology shown in Figure C.4 which only has the fixed nodes. By means of consecutive splitting operations we obtain the topology shown in Figure C.4, which we will take as the "primary optimal solution" and has 16 Steiner nodes. On this network, we iteratively apply some $u - tree$ insertions followed of $H - path$ insertion a certain number of times until Network 6 is formed. In this way, the new Steiner

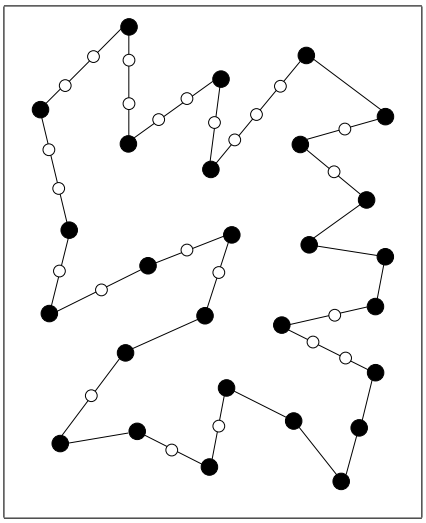


Figure C.3: Primary optimal solution for Network 5.

nodes could be considered in the optimal solution search for the instance. The resultant network is shown in Figure 4.16. There are six square nodes: the set $\{v_1, v_2, v_3\}$ and the set $\{w_1, w_2, w_3\}$. Our objective is to find a minimum cost sub-network whit the following properties:

- there exist at least 3-node-disjoint paths between the fixed nodes $v_i$ and $w_i$, $i \in 1..3$;
- there exist at least 2-node-disjoint paths between any pair of fixed nodes.

As in the other previous cases, the costs assigned to the network edges satisfy the presented properties guaranteeing therefore the optimality non-loss of the "primary solution", of cost 3041. This process does not restrict the appearance of new optimal feasible topologies for this instance. Particularly, we used the propositions C.0.9, C.0.11, C.0.12, C.0.13 and C.0.14. The costs were selected within the range of $[10, 200]$.

For the instances 3, 4, 5 and 6, we also created other test cases having the same topologies but with other edge costs (selected within more restricted value ranges), preserving always the the optimum values and the optimal primary solutions discussed above. The computational experiments with these additional test cases were similar to those shown in Section 4.6, as the proposed GRASP algorithm found a global optimal solution in every case.

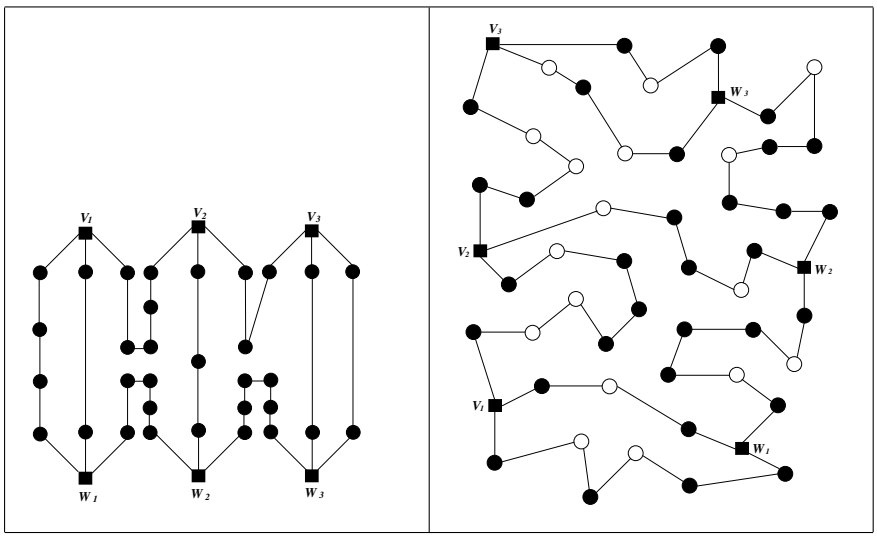



Figure C.4: Initial network for Network 6 and the primary solution.

# Bibliography


[1] A. Agrawal, P. Klein, and R. Ravi. When trees collide: an approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal on Computing*, 24(3):440–456, 1995.

[2] K. Altinkemer and A. Chaturvedi. Neural networks for topological design of local access tree networks. In *Proceeding of Telecommunication Systems, Modelling and Analysis Conference*, pages 256–263, 1993.

[3] K. Altinkemer and Z. Yu. Topological design of wide area communication networks. *Annals of Operations Research*, 36:365–382, 1992.

[4] M. Andrews and L. Zhang. The Access Network Design Problem. In *39th Annual Symposium on Foundations of Computer Science*, pages 40–49, 1998.

[5] V. Auletta, Y. Dinitz, Z. Nutov, and D. Parente. A 2-approximation algorithm for finding an optimum 3-vertex-connected spanning subgraph. *Journal of Algorithms*, 32(1):21–30, 1999.

[6] Mourad Baïou. *Le problème du suos-graphe Steiner 2-arête connexe: approache polyédrale*. PhD thesis, Université de Rennes I, 1996.

[7] H. Bakircioglu and T. Koçak. Survey of random neural network applications. *European Journal of Operational Research*, 126:319–330, 2000.

[8] A. Balakrishnan, T. Magnanti, and P. Mirchandi. A dual-based algorithm for multi-level network design. *Management Science*, 40:567–581, 1994.

[9] A. Balakrishnan, T. Magnanti, and P. Mirchandi. Connectivity-splitting models for survivable network design. *Networks*, 43(1):10–27, 2004.

[10] A. Balakrishnan, T.L. Magnanti, and P. Mirchandani. Modeling and heuristic worst-case performance analysis of the two-level network design problem. *Management Science*, 40(7):846–867, 1994.

[11] A. Balakrishnan, T.L. Magnanti, and P. Mirchandani. Designing hierarchical survivable networks. *Operations Research*, 46(1):116–136, 1998.

[12] R.T. Berger and S. Raghavan. Long-Distance Access Network Design. *Management Science*, 50:309–325, 2004.

[13] D. Bienstock, E.F. Brickell, and C.L. Monma. On the structure of minimum weight k-connected spanning networks. *SIAM Journal on Discrete Mathematics*, 3(3):320–329, 1990.

[14] H.J. Böckenhauser, D. Bongartz, J. Hromkovič, R. Klasing, G. Proietti, S. Seibert, and W. Unger. On the hardness of constructing minimal 2-connected spanning subgraphs in complete graphs with sharpened triangle inequality. In *Proceedings of the 22nd Conference Kanpur on Foundations of Software Technology and Theoretical Computer Science*, pages 59–70. Springer-Verlag, 2002.

[15] Z.R. Bogdanowicz. A new optimal algorithm for backbone topology design in communications networks. *Mathematical and Computer Modelling*, 17(8):49–61, 1993.

[16] R.R. Boorstlyn and H. Frank. Large-scale network topological optimization. *IEEE Transactions on Communications*, 25(1):29–47, 1977.

[17] H. Cancela, F. Robledo, and G. Rubino. A GRASP algorithm for designing a Wide Area Network backbone. In *Proceedings of the International Network Optimization Conference (INOC'03)*, pages 138–143, Evry/Paris, France, October 2003.

[18] H. Cancela, F. Robledo, and G. Rubino. Network design with node connectivity constraints. In *Proceedings of the IFIP/ACM Latin America Networking Conference (LANC'03)*, pages 13–20, La Paz, Bolivia, October 2003.

[19] H. Cancela, F. Robledo, and G. Rubino. Finding Steiner trees with degree 1 terminal nodes. *IEICE Electronics Express (ELEX)*, 1(9):258–262, 2004.

[20] H. Cancela, F. Robledo, and G. Rubino. A GRASP algorithm with RNN based local search for designing a WAN access network. In *Electronic Notes in Discrete Mathematics - special issue including the Proceedings of the Latin-American Conference on Combinatorics, Graphs and Applications (LACGA'04)*, volume 18C, pages 53–58, 2004.

[21] H. Cancela, F. Robledo, and G. Rubino. A GRASP algorithm with tree based local search for designing a Wide Area Network backbone. *Journal of Computer Science and Technology*, 4(1):52–58, 2004.

[22] H. Cancela, F. Robledo, and G. Rubino. Designing low-cost access network topologies. In *Proceeding of the International Network Optimization Conference (INOC'05)*, University of Lisbon, Portugal, October 2005.

[23] H. Cancela, F. Robledo, and O. Viera. A parallel algorithm for the Steiner 2-edge-survivable network problem. *Journal of ICHIO (Chilean Institute of Operations Research)*, 9 (to appear), 2004.

[24] S. Chamberland and B. Sansò. On the design problem of multitechnology networks. *INFORMS Journal on Computing*, 13(3):245–256, 2001.

[25] S. Chamberland, B. Sansò, and O. Marcotte. Topological design of two-level telecommunication networks with modular switches. *Operations Research*, 48(5):745–760, 2000.

[26] N.G. Chattopadhyay, T.W. Morgan, and A. Ranghuram. An innovative technique for backbone network design. *IEEE Transactions on Systems, Man and Cybernetics*, 19(5):1122–1132, 1989.

[27] Zhi-Zhong Chen. Approximating unweighted connectivity problems in parallel. *Information and Computation*, 171:125–126, 2001.

[28] Sheng-Tzong Cheng. Topological optimization of a reliable communication network. *IEEE Transactions on Reliability*, 47(3):225–233, 1998.

[29] J. Cheriyan, T. Jordan, and Z. Nutov. On rooted node-connectivity problems. *Algorithmica*, 30(3):353–375, 2001.

[30] J. Cheriyan, A. Sebő, and Z. Szigeti. Improving on the 1.5-approximation of a smallest 2-edge connected spanning subgraph. *SIAM Journal on Discrete Mathematics*, 14(2):170–180, 2001.

[31] J. Cheriyan and R. Thurimella. Approximating minimum-size k-connected spanning subgraphs via matching. *SIAM Journal on Computing*, 30:528–560, 2000.

[32] J. Cheriyan, S. Vempala, and A. Vetta. Approximation algorithms for minimum-cost k-vertex-connected subgraphs. In *Proceedings of the 34th Annual ACM Symposium on the Theory of Computing*, pages 306–312, 2002.

[33] S. Chopra. Polyhedra of the equivalent subgraph problem and some edge connectivity problems. *SIAM Journal on Discrete Mathematics*, 5(3):321–337, 1992.

[34] W. Chou and H. Frank. Survivable communication networks and the terminal capacity matrix. *IEEE Transactions on Circuit Theory*, 17:192–197, 1970.

[35] C. Christofides and C. A. Whitlock. Network synthesis with connectivity constraints-a survey. *Operational Research*, 81:705–723, 1981.

[36] N. Christofides and C.A. Whitlock. An algorithm for the design of optimal invulnerable networks. Technical Report IC-OR-81-6, Imperial College, London, 1981.

[37] R. Coullard, A. Rais, R.L. Rardin, and D.K. Wagner. Linear-time algorithm for the 2-connected Steiner subgraph problem on special classes of graphs. Technical Report No. 91-25, School of Industrial Engineering, Purdue University, 1991.

[38] F.R.B. Cruz, J. MacGregor Smith, and G.R. Mateus. Algorithms for a multi-level network optimization problem. *European Journal of Operational Research*, 118:164–180, 1999.

[39] B. Csaba, M. Karpinski, and P. Krysta. Approximability of dense and sparse instances of minimum 2-connectivity, TSP and path problems. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 74–83, 2002.

[40] A. Czumaj and A. Lingas. On approximability of the minimum-cost k-connected spanning subgraph problem. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 281–290, 1999.

[41] L. Davis, D. Orvosh, A. Cox, and Y. Qiu. A genetic algorithm for survivable network design. In *Proceedings of the Fifth International Conference on Genetic Algorithms (San Mateo, CA, USA)*, pages 408–415, 1993.

[42] M. Poggi de Aragão, C.C. Ribeiro, E. Uchoa, and R.F. Werneck. Hybrid local search for the Steiner problem in graphs. In *Extended Abstracts of the 4th Metaheuristics International Conference (MIC 2001)*, pages 429–433, 2001.

[43] R. Diestel. *Graph theory*. Springer-Verlag, 2 edition, 2000.

[44] T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computionally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

[45] T.A. Feo and M.G.C. Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[46] C.G. Fernandes. A better approximation ratio for the minimum k-edge-connected spanning subgraph problem. *Journal of Algorithms*, 28(1):105–124, 1998.

[47] P. Festa and M.G.C. Resende. An annotated bibliography of GRASP. Technical Report TD-5WYSEW, AT&T Labs Research, 2004.

[48] H. Frank and W. Chou. Connectivity considerations in the design of survivable networks. *IEEE Transactions on Circuit Theory*, 17:486–490, 1970.

[49] L.F. Frantzeskakis and H. Luss. The network redesign problem for access telecommunications networks. *Naval Research Logistics*, 46:487–506, 1999.

[50] G.N. Frederickson and J. Jàjà. On the relationship between biconnectivity augmentation and traveling salesman problem. *Theoretical Computer Science*, 19:189–201, 1982.

[51] H.N. Gabow, M.X. Goemans, and D.P. Williamson. An efficient approximation algorithm for the survivable network design problem. In *Proceedings of the 3rd MPS Conference on Integer Programming and Combinatorial Optimization*, pages 57–74, 1993.

[52] A. Galluccio and G. Proietti. Polynomial time algorithms for edge-connectivity augmentation problems. *Algorithmica*, 36(4):361–374, 2004.

[53] B. Gavish. A general model for the topological design of computer networks. In *GLOBECOM'86 - IEEE International Global Telecommunications Conference*, pages 1584–1588, 1986.

[54] B. Gavish. Topological design of telecommunication networks - local access design methods. *Annals of Operations Research*, 33:17–71, 1991.

[55] B. Gavish. Configuring wide area computer networks-problems and models. *OR Spektrum*, 14(3):115–128, 1992.

[56] B. Gavish. Topological design of computer communication networks-the overall design problem. *European Journal of Operations Research*, 58(2):149–172, 1992.

[57] B. Gavish and K. Altinkemer. Parallel savings heuristics for the topological design of local access tree networks. In *Proceedings of IEEE INFOCOM'86. Fifth Annual Conference on Computers and Communications Integration Design, Analysis, Management*, pages 130–139, 1986.

[58] E. Gelenbe. Random neural network with negative and positive signals and product form solution. *Neural Computation*, 1(4):502–511, 1989.

[59] E. Gelenbe. Stability of the random neural network model. *Neural Computation*, 2(2):239–247, 1990.

[60] E. Gelenbe. Hopfield energy of the random neural network. In *Proceedings of the IEEE World Congress on Computational Intelligence*, volume 7, pages 4681–4686, 1994.

[61] E. Gelenbe and F. Batty. Minimum cost graph covering with the random neural network. *Computer Science and Operations Research. (New York: Pergamon)*, pages 139–147, 1992.

[62] E. Gelenbe, A. Ghanwani, and V. Srinivasan. Improved neural heuristics for multicast routing. *IEEE Journal on Selected Areas in Communications*, 15(2):147–155, 1997.

[63] E. Gelenbe, V. Koubi, and F. Pekergin. Dynamical random neural network approach to the traveling salesman problem. In *Proceedings of the IEEE Symposium on Systems Engineering in the Service of Humans*, pages 630–635. Systems, Man and Cybernetics, 1993.

[64] E. Gelenbe and A. Stafylopatis. Global behaviour of homogeneous random neural systems. *Applied Mathematical Modelling*, 15:534–541, 1991.

[65] A. Ghanwani. A qualitative comparison of neural networks models applied to the vertex covering problem. *Elektrik*, 2(1):11–18, 1994.

[66] L. Ghosh, A. Mukherjee, and D. Saha. Design of 1-ft communication network under budget constraint. In *Proceedings of Distributed Computing. Mobile and Wireless Computing. 4th International Workshop - IWDC 2002 (Calcutta, India)*, pages 300–311, 2002.

[67] L. Ghosh, A. Mukherjee, and D. Saha. Optimal design of backbone topology for a communication network cost constraint. In *Proceedings of ICCC 2002 - 15th International Conference on Computer Communication*, volume 2, pages 471–485, 2002.

[68] A. Girard, B. Sansò, and L. Dadjo. A tabu search algorithm for access network design. *Annals of Operations Research*, 106(1-4):229–262, 2001.

[69] M.X. Goemans and D.J. Bertsimas. Survivable networks, linear programming relaxations and the parsimonious property. *Mathematical Programming*, 60:143–166, 1993.

[70] M.X. Goemans, A.V. Goldberg, S. Plotkin, É. Tardos, and D.P. Williamson. Improved approximation algorithms for network design problems. In *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 223–232, 1994.

[71] M.X. Goemans and D.P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1992.

[72] L. Gouveia and M.J. Lopes. Using generalized capacitated trees for designing the topology of local access networks. *Telecommunications Systems*, 7(4):315–337, 1997.

[73] R. Grimaldi. *Discrete and combinatorial mathematics. An applied introduction*. Addison-Wesley, 1994.

[74] M. Grötschel and C.L. Monma. Integer polyhedra associated with certain network design problems with connectivity constraints. *SIAM Journal on Discrete Mathematics*, 3:502–523, 1990.

[75] M. Grötschel, C.L. Monma, and M. Stoer. Polyhedral Approaches to Network Survivability. In F. Roberts, F. Hwang, and C.L. Monma, editors, *Reliability of Computer and Communication Networks, Proc. Workshop 1989, New Brunswick, NJ/USA*, volume 5 of *Series in Discrete Mathematics and Theoretical Computer Science*, pages 121–141. American Mathematical Society, 1991.

[76] M. Grötschel, C.L. Monma, and M. Stoer. Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints. *Operations Research*, 40(2):309–330, 1992.

[77] M. Grötschel, C.L. Monma, and M. Stoer. Facets for polyhedra arising in the design of communication networks with low-connectivity constraints. *SIAM Journal on Optimization*, 2(3):474–504, 1992.

[78] M. Grötschel, C.L. Monma, and M. Stoer. Polyhedral and computational investigations for designing communications networks with high survivability requirements. *Operations Research*, 43(6):1012–1024, 1995.

[79] Jae gyun Kim and Dong wan Tcha. Optimal design of a two-level hierarchical network with tree-star configuration. *Computers & Industrial Engineering*, 22(3):273–281, 1992.

[80] F. Harary. The maximum connectivity of a graph. In *Proceedings of the National Academy of Sciences (USA)*, volume 48, pages 1142–1146, 1962.

[81] Sung hark Chung, Young soo Myung, and Dong wan Tcha. Optimal design of a distributed network with two-level hierarchical structure. *European Journal of Operational Research*, 62(1):105–115, 1992.

[82] K. Jain. A 3-approximation algorithm for finding optimum 4,5-vertex-connected spanning subgraphs. *Journal of Algorithms*, 32(1):31–40, 1999.

[83] K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21:39–60, 2001.

[84] T. Jordan. On the optimal vertex-connectivity augmentation. *Journal of Combinatorial Theory*, Series B 63:8–20, 1995.

[85] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–104. Plenum Press, NY, 1972.

[86] S. Khuller and B. Raghavachari. Improved approximation algorithms for uniform connectivity problems. *Journal of Algorithms*, 21(2):434–450, 1996.

[87] S. Khuller and R. Thurimella. Approximation algorithms for graph augmentation. *Journal of Algorithms*, 4(2):214–225, 1991.

[88] S. Khuller and U. Vishkin. Biconnectivity approximations and graph carvings. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on the Theory of Computing*, pages 759–770, 1992.

[89] C.W. Ko and C.L. Monma. Heuristics methods for designing highly survivable communication networks. Technical report, Bellcore, 1989.

[90] T. Koch, A. Martin, and S. Voß. SteinLib: An updated library on Steiner tree problems in graphs. Technical Report ZIB-Report 00-37, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustr. 7, Berlin, 2000.

[91] A. Konak and A.E. Smith. A hybrid genetic algorithm approach for backbone design of communication networks. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Washington, DC, USA)*, volume 3, pages 1817–1823, 1999.

[92] G. Kortsarz, R. Krauthgamer, and J. R. Lee. Hardness of approximation for vertex-connectivity network-design problems. *SIAM Journal on Computing*, 33(3):704–720, 2004.

[93] G. Kortsarz and Z. Nutov. Approximating node connectivity problems via set covers. *Algorithmica*, 37(2):75–92, 2003.

[94] M. Kos, M. Mikac, and D. Mikac. Topological planning of communication networks. *Journal of Information and Organizational Sciences*, 26(1-2):57–68, 2002.

[95] P. Krysta and V.S.A. Kumar. Approximation algorithms for minimum size 2-connectivity problems. In *Proceedings of the 18th Annual Symposium Theoretical Aspects of Computer Science (STACS'01, Berlin), Lecture Notes in Computer Science*, volume 2010, pages 431–442. Springer, 2001.

[96] B. Lukic. An approach of designing local access network using Simulated Annealing method. In *ConTEL'99 - 5th International Conference on Telecommunications*, pages 241–247, 1999.

[97] S. Mandal, D. Saha, R. Mukherjee, and A. Roy. An efficient algorithm for designing optimal backbone topology for a communication networks. In *Proceedings of ICCT 2003 - International Conference on Communication Technology (Beijing, China)*, volume 1, pages 103–106, 2003.

[98] S.L. Martins, M.G.C. Resende, C.C. Ribeiro, and P.M. Pardalos. A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. *Journal of Global Optimization*, 17:267–283, 2000.

[99] G.R. Mateus and R.V.L. Franqueira. Model and heuristics for a generalized access network design problem. *Telecommunication Systems - Modelling, Analysis, Design and Management*, 15(3-4):257–271, 2000.

[100] T. Mavridou, P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A GRASP for the bicuadratic assignment problem. *European Journal of Operational Research*, 105:613–621, 1998.

[101] M. Minoux. Efficient greedy heuristics for Steiner tree problems using reoptimization and supermodularity. *INFOR*, 28:221–233, 1990.

[102] C.L. Monma, B.S. Munson, and W.R. Pulleyblank. Minimum-weight two connected spanning networks. *Mathematical Programming*, 46:153–171, 1990.

[103] C.L. Monma and D.F. Shallcross. Methods for designing communication networks with certain two-connected survivability constraints. *Operations Research*, 37:531–541, 1989.

[104] Z. Nutov and M. Penn. Faster approximation algorithms for weighted triconnectivity augmentation problems. *Operations Research Letters*, 21:219–223, 1997.

[105] P.M. Pardalos, T. Qian, and M.G.C. Resende. A greedy randomized adaptive search procedure for the feedback vertex set problem. *Journal of Combinatorial Optimization*, 2:399–412, 1999.

[106] Michael Penn and Haya Shasha-Krupnik. Improved approximation algorithms for weighted 2- and 3- vertex connectivity augmentation problems. *Journal of Algorithms*, 22:187–196, 1997.

[107] S. Pierre and A. Elgibaoui. A tabu-search approach for designing computer-network topologies with unreliable components. *IEEE Transactions on Reliability*, 46(3):350–359, 1997.

[108] M. Priem and F. Priem. *Ingénierie des WAN (text in French)*. Dunod InterEditions, 1999.

[109] J.S. Provan and R.C. Burk. Two-connected augmentation problems in planar graphs. *Journal of Algorithms*, 32:87–107, 1999.

[110] C.D. Randazzo and H.P.L. Luna. A comparison of optimal methods for local access uncapacited network design. *Annals of Operations Research*, 106:263–286, 2001.

[111] C.D. Randazzo, H.P.L. Luna, and P. Mahey. Benders decomposition for local access network design with two technologies. *Discete Mathematics & Theoretical Computer Science*, 4:235–246, 2001.

[112] R. Ravi and P.N. Klein. When cycles collapse: a general approximation technique for constrained 2-connectivity problems. In *Proceedings of the 3rd Symposium on Integer Programming and Combinatorial Optimization*, pages 39–55, 1993.

[113] R. Ravi and D.P. Williamson. An approximation for minimum-cost vertex-connectivity problems. *Algorithmica*, 18(1):21–43, 1997.

[114] R. Ravi and D.P. Williamson. Erratum: An approximation for minimum-cost vertex-connectivity problems. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1000–1001, 2002.

[115] G. Reinelt. TSPLIB library. http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsplib.html, 2004.

[116] M.G.C. Resende. Computing approximate solutions of the maximum covering problem using GRASP. *Journal of Heuristics*, 4:161–171, 1998.

[117] M.G.C. Resende and C.C. Ribeiro. A GRASP for graph planarization. *Journal of Heuristics*, 4:161–171, 1998.

[118] M.G.C. Resende and C.C. Ribeiro. Greedy Randomized Adaptive Search Procedures. Technical Report TD-53RSJY, AT&T Labs Research, 2002.

[119] C.C. Ribeiro, E. Uchoa, and R.F. Werneck. A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing*, 14(3):228–246, 2002.

[120] F. Robledo. Diseño topológico de redes; casos de estudio: the Generalized Steiner Problem and the Steiner 2-Edge-Connected Subgraph Problem (text in Spanish). Master Thesis, Universidad de la República-Facultad de Ingeniería, J. Herrera y Reissig 565, Montevideo, Uruguay, 2000.

[121] F. Robledo. A parallel algorithm for the Steiner 2-edge-survivable network problem. Technical Report PI 1504, IRISA/INRIA, 2002.

[122] F. Robledo. A GRASP algorithm with MST based local search for designing a WAN access network. In *Proceedings of the 6 ème Journées Doctorales Informatique et Réseau (JDIR'04)-France Télécom R&D*, Lannion, France, November 2004.

[123] F. Robledo, R. Maba, I. Manzo, and D. Nachman. Using GRASP for designing low-cost access topologies. In *International Conference on Industrial Logistics (ICIL'05)*, Universidad de la República-Facultad de Ingeniería, Montevideo, Uruguay, February 2005.

[124] I. Rosseti, M. Poggi de Aragão, C.C. Ribeiro, E. Uchoa, and R.F. Werneck. New benchmark instances for the Steiner problem in graphs. In *Extended Abstracts of the 4th Metaheuristics International Conference (MIC 2001)*, pages 557–561, 2001.

[125] K. Steiglitz, P. Weiner, and D.J. Kleitman. The design of minimum-cost survivable networks. *IEEE Transactions on Circuit Theory*, 16:455–460, 1969.

[126] M. Stoer. *Design of survivable networks*, volume 1531 of *Lecture Notes in Mathematics*. Springer-Verlag, 1992.

[127] H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Mathematica Japonica*, 24:537–577, 1980.

[128] T. Thomadsen and J. Clausen. Hierarchical network design using simulated annealing. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 305, DK-2800 Kgs. Lyngby, sep 2002.

[129] L. Tran and P.A. Beling. A heuristic for the topological design of two-tiered networks. In *SMC'98 - IEEE International Conference on Systems, Man, and Cybernetics*, volume 3, pages 2962–2967, 1998.

[130] S. Vempala and A. Vetta. Factor 4/3 approximations for minimum 2-connected subgraphs. In *Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (Berlin, Germany)*, pages 262–273, 2000.

[131] M.G.A. Verhoeven and M.E.M. Severens adn E.H.L. Aarts. Local search for Steiner trees in graphs. In V.J. Rayward Smith, I.H. Osman, C.R. Reeves, and G.D. Smith, editors, *Modern Heuristics Search Methods*, pages 117–129. Jhon Wiley, 1996.

[132] T. Watanabe and A. Nakamura. A minimum 3-connectivity augmentation of a graph. *Joournal of Computer and System Sciences*, 46(1):91–128, 1993.

[133] D.P. Williamsom, M.X. Goemans, M. Mihail, and V.V. Vazirani. A primal-dual approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 15:435–454, 1995.

[134] P. Winter. Generalized Steiner problem in outerplanar graphs. *BIT*, 25(3):485–496, 1985.

[135] P. Winter. Generalized Steiner problem in series-parallel networks. *Journal of Algorithms*, 7:549–566, 1986.

[136] P. Winter. Steiner problem in networks: A survey. *Networks*, 17(2):129–167, 1987.

[137] J.Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17:712–716, 1971.