Università degli Studi di Bologna

SCUOLA DI INGEGNERIA E ARCHITETTURA
Corso di Laurea Magistrale in Telecommunications Engineering
Optimization Models and Algorithms M

Scalable Algorithms for Cloud Radio Access Network (C-RAN) Optimization

TESI DI LAUREA DI:
NICOLA DI CICCO

RELATORE:
Prof.ssa VALENTINA CACCHIANI

CORRELATORE:
Prof.ssa CARLA RAFFAELLI

Anno Accademico 2020/21

Contents

Abstract

In the evolving scenario of 5G end-to-end networks, the Cloud Radio Access Network (C-RAN) model has proven to be the key for managing ever increasing Capital Expenditure (CAPEX) and Operating Expenditure (OPEX) for mobile networks while ensuring high Quality of Service (QoS). In a framework of end-to-end services with stringent QoS requirements, it is of critical importance to design resource allocation algorithms for the C-RAN that are not only able to utilize efficiently the resources made available by the network, but that are also fast enough for network reconfiguration in dynamic traffic scenarios.

In Chapter 1 a brief overview of the main elements of the C-RAN and of the methodologies that are employed in this work is provided.

In Chapter 2, an exact scalable methodology for a static traffic scenario, based on lexicographic optimization, is proposed for the solution of a multi-objective optimization problem to achieve, among other goals, the minimization of the number of active nodes in the C-RAN while supporting reliability and meeting latency constraints. The optimal solution of the most relevant objectives for networks of several tens of nodes is obtained in few tens of seconds of computational time in the worst case. For the least relevant objective, a simple heuristic is developed in order to provide near optimal solutions in few seconds of computing time. This Chapter details and expands the work that has been presented in [8].

In Chapter 3, an optimization framework for dynamic C-RAN reconfiguration is developed. The objective is to maintain C-RAN cost optimization, while minimizing the cost of virtual network function migration. Significant savings in terms of migrations (above 82% for primary virtual BBU functions and above 75% for backup virtual BBU functions) can be obtained with respect to a static traffic scenario, with execution time of the optimization algorithm below 20 seconds in the worst cases, making its application feasible for dynamic scenarios. This Chapter details the work that has been developed for [7].

In Chapter 4, an alternative Column Generation model formulation is developed, and the quality of the computed lower bounds is evaluated. Further extensions from this baseline (e.g. heuristics based on Column Generation, exact Branch&Price algorithms) are left as future work.

In Chapter 5, the main results achieved in this work are summarized, and several possible extensions are proposed.

Chapter 1

Introduction

In this chapter, the motivations behind this work are presented. In particular, an overview of the considered application scenario, that is the Cloud Radio Access Network (C-RAN) paradigm, is outlined. Finally, the main methodologies that will be used in this work, which are going to be declined in the network optimization and planning context in the following chapters, are briefly described.

1.1 Cloud Radio Access Network (C-RAN)

In the context of heterogeneous, ever rising mobile traffic, network operators need to upgrade their infrastructure in order to cope with ownership costs. Relevant figures of merit are the Capital Expenditure (CAPEX) and Operating Expenditure (OPEX) of a network. Briefly, CAPEX accounts for all the costs in building the network, such as site planning and acquisition, hardware/software deployment, and so on. OPEX accounts for all the costs needed to operate the network, such as ordinary and extraordinary maintenance, energy consumption, hardware/software upgrades, and so on [14]. In particular, the largest contribution in CAPEX is given by base stations, which are the most expensive components, while OPEX increases as cell sites requires a considerable amount of energy for operating: for instance, China Mobile estimates that cell sites contribute up to 72% to the total energy consumption [5]. Therefore, novel networking paradigms need to be developed for coping with ever increasing ownership costs and tight Quality of Service (QoS) requirements.

The root of the C-RAN model lies in the full decoupling of baseband functions from radiofrequency functions, in the form of Baseband Units (BBUs) from the Remote Radio Units (RRUs) [5]. In particular, the BBUs are virtualized and centralized in general-purpose processing nodes, called BBU Hotels, which are able to process traffic pertaining to different RRUs (Figure 1.1). An evolution of the pure C-RAN scheme is represented by the Next Generation Fronthaul Interface that introduces packet-based interconnection and further functional split in the optical transport network [11]. The advantages provided by such an architecture are manyfold.

First, centralization of BBUs implies that not all transport nodes able to do processing need to be active at the same time. This can lead to tremendous savings in

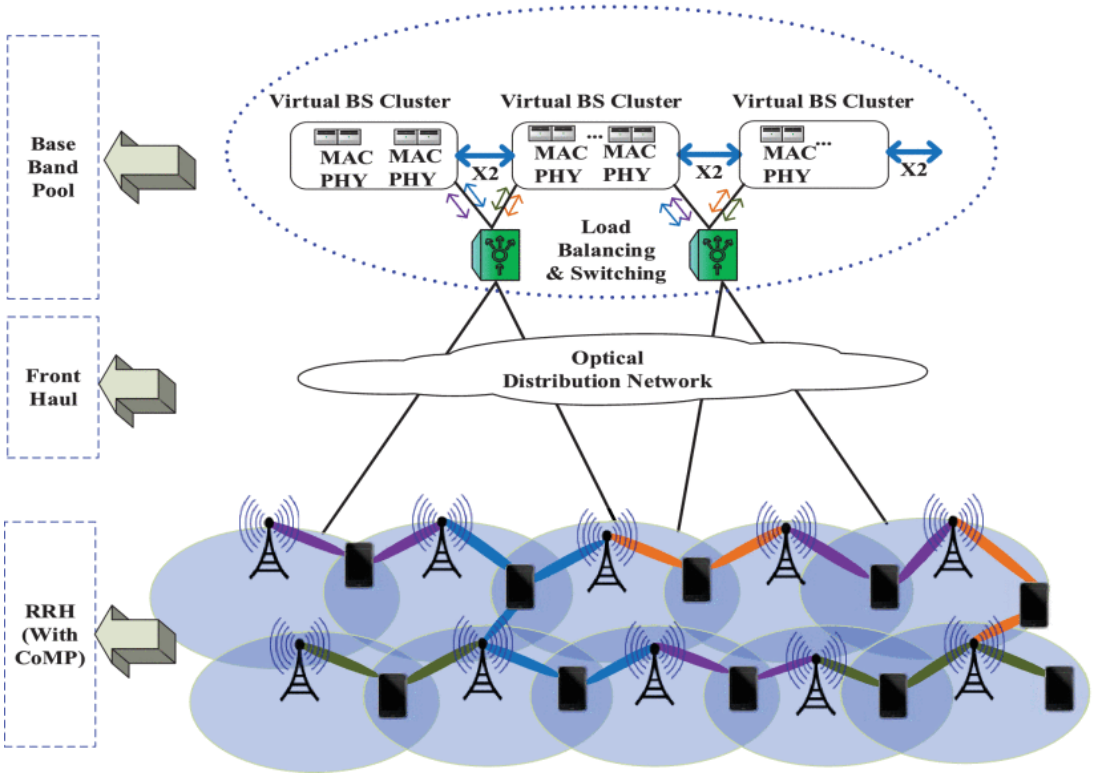


Figure 1.1: Cloud Ran (C-RAN) network model for 5G [2].

terms of energy consumption and computing power utilization (up to 80% with respect to traditional RAN), as only a fraction of the whole set of transport nodes needs to be activated [4]. This feature makes the C-RAN scalable with respect to the ever increasing traffic volume in mobile networks.

Second, virtualization of the BBUs allows dynamic reconfiguration of the virtual network topology in case of dynamic, nonuniform traffic. In the traditional RAN, a sizeable amount of energy would get wasted during time periods in which the offered mobile traffic was considerably lower than the available network capacity. With the C-RAN, BBU Hotels can be instantiated and reconfigured according to the time and space distribution of the mobile traffic, with better network capacity utilization [5]. In particular, one can define the Multiplexing Gain of the C-RAN network versus a traditional RAN as follows:

$$\text{Multiplexing Gain} = \frac{C_{\text{BS, tot}}}{C_{\text{Hotels, tot}}} = \frac{\sum_{i=1}^{N_{\text{BS}}} C_{\text{BS},i}}{\sum_{j=1}^{N_{\text{Hotels}}} C_{\text{Hotel},j}} \tag{1.1}$$

Where $C_{\text{BS, tot}}$ is the total capacity utilized by the Base Station in the traditional RAN, whereas $C_{\text{Hotels, tot}}$ is the total capacity required by the BBU Hotels in the C-RAN. Generally, it holds that $C_{\text{Hotels, tot}} < C_{\text{BTS, tot}}$, meaning that the multiplexing gain accounts for the more efficient utilization of available network resources in the same offered traffic conditions.

Third, centralization of BBU functions allows to perform also radio functions in a more optimized way. For instance, in the case of handover between RRUs managed by the same BBU Hotel, there is no need of signaling between base sta-

tions: virtualized baseband functions can communicate directly within the centralized compute unit, reducing delays. Another application example can be interference management via Coordinated Multipoint Transmission and Reception (CoMP): if all RRUs within a CoMP are managed by the same BBU Hotel, significant throughput gains can be achieved thanks to the tighter interaction among baseband functions [2] [3].

One of the main challenges in C-RAN concerns the huge overhead imposed on the fronthaul links connecting BBU Hotels and RRUs: indeed, traffic needs to be aggregated at the BBU Hotels. A possible solution for fronthaul connectivity would be the utilization of dark fibers employed in a circuit switched way. Although dark fiber deployment is of relatively low cost, network extensibility might be troublesome as significant fiber resources could be needed. Wavelength Division Multiplexing (WDM) schemes can be envisaged in scenarios with fiber scarcity, for which 40-80 wavelengths per links can be multiplexed in a single fiber, with the drawback of a higher upgrade cost with respect to dark fibers [4].

In the evolving scenario of end-to-end service provisioning in 5G networks, three main service classes have been defined: Massive Machine Type Communications (MMTC), Ultra Reliable Low Latency Communications (URLLC) and Enhanced Mobile Broadband (EMBB). For the URLLC service class in particular, tight constraints in terms of latency, aiming at sub-ms latency, and reliability, aiming at 99.999% ("five nines") reliability, must be satisfied [1]. Virtual resource allocation in C-RAN quickly becomes a complex issue as service constraints are taken into account in the network optimization and planning problem. In particular, in order to account to such requirements, proper redundancy of the BBU Hotels for robustness to hardware/software failures and latency constraints need to be taken into account in network optimization and planning models, adding a further degree of complexity to the problem [12] [18] [21].

1.2 Integer Linear Programming

Integer Linear Programming (ILP) problems are a relevant class of problems for network optimization and planning. In an ILP optimization problem, given a set of integer decision variables, the aim is to minimize/maximize an objective function under a set of linear constraints. A general formulation for an ILP optimization problem can be expressed as follows:

$$\begin{aligned} & \min \mathbf{c}^T \cdot \mathbf{x} \\ \text{subject to} \quad & \mathbf{A} \cdot \mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq 0, \text{ integer} \end{aligned}$$

Where $\mathbf{c} \in \mathbb{R}^n$ is the cost vector, $\mathbf{x} \in \mathbb{R}^n$ are the decision variables, $\mathbf{A} \in \mathbb{R}^{m \times n}$ is the constraint matrix, and $b \in \mathbb{R}^m$ is the right-hand side vector of the constraints. Integrality constraints on the decision variables are equivalent to nonlinear constraints in the form $\sin(\pi \cdot x) = 0$. Many relevant problems in network optimization and planning can be formulated as ILP, such as the Routing and Wavelength Assignment Problem (RWA) [20], the traffic grooming problem in Elastic Optical

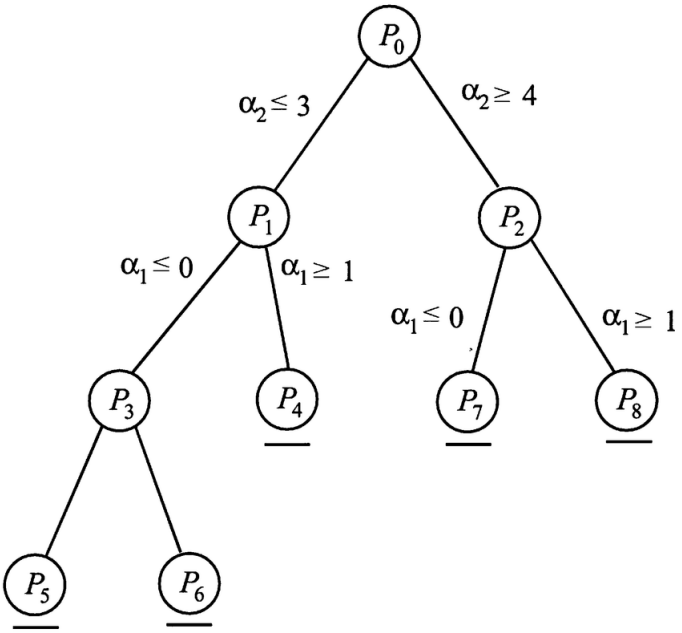


Figure 1.2: Branch-and-Bound decision tree.

Networks (EON) [23], and the optimal BBU Hotel placement problem that will be studied in this work. In general, computational complexity of ILP problems in optimization form is NP-Hard, therefore handcrafted heuristics are often used to obtain good quality solutions in reasonable computing time.

Often, exact solution methods for ILPs rely on the Branch-and-Bound algorithm, which adopts a divide-and-conquer approach by dividing (i.e. "branching") the original problem into smaller subproblems and by solving their linear relaxation. Branch-and-Bound behaves as an enumerative algorithm, for which non-optimal solutions are discarded (i.e. "bounding") according to the computed lower bounds value. Execution of the Branch-and-Bound algorithm can be figuratively represented by a decision tree, as shown in Figure 1.2.

Examples of solvers for ILP optimization problems implementing the Branch-

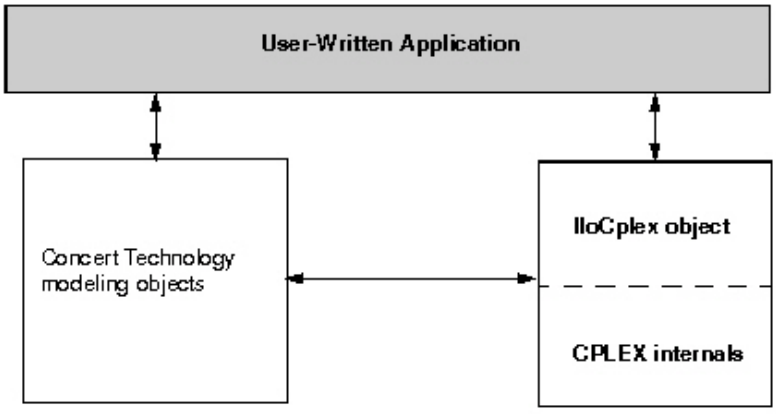


Figure 1.3: Concert Technology application high-level architecture.

and-Bound algorithm are CPLEX, Gurobi, XPress, SCIP. It is important to note that, in addition to the core Branch-and-Bound, commercial solvers include a plethora of complex subroutines such as cutting planes, presolve (transformation of the original problem in an equivalent, easier one), primal heuristics (heuristic algorithms executed in promising subproblems that may yield integer solutions), whose behaviour can be tuned via several parameters.

In this work, the commercial solver CPLEX via its Concert Technology C++ API has been used, which provides a library of classes and functions for optimization. The high-level architecture of a Concert Technology application is illustrated in 1.3. In summary, modelling objects are provided for building the optimization problems. The model is passed to IloCplex class objects, which solve the problem by interfacing with the CPLEX internals, and can then be queried for extracting solution information.

1.3 Lexicographic Optimization

A generic Multi-Objective optimization problem can be expressed as follows:

$$\begin{aligned} &\min \mathbf{f}(\mathbf{x}) = \min\left(f_1(\mathbf{x}), \ldots, f_n(\mathbf{x})\right) \\ &\text{subject to} \quad \mathbf{x} \in \mathbf{X} \end{aligned}$$

where $\mathbf{X}$ denotes the feasible set.

Since in general it is not possible to minimize all objectives simultaneously, the fundamental concept for defining optimality in multi-objective optimization problems is the one of Pareto optimality. In particular, a solution $\mathbf{x}^*$ is said to be Pareto optimal if there does not exist $\mathbf{x} \in \mathbf{X}$ such that:

$$\mathbf{f}(\mathbf{x}) \leq \mathbf{f}(\mathbf{x}^*) \;\text{ with }\; f_i(x^*) < f_i(x) \text{ for some } i \tag{1.2}$$

meaning that no objective can be further improved without worsening some other objectives. Pareto optimal solutions lie in the Pareto Frontier within the objective functions space, as illustrated in Figure 1.4 for an example in two dimensions with two objective functions.

A-priori without any domain knowledge the whole Pareto frontier should be explored for finding the best trade-off among objectives, which can be in many cases computationally cumbersome. On the other hand, if a ranking of importance between the objective functions is made available by the application scenario, the Lexicographic method can be applied [9].

Assume that objectives are ranked in importance such that minimization of $f_i(\mathbf{x})$ is infinitely more important than minimization of $f_{i+1}(\mathbf{x}),\ i = 1, \ldots, n-1$. Objectives ranked in this way are said to be lexicographically ordered. The Lexicographic method consists in solving a sequence of n single-objective optimization problems in the following form:

$$\min f_j(\mathbf{x}) \tag{1.3}$$

$$\text{subject to} \quad \mathbf{x} \in \mathbf{X} \tag{1.4}$$

$$f_i(\mathbf{x}) \leq \mathbf{y}_i^* \quad \forall i < j \tag{1.5}$$

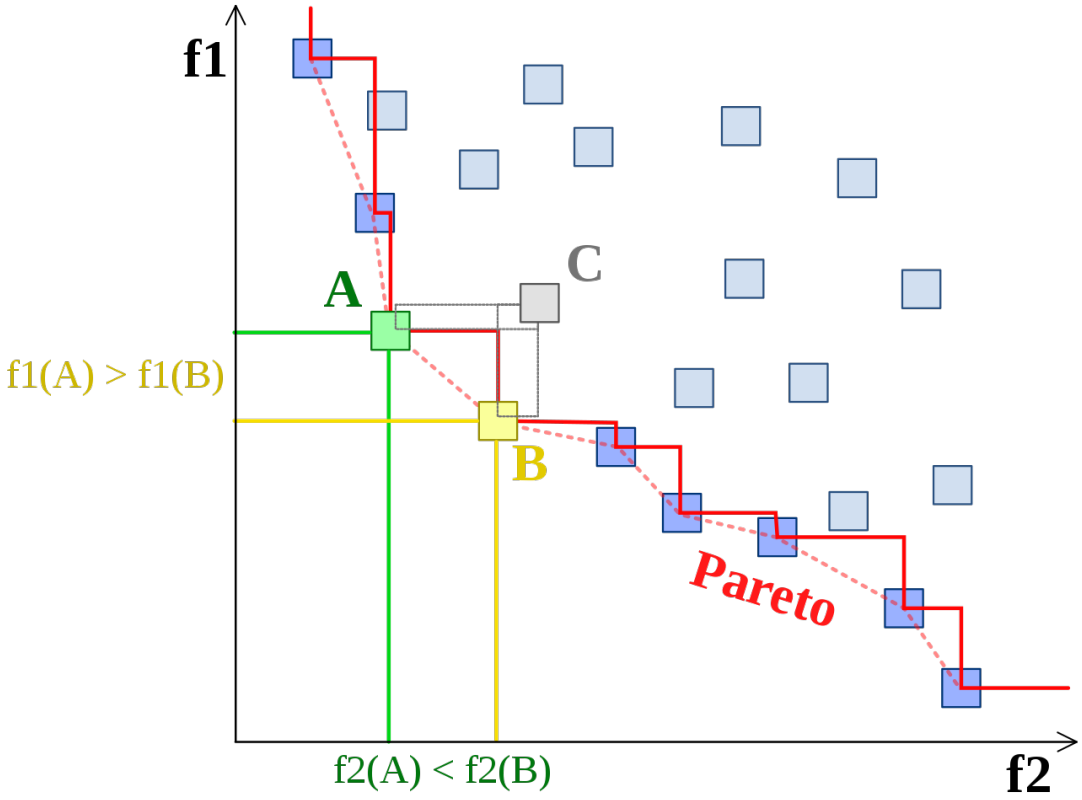


Figure 1.4: Pareto frontier for a minimization problem.

where y_i^* is the optimal solution value for the i-th single-objective problem. The above formulation means that the objectives are solved independently in order of importance, and subsequent single-objective optimization problems are constrained in order not to worsen the solutions computed for the previous higher priority objectives. Such an approach is said to be non-scalarized, since the objective functions are not multiplied by scalars according to their importance, avoiding potential numerical issues due to very large weights. It is important to note that the solution found by the Lexicographic method shall be within the subset of Pareto optimal solutions that also satisfy the lexicographic ordering.

With respect to a scalarized approach, where objective functions are multiplied by weights according to their priority and summed, decoupling the objectives has the potential of decomposing a single "harder to solve" multi-objective optimization problem into several "simpler to solve" single-objective optimization problems. This consideration however is not general, and there may be problems or instances for which the application of the Lexicographic method is not beneficial. For instance, it may happen that even though the computational time for solving a single-objective problem is smaller than the one for solving the multi-objective problem, the overall running time for solving all the single-objective problems may be actually larger. This depends on the particular problem or instance at hand, and needs to be evaluated a-posteriori. In addition, the lexicographic approach is only possible when a clear ranking of the objectives is available. Previous applications of Lexicographic optimization in telecommunications problems can be found in literature, such as in [16] [10].

1.4 Column Generation

Column Generation is an algorithm used to solve Linear Programming (LP) problems with an exponential number of variables (e.g. linear relaxation of the Vehicle Routing Problem in Set Partitioning formulation). Consider a generic LP formulation:

$$\begin{aligned} &\min \mathbf{c}^T \cdot \mathbf{x} \\ &\text{subject to} \quad \mathbf{A} \cdot \mathbf{x} \geq \mathbf{b} \\ &\qquad\qquad \mathbf{x} \geq 0 \end{aligned}$$

Where $\mathbf{c} \in \mathbb{R}^n$ is the cost vector, $\mathbf{x} \in \mathbb{R}^n$ are the decision variables, $\mathbf{A} \in \mathbb{R}^{m \times n}$ is the constraint matrix, and $b \in \mathbb{R}^m$ is is the right-hand side vector of the constraints. The dual formulation for the above LP problem is the following:

$$\begin{aligned} &\max \mathbf{y}^T \cdot \mathbf{b} \\ &\text{subject to} \quad \mathbf{y}^T \cdot \mathbf{A} \leq \mathbf{c}^T \\ &\qquad\qquad \mathbf{y} \geq 0 \end{aligned}$$

Where $\mathbf{y} \in \mathbb{R}^m$ are the dual variables.

It is assumed that number of decision variables in $\mathbf{x}$ is very large. In the worst case, the number of variables is exponential, therefore it would be impossible to fit the model within the computer memory or it would be computationally expensive for the solver to deal with this large set of variables. The Column Generation algorithm initializes the model with a subset $\tilde{\mathbf{x}}$ of the variables, solving a reduced problem. Optimality of the computed solution is given by verifying feasibility for the dual problem. Each variable in the primal problem corresponds to a constraint in the dual, therefore non-optimal solution for the restricted primal problem will result infeasible for the dual. The Column Generation algorithm is outlined as follows:

1. Initialize the problem with a subset of columns of $\tilde{\mathbf{A}}$, $\tilde{\mathbf{x}}^T$, $\tilde{\mathbf{c}}^T$
2. Solve the reduced primal, get the dual variables $\mathbf{y}^*$
3. If $\mathbf{y}^*$ does not violate any constraint in $\mathbf{y}^T \cdot \mathbf{A} \leq \mathbf{c}^T$ then STOP. Otherwise, add to the primal one or more columns associated with violated dual constraints, and GOTO 2.

Note that, given dual variables $\mathbf{y}^*$ associated with the solution of the restricted primal, finding a violated constraint is equivalent to finding a column x_j with constraint coefficients defined by $\mathbf{A}_j$ such that $\mathbf{c}^T - \mathbf{y}^{*T} \cdot \mathbf{A}_j < 0$. This expression is the reduced cost associated with variable x_j, which represents the change in the primal objective function value per unit increase of x_j. Therefore finding a violated dual constraint is equal to finding a column x_j with negative reduced cost, corresponding to a variable that, if added to the restricted problem, can improve the objective function value.

Chapter 2

Static Traffic Scenario

In this chapter, an exact algorithm for the optimal BBU Hotel placement problem, accounting for reliability and latency constraints, is developed for a static traffic scenario. This algorithm employs the framework of lexicographic optimization, which relies on the priority among the objectives, dictated by the considered application scenario. Along with the mathematical formulation of the problem, a sample C++ code implementation will be provided for modelling and solving the problem via CPLEX Concert Technology. The performance of this algorithm is compared with an aggregate optimization approach. The lexicographic optimization approach finds the optimal solution for the most relevant objectives in networks of several tens of nodes, and computes much better solutions than the aggregate one for the largest network of 100 nodes. This Chapter details the work that has been presented in [8].

2.1 Problem Statement

The main elements of a transport networks to support C-RAN are the nodes, the ports within nodes and the wavelengths on each link. As discussed in the previous chapter, the C-RAN model allows to centralize BBU functions in a few nodes, thus reducing the number of nodes in the transport network that needs to be activated, which means sizeable cost savings in terms of power consumption and network management. At the same time a larger number of wavelengths is required when centralizing due to longer paths traversed to reach the BBU Hotel. Furthermore, a higher number of hops, and consequently higher delay, is introduced with centralization.

In Figure 2.1 the reference transport network with the main elements considered in the model, as previously discussed in 1.1, is presented, according to [12]. A set of RRUs equipped with antennas covers a geographical area, and each RRU is connected to a node to access the transport network where the BBU functionality is located. The transport network consists of a set of nodes interconnected by WDM optical fibers acting as fronthaul segments, according to the C-RAN principle. Support for reliability is provided with reference to single BBU hotel failure, by assigning primary and backup BBUs to each RRU in distinct nodes. The point of access to BBU functionalities, either primary or backup, is referred

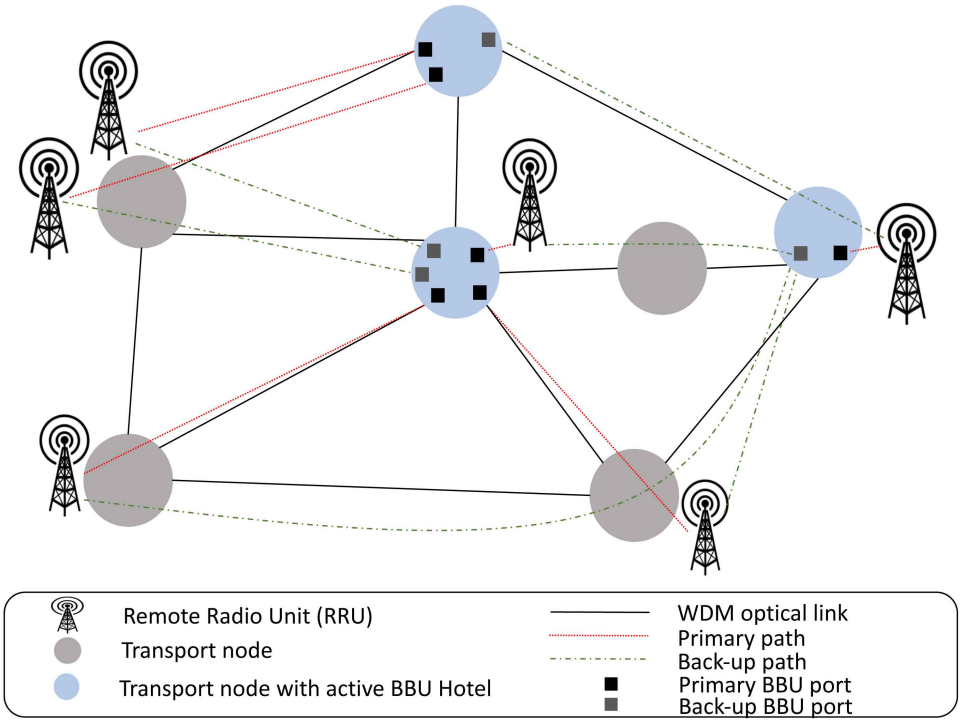


Figure 2.1: Main elements of the C-RAN transport network.

to as port. Backup ports can be shared by RRUs with different primary nodes: in case of single primary node failure, only RRUs assigned to the failed primary will switch to the assigned backup port. Thus, backup ports sharing allows to save resources on the computing nodes. Conversely, RRUs with the same primary node need to have distinct backup ports, otherwise, in case of a failure of a shared primary, there would be conflict between two sets of RRUs trying to access the same backup port. A static traffic scenario is assumed, meaning that the number of active RRUs per node, and thus the overall wavelength demand, are a-priori known. Considering the aforementioned elements, the problem statement for the BBU Hotel assignment problem in a static traffic scenario can be defined.

Given the network topology and the number of active RRUs per node, find a BBU Hotel assignment that minimizes the number of active BBU Hotels, the total number of hops between BBU Hotels and RRUs, and the total number of backup ports. The BBU Hotel assignment must be subject to maximum distance constraints between BBU Hotels and their RRUs, in order to ensure low latency, and to maximum wavelength constraints, to ensure that capacity in each WDM fronthaul link is not exceeded.

2.2 Optimization Model

The BBU Hotel assignment problem can be formulated as an ILP optimization problem. In the past, different approaches, based on ILP models or heuristics, have been proposed for the solution of this problem, each one showing its pros and cons [12], [15], [13]. In particular, solving the ILP models by the means of

Table 2.1: Model parameters and variables

	Parameters
S	Set of transport nodes. $\|S\| = s$
H	$s \times s$ matrix. h_{ij} is the distance in hops between nodes i and j computed with the shortest path.
α	Weight for the distance in the cost function.
β	Activation cost for a single BBU hotel.
γ	Cost for a BBU hotel port.
r_i	Number of RRUS at site i, $i \in S$.
δ_{ij}^l	1 if shortest path between i and j is using link l, 0 otherwise, i, $j \in S$, $l \in L$
M_W	Maximum number of wavelengths available in each link.
M_H	Maximum allowed distance in hops between RRU and BBU.
L	Set of links.
	Variables
B_j	1 if node $j \in S$ hosts a BBU hotel, 0 otherwise
p_{ij}	1 if BBU hotel j is assigned as primary for RRUs at node i, i, $j \in S$, 0 otherwise.
b_{ij}	1 if BBU hotel j is assigned as backup for RRUs at node i, i, $j \in S$, 0 otherwise.
y_j	Number of BBU ports required at hotel site j for backup purposes, $j \in S$.
$c_{ijj'}$	1 if RRUs at node i are using destination j as primary and j' as backup hotel site, i, j, $j' \in S$, 0 otherwise.

a solver has shown some scalability limitations that the heuristics are typically suited to overcome, albeit with some degradation in solution quality [13].

2.2.1 Aggregate Method

The aggregate method employs a scalarization of the objective function, in which objectives are weighed with positive scalars, whose magnitude is proportional to the objective priority, and summed. In this way, all three objectives are optimized at the same time, hence the name "aggregate".

Notation for the model parameters and variables is reported in Table 2.1. Then, the scalarized costs for the objective functions can be defined as follows:

$$C_B = \beta \cdot \sum_{j \in S} B_j \quad \text{Cost for the activation of nodes.} \tag{2.1}$$

$$C_H = \alpha \cdot \sum_{i \in S} \sum_{j \in S} (p_{ij} + b_{ij}) \cdot h_{ij} \quad \text{Cost for the total hops.} \tag{2.2}$$

$$C_P = \gamma \cdot \sum_{j \in S} y_j \quad \text{Cost for the backup BBU Hotel ports.} \tag{2.3}$$

Therefore, given the problem, the ILP model for the BBU Hotel assignment problem in a static traffic scenario can be formulated as follows:

$$\min C = C_B + C_H + C_P \tag{2.4}$$

$$\sum_{j \in S} p_{ij} = 1 \quad \forall i \in S \tag{2.5}$$

$$\sum_{j \in S} b_{ij} = 1 \quad \forall i \in S \tag{2.6}$$

$$p_{ij} + b_{ij} \leq B_j \quad \forall i, j \in S \tag{2.7}$$

$$(p_{ij} + b_{ij}) \cdot h_{ij} \leq M_H \quad \forall i, j \in S \tag{2.8}$$

$$\sum_{i \in S} \sum_{j \in S} (p_{ij} + b_{ij}) \cdot \delta_{ij}^{l} \cdot r_i \leq M_W \quad \forall l \in L \tag{2.9}$$

$$c_{ijj'} \geq p_{ij} + b_{ij'} - 1 \quad \forall i, j, j' \in S,\ j \neq j' \tag{2.10}$$

$$y_{j'} \geq \sum_{i \in S} c_{ijj'} \cdot r_i \quad \forall j, j' \in S,\ j \neq j' \tag{2.11}$$

$$B_j \in \{0, 1\} \quad \forall j \in S \tag{2.12}$$

$$p_{ij} \in \{0, 1\} \quad \forall i \in S, j \in S \tag{2.13}$$

$$b_{ij} \in \{0, 1\} \quad \forall i \in S, j \in S \tag{2.14}$$

$$c_{ijj'} \in \{0, 1\} \quad \forall i \in S, j \in S, j' \in S, j \neq j' \tag{2.15}$$

$$y_j \geq 0, \quad \textit{integer} \quad \forall j \in S \tag{2.16}$$

The objective function (2.4) minimizes the sum of the three objectives, weighted by the coefficients α, β, γ. Constraints (2.5) and (2.6) impose that each RRU has a primary and backup BBU Hotel assigned, in order to ensure reliability. Constraints (2.7) impose that for each RRU backup and primary BBU Hotels are distinct, and are used to count the number of active BBU Hotels. Constraints (2.8) impose that the distance between each RRU and their primary and backup BBU Hotels does not exceed M_H. Note that, even though in this work the distance is expressed in number of hops, any distance metric can be employed. Constraints (2.9) impose that the number of wavelengths in each WDM fronthaul link does not exceed M_W. Constraints (2.10) and (2.11) are used jointly in order to determine the number of backup ports after sharing. Constraints (2.10) are used to define if a node i is using destination j as primary and j' as backup nodes ($i, j, j' \in S$, $j \neq j'$), and constraints (2.11) to count the number of needed backup ports for each active BBU Hotel. Recall that the sharing policy was the following: backup ports can only be shared between RRUs that do not share the same primary BBU Hotel. Given a backup BBU Hotel at node j', constraints (2.11) impose that the number of backup ports is greater than (i.e. equal, since their total number is minimized by the objective) to the maximum number of RRUs assigned at node j' that share the same primary BBU Hotel: the remaining ports can then be shared. Finally, constraints (2.12) - (2.16) define the variable domains. Note that due to constraints (2.7) and to the fact that p_{ij}, b_{ij} and B_j are binary, the term $(p_{ij} + b_{ij})$ in constraints (2.8) and (2.9) can be at most equal to 1.

Implementation

In the following, a sample C++ implementation in CPLEX Concert Technology for the aggregate method problem shall be provided. The following lines are used to include the Concert Technology libraries and to ensure code portability.

```
#include <ilcplex/ilocplex.h>
ILOSTLBEGIN
```

The macro "ILOSTLBEGIN" translates to "using namespace std" when the C++ Standard Library is used, otherwise its value is null.

As a first step, an "IloEnv" environment object must be declared for managing all the modeling objects that will be later created. The IloEnv object must then be destroyed at the end of the program by calling the "end" method, in order to correctly free the allocated memory.

```
int main()
{
  IloEnv env;
  // Model building and optimization
  env.end();
  return 0
}
```

The C++ Concert library provides arrays up to dimension 4 to hold both model parameters. Arrays can be directly read from input data files via the overloaded stream extraction operator ">>", but the input data files must be formatted correctly. In general, arrays must be formatted as "[x, y, ... z]" where x is an element of the array. An example of a .dat file containing a scalar, one array of dimension 1, one array of dimension 2 and one array of dimension 3 is:

```
100
[1.5, 2, 4, 3.2]
[[1, 0, 1, 1, 0],
[0, 0, 1, 1, 1],
[1, 1, 0, 0, 1]]
[[[1, 0],
[0, 1]],
[[2, 0],
[0, 2]]]
```

Assuming the data file to be called "input.dat", one can create the arrays for storing the input data and read it as follows:

```
IloInt data1; // integer scalar
IloNumArray data2(env, 4); // float array of dimension 1
IloIntArray2 data3(env, 3) // int array of dimension 2
for (IloInt i = 0; i < 3; ++i) // array initialization
   data3[i] = IloIntArray(env, 4);
IloIntArray3 data4(env, 2); // int array of dimension 3
```

```
for (IloInt i = 0; i < 2; ++i) {
   data4[i] = IloIntArray2(env, 2);
   for (IloInt j = 0; j < 2; ++j}
      data4[i][j] = IloIntArray3(env, 2);
}
ifstream inputfile("input.dat");
input >> data1 >> data2 >> data3 >> data4;
```

In the following, it will be assumed that the model parameters have been imported from .dat file and stored in appropriate data structures in a way similar to what has been shown. Decision variables are instantiated via the "IloNumVar" class. Multidimensional arrays of decision variables can be defined via similar definitions:

```
typedef IloArray<IloNumVarArray> IloNumVarArray2;
typedef IloArray<IloNumVarArray2> IloNumVarArray3;
```

Considering the optimization model, decision variables are instantiated as follows:

```
// B_j
IloNumVarArray activeHotel(env, nbNodes, 0, 1, ILOINT);
// p_ij and b_ij
IloNumVarArray2 primaryHotel(env, nbNodes);
for (IloInt i = 0; i < nbNodes; ++i)
  primaryHotel[i] = IloNumVarArray(env, nbNodes, 0, 1, ILOINT);
IloNumVarArray2 backupHotel(env, nbNodes);
for (IloInt i = 0; i < nbNodes; ++i)
  backupHotel[i] = IloNumVarArray(env, nbNodes, 0, 1, ILOINT);
// y_j
IloNumVarArray backupPorts(env, nbNodes, 0, IloInfinity,
   ILOINT);
// c_ijj'
IloNumVarArray3 commonPorts(env, nbNodes);
for (IloInt i = 0; i < nbNodes; ++i) {
  commonPorts[i] = IloNumVarArray2(env, nbNodes);
  for (IloInt j = 0; j < nbNodes; ++j) {
    commonPorts[i][j] = IloNumVarArray(env, nbNodes, 0, 1,
       ILOINT);
  }
}
```

Note that instantiation of decision variables requires the specification of the optimization environment, the array dimension, the lower and upper bounds for the variables, and the variable type (in our case integer, specified by "ILOINT").

The optimization model is instantiated via the "IloModel" class, which provides several methods for model building. In this implementation the "add" method has been used, which allows to add expressions, functions of the decision variables, for the objective functions and the constraints. Thus, the implementation for the model is done as follows:

```
IloModel model(env);

// Objective function (2.8)
IloExpr obj(env); // Initialize an empty expression
obj += beta * IloSum(activeHotel) + gamma * IloSum(backupPorts);
for (IloInt s = 0; s < nbNodes; ++s) {
  obj += alpha * IloScalProd(primaryHotel[s],
     connectionCost[s]) + alpha * IloScalProd(backupHotel[s],
     connectionCost[s]);
}
model.add(IloMinimize(env, obj));
obj.end();

// Constraints (2.9)
for (IloInt s = 0; s < nbNodes; ++s)
  model.add(IloSum(primaryHotel[s]) == 1);

// Constraints (2.10)
for (IloInt s = 0; s < nbNodes; ++s)
  model.add(IloSum(backupHotel[s]) == 1);

// Constraints (2.11)
for (IloInt s = 0; s < nbNodes; ++s) {
  for (IloInt d = 0; d < nbNodes; ++d)
    model.add(primaryHotel[s][d] + backupHotel[s][d] <=
       activeHotel[d]);
}

// Constraints (2.12)
for (IloInt s = 0; s < nbNodes; ++s) {
  for (IloInt d = 0; d < nbNodes; ++d)
    model.add((primaryHotel[s][d] + backupHotel[s][d]) *
       connectionCost[s][d] <= maxDistance);
}

// Constraints (2.13)
for (IloInt l = 0; l < nbLinks; ++l) {
  IloExpr v(env);
  for (IloInt a = 0; a < nbNodes; ++a) {
    for (IloInt b = 0; b < nbNodes; ++b)
        v += (primaryHotel[a][b] + backupHotel[a][b]) *
           delta[l][a][b] * rrusAtNode[a];
  }
  model.add(v <= maxWavelengths);
  v.end();
}

// Constraints (2.14)
for (IloInt s = 0; s < nbNodes; ++s) {
```

```
  for (IloInt d = 0; d < nbNodes; ++d) {
    for (IloInt d1 = 0; d1 < nbNodes; ++d1) {
      if (d != d1)
           model.add(commonPorts[s][d][d1] >=
              primaryHotel[s][d] + backupHotel[s][d1] - 1);
    }
  }
}

// Constraints (2.15)
for (IloInt d = 0; d < nbNodes; ++d) {
  for (IloInt d1 = 0; d1 < nbNodes; ++d1) {
    if (d != d1) {
      IloExpr v(env);
      for (IloInt s = 0; s < nbNodes; ++s)
           v += commonPorts[s][d][d1] * rrusAtNode[s];
      model.add(backupPorts[d1] >= v);
      v.end();
    }
  }
}
```

Objects of class "IloExpr" are used to define expressions function of the decision variables, such as the objective function and the constraints. Function "IloAdd" allows to compute the sum of all elements in an array. Function "IloScalProd" allows to compute the scalar product between two arrays of equal size. Function "IloMinimize" allows to instantiate an object of class "IloObjective" for defining an objective function for a minimization problem. Expression "obj" is then destroyed for freeing memory as it is not anymore needed in the following.

The syntax for defining the constraints is quite similar to how the mathematical model is formulated, but particular attention needs to be put when building an expression iteratively, as for constraints (2.13) and (2.15). In these constraints, a dummy "IloExpr v(env)" is created in order to construct the required expression within a for cycle. After such expression is then added in the model within a constraint, one must take care in destroying it to avoid undesirable memory bloating effects.

In order for the model to be solved, it needs to be extracted by an object of class "IloCplex", which will store the model into the proper efficient data structures and will interface with the internals of CPLEX for solving it. From the IloCplex object, the solver properties can be set. The implementation is the following:

```
IloCplex cplex(env);
cplex.setParam(IloCplex::Param::TimeLimit, 3600);
cplex.setParam(IloCplex::Param::WorkMem, 28000);
cplex.setParam(IloCplex::Param::ClockType, 2);
cplex.setParam(IloCplex::Param::MIP::Display, 3);
cplex.setParam(IloCplex::Param::MIP::Tolerances::MIPGap, 1e-9);
cplex.extract(model);
```

An equivalent one-liner for instantiating the IloCplex object and extracting the model is the following:

```
IloCplex cplex(model);
```

Several solver parameters have been set. Execution time limit was set to 3600 seconds (1 hour), the maximum memory occupation before swapping/compression was set to 28000MB. The clock type, for the function "cplex.getCplexTime()", was set to 2, which is the wall clock time (1 is CPU time). CPLEX log display was set to 3, which means that a timestamp for each best integer feasible solution found is provided. Finally, tolerance for the MIP relative gap was set to 10^{-9}, meaning that if the relative MIP gap is less than 10^{-9} the solution is considered to be optimal.

Finally, for solving the model the "solve" method is called:

```
cplex.solve();
```

The solver will start displaying on the terminal (or writing on a log file, if the output was redirected) the CPLEX log, with the set display rules. It can be useful to print, after the log, some information regarding the solution status, for example:

```
cplex.out() << "Solution status: " << cplex.getStatus() << endl;
cplex.out() << "Optimal value: " << cplex.getObjValue() << endl;
cplex.out() << "Best bound: " << cplex.getBestObjValue() <<
   endl;
cplex.out() << "MIP relative gap: " <<
   cplex.getMIPRelativeGap() << endl;
```

An example CPLEX log is reported:

```
Version identifier: 20.1.0.0 | 2020-11-10 | 9bedb6d68
CPXPARAM_MIP_Display                 3
CPXPARAM_TimeLimit                   3600
CPXPARAM_WorkMem                     28000
CPXPARAM_MIP_Tolerances_MIPGap       1.0000000000000001e-09
Tried aggregator 1 time.
MIP Presolve eliminated 275 rows and 16 columns.
MIP Presolve modified 308 coefficients.
Reduced MIP has 4389 rows, 4384 columns, and 18120 nonzeros.
Reduced MIP has 4368 binaries, 16 generals, 0 SOSs, and 0
   indicators.
Presolve time = 0.00 sec. (11.12 ticks)
Found incumbent of value 1668560.000000 after 0.01 sec. (13.85
   ticks)
Probing time = 0.00 sec. (2.04 ticks)
Tried aggregator 1 time.
Detecting symmetries...
Reduced MIP has 4389 rows, 4384 columns, and 18120 nonzeros.
Reduced MIP has 4368 binaries, 16 generals, 0 SOSs, and 0
   indicators.
Presolve time = 0.02 sec. (14.76 ticks)
```

```
Probing time = 0.02 sec. (2.01 ticks)
Clique table members: 288.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 16 threads.
Root relaxation solution time = 0.03 sec. (32.49 ticks)

      Nodes                                         Cuts/
   Node Left     Objective IInf Best Integer    Best Bound    ItCnt     Gap

*     0+  0                      1668560.0000        0.0000           100.00%
Found incumbent of value 1668560.000000 after 0.08 sec. (71.94
   ticks)
*     0+  0                       868352.0000        0.0000           100.00%
Found incumbent of value 868352.000000 after 0.08 sec. (72.03
   ticks)
*     0+  0                       868144.0000        0.0000           100.00%
Found incumbent of value 868144.000000 after 0.08 sec. (72.06
   ticks)
*     0+  0                       867936.0000        0.0000           100.00%
Found incumbent of value 867936.000000 after 0.08 sec. (72.09
   ticks)
*     0+  0                       867728.0000        0.0000           100.00%
Found incumbent of value 867728.000000 after 0.08 sec. (72.13
   ticks)
*     0+  0                       867520.0000        0.0000           100.00%
Found incumbent of value 867520.000000 after 0.08 sec. (72.16
   ticks)
*     0+  0                       867312.0000        0.0000           100.00%
Found incumbent of value 867312.000000 after 0.08 sec. (72.19
   ticks)
*     0+  0                       867104.0000        0.0000           100.00%
Found incumbent of value 867104.000000 after 0.08 sec. (72.23
   ticks)
      0     0   264000.0000    90  867104.0000   264000.0000      686   69.55%
*     0+  0                       264160.0000   264000.0000              0.06%
Found incumbent of value 264160.000000 after 0.14 sec. (129.25
   ticks)
      0     0   264000.0000    72  264160.0000  Cuts: 26      706    0.06%
      0     0   264000.0000    75  264160.0000  Cuts: 85      792    0.06%
      0     0   264000.0000    84  264160.0000 ZeroHalf: 84    848  0.06%
      0     0        cutoff      264160.0000                  848    0.00%
Elapsed time = 0.23 sec. (234.79 ticks, tree = 0.01 MB,
   solutions = 9)

Implied bound cuts applied: 7
Zero-half cuts applied: 46
Lift and project cuts applied: 1

Root node processing (before b&c):
```

```
  Real time        =   0.23 sec. (235.24 ticks)
Parallel b&c, 16 threads:
  Real time        =   0.00 sec. (0.00 ticks)
  Sync time (average) = 0.00 sec.
  Wait time (average) = 0.00 sec.
                    ------------
Total (root+branch&cut) = 0.23 sec. (235.24 ticks)
Solution status: Optimal
Optimal value: 264160
Best bound: 264160
MIP relative gap: 0
```

From the log, several information can be extracted. Integer feasible solutions that improve the current objectives are marked with an asterisk (*) and associated with a timestamp, as IloCplex::Param::MIP::Display was set to 3. In particular, integer feasible solution found with a heuristic are marked with a plus (+). Then, the columns display, in order, the current node in exploration, the number of nodes left to be explored, the objective function best value, the best lower bound from the LP relaxations, the cumulative iteration count of the algorithm solving the subproblems and the relative gap.

2.2.2 Lexicographic Method

Considering the aforementioned problem and the reference application scenario, it is possible to define a priority ordering among objectives. In particular, for a solution to be relevant in the context of C-RAN optimization, it must hold that $\alpha \gg \beta \gg \gamma$. This means that BBU Hotel minimization has the highest priority, followed by hops minimization and port minimization. The practical reasons for imposing such an ordering are manyfold. Firstly, since the main objective of the C-RAN model is to reduce CAPEX and OPEX, and it has been shown that the majority of costs come from the energy consumption within active transport nodes, it is paramount to activate as few nodes as possible in the network in order to maximize the savings. Secondly, imposing a priority for the hops or the ports higher than the nodes would lead to useless solutions from the practical point of view. If hops were given a higher priority than BBU Hotels, all transport nodes would then need to host a BBU Hotel, as RRUs would get assigned to their transport node for either primary or backup purposes. If ports were given a higher priority than BBU Hotels one would also get a very high number of BBU Hotels. This is because according to the sharing policy, backup ports can only be shared among RRUs that do not share the same primary BBU, thus each RRU would get assigned to its transport node for primary purposes, so that RRUs would have no primary BBU Hotel in common and maximum sharing is possible. Finally, since latency is a major requirement for the URLLC service class, it makes sense to give higher priority to the hops than to the ports, also considering that the major savings in terms of energy consumption are made by the minimization of the number of active BBU Hotels. Therefore, given this lexicographic ordering among objectives, the three-steps lexicographic method is formulated as follows.

Step 1: Minimization of the number of active BBU Hotels

This step is used to determine the optimal activation cost of BBU hotels in transport nodes, which is the objective of highest priority. The ILP model solved in this step reads as follows:

$$\min C_B = \sum_{j \in S} B_j \tag{2.17}$$

$$\sum_{j \in S} p_{ij} = 1 \quad \forall i \in S \tag{2.18}$$

$$\sum_{j \in S} b_{ij} = 1 \quad \forall i \in S \tag{2.19}$$

$$p_{ij} + b_{ij} \leq B_j \quad \forall i, j \in S \tag{2.20}$$

$$(p_{ij} + b_{ij}) \cdot h_{ij} \leq M_H \quad \forall i, j \in S \tag{2.21}$$

$$\sum_{i \in S} \sum_{j \in S} (p_{ij} + b_{ij}) \cdot \delta_{ij}^{l} \cdot r_i \leq M_W \quad \forall l \in L \tag{2.22}$$

$$B_j \in \{0, 1\} \quad \forall j \in S \tag{2.23}$$

$$p_{ij} \in \{0, 1\} \quad \forall i \in S, j \in S \tag{2.24}$$

$$b_{ij} \in \{0, 1\} \quad \forall i \in S, j \in S \tag{2.25}$$

The objective function (2.17) requires to minimize the activation cost C_B, expressed as the number of nodes hosting a BBU Hotel. Constraints (2.18) and (2.19) impose, respectively, that one primary node and one backup node are associated with each RRH, in order to ensure reliability for single BBU Hotel failure. Constraints (2.20) impose that primary and backup BBU Hotels for each RRU must be distinct in order to ensure reliability for single BBU Hotel failure, and are also used for counting the number of active BBU Hotels. Constraints (2.21) impose that the maximum distance between RRUs and their primary and backup BBU Hotels does not exceed the M_H. Note that in this work hops are used for measuring the distance between two nodes, but any distance metric can be used. Constraints (2.22) impose that the number of wavelengths over each WDM fronthaul link is at most M_W. Finally, constraints (2.23)-(2.25) define the variable domains. Note that due to constraints (2.20) and to the fact that all decision variables are binary, the term $(p_{ij} + b_{ij})$ in constraints (2.21) and (2.22) is at most equal to 1.

Step 2: Minimization of the number of hops

In this step, the objective is the minimization of the distance, expressed as the number of hops needed to connect BBU hotels and RRUs. The ILP model solved in this step reads as follows:

$$\min C_H = \sum_{i \in S} \sum_{j \in S} (p_{ij} + b_{ij}) \cdot h_{ij} \tag{2.26}$$

$$(2.18) - (2.25)$$

$$\sum_{j \in S} B_j \leq C_B^* \tag{2.27}$$

The objective function (2.26) is the minimization of the total number of hops. All the constraints defined in the ILP model of step 1 are imposed: indeed, in this step we have to define the optimal assignment of primary and backup nodes (and, consequently, the number of active nodes) so as to minimize the number of hops. However, the number of active nodes is limited to C_B^* with constraint (2.27), in order not to worsen the solution computed at Step 1, of higher priority. In this way, the lexicographic ordering among objectives is imposed.

Step 3: Minimization of the number of ports

In this step, the objective is the minimization of the total number of backup ports. Recall that backup ports can only be shared among RRUs that do not share the same primary BBU Hotel. The ILP model solved in this step reads as follows:

$$\min C_P = \sum_{j \in S} y_j \tag{2.28}$$

$$(2.18) - (2.25), \quad (2.27)$$

$$c_{ijj'} \geq p_{ij} + b_{ij'} - 1 \quad \forall i, j, j' \in S,\ j \neq j' \tag{2.29}$$

$$y_{j'} \geq \sum_{i \in S} c_{ijj'} \cdot r_i \quad \forall j, j' \in S,\ j \neq j' \tag{2.30}$$

$$\sum_{i \in S} \sum_{j \in S} (p_{ij} + b_{ij}) \cdot h_{ij} \leq C_H^* \tag{2.31}$$

$$\sum_{j \in S} y_j \geq \frac{\sum_{i \in S} r_i}{C_B^* - 1} \tag{2.32}$$

$$y_j \geq 0, \quad \textit{integer} \quad \forall j \in S \tag{2.33}$$

$$c_{ijj'} \in \{0, 1\} \quad \forall i \in S, j \in S, j' \in S, j \neq j' \tag{2.34}$$

The objective function (2.28) requires to minimize the total number of backup ports. As per Step 2, all constraints of the previous steps are imposed. However, we limit the search space by the optimal values obtained in the previous steps (constraints (2.27) and (2.31)), ensuring that the solutions computing at the higher priority steps are not worsened. Constraints (2.29) is used to define if a node i is using destination j as primary and j' as backup nodes ($i, j, j' \in S$, $j \neq j'$), and constraints (2.30) determine the number of needed backup ports for each active BBU Hotel. As expected, this step is the hardest to be solved, especially due to the large number of variables $c_{ijj'}$. By preliminary computational experiments, it was observed that due to constraints (2.29) weak lower bounds were obtained by solving the Linear Programming relaxation of this model, as fractional solutions with $p_{ij} = b_{ij'} = 0.5$ would lead to a lower bound of 0. In order to improve this lower bound constraint (2.32) was added, that computes the minimum number of backup ports required in any optimal solution. It corresponds to the ratio between the total number of RRUs and the number of active nodes minus one. Indeed, recall that RRUs can share backup ports only if they have different primary BBU hotels. Therefore, the minimum number of backup ports is obtained by considering the largest number of different primary nodes: the latter coincides with the number of active nodes, but we cannot assign the same node as

primary and as backup (see constraints (2.20)), hence we subtract one. Note that constraint (2.32) can only be imposed within the Lexicographic approach, since C_B^* is computed at a previous optimization step. In the aggregate approach, since C_B is a function of the decision variables, the constraint would become nonlinear, leading to a nonlinear feasible set, therefore it cannot be inserted in the model.

Implementation

In the following, two sample C++ implementations in CPLEX Concert Technology for the Lexicographic method shall be provided. The first presented approach, which is simpler, employs the function "IloStaticLex" provided by the Concert library, however it does not allow customization of the optimization parameters for each optimization step. The second presented approach will define three optimization models, as they were defined in the mathematical formulation, and solve them in sequence.

Using IloStaticLex. In this first implementation, the only change that needs to be performed in the code for the aggregate approach is in the objective function, which is defined via the "IloStaticLex" function. This functions needs as arguments the optimization environment, and arrays of equal size containing the expressions of the objectives, their priorities, their weights, their absolute tolerance with respect to the lower bound and their relative tolerance with respect to the lower bound. The syntax is the following:

```
IloExpr objNodes(env);
IloExpr objHops(env);
IloExpr objPorts(env);
objNodes = IloSum(activeHotel);
for (IloInt s = 0; s < nbNodes; ++s)
   objHops += IloScalProd(primaryHotel[s], connectionCost[s]) +
      IloScalProd(backupHotel[s], connectionCost[s]);
objPorts = IloSum(backupPorts);

// Define an array with the objectives of the three steps
IloExprArray objArray(env);
objArray.add(objNodes);
objArray.add(objHops);
objArray.add(objPorts);

// Assign priorities to the objectives
IloIntArray priorities(env, 3);
priorities[0] = 2;
priorities[1] = 1;
priorities[2] = 0;

// Assign weights to the objectives (e.g. if two or more
   objectives have the same priority)
IloNumArray weights(env, 3);
weights[0] = 1.0;
```

```
weights[1] = 1.0;
weights[2] = 1.0;

// Absolute tolerances for each objective (equivalent to
   IloCplex::Param::MIP::Tolerances::AbsMIPGap)
IloNumArray absTols(env, 3);
absTols[0] = 0.0;
absTols[1] = 0.0;
absTols[2] = 0.0;

// Relative tolerances for each objective (equivalent to
   IloCplex::Param::MIP::Tolerances::MIPGap)
IloNumArray relTols(env, 3);
relTols[0] = 0.0;
relTols[1] = 0.0;
relTols[2] = 0.0;

model.add(IloMinimize(env, IloStaticLex(env, objArray, weights,
   priorities, absTols, relTols)));
objNodes.end();
objHops.end();
objPorts.end();
```

When solving a multi-objective optimization problem in this way, for having the CPLEX log to be outputted with the same level of detail of what has been shown for the aggregate code, the parameter to be set is the following:

```
cplex.setParam(IloCplex::Param::MultiObjective::Display, 2);
```

Defining three models. In this second implementation, according to the previously developed mathematical formulations, three optimization models are defined and solved in order. The three models can be instantiated and built in the following way:

```
IloModel model(env); // Optimize number of nodes
IloModel model2(env); // Optimize hops
IloModel model3(env); // Optimize ports

// Objective (2.5)
IloExpr cNodes(env);
cNodes = IloSum(activeHotel);

// Objective (2.6)
IloExpr cHops(env);
for (IloInt s = 0; s < nbNodes; ++s)
  cHops += IloScalProd(primaryHotel[s], connectionCost[s]) +
     IloScalProd(backupHotel[s], connectionCost[s]);

// Objective (2.7)
```

```
IloExpr cPorts(env);
cPorts = IloSum(backupPorts);

model.add(IloMinimize(env, cNodes));
model2.add(IloMinimize(env, cHops));
model3.add(IloMinimize(env, cPorts));
cNodes.end();
cHops.end();
cPorts.end();

// Constraints (2.9)
for (IloInt s = 0; s < nbNodes; ++s) {
  model.add(IloSum(primaryHotel[s]) == 1);
   model2.add(IloSum(primaryHotel[s]) == 1);
   model3.add(IloSum(primaryHotel[s]) == 1);
}

// Constraints (2.10)
for (IloInt s = 0; s < nbNodes; ++s) {
  model.add(IloSum(backupHotel[s]) == 1);
   model2.add(IloSum(backupHotel[s]) == 1);
   model3.add(IloSum(backupHotel[s]) == 1);
}

// Constraints (2.11)
for (IloInt s = 0; s < nbNodes; ++s) {
  for (IloInt d = 0; d < nbNodes; ++d) {
    model.add(primaryHotel[s][d] + backupHotel[s][d] <=
       activeHotel[d]);
    model2.add(primaryHotel[s][d] + backupHotel[s][d] <=
       activeHotel[d]);
    model3.add(primaryHotel[s][d] + backupHotel[s][d] <=
       activeHotel[d]);
  }
}

// Constraints (2.12)
for (IloInt s = 0; s < nbNodes; ++s) {
  for (IloInt d = 0; d < nbNodes; ++d) {
    model.add((primaryHotel[s][d] + backupHotel[s][d]) *
       connectionCost[s][d] <= maxDistance);
    model2.add((primaryHotel[s][d] + backupHotel[s][d]) *
       connectionCost[s][d] <= maxDistance);
    model3.add((primaryHotel[s][d] + backupHotel[s][d]) *
       connectionCost[s][d] <= maxDistance);
  }
}

// Constraints (2.13)
for (IloInt l = 0; l < nbLinks; ++l) {
```

```
  IloExpr v(env);
  for (IloInt a = 0; a < nbNodes; ++a) {
    for (IloInt b = 0; b < nbNodes; ++b)
        v += (primaryHotel[a][b] + backupHotel[a][b]) *
           delta[l][a][b] * rrusAtNode[a];
  }
  model.add(v <= maxWavelengths);
  model2.add(v <= maxWavelengths);
  model3.add(v <= maxWavelengths);
  v.end();
}

// Constraints (2.14)
for (IloInt s = 0; s < nbNodes; ++s) {
  for (IloInt d = 0; d < nbNodes; ++d) {
    for (IloInt d1 = 0; d1 < nbNodes; ++d1) {
      if (d != d1)
          model3.add(commonPorts[s][d][d1] >=
             primaryHotel[s][d] + backupHotel[s][d1] - 1);
    }
  }
}

// Constraints (2.15)
for (IloInt d = 0; d < nbNodes; ++d) {
  for (IloInt d1 = 0; d1 < nbNodes; ++d1) {
    if (d != d1) {
      IloExpr v(env);
      for (IloInt s = 0; s < nbNodes; ++s)
          v += commonPorts[s][d][d1] * rrusAtNode[s];
      model3.add(backupPorts[d1] >= v);
      v.end();
    }
  }
}
```

In principle one could instantiate three IloCplex objects, one for each problem, but in fact only one is sufficient, provided that model extractions are placed correctly within the code.

This approach allows to individually set solver parameters (e.g. maximum execution time, emphasis on feasibility/optimality, etc.) for each optimization step. For solving Step 1, the syntax is the following:

```
IloCplex cplex(env);
cplex.setParam(IloCplex::Param::TimeLimit, 200);
cplex.setParam(IloCplex::Param::WorkMem, 28000);
cplex.setParam(IloCplex::Param::MIP::Display, 3);
cplex.setParam(IloCplex::Param::ClockType, 2);
cplex.setParam(IloCplex::Param::MIP::Tolerances::MIPGap, 1e-9);
```

```
cplex.extract(model);
IloNum start1 = cplex.getCplexTime();
cplex.solve();
IloNum time1 = cplex.getCplexTime() - start1;
```

where variable "time1" is used for storing the computing times of each step.

Now, constraint (2.27) needs to be added to the models for both Step 2 and Step 3, with the following:

```
model2.add(IloSum(activeHotel) <= cplex.getObjValue());
model3.add(IloSum(activeHotel) <= cplex.getObjValue());
```

Furthermore, one can also add constraint (2.32), which imposes a lower bound on the minimum number of backup ports, to the model for Step 3:

```
model3.add(IloSum(backupPorts) * (cplex.getObjValue() - 1) >=
   IloSum(rrusAtNode));
```

Before extracting the model for Step 2, an important step that should be performed is to provide for the following step an initial solution, or MIP start. In fact, the computed solution at Step 1 is feasible, albeit likely non-optimal, for Step 2. Thus, providing such solution to Step 2 allows to save computational time for finding an initial feasible solution. This is particularly critical for lesser priority objectives, since constraints from the lexicographic ordering greatly reduce the search space, thus finding an initial feasible solution can become a cumbersome task. For the problem at hand, if a starting feasible solution is not provided to Step 3, the solver would run for an hour without finding even one single integer feasible solution. For instructing CPLEX to process MIP starts, the following parameter needs to be set:

```
cplex.setParam(IloCplex::Param::Advance, 1);
```

In order to define a MIPstart, one possible approach is to use the IloCplex method "addMIPStart". For this, one must define two one-dimensional arrays of equal length, one with the variables for which they want to provide an initial value, and another with the initial values. Note that this method is not incremental, that is, successive calls of the "addMIPStart" method overwrite any existing MIP start. The syntax for defining the MIP start for Step 2 is following:

```
IloNumVarArray startVar2(env); // Array of variables
IloNumArray startVal2(env); // Array of starting values

for (IloInt i = 0; i < nbNodes; ++i) {
  startVar2.add(activeHotel[i]);
  startVal2.add(cplex.getValue(activeHotel[i]));
}

for (IloInt i = 0; i < nbNodes; ++i) {
  for (IloInt j = 0; j < nbNodes; ++j) {
    startVar2.add(primaryHotel[i][j]);
```

```
      startVal2.add(cplex.getValue(primaryHotel[i][j]));
    }
}

for (IloInt i = 0; i < nbNodes; ++i) {
  for (IloInt j = 0; j < nbNodes; ++j) {
      startVar2.add(backupHotel[i][j]);
      startVal2.add(cplex.getValue(backupHotel[i][j]));
    }
}
```

Then, one can add the MIP start to the model for Step 2 and solve it with the following syntax:

```
cplex.setParam(IloCplex::Param::TimeLimit, 200);
cplex.extract(model2);
cplex.addMIPStart(startVar2, startVal2);
IloNum start2;
cplex.solve();
IloNum time2 = cplex.getCplexTime() - start2;
```

Note that the extraction takes place before addMIPStart. This is because, before the instruction "cplex.extract(model2)", the active model, that is the one currently extracted, was the one for Step 1. Had one called addMIPStart before extract, the MIP start would've been added to the model for Step 1, which would have been then immediately overwritten by the extract method.

The CPLEX log will display in the first line whether a MIP start was accepted as an initial solution and will display the initial solution value. In this problem, since the solution of a step is feasible also for the following ones, the MIPStart will be always accepted. An example can be the following:

```
Version identifier: 20.1.0.0 | 2020-11-10 | 9bedb6d68
CPXPARAM_MIP_Display                 3
CPXPARAM_TimeLimit                  3200
CPXPARAM_WorkMem                    28000
CPXPARAM_MIP_Tolerances_MIPGap      0
Processing 1 MIP starts.
MIP start 'm1' defined solution with objective 230.0000.
1 of 1 MIP starts provided solutions.
MIP start 'm1' defined initial solution with objective 230.0000.
```

Finally, one can add constraint (2.31) to the model for Step 3, provide a MIP start and solve the model in a very similar way to what has been done for Step 2, with the following syntax:

```
model3.add(totalHops <= cplex.getObjValue());

IloNumVarArray startVar3(env);
IloNumArray startVal3(env);
```

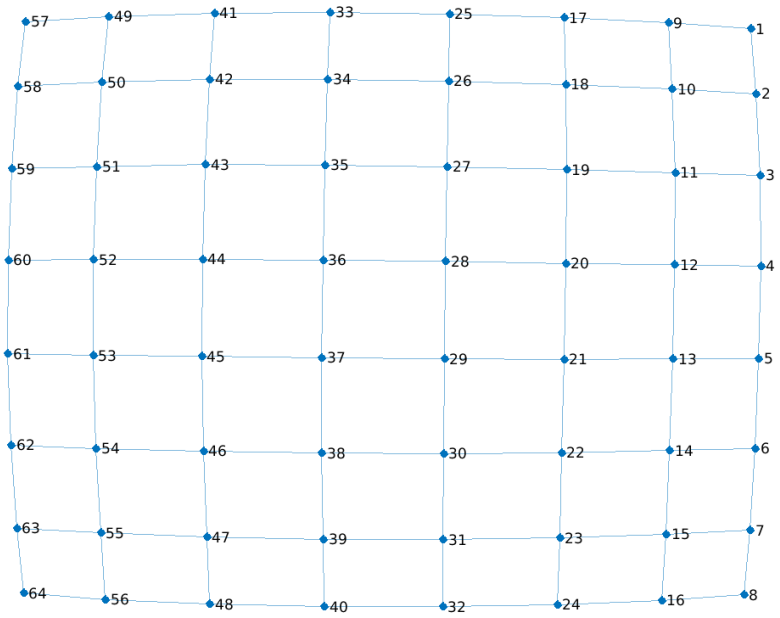
```
for (IloInt i = 0; i < nbNodes; ++i) {
  startVar3.add(activeHotel[i]);
  startVal3.add(cplex.getValue(activeHotel[i]));
}

for (IloInt i = 0; i < nbNodes; ++i) {
  for (IloInt j = 0; j < nbNodes; ++j) {
    startVar3.add(primaryHotel[i][j]);
    startVal3.add(cplex.getValue(primaryHotel[i][j]));
  }
}

for (IloInt i = 0; i < nbNodes; ++i) {
  for (IloInt j = 0; j < nbNodes; ++j) {
    startVar3.add(backupHotel[i][j]);
    startVal3.add(cplex.getValue(backupHotel[i][j]));
  }
}

cplex.setParam(IloCplex::Param::TimeLimit, 3200);
cplex.extract(model3);
cplex.addMIPStart(startVar3, startVal3)
IloNum start3 = cplex.getCplexTime();;
cplex.solve();
IloNum time3 = cplex.getCplexTime() - start3;
```

2.3 Numerical Evaluations

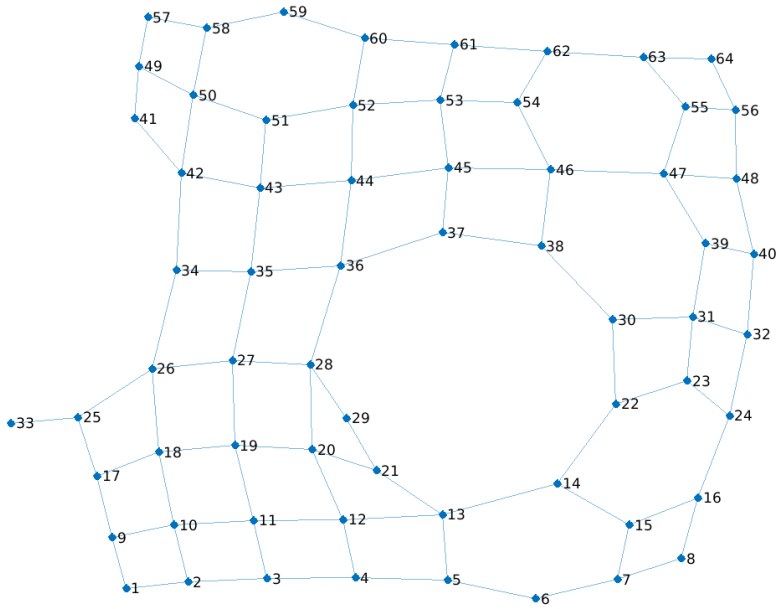
In this section, numerical results and performance comparisons between the aggregate and the lexicographic methods will be detailed. In particular, particular emphasis will be put on algorithm scalability to networks with a large number of transport nodes, up to 100.

2.3.1 Problem Instances

The numerical results were obtained using the commercial solver CPLEX 12.10, running on an Intel Core i9-9900K@4.8GHz with 32GB@3000MHz RAM. The time limit for execution was set to 1 hour. In the lexicographic approach, the time limit was set to 200 seconds for Step 1, 200 seconds for Step 2, and 3200 seconds for Step 3. Four regular Lattice networks (e.g. Fig. 2.2a) of 36, 49, 64 and 100 nodes were considered, with $r_i = 10$ RRU per node and a maximum of $M_W = 80$ wavelengths per link. In order to test the effectiveness of the algorithm also on less regular networks, tests on 49 and 64 nodes networks, where 10 random links are removed with respect to the regular Lattice networks (e.g. Fig. 2.2b), were performed. In the numerical evaluations, $\beta = 10^6$, $\alpha = 10^3$, and, $\gamma = 1$ were used as weights in the multi-objective function of the aggregate model as in [12], thus



(a) 64-nodes regular Lattice network.



(b) 64-nodes regular Lattice network with 10 random links removed.

Figure 2.2: 64-nodes networks considered for the problem instances.

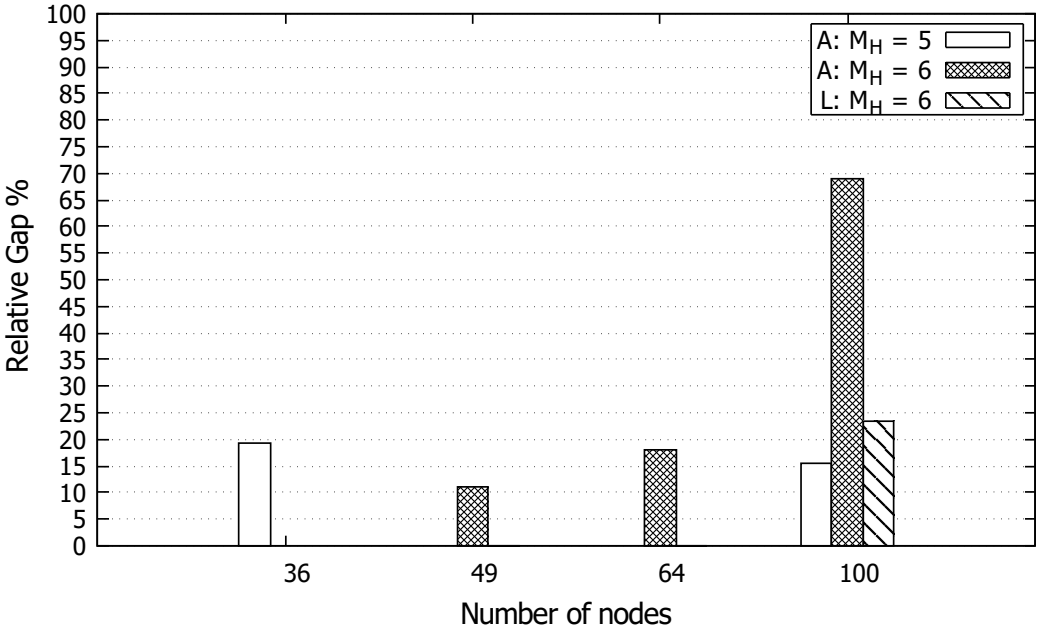


Figure 2.3: Relative gaps of the two approaches for different instances, varying the maximum distance M_H. A:= Aggregate, L = Lexicographic.

imposing the hierarchy among objectives required by the application scenario.

An important figure of merit for performance comparison, other than the execution time, is the relative gap of the best integer solution with respect to the lower bound. However, the aggregate and the lexicographic method compute such gap in two different ways. For the aggregate, there is one value for the gap, that is for the single objective function that is being optimized. For the lexicographic, there are three different values for the gap, one for each step. Evaluating the gap of the lexicographic individually can be deceiving as, for instance, a very large gap on the least-priority objective can give the impression that the algorithm is performing poorly. Therefore, in order to compare the two approaches under the same metric, the following formula is used for computing an equivalent gap for the lexicographic by weighing the objectives and the lower bounds of each step by the same weights used in the aggregate:

$$\mathrm{G}_{\mathrm{eq}} = \frac{C_{\mathrm{eq}} - \mathrm{LB}_{\mathrm{eq}}}{C_{\mathrm{eq}}} \tag{2.35}$$

$$C_{\mathrm{eq}} = \beta \cdot C_B^* + \alpha \cdot C_H^* + \gamma \cdot C_P^* \tag{2.36}$$

$$\mathrm{LB}_{\mathrm{eq}} = \beta \cdot \mathrm{LB}(C_B) + \alpha \cdot \mathrm{LB}(C_H) + \gamma \cdot \mathrm{LB}(C_P) \tag{2.37}$$

where $\mathrm{LB}(\cdot)$ is the best lower bound value achieved by the solver in the execution time limit for the corresponding model in each step, and C^* is the objective value of the best solution found in that step.

2.3.2 Results

Figure 2.3 reports the relative percentage gaps of the two approaches for the four networks and M_H equal to 5 or 6 (see Table 2.2 for more details). When $M_H = 5$,

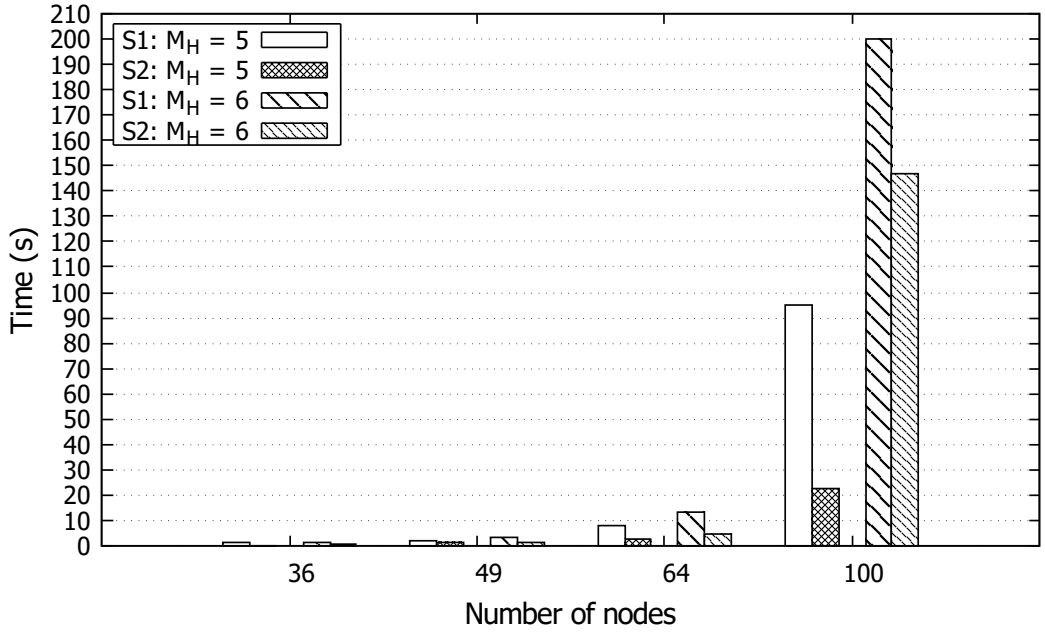


Figure 2.4: Execution times of the first two steps of the lexicographic for different instances. S1 = Step 1, S2 = Step 2, varying the maximum distance M_H.

the relative percentage gap of the lexicographic approach is at most 0.0042% for all instances, hence it is not reported, while the aggregate model has a gap of about 15% for two instances. When $M_H = 6$, near-optimal solutions are obtained by the lexicographic approach for all the instances with up to 64 nodes, while the aggregate model shows much larger gaps (more than 10%). For the network with 100 nodes, the relative gap increases also for the lexicographic approach, in particular for $M_H = 6$, but it is significantly smaller (about $1/3$) than the gap of the aggregate model.

In Figures 2.4 and 2.5 the computing times required by the lexicographic approach (steps 1 and 2) and by the aggregate model are reported, respectively.

For all instances but the largest one (no matter the M_H value), the lexicographic approach requires very short computing times and obtains optimal or near-optimal solutions. For the network with 100 nodes, the computing time increases: when $M_H = 5$, the lexicographic approach obtains a near-optimal solution, while, when $M_H = 6$, step 1 reaches the time limit of 200 seconds. On the other hand, the aggregate model requires much longer computing times (see Figure 2.5) and reaches the time limit of 3600 seconds for most instances. Although the third step of the lexicographic approach reaches the time limit for most instances, the advantage of this method is that it is able to obtain proven optimal solutions for what concerns the first two objectives. Since the third component of the objective function has a much smaller weight than the first two, these solutions are also near-optimal ones for the multi-objective problem.

In Figure 2.6 the required number of BBU Hotels in the considered networks for different distance values are shown. As the distance value increases, the number of required BBU hotels decreases sharply. Correspondingly, as shown in Figure 2.7, the total number of wavelengths increases with the distance value. In fact,

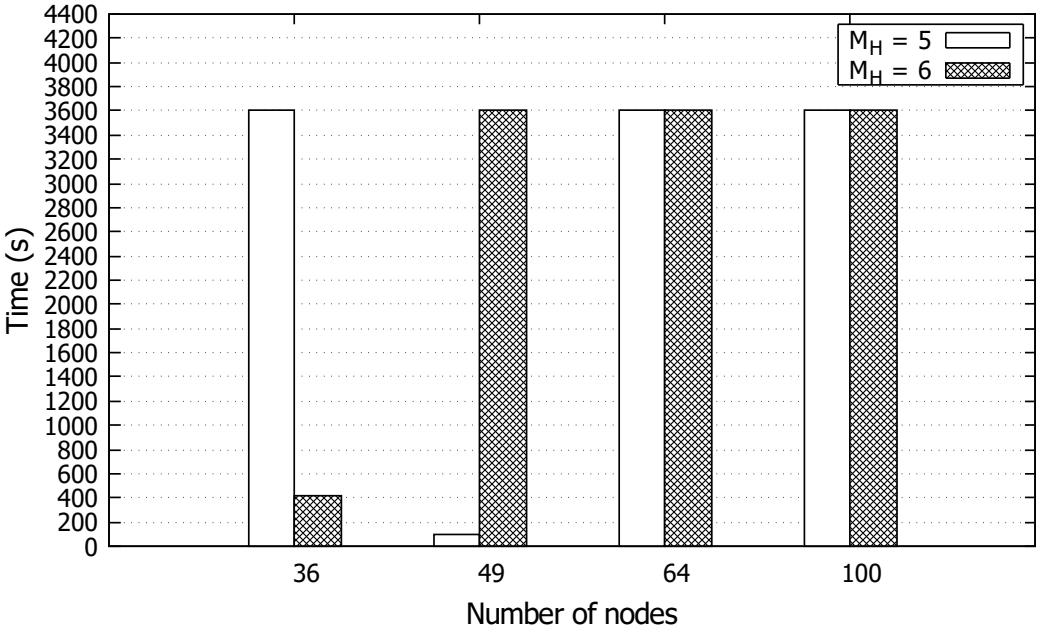


Figure 2.5: Execution times of the aggregate approach for different instances, varying the maximum distance M_H. The maximum value corresponds to the time limit set for execution.

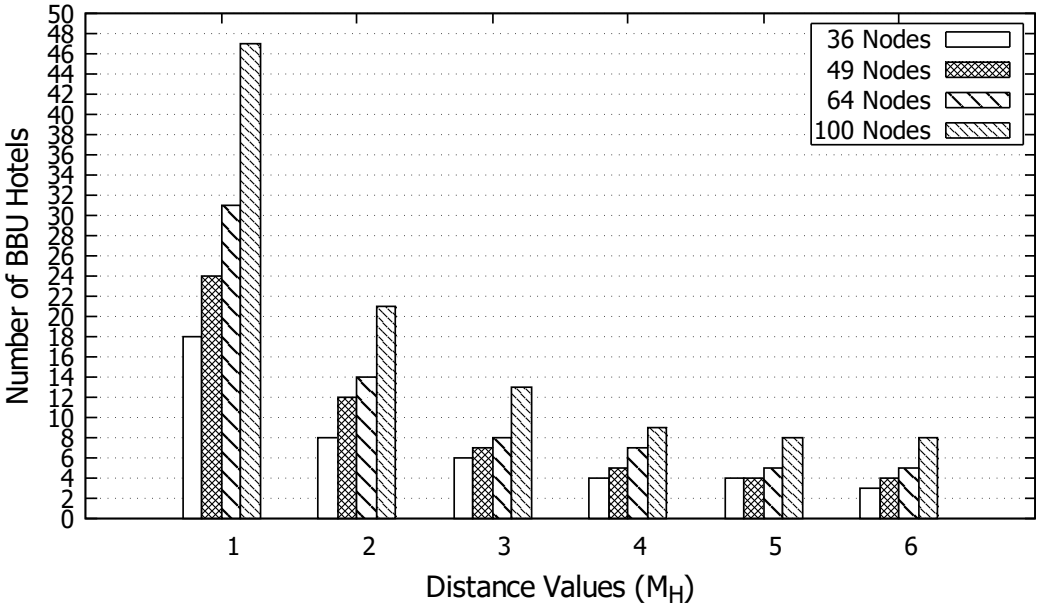


Figure 2.6: Number of active BBU hotels as a function of the distance value varying the network size.

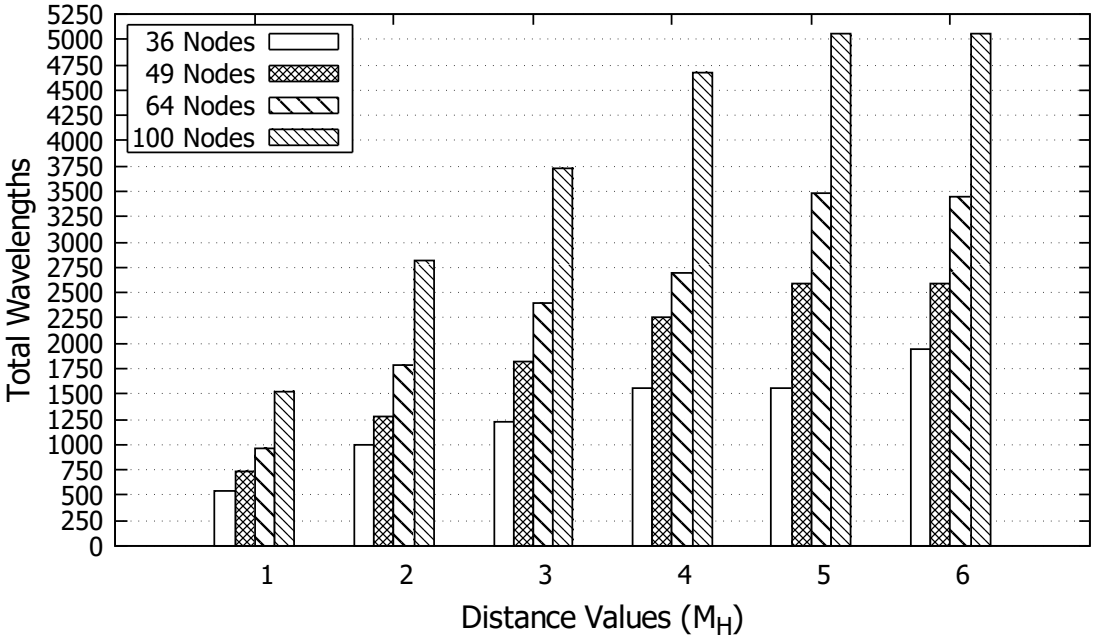


Figure 2.7: Total number of wavelengths as a function of the distance value varying the network size.

as the number of BBU hotels decreases, more wavelengths are required in the interconnection networks to connect RRUs to their assigned primary and backup BBU hotels. In addition, as the distance value increases, further minimization of the number of BBU hotels becomes increasingly difficult. This is because as the number of active BBU hotels decreases, constraints (2.22) become more tight due to the greater number of total wavelengths to distribute in the fronthaul WDM links.

In Figure 2.8 the average number of hops required by RRUs to reach their BBU primary or backup hotel is shown. In the reference networks, as the distance constraints get less tight, the average number of hops does not exceed 3. In particular, considering figure 2.6, one can observe that centralization does not result in a severe increase of the average number of hops, furthering the choice for the lexicographic ordering of the objectives. In addition, since the reference networks have the same structure, the average number of hops results to be mostly independent from the number of nodes.

In Figure 2.9 the number of needed primary and backup wavelengths for the 64 node network is shown. The additional backup wavelengths, for each distance value, show similar values as the needed primary wavelengths. This is because primary and backup hops were given the same level of priority in the lexicographic ordering.

In Figure 2.10 the number of needed primary and backup ports for the 64 nodes network is shown. As mentioned before, the number of primary ports is constant, since they cannot be shared, and equal to the sum of all RRUs. The overall sum of backup ports is minimized, albeit with lowest priority, therefore a non-monotonic behaviour of the solution value with respect to the instance size

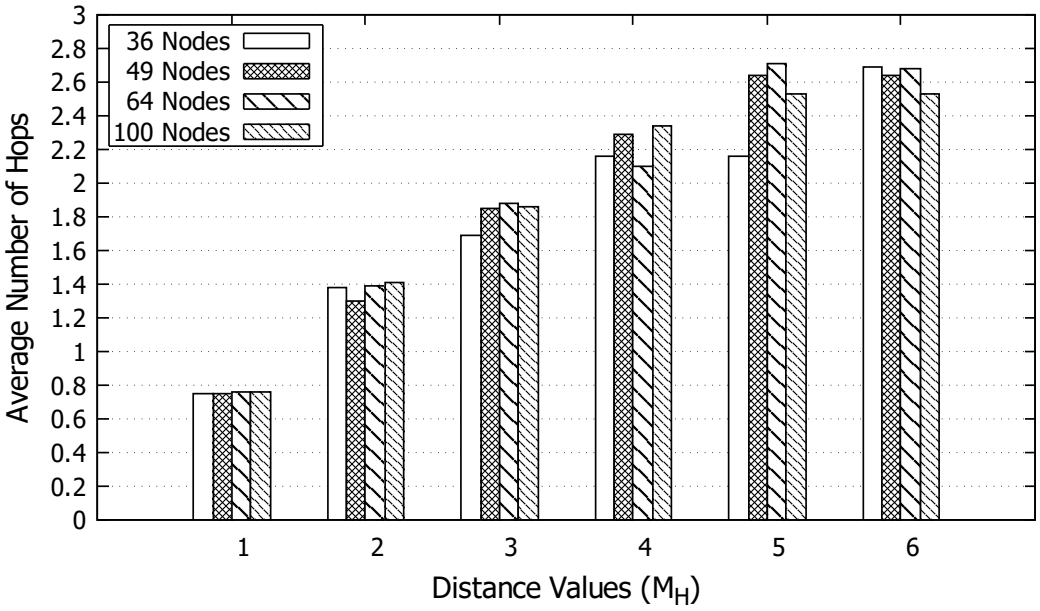


Figure 2.8: Average number of hops required in different networks, for different distance values.

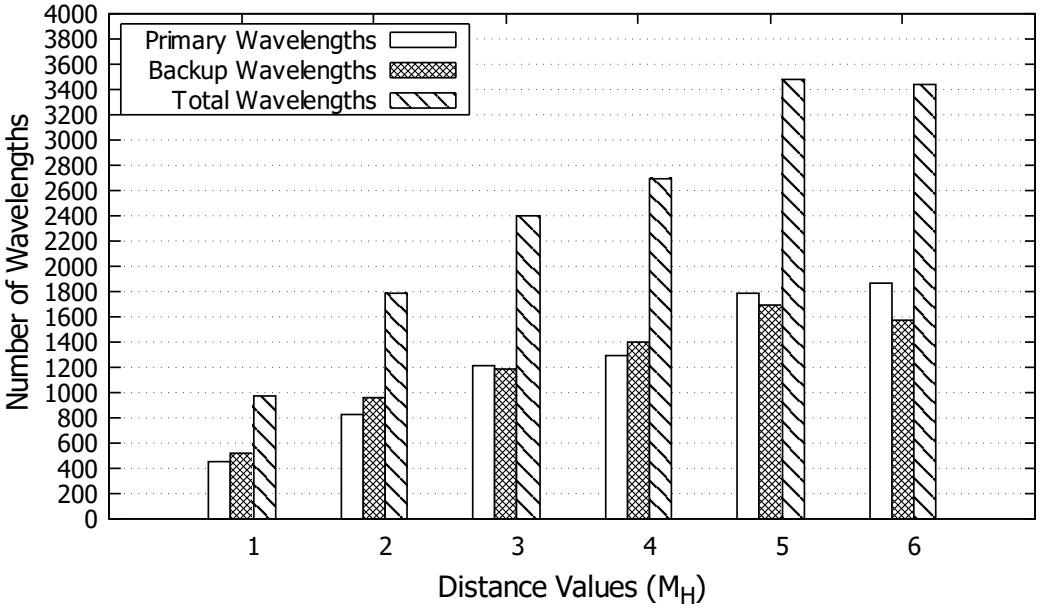


Figure 2.9: Number of primary and backup wavelengths required in the 64 nodes network for different distance values.

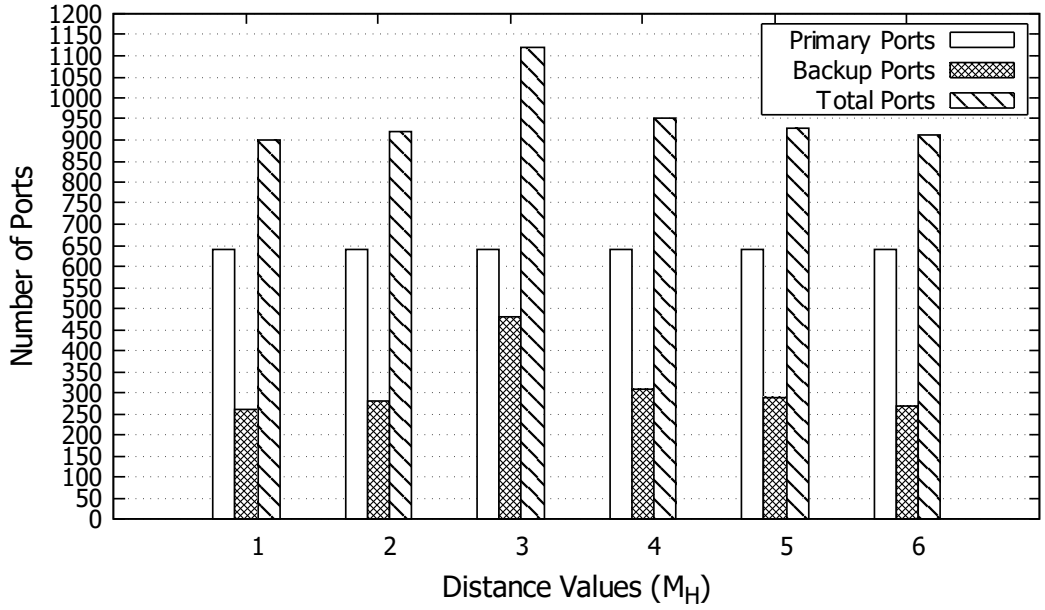


Figure 2.10: Number of primary and backup ports required in the 64 nodes network for different distance values.

is to be expected. As an example, for M_H = 3 a larger number of backup ports is obtained as a consequence of the higher priority in minimizing the total number of hops.

In Table 2.2 the objective values and relative percentage gaps obtained by the lexicographic and aggregate approaches for the regular Lattice networks are reported. Results for the aggregate approach are based on [12]. Except for a few cases (36 nodes with $M_H = 6$, 49 nodes with $M_H = 5$, and 64 nodes with $M_H = 5$) in which both methods have comparable performance, the lexicographic approach performs significantly better than the aggregate model. For instances with up to 64 nodes, it finds optimal or near-optimal solutions, being able to guarantee the optimality of the first two steps, while the aggregate model shows, in some cases, much larger gaps, thus providing worse information regarding the actual quality of the solutions obtained. Not only the relative gap is smaller for the lexicographic approach, but also better solutions are obtained. For the largest instance, it finds a near-optimal solution, characterized by a smaller number of hops than the aggregate model, when $M_H = 5$. When $M_H = 6$, even if the lexicographic approach has 23.5% gap, it finds a significantly better solution with 8 active nodes and 506 hops compared to the 18 nodes and 607 hops required by the aggregate model.

The major bottleneck in the algorithm consists in the port optimization, which even though is the lowest priority objective, in only two instances ($|S| = 36$ with $M_H = 5$, and $|S| = 36$ with $M_H = 6$) converged to the optimal solution within the $3200s$ time limit. Therefore, given the fact that port optimization is the lowest priority objective and in order to mitigate the bottleneck, a simple heuristic algorithm for Step 3 has been developed.

The heuristic algorithm consists in fixing the variables B_j as they are found at

Table 2.2: Results of the lexicographic and aggregate approaches for the regular Lattice networks.

Approach	$\|S\|$	M_H	C_B	C_H	C_P	Gap%
Lexicographic	36	5	4	156	180	0
Aggregate	36	5	4	156	180	19.16
Lexicographic	36	6	3	194	180	0
Aggregate	36	6	3	194	180	0
Lexicographic	49	5	4	259	250	0.0022
Aggregate	49	5	4	259	250	0
Lexicographic	49	6	4	259	250	0.0022
Aggregate	49	6	4	259	250	11.21
Lexicographic	64	5	5	348	300	0.0026
Aggregate	64	5	5	348	290	0.0002
Lexicographic	64	6	5	344	270	0.002
Aggregate	64	6	5	356	220	18.16
Lexicographic	100	5	8	506	500	0.0042
Aggregate	100	5	8	535	470	15.37
Lexicographic	100	6	8	506	540	23.5
Aggregate	100	6	18	607	280	68.9

Table 2.3: Results of the heuristic for Step 3 for the regular Lattice networks of 36, 49, 64 and 100 nodes.

$\|S\|$	M_H	T_{elapsed} (s)	C_P^{start}	C_P	C_P^{best}
36	5	0.17	190	180	180
36	6	0.20	220	180	180
49	5	0.38	270	250	250
49	6	0.42	300	250	250
64	5	0.86	340	290	290
64	6	0.86	310	260	270
100	5	2.94	620	510	550
100	6	2.88	540	530	530

Table 2.4: Results of the lexicographic and aggregate for the regular Lattice networks of 49 and 64 nodes, where 10 random links are removed.

Approach	$\|S\|$	M_H	C_B	C_H	C_P	Gap%
Lexicographic	49	5	5	226	230	0
Aggregate	49	5	5	229	210	17.98
Lexicographic	49	6	4	269	220	0.0013
Aggregate	49	6	4	269	220	9.21
Lexicographic	64	5	6	305	320	0.0030
Aggregate	64	5	6	306	380	14.86
Lexicographic	64	6	5	363	340	0.0034
Aggregate	64	6	6	331	230	30.43

the end of Step 2, therefore optimizing, in Step 3, only decision variables p_{ij}, b_{ij}, $c_{ijj'}$ and y_j.

In table 2.3 results for the heuristic algorithm for Step 3 for the considered regular Lattice networks are shown. It can be observed that computational times are relatively small, at most $3s$ for the 100-nodes network. Column T_{elapsed} (s) reports the execution time of the heuristic algorithm. Column C_P^{start} reports the number of ports resulting from the MIPStart, that is the starting feasible solution from Step 2. It can be observed that for all instances, the solution from the MIPStart is worse than the best one, therefore plainly stopping execution at the end of Step 2 might not be the most effective strategy. Column C_P reports the number of ports after fixing B_j according to the results of Step 2 and optimizing the remaining variables. Column C_P^{best} reports the best solution found by Step 3 in $3200s$ of computational time. It can be observed that for all instances results are either equal or better (namely, for $|S| = 64$ with $M_H = 6$, $|S| = 100$ with $M_H = 5$, and $|S| = 100$ with $M_H = 6$) with respect to the best ones, all in a couple seconds of computational time. Therefore, this simple heuristic algorithm is effective enough for handling the bottleneck induced by solving Step 3 in an exact way.

In table 2.4 the objective values and relative percentage gap obtained by the lexicographic and aggregate approaches for the regular Lattice networks with 10 removed random links are shown. It can be observed that results show similar trends to what was shown in table 2.2, if not better in terms of percentage gap (e.g. $|S| = 64$ with $M_H = 6$). Therefore, one can conclude that the lexicographic method works efficiently also on non-regular network topologies.

2.4 Conclusions

A lexicographic approach is proposed to solve a multi-objective optimization problem for C-RAN optimization in a static traffic scenario, aiming at achieving scalability in optimal transport network design. The multi-objective problem is divided into three single-objective steps which are analyzed in terms of execution time and accuracy. Compared to a previously defined aggregate model, the lexi-

cographic approach shows much better performance and accuracy when applied to large networks: it allows to calculate the optimal (or near-optimal) solution in a few tens of seconds for the most relevant objectives, also in those situations that the aggregate model was not able to solve. The main bottleneck of the approach is represented by port optimization: a simple heuristic algorithm was developed, which returned in few seconds of computational times equal or better than the best ones obtained by optimizing the ports in an exact way.

Chapter 3

Dynamic Traffic Scenario

In this chapter, an exact algorithm for the optimal BBU placement problem is developed for a dynamic traffic scenario. This algorithm employs the framework of lexicographic optimization developed in the previous chapter. Differently from the static traffic scenario, the algorithm takes into account reconfiguration needs of the C-RAN as a consequence of traffic changes: the objective is to maintain C-RAN cost optimization, while minimizing the cost of virtual network function migration. Even though some works in literature regarding C-RAN optimization with migration costs exist [22] [19], as far as my knowledge goes, this is the first approach taking also into account reliability and latency constraints. Along with the mathematical formulation of the problem, a sample C++ code implementation will be provided for modelling and solving the problem via CPLEX Concert Technology. The algorithm leads to significant savings in the total number of migrations (above 82% for primary virtual BBU functions and above 75% for backup virtual BBU function). The execution time of the optimization algorithm is below 20 seconds in most cases, making its application feasible for dynamic scenarios. This Chapter details the work that was developed for [7].

3.1 Problem Statement

In the static traffic scenario, a constant number of active RRUs per transport node was assumed. In dynamic traffic scenarios this may not be the case, as the traffic volume varies through the day, following some general daily trends such as the one shown in [6] [17]. In particular, a BBU Hotel assignment computed for a certain traffic volume becomes infeasible if an increase in the traffic would imply exceeding the capacity of one or more fronthaul WDM links, therefore requiring a reconfiguration. However, even though the algorithm for the static traffic scenario has been proven to be efficient in terms of computational times and solution quality, it is completely blind to virtual network function migrations. In fact, when reconfiguring the network after a traffic increase, instead of completely changing the assignment for the sake of optimality, one should take into account the displacement, i.e., the number of migrations, with respect to the previous, now infeasible, solution. Similarly, when the traffic decreases, reconfiguration is useful to reduce the energy consumption: in this case, one would like to de-

activate nodes if possible, but should also to take into account the number of migrations induced by a deactivation.

Therefore, in a dynamic traffic scenario, the aim is to minimize the cost of migrating network functions in relation to the traffic changes in time, including nodes and hops optimization, subject to constraints on the maximum latency with support for reliability to single BBU Hotel failure. In practice, the migration cost accounts for the time and processing needed to reconfigure the assignment.

3.2 Optimization Model

The optimization model is solved multiple times according to a predefined time granularity (e.g. one instance every 60 seconds) within a reference period (e.g. a day). For example, a 60 seconds time granularity over a time period of one day would correspond to a total of 1440 problem instances to be solved in sequence. In each time interval, according to the traffic values sampled at the beginning of the interval, the model optimizes the number of network function migrations while preserving as much as possible C-RAN optimality, with respect to the static traffic scenario, in terms of active BBU Hotel and total hops between BBU Hotels and their assigned RRUs. In particular, migrations are considered with respect to the solution that was computed for the previous time interval. Therefore, after solving the model in a time interval, the current solution is stored to be used as a reference for the successive time interval. A starting reference solution is provided by solving an initial instance via the lexicographic algorithm for a static traffic scenario developed in 2.2.2. The model is solved using a lexicographic method, since it was proven to be computationally much faster than an aggregate approach.

For what concerns the definition of the objectives, several considerations need to be made in order to properly account for the migrations. Indeed, considering the starting solution from a previous time instant, a cost must be associated with activating a previously idle node. This alone would lead to keeping the previous solution, if feasible, without any change. However, from an application point of view, one must also take into account, other than the allocation of resources, their release when they are no more needed. Therefore, a reward must be associated with deactivating a previously active node. However, the cost for activation must be greater than the reward for deactivation: otherwise a deactivation followed by an activation, an undesirable event from the migrations point of view, would result in a net cost less or equal than 0.

To summarize, the highest priority objective, optimizing the number of active nodes, must account for the following:

- A cost for activating nodes that were not active in the previous solution,
- A reward for deactivating nodes that were active in the previous solution,
- The cost for activating previously-inactive nodes must be greater than the reward for deactivating previously-active nodes.

This first objective therefore optimizes the number of active BBU Hotels, minimizing the displacement from the previous solution in terms of active BBU Hotels and releasing resources whenever possible, driven by the reward.

The second objective, given the objective function value from the previous step, will then minimize the displacement from the previous solution in terms of primary and backup assignments for each BBU Hotel, i.e. minimize the total number of migrations.

The third and final objective will optimize the total number of hops, in the same way that has been discussed for the static traffic scenario. Port optimization, given that according to the considered application is the least priority objective, is neglected for the dynamic traffic scenario, since its optimization requires longer computing times.

Overall, the optimization consists in two ILPs to be optimized in sequence, imposing the lexicographic ordering of the objectives. The first ILP optimizes the first objective, which is function of variables B_j. The second ILP optimizes jointly the second and the third objectives, which are both function of variables p_{ij} and b_{ij}. This is because separating the objectives according to the decision variables appearing in the objective functions was proven to be computationally more efficient than having one optimization step per objective. The parameters and decision variables are reported in Table 3.1.

3.2.1 Step 1: Optimal BBU activation/deactivation

This step is used to determine the optimal activation cost for BBU Hotels in transport nodes. The ILP model of this step reads as follows:

$$\min C_B = \sum_{j \in S_0} B_j - \frac{1}{2} \sum_{j \in S_1} (1 - B_j) \tag{3.1}$$

$$\sum_{j \in S} p_{ij} = 1 \quad \forall i \in S \tag{3.2}$$

$$\sum_{j \in S} b_{ij} = 1 \quad \forall i \in S \tag{3.3}$$

$$p_{ij} + b_{ij} \leq B_j \quad \forall i, j \in S \tag{3.4}$$

$$(p_{ij} + b_{ij}) \cdot h_{ij} \leq M_H \quad \forall i, j \in S \tag{3.5}$$

$$\sum_{i \in S} \sum_{j \in S} (p_{ij} + b_{ij}) \cdot \delta_{ij}^{l} \cdot r_i \leq M_W \quad \forall l \in L \tag{3.6}$$

$$B_j \in \{0, 1\} \quad \forall j \in S \tag{3.7}$$

$$p_{ij} \in \{0, 1\} \quad \forall i \in S,\ j \in S \tag{3.8}$$

$$b_{ij} \in \{0, 1\} \quad \forall i \in S,\ j \in S \tag{3.9}$$

The objective function (3.1), as discussed, penalizes the activation of new BBU Hotels with respect to the solution of the previous time interval and rewards the deactivation of a BBU Hotel active in the previous solution. The penalty is larger than the reward, so that the deactivation of a BBU Hotel followed by the activation of a BBU Hotel results in a penalty. Overall, this step minimizes the

Table 3.1: Model parameters and variables

	Parameters
S	Set of transport nodes. $\|S\| = s$
S_0	Set of inactive nodes in the previous solution.
S_1	Set of active nodes in the previous solution.
EP_0	Set of inactive primary assignments in the previous solution.
EB_0	Set of inactive backup assignments in the previous solution.
L	Set of links.
H	$s \times s$ matrix. h_{ij} is the distance in hops between nodes i and j computed with the shortest path.
α	Migration cost of a primary BBU function.
β	Migration cost of a backup BBU function.
γ	Weight for the distance in the cost function.
r_i	Number of active RRUS at site i, $i \in S$.
δ_{ij}^l	1 if shortest path between i and j is using link l, 0 otherwise, i, $j \in S$, $l \in L$
M_W	Maximum number of wavelengths available in each link.
M_H	Maximum allowed distance in hops between RRU and BBU.
	Variables
B_j	1 if node $j \in S$ hosts a BBU hotel, 0 otherwise
p_{ij}	1 if BBU hotel j is assigned as primary for RRUs at node i, i, $j \in S$, 0 otherwise.
b_{ij}	1 if BBU hotel j is assigned as backup for RRUs at node i, i, $j \in S$, 0 otherwise.

displacement in terms of active BBU Hotels with respect to the previous solution, while deactivating as much BBU Hotels as possible. Constraints (3.2) and (3.3) impose that each RRU has a primary and backup BBU Hotel assigned. Constraints (3.4) are used to count active BBU Hotels and to impose that primary and backup BBU Hotels for each RRU are distinct. Constraints (3.5) impose that the distance of each RRU from its primary and backup BBU Hotels does not exceed M_H. Constraints (3.6) impose that the number of wavelengths per WDM link does not exceed M_W. Finally, constraints (3.7)-(3.9) define the variable domains.

3.2.2 Step 2: Optimal Number of Migrations and Hops

This step is used to minimize the number of migrations and the distance between RRUs and BBU Hotels, which is expressed in number of hops. The ILP model solved in this step reads as follows:

$$\min C_M + C_H = \alpha \sum_{(i,j)\in EP_0} p_{ij} + \beta \sum_{(i,j)\in EB_0} b_{ij} + \gamma \sum_{i\in S}\sum_{j\in S}(p_{ij} + b_{ij}) \cdot h_{ij} \tag{3.10}$$

$$(3.2) - (3.9)$$

$$\sum_{j\in S_0} B_j - \frac{1}{2}\sum_{j\in S_1}(1 - B_j) \leq C_B^* \tag{3.11}$$

The weighted multi-objective function (3.10) minimizes the total primary and backup migrations, and the total number of hops. All the constraints defined in the ILP model of Step 1 are imposed, since the choice of active BBU Hotels is not fixed from the previous step. Constraint (3.11) reduces the search space and guarantees that the optimal solution of the previous higher priority step, that is the net cost resulting from activations and deactivations, is not worsened.

3.2.3 Implementation

In the following, a sample C++ implementation in CPLEX Concert Technology for the NFMig algorithm shall be provided. It will be assumed that the model parameters have been imported from properly formatted .dat files and stored in appropriate data structures.

Given that the algorithm will be called repeatedly for each time instant according to a predefined time granularity, it is advisable to automate it by putting the optimization procedure within a for loop such as the following:

```
const IloInt totInstances = 1440; // 1m granularity
for (IloInt t = 0; t < totInstances; ++t) {
   IloEnv env;
   // Optimization procedure
   // Update reference solution
   env.end();
}
```

For instance, in the above example the algorithm is called 1440 times for a time granularity of 1 minute, for an overall time period of one day.

The decision variables are defined in the same way as for the implementation in the static traffic scenario:

```
IloNumVarArray activeHotel(env, nbNodes, 0, 1, ILOINT);
IloNumVarArray2 primaryHotel(env, nbNodes);
for (IloInt i = 0; i < nbNodes; ++i)
   primaryHotel[i] = IloNumVarArray(env, nbNodes, 0, 1, ILOINT);
IloNumVarArray2 backupHotel(env, nbNodes);
for (IloInt i = 0; i < nbNodes; ++i)
  backupHotel[i] = IloNumVarArray(env, nbNodes, 0, 1, ILOINT);
IloNumVarArray primaryPorts(env, nbNodes, 0, IloInfinity,
   ILOINT);
IloNumVarArray backupPorts(env, nbNodes, 0, IloInfinity,
   ILOINT);
```

It is reminded that IloIntArray2 is not a data type defined in Concert and needs to be defined via:

```
typedef IloArray<IloNumVarArray> IloNumVarArray2;
```

The previous solution is stored in a properly formatted .dat file and imported in the proper data structures:

```
IloIntArray solActiveHotel(env, nbNodes);
IloIntArray2 solPrimaryHotel(env, nbNodes);
IloIntArray2 solBackupHotel(env, nbNodes);

//Read previous sol. from properly formatted .dat file
const char* solfile = "PATH-TO-DATA";
ifstream sfile(solfile);
sfile >> solActiveHotel >> solPrimaryHotel >> solBackupHotel;
sfile.close();
```

As for the previously developed Lexicographic method, two optimization models to be solved in sequence are defined and populated with their respective objective functions and constraints according to the mathematical formulations:

```
IloModel model(env); // Step 1
IloModel model2(env); // Step 2

// Objective functions definitions
IloExpr objNodes(env);
IloExpr objMig(env);
IloExpr objHops(env);
// Objective 3.1
for (IloInt i = 0; i < nbNodes; ++i) {
  if (solActiveHotel[i] == 0)
    objNodes += activeHotel[i];
  else
    objNodes += -0.5 * (1 - activeHotel[i]);
}
```

```
// Objective 3.10
for (IloInt i = 0; i < nbNodes; ++i) {
  for (IloInt j = 0; j < nbNodes; ++j) {
    if (solPrimaryHotel[i][j] == 0)
      objMig += alpha * primaryHotel[i][j];
    if (solBackupHotel[i][j] == 0)
      objMig += beta * backupHotel[i][j];
    objHops += gamma * (primaryHotel[i][j] + backupHotel[i][j])
       * connectionCost[i][j];
  }
}
model.add(IloMinimize(env, objNodes));
model2.add(IloMinimize(env, objMig + objHops));
objMig.end();
objHops.end();

// Constraints definitions
// Constraints (3.2)
for (IloInt s = 0; s < nbNodes; ++s) {
  model.add(IloSum(primaryHotel[s]) == 1);
  model2.add(IloSum(primaryHotel[s]) == 1);
}

// Constraints (3.3)
for (IloInt s = 0; s < nbNodes; ++s) {
  model.add(IloSum(backupHotel[s]) == 1);
  model2.add(IloSum(backupHotel[s]) == 1);
}

// Constraints (3.4)
for (IloInt s = 0; s < nbNodes; ++s) {
  for (IloInt d = 0; d < nbNodes; ++d) {
    model.add(primaryHotel[s][d] + backupHotel[s][d] <=
       activeHotel[d]);
    model2.add(primaryHotel[s][d] + backupHotel[s][d] <=
       activeHotel[d]);
  }
}

// Constraints (3.5)
for (IloInt s = 0; s < nbNodes; ++s) {
   for (IloInt d = 0; d < nbNodes; ++d) {
    model.add((primaryHotel[s][d] + backupHotel[s][d]) *
       connectionCost[s][d] <= maxDistance);
    model2.add((primaryHotel[s][d] + backupHotel[s][d]) *
       connectionCost[s][d] <= maxDistance);
  }
}
```

```
    // Constraints (3.6)
for (IloInt l = 0; l < nbLinks; ++l) {
  IloExpr v(env);
  for (IloInt s = 0; s < nbNodes; ++s) {
    for (IloInt d = 0; d < nbNodes; ++d) {
      v += (primaryHotel[s][d] + backupHotel[s][d]) *
          delta[l][s][d] * rrusAtNode[t][s];
    }
  }
  model.add(v <= maxWavelengths);
  model2.add(v <= maxWavelengths);
  v.end();
}
```

Given the potentially high number of instances to be solved in sequence, it is advisable to redirect the output log to a file, via the following:

```
ofstream fout("PATH-TO-LOGFILE");
cplex.setOut(fout);
cplex.setWarning(fout);
cplex.setError(fout);
```

Then, a IloCplex object needs to be instantiated and the parameters for solving Step 1 need to be set. Then, the model for Step 1 is extracted and solved, and the solving time is stored.

```
IloCplex cplex(env);
Cplex.setParam(IloCplex::Param::TimeLimit, 35);
cplex.setParam(IloCplex::Param::WorkMem, 28000);
cplex.setParam(IloCplex::Param::ClockType, 2);
cplex.setParam(IloCplex::Param::MIP::Display, 3);
cplex.setParam(IloCplex::Param::Advance, 1);
cplex.setParam(IloCplex::Param::MIP::Tolerances::MIPGap, 1e-9);

float start1 = cplex.getCplexTime();
cplex.extract(model);
cplex.solve();
float time1 = end1 - cplex.getCplexTime();
```

Information on the solution status, objective function value, lower bound and MIP gap can be printed in the log file with the following:

```
cplex.out() << "Solution status: " << cplex.getStatus() << endl;
cplex.out() << "Optimal value: " << cplex.getObjValue() << endl;
cplex.out() << "Lower bound: " << cplex.getBestObjValue() <<
   endl;
cplex.out() << "MIP relative gap: " <<
   cplex.getMIPRelativeGap() << endl;
```

Then, from the solution of Step 1, a MIPStart is provided to Step 2, and the con-

straint enforcing the lexicographic ordering of the objectives is added.

```
IloNumVarArray startVar2(env);
IloNumArray startVal2(env);

for (IloInt i = 0; i < nbNodes; ++i) {
startVar2.add(activeHotel[i]);
startVal2.add(cplex.getValue(activeHotel[i]));
}

for (IloInt i = 0; i < nbNodes; ++i) {
  for (IloInt j = 0; j < nbNodes; ++j) {
    startVar2.add(primaryHotel[i][j]);
    startVal2.add(cplex.getValue(primaryHotel[i][j]));
  }
}

for (IloInt i = 0; i < nbNodes; ++i) {
  for (IloInt j = 0; j < nbNodes; ++j) {
    startVar2.add(backupHotel[i][j]);
    startVal2.add(cplex.getValue(backupHotel[i][j]));
  }
}

// Constraint 3.11
model2.add(objNodes <= cplex.getObjValue());
objNodes.end();
```

For this particular implementation, a time granularity of one minute was assumed. Therefore, assuming a backoff of 10 seconds needed to propagate the NFMig solution and to perform migrations, the overall time limit for Step 1 plus Step 2 is set to 50 seconds. In particular, it was set at most 35 seconds for Step 1 and the residual time to Step 2. This can be done with the following:

```
float timelim2 = 50 - time1;
timelim2 = max(15.0f, timelim2);
cplex.setParam(IloCplex::Param::TimeLimit, timelim2);
```

Finally, the model for Step 2 is extracted, the MIPStart is added, the model is solved, and the computing time is stored:

```
float start2 = cplex.getCplexTime();
cplex.extract(model2);
cplex.addMIPStart(startVar2, startVal2);
cplex.solve();
float time2 = end2 - cplex.getCplexTime();
```

At this point, the file containing the previous solutions must be updated with the current solution, to be used at reference for the next time instant. Given the formatting required by Concert and the previously shown syntax for importing

the data, the solution file can be updated as follows:

```
ofstream updatefile("PATH-TO-SOLFILE");
IloInt i;
IloInt j;
updatefile << "[";
for (i = 0; i < nbNodes - 1; ++i)
  updatefile << cplex.getValue(activeHotel[i]) << ", ";
updatefile << cplex.getValue(activeHotel[i]) << "]" << endl;

updatefile << "[";
for (i = 0; i < nbNodes - 1; ++i) {
  updatefile << "[";
  for (j = 0; j < nbNodes - 1; ++j)
    updatefile << cplex.getValue(primaryHotel[i][j]) << ", ";
  updatefile << cplex.getValue(primaryHotel[i][j]) << "]," <<
     endl;
}
updatefile << "[";
for (j = 0; j < nbNodes - 1; ++j)
  updatefile << cplex.getValue(primaryHotel[i][j]) << ", ";
updatefile << cplex.getValue(primaryHotel[i][j]) << "]]" <<
   endl;

updatefile << "[";
for (i = 0; i < nbNodes - 1; ++i) {
  updatefile << "[";
  for (j = 0; j < nbNodes - 1; ++j)
    updatefile << cplex.getValue(backupHotel[i][j]) << ", ";
  updatefile << cplex.getValue(backupHotel[i][j]) << "]," <<
     endl;
}
updatefile << "[";
for (j = 0; j < nbNodes - 1; ++j)
  updatefile << cplex.getValue(primaryHotel[i][j]) << ", ";
updatefile << cplex.getValue(backupHotel[i][j]) << "]]" << endl;
updatefile.close();
```

3.3 Numerical Evaluations

The reference network consists in 38 transport nodes, interconnected as shown in Figure 3.1. The mean traffic offered to the network during the day, based on the general trends described in [6], [17] and expressed in average RRUs per node, is shown in Figure 3.2. Such traffic behaviour is comprehensive of both low and high traffic, and of both positive, negative and null gradient with respect to time.

For each time instant, the number of active RRUs in a node consists of the mean value, illustrated in Figure 3.2, plus a random contribution uniformly distributed between $[-2, 2]$ active RRUs, in order to take into account small devia-

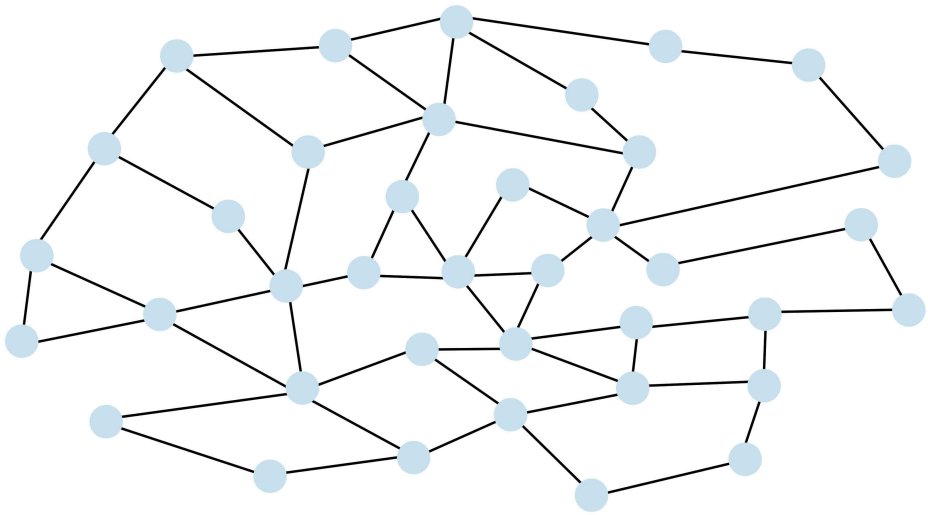


Figure 3.1: 38-node C-RAN used for the numerical evaluations

tions with respect to the average traffic values in each transport node.

For the following comparisons, the developed algorithm for the dynamic traffic scenario shall be referred to as "NFMig". The described NFMig algorithm is applied to maintain the virtual BBU function assignment optimized in relation to traffic variation. The instants of application can be set periodically with a given time granularity that can be constant or variable during the observation period. It can also be triggered by sudden changes in traffic. A constant interval is here chosen and the sensitivity to its length is evaluated. One main requirement is that the execution time of the algorithm is enough shorter than the time period. In this way, time for the consequent network re-configurations and network function migrations is accounted for. Intervals from a few tens of seconds to a few tens of minutes can be assumed. When the time granularity gets larger, optimization is done less frequently leading to insufficient resource allocation, especially in the case of increasing traffic.

In this section the performance of the NFMig algorithm with respect to the aforementioned scenarios is shown. The numerical results were obtained using the commercial solver CPLEX v12.10, running on an Intel Core i9-9900K@4.8GHz with 32GB@3000MHz RAM. A maximum of 6 hops between BBU Hotels and RRUs and 80 wavelengths per link were imposed. For what concerns Step 2, $\alpha \gg \beta$ was considered in order to penalize more primary migrations with respect to backup migrations, according to the application scenario. Furthermore, $\beta \gg \gamma$ was considered in order to prioritize the minimization of migrations with respect to the total number of hops. An optimization model for static traffic conditions was presented in [8] and detailed in Chapter 2, solved by a lexicographic method, and will be referred to as "NFStat" for the following comparisons.

Figure 3.3 shows the execution times of the instances calculated by NFMig and NFStat, with time granularity equal to 1 minute, assuming the traffic dynamics of Figure 3.2. The execution time of NFMig depends on the traffic variations, but it is on average below $20s$. NFMig obtains optimized results in a time which is much shorter than for NFStat (average $0.7885s$ for NFMig while $5.746s$ for NFStat) and

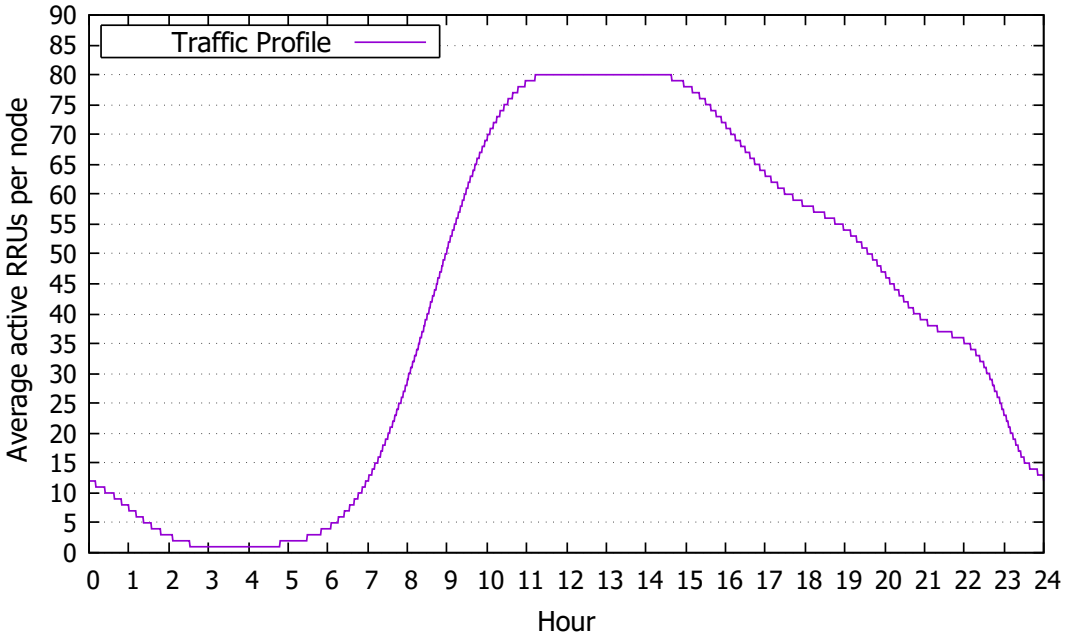


Figure 3.2: Average traffic offered to the network during 24 hours expressed in active RRUs per node

always far below 1 minute, that can be chosen as the basic granularity for further evaluations.

3.3.1 Fine time granularity evaluations

Firstly, NFMig was applied with a fine time granularity, which is every 60 seconds in the 24 hour range. The time limit was set to 50 seconds, with at most 35 seconds for Step 1 and the residual time for Step 2. For NFStat, at most 40 seconds were given to BBU Hotel optimization and the residual time to hop optimization.

Figure 3.4 shows the number of active BBU Hotels computed by NFMig in comparison with the number of active BBU Hotels obtained by NFStat, with the same time granularity. NFMig and NFStat are called once every minute throughout a period of one day, for a total of 1440 problem instances. NFMig reached the time limit in only 4 instances out of 1440: also for those 4 instances the optimal solution was found, missing the proof of optimality only. NFStat reached the time limit in several instances, nevertheless it generally computed solutions with the same or a better number of active nodes than NFMig. The reason is that NFStat, myopic to migrations, completely neglects the solution computed in the previous time period. However, one can observe that on average there is not much difference between the two sets of results. In particular, the average worsening throughout all instances is equal to 0.1169 BBU Hotels. Focusing on the time intervals in which the number of active BBU Hotels varied the most, between 6:00 and 9:00 the average worsening is equal to 0.1018 BBU Hotels, whereas between 20:00 and 24:00 is equal to 0.5991 BBU Hotels. Therefore, the solutions obtained by NFMig are either optimal or near optimal even if NFMig also accounts for migrations, and one can conclude that the minimization of the displacement from

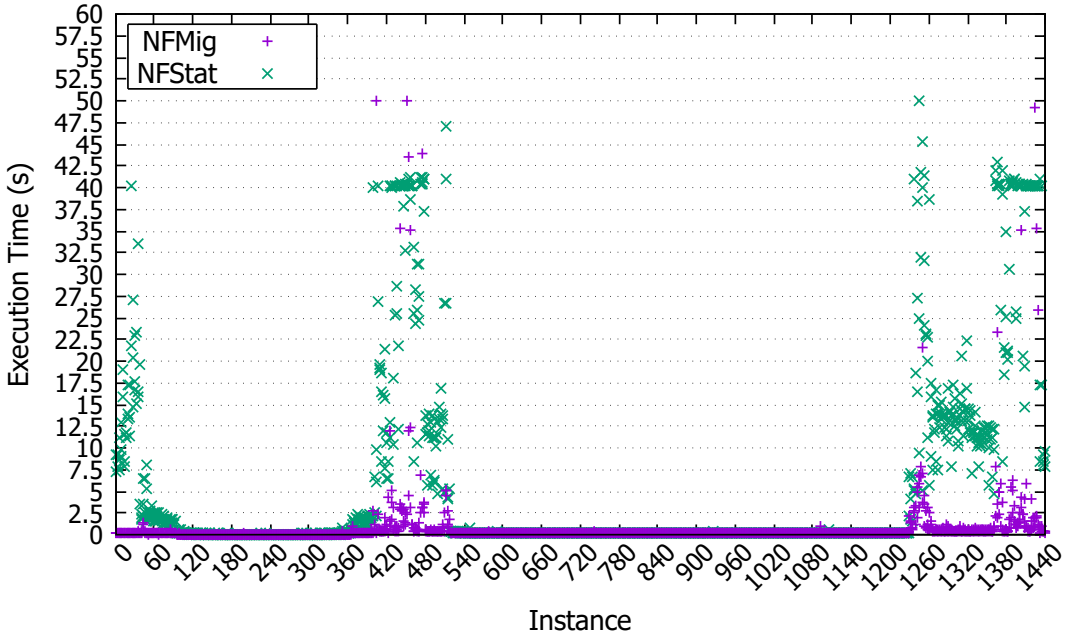


Figure 3.3: Execution time of each instance for NFMig and NFStat, with time granularity equal to 1 minute over 24 hours.

the previous time instance does not hinder the number of active BBU Hotels.

In Figure 3.5 the total number of hops computed by NFMig is compared with that obtained by NFStat in the same time limit. The average worsening throughout all instances is equal to 7.333 hops, between 6:00 and 9:00 is equal to 21.61 hops, and between 20:00 and 24:00 is equal to 9.390 hops. Since the minimization of the total number of hops has the lowest priority among all objectives, it is reasonable to observe a larger worsening with respect to optimal or near-optimal static traffic solutions. Nevertheless, the obtained solutions are of acceptable quality (6.43% average difference from NFStat optimum) and most importantly compliant with the maximum distance constraint, which is paramount for latency requirements.

In Figures 3.6 and 3.7 the total number of primary migrations and backup migrations are shown. One can observe that for NFStat, which completely disregards migrations, the overall number of migrations is completely out of control except for the instances in which the fronthaul links capacity approaches saturation. In all other instances, the small random contribution in the number of active RRUs per node is enough to completely destabilize NFStat. In particular, the total number primary and backup migrations for NFStat are respectively equal to 15703 and 15307, whereas for NFMig they are equal to 648 and 925. One can compute the improvement provided by NFMig with respect to NFStat as follows:

$$\text{Improvement\%}_{\text{NFMig}} = \frac{\#\text{Migs}_{\text{NFStat}} - \#\text{Migs}_{\text{NFMig}}}{\#\text{Migs}_{\text{NFStat}}} \cdot 100$$

Therefore, there are 95.87% less primary migrations and 93.95% less backup migrations than in NFStat, with a worst case improvement of 18.75% and 25% for primary and backup migrations respectively.

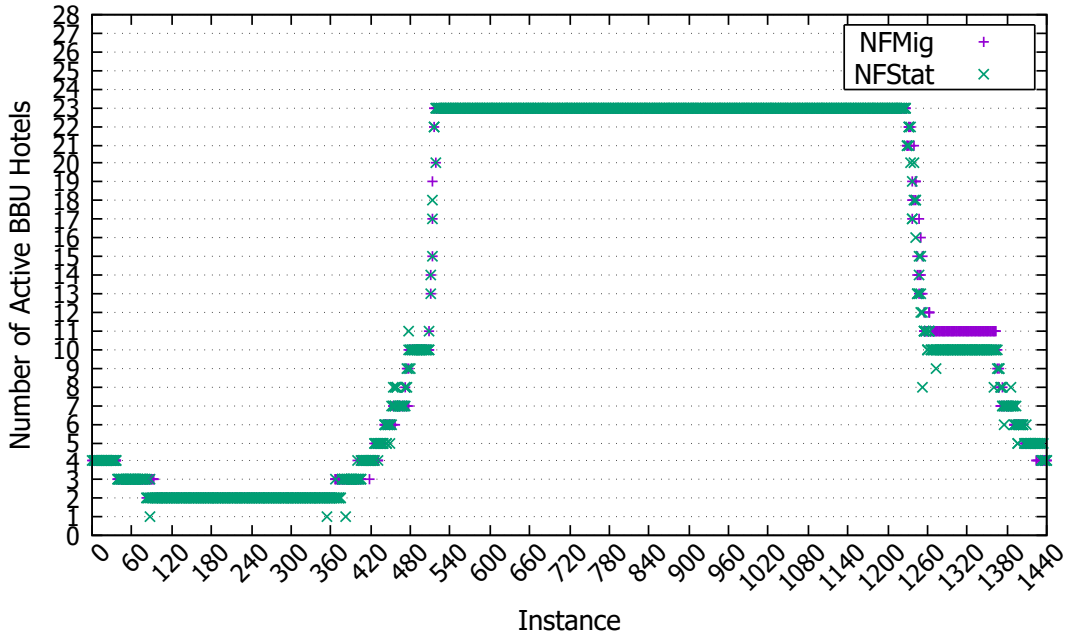


Figure 3.4: Number of active BBUs of each instance for NFMig and NFStat, with time granularity equal to 1 minute over 24 hours.

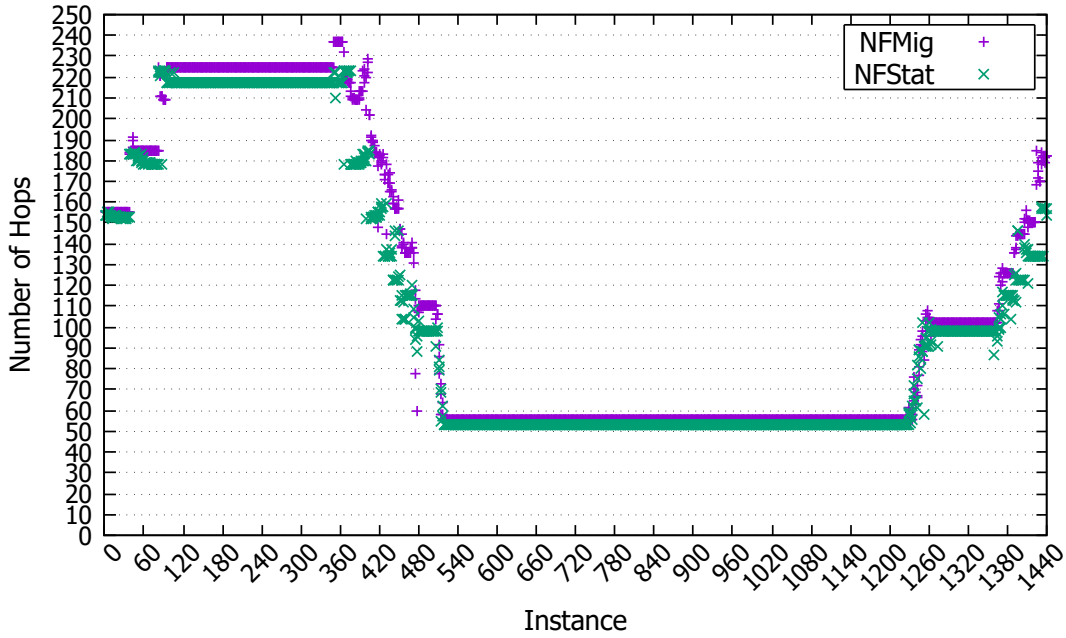


Figure 3.5: Total number of hops of each instance for NFMig and NFStat, with time granularity equal to 1 minute over 24 hours.

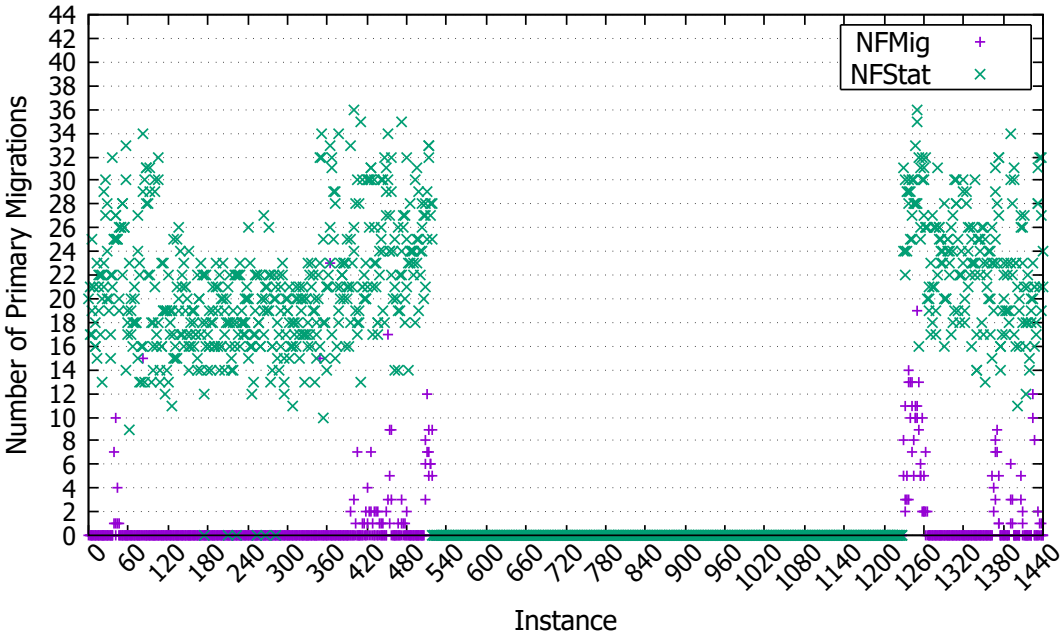


Figure 3.6: Primary BBU virtual function migrations of each instance for NFMig and NFStat, with time granularity equal to 1 minute over 24 hours.

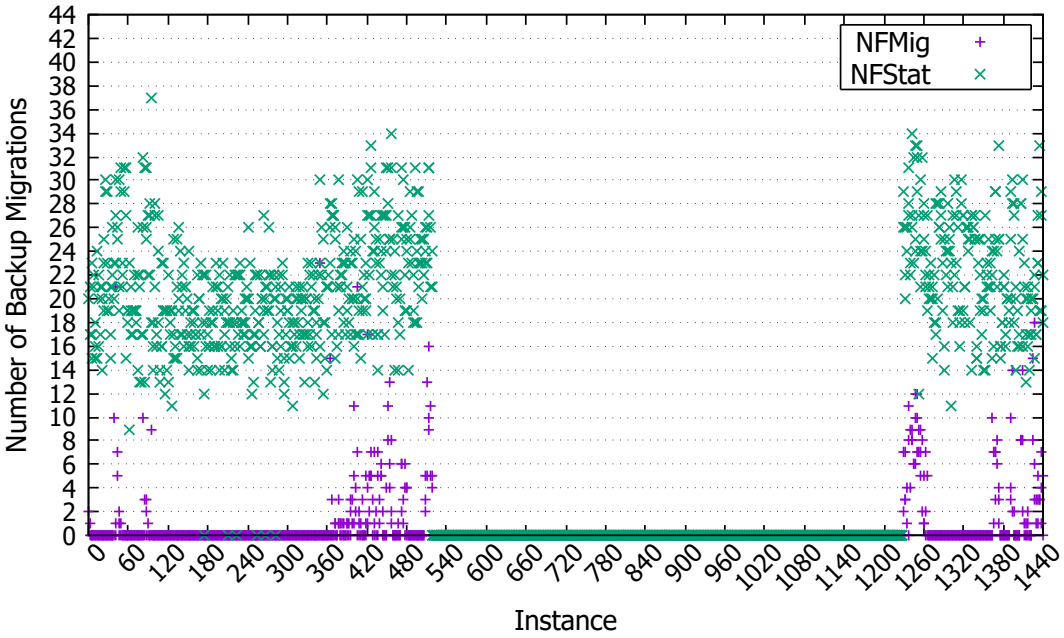


Figure 3.7: Backup BBU virtual function migrations of each instance for NFMig and NFStat, with time granularity equal to 1 minute over 24 hours.

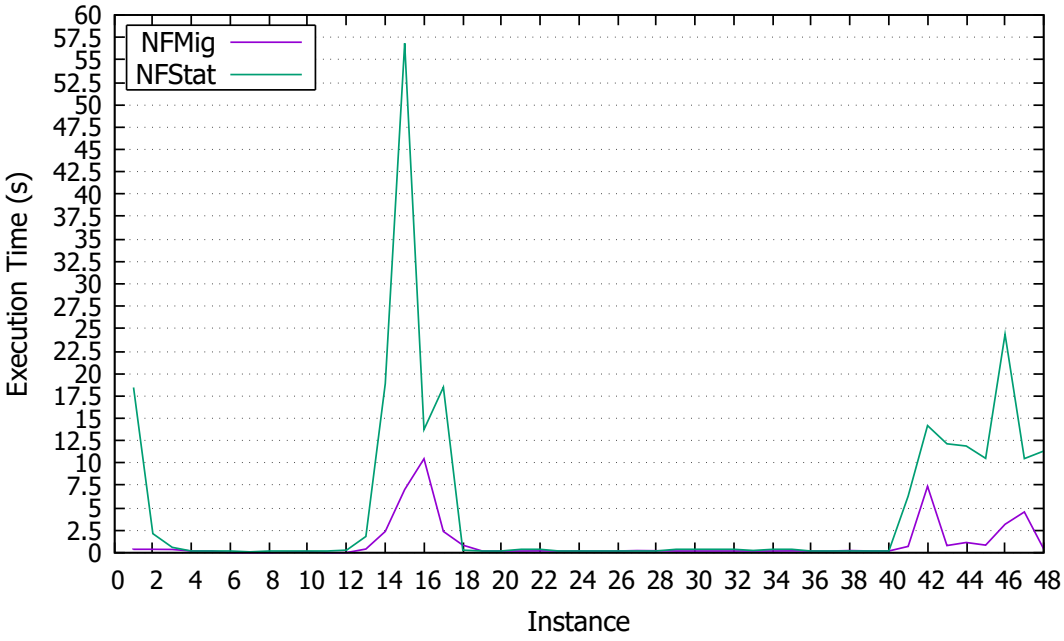


Figure 3.8: Execution time of each instance for NFMig and NFStat, with time granularity equal to 30 minutes over 24 hours.

Furthermore, for NFMig, one can observe that the large majority of migrations is concentrated in the time instants in which new nodes are either activated or deactivated, as expected. Considering the mean traffic profile, it is reasonable to execute the algorithm between short time intervals when the traffic is rising, so as to maximize responsiveness. However, when the mean traffic is decreasing, it may be beneficial to execute the algorithm between longer time intervals, and perform BBU Hotels de-activations and consequent migrations only a few times over the descending slope.

3.3.2 Coarse time granularity evaluations

Secondly, the algorithm was applied with a coarser time granularity, which is every thirty minutes. The time limit was set to 1700 seconds, with at most 1200 seconds for Step 1 and 500 seconds for Step 2.

Figure 3.8 shows the computing times of NFMig and NFStat for each instance. One can observe that even though the time granularity is much coarser than in the previous case, and therefore traffic variations between two instances are much larger, the average computing times remain in the same order of magnitude as shown in Figure 2.4.

Figure 3.9 shows the number of active BBU hotels of NFMig versus NFStat. NFMig and NFStat are called once every 30 minutes throughout a period of one day, for a total of 48 problem instances. The average worsening of NFMig with respect to NFStat over the entire day is equal to 0.1667 BBU Hotels, between 6:00 and 9:00 is equal to 0.2857 BBU Hotels and between 20:00 and 24:00 is equal to 0.5556 BBU Hotels. Thus, even though a coarser time granularity was considered and therefore larger traffic variations, also in this case the minimization of the

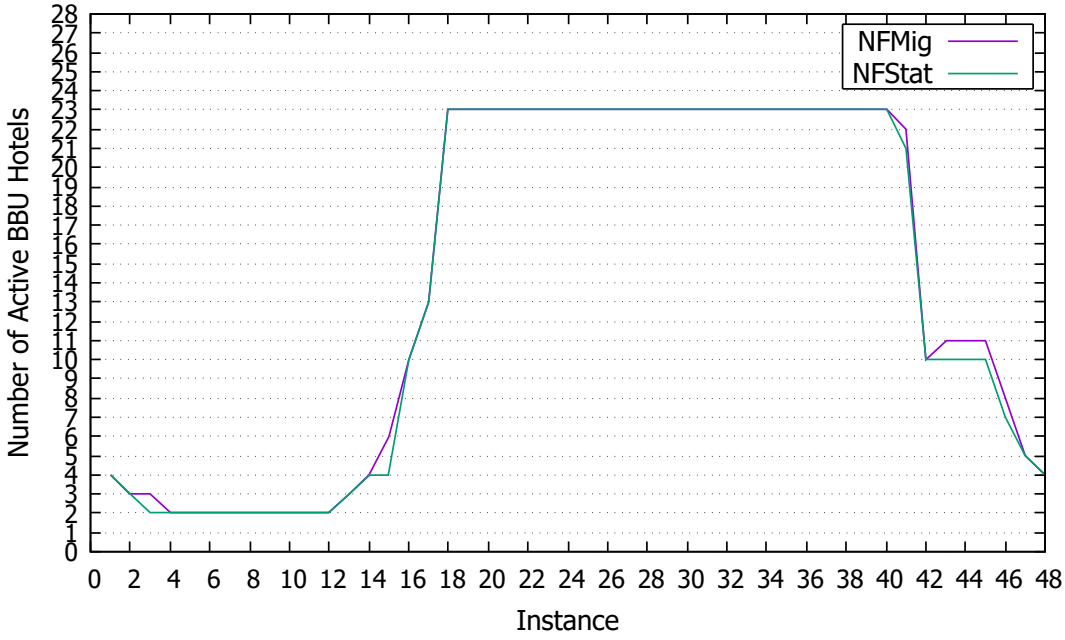


Figure 3.9: Number of active BBUs of each instance for NFMig and NFStat, with time granularity equal to 30 minutes over 24 hours.

displacement from the previous solution does not hinder the number of active BBU Hotels.

Figure 3.10 shows the total number of hops of NFMig versus NFStat. The average worsening of NFMig with respect to NFStat over the entire day is equal to 5.511 hops, between 6:00 and 9:00 is equal to 13.66 hops and between 20:00 and 24:00 is equal to 10.37 hops. As for the scenario with finer time granularity, we still get solutions of acceptable quality (5.147% average difference from NFStat optimum), even though minimization of the total number of hops is assigned the lowest priority among objectives.

Figures 3.11 and 3.12 show the number of primary and backup migrations, respectively, of NFMig versus NFStat. Overall, NFMig performs 82.58% and 75.04% less primary and backup migrations, respectively, than NFStat. In the worst case, NFMig performs 33.3% and 20.83% less primary and backup migrations, respectively, than NFStat. The total number of migrations is much smaller with respect to the 1m time granularity: this can be of particular interest in case of decreasing traffic, where the timeliness of VNFs reconfiguration is not crucial, thus saving computing power. As an example, the scenario between 20:00 and 24:00 shows 830 migrations with 1m time granularity, and 90 migrations with 30m time granularity. Therefore, since with decreasing traffic the current solution maintains feasibility, the advantage of adopting a coarser time granularity is twofold: firstly, the algorithm can be called fewer times, thus saving computational power; secondly, the total number of migrations to be performed is much smaller, avoiding redundant re-configurations while deactivating unnecessary BBU Hotels. Instead, when the traffic is increasing, it is paramount to reconfigure the network as fast as possible, therefore a finer granularity should be adopted.

In Figure 3.13a the active BBU Hotels, highlighted in red, at 7:30 are shown. At

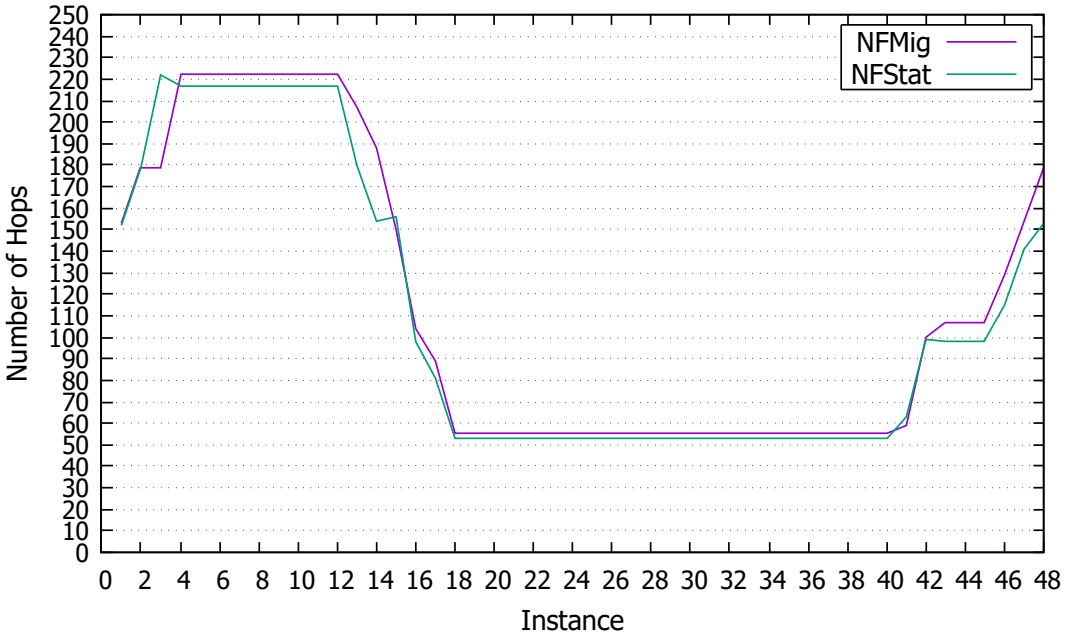


Figure 3.10: Total number of hops of each instance for NFMig and NFStat, with time granularity equal to 30 minutes over 24 hours.

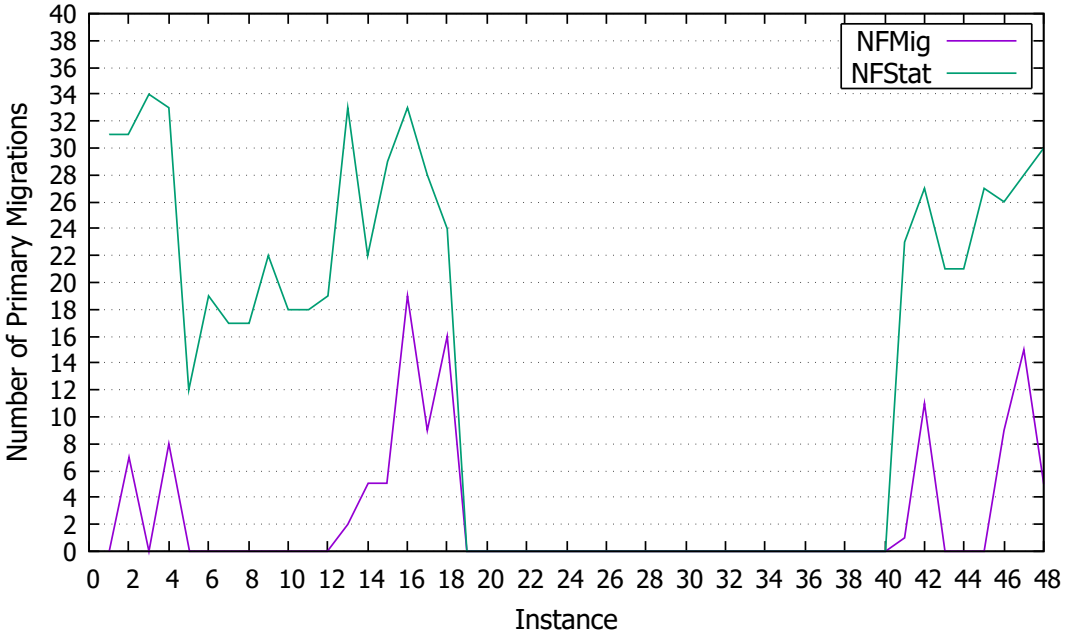


Figure 3.11: Primary BBU virtual function migration of each instance for NFMig and NFStat, with time granularity equal to 30 minutes over 24 hours.

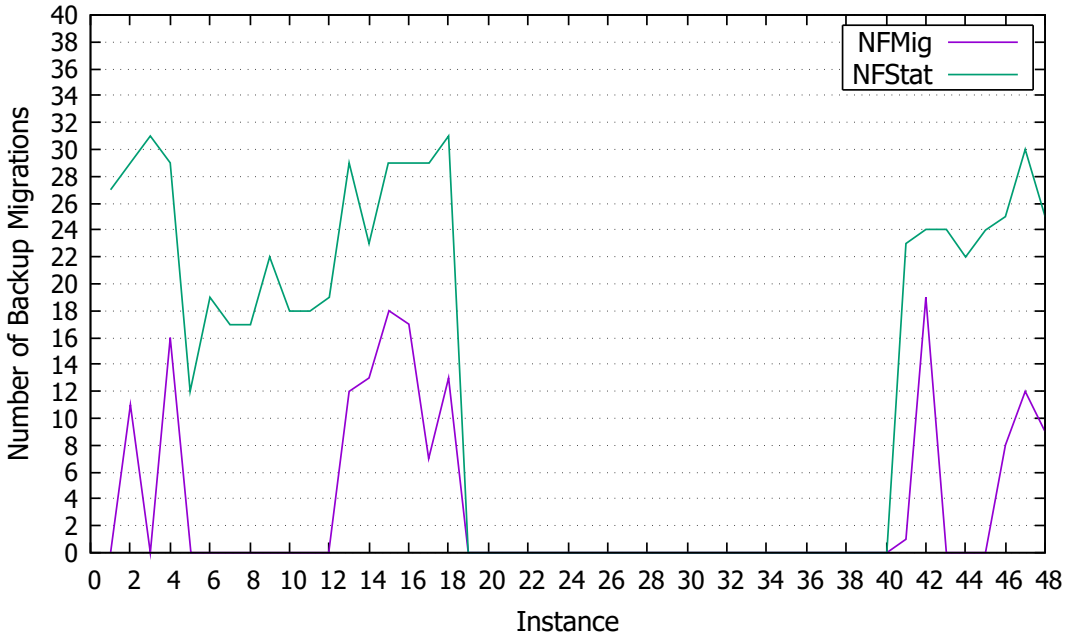
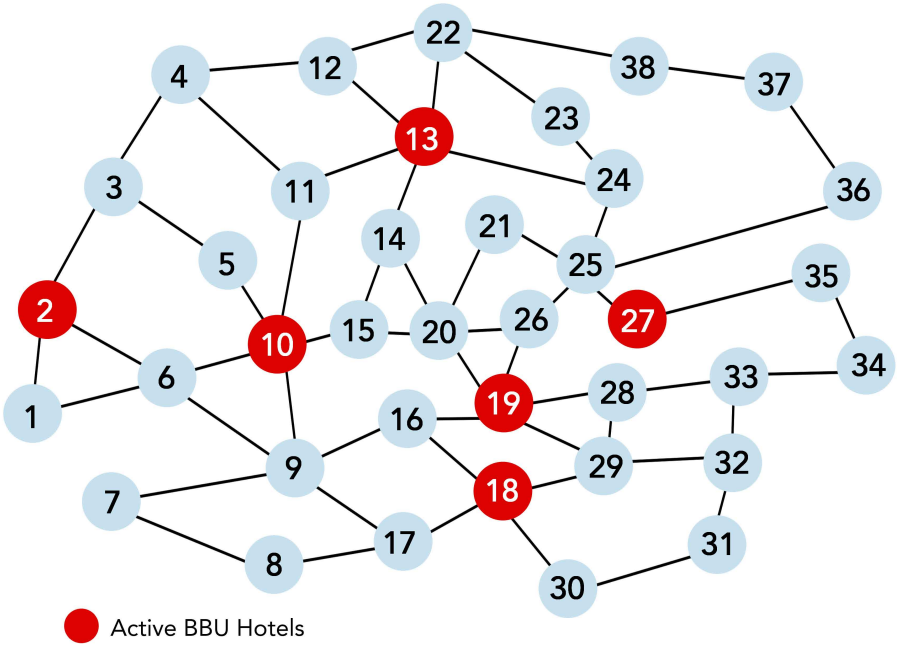


Figure 3.12: Backup BBU virtual function migration as a function of time performed by NFMig and NFStat, with time granularity equal to 30 minutes.

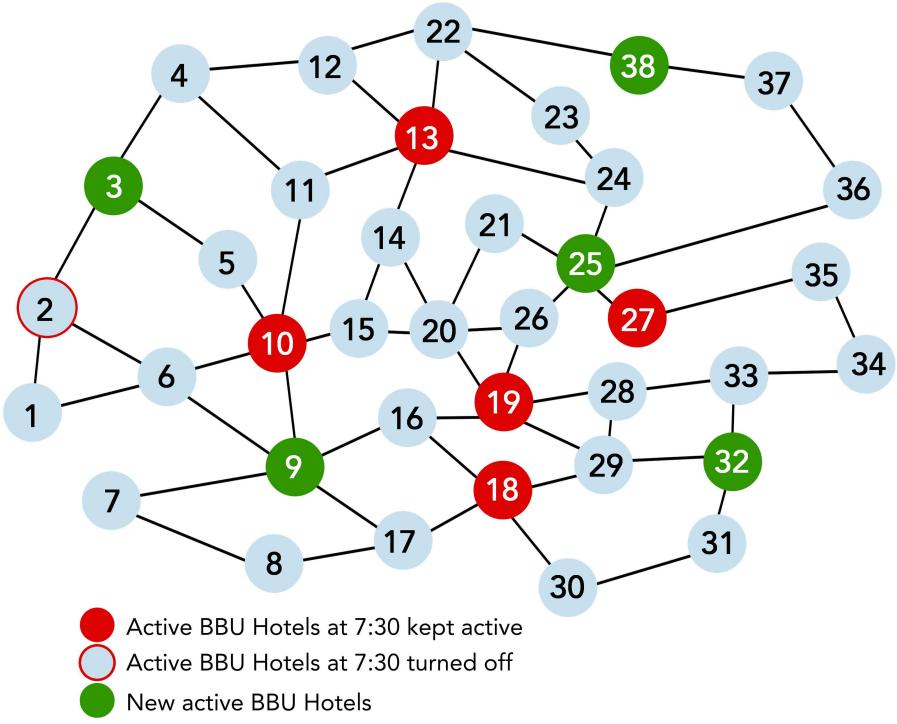
7:30 there are 6 active BBU Hotels (2, 10, 13, 18, 19, 27), as reported in Figure 3.9. In Figure 3.13b the active BBU Hotels at 8:00 are reported after a traffic increase with respect to 7:30. In particular, previously active BBU Hotels which are kept active are highlighted in red, previously active BBU Hotels that have been deactivated are circled in red, and new active BBU Hotels are highlighted in green. As reported in Figure 3.9, at 8:00 there are 10 active BBU Hotels, 6 more than 7:30. One can observe that 5 new nodes (3, 9, 25, 32, 38) have been activated: had one fixed the previous solution, that would have led to 11 active BBU Hotels. Since the objective function of NFMig rewards the deactivation of previously activated BBU Hotels when they are not anymore needed, node 2 has been deactivated. This allows NFMig to reach the optimal NFStat solution value of 10 active BBU Hotels, in shorter computing time and by also minimizing the total number of migrations to be performed.

3.4 Conclusions

An optimization algorithm, based on sequentially solving an ILP model by a lexicographic method, is proposed to design reliable latency-constrained C-RAN with dynamic traffic. The model accounts for virtual BBU migration cost and for penalties and rewards in the presence of activation or de-activation of BBU Hotels between two time intervals. The algorithm is applied to a 24 hour traffic profile to show its effectiveness in maintaining optimized resource assignment according to the chosen time granularity. The results obtained with the proposed NFMig algorithm are compared with those of the NFStat algorithm illustrated in 2.2.2, defined for static traffic. It is shown that NFMig shows negligible degrada-



(a) Active BBU Hotels at 7:30



(b) Active BBU Hotels at 8:00

Figure 3.13: Active BBU Hotels activations and deactivations between 7:30 and 8:00, with time granularity of 30 minutes.

tion in terms of number of active BBU Hotels and total hops with respect to the optimum achieved by NFStat, with remarkable reduction in the total number of virtual function migrations, 82% and higher for primary and 75% and higher for backup functions. The execution time, in the range of few tens of seconds, allows fine granularity in optimized design adaptation to traffic changes.

Chapter 4

Column Generation Model

In this chapter, an alternative model for the optimal BBU placement problem for a static traffic scenario will be developed, and its linear relaxation solved via a Column Generation procedure. A basic Column Generation framework aimed at the minimization of the number of active BBU Hotels is developed, that can be used as a baseline for extensions aimed at more complex modelling, exact branch-and-price algorithms or Column Generation based heuristics.

4.1 Problem Statement

The problem considered in this chapter is the Linear Programming relaxation of the first step of the lexicographic method developed for the static traffic scenario, that is, the minimization of the number of active BBU Hotels under maximum distance and maximum wavelengths constraints.

The main difference from the previous approach is that now each variable models a set of assignments of RRUs to a certain BBU Hotel, greatly reducing the symmetry present in the original formulation, at the price of a much larger number of decision variables. Given the fact that in general the number of possible assignments is exponential, and therefore it is impractical to include in the model all variables, the model is initialized with an initial subset of variables ensuring the existence of at least one feasible solutions, and variables are generated dynamically via a Column Generation procedure.

4.2 Optimization Model

The optimization model consists in a Master Problem and in a Column Generation Subproblem. The purpose of the Master problem is to minimize the number of active BBU Hotels, that is the primary objective in the problem statement. Due to the fact that the number of variables can be too large to be effectively managed by the solver, the Master problem is initialized with a subset of the variables. Thus, the purpose of the Column Generation subproblem is to generate feasible assignments that improve the current objective function value of the Master problem.

Table 4.1: Models parameters and variables

	Parameters
S	Set of transport nodes.
L	Set of links.
A	Set of feasible RRU assignments.
H	$s \times s$ matrix. h_{ij} is the distance in hops between nodes i and j computed with the shortest path.
r_i	Number of RRUS at site i, $i \in S$.
δ_{ij}^l	1 if shortest path between i and j is using link l, 0 otherwise, i, $j \in S$, $l \in L$
$b_{i,j}$	1 if node j is the BBU Hotel for assignment i, 0 otherwise, $j \in S$, $i \in A$
$a_{i,j}$	1 if RRUs at node j are in assignment i, 0 otherwise, $j \in S$, $i \in A$
$w_{i,l}$	Number of wavelengths occupied in link l by assignment i, $l \in L$, $i \in A$
M_W	Maximum number of wavelengths available in each link.
M_H	Maximum allowed distance in hops between RRUs and BBUs.
M	A large number
	Variables: Master Problem
x_i	1 if assignment i is selected, 0 otherwise, $i \in A$
	Variables: Column Generation Subproblem
α_j	1 if RRUs at node j are in the assignment, 0 otherwise, $j \in S$
β_j	1 if node j is the BBU Hotel for the assignment, 0 otherwise, $j \in S$
λ_l	number of wavelengths of the assignment on link l, $l \in L$
π_j	Dual variables for constraints (4.2), $j \in S$, ≤ 0
μ_j	Dual variables for constraints (4.3), $j \in S$, free
γ_l	Dual variables for constraints (4.4), $l \in L$, ≤ 0

The optimization model for the Master problem reads as follows:

$$\min C_m = \sum_i x_i \tag{4.1}$$

$$\sum_{i \in A} x_i \cdot b_{i,j} \leq 1 \quad \forall j \in S \tag{4.2}$$

$$\sum_{i \in A} x_i \cdot a_{i,j} = 2 \quad \forall j \in S \tag{4.3}$$

$$\sum_{i \in A} x_i \cdot w_{i,l} \leq M_W \quad \forall l \in L \tag{4.4}$$

$$x_i \in \{0, 1\} \tag{4.5}$$

Objective function (4.1) minimizes the number of active BBU Hotels. In fact, each decision variable represents a feasible assignment, that is one active BBU Hotel and all of its assigned RRUs. Therefore, minimizing the number of utilized assignments is equivalent to minimizing the number of active BBU Hotels. Constraints (4.2) ensures that only one assignment per possible active BBU Hotel can be selected. In fact, many feasible assignment per active BBU Hotel may exists, but ultimately at most one of them can be selected for the final solution. Constraints (4.3) ensure that each RRU has a primary and backup BBU Hotel assigned. This is equivalent to imposing that each RRU must appear in exactly two assignment. Due to constraints (4.3) imposing that only one assignment can be selected per possible active BBU Hotel, it is ensured that primary and backup BBU Hotels are also distinct. Constraints (4.4) impose that the maximum number of wavelengths in each WDM fronthaul link does not exceed M_W. Finally, constraints (4.5) define the variable domains.

The Master problem is initialized with a subset of all possible assignments. In order to add to the problem only assignments that can improve the current solution, the reduced cost of the new potential assignment is employed as the objective function for the Column Generation subproblem. The optimization model for the Column Generation subproblem reads as follows:

$$\min C_s = 1 - \sum_{j=1}^{|S|} \beta_j \cdot \pi_j - \sum_{j=1}^{|S|} \alpha_j \cdot \mu_j - \sum_{l=1}^{|L|} \lambda_l \cdot \gamma_l \tag{4.6}$$

$$\sum_{j=1}^{|S|} \beta_j = 1 \tag{4.7}$$

$$\alpha_j \cdot h_{k,j} \leq M_H + M \cdot (1 - \beta_j) \quad \forall j, k \in S \tag{4.8}$$

$$\lambda_l \geq \sum_{j=1}^{|S|} \alpha_j \cdot \delta_{k,j}^l \cdot r_j - M \cdot (1 - \beta_j) \quad \forall k \in S,\, l \in L \tag{4.9}$$

$$\lambda_l \leq M_w \quad l \in L \tag{4.10}$$

$$\beta_j \in \{0, 1\} \tag{4.11}$$

$$\alpha_j \in \{0, 1\} \tag{4.12}$$

$$\lambda_l \in \mathbb{N} \tag{4.13}$$

Objective function (4.6) is the reduced cost of the assignment. A negative reduced cost for a variable expresses the improvement in the objective function value of the Master problem per unit increase of such variable. Therefore, finding a variable with negative reduced cost implies finding a feasible assignment that, added to the Master problem, will improve the current solution. On the other hand, not finding any variable with negative reduced cost would imply that no assignment exist that can improve the current solution, and thus the current solution is also the optimal one. Constraint (4.7) impose that the assignment must have exactly one active BBU Hotel. Constraints (4.8) impose that the distance in hops between the RRUs in the assignment and the active BBU Hotel does not exceed M_H. Constraints (4.9) count the number of wavelengths over each link with respect to each path from the active BBU Hotel to each assigned RRU. Constraints (4.10) impose that the total number of wavelength per link does not exceed the maximum capacity M_W. Finally, constraints (4.11)-(4.13) define the variable domains.

An alternative formulation for the Column Generation subproblem is to fix one active BBU Hotel and to find an improving assignment for that active BBU Hotel only. Differently from the previous formulation, $|S|$ subproblems need to be solved per iteration, with at most $|S|$ generated columns per iteration. The alternative Column Generation subproblem formulation reads as follows:

$$\forall k \in S :$$

$$1 - \pi_k + \min - \sum_{j=1}^{|S|} \alpha_j \cdot \mu_j - \sum_{l=1}^{|L|} \lambda_l \cdot \gamma_l \tag{4.14}$$

$$\alpha_j \cdot h_{k,j} \leq M_H \quad \forall j \in S \tag{4.15}$$

$$\lambda_l = \sum_{j=1}^{|S|} \alpha_j \cdot \delta_{k,j}^{l} \cdot r_j \quad \forall l \in L \tag{4.16}$$

$$\lambda_l \leq M_w \quad \forall l \in L \tag{4.17}$$

$$\alpha_j \in \{0, 1\} \tag{4.18}$$

$$\lambda_l \in \mathbb{N} \tag{4.19}$$

Objective function (4.14) is the reduced cost of the assignment, having the BBU Hotel active at node k. Constraints (4.15) impose that the maximum distance between the BBU Hotel at node k and its assigned RRUs does not exceed M_H. These constraints can be eliminated via a simple preprocessing operation, fixing to 0 all variables α_j that violate the constraints. Constraints (4.16) and (4.17) count the number of wavelengths in the link between the BBU Hotel at node k and the assigned RRUs, and ensure that the number of wavelengths per link does not exceed the maximum M_W. Finally, constraints (4.18)-(4.19) define the variable domains.

4.3 Implementation

In this section, a sample C++ implementation in CPLEX Concert Technology for the NFMig algorithm shall be provided. In particular, it will be shown how to

set up the Column Generation loop for the case in which the Column Generation subproblem has to be solved for each BBU Hotel. In this implementation, for demonstrative purposes, the Column Generation subproblem at each iteration is solved exactly, by solving the model with the solver. Implementation and testing of a heuristic pricing algorithm for the subproblem is left as future work. It will be assumed that the model parameters have been imported from properly formatted .dat files and stored in appropriate data structures.

The first step is to build the Master problem model and initialize it with a set of initial variables that provide an initial feasible solution. For instance, a simple initialization set can be built considering all BBU Hotels to be active and to consider them in pairs of adjacent neighbors, assigning for each one the RRUs at their own transport node and the RRUs at the other node in the pair. In this way, one easily ensures that each RRU has a primary and a backup BBU Hotel.

Initialization vectors are stored in properly formatted .dat files. The syntax for loading the initialization data into the proper data structure is the following:

```
IloIntArray2 bbu_init(env, nbNodes);
for (IloInt i = 0; i < nbNodes; ++i)
   bbu_init[i] = IloIntArray(env, nbNodes);
ifstream bbuInit("bbu_init.dat");
bbuInit >> bbu_init;
bbuInit.close();

IloIntArray2 rru_init(env, nbNodes);
for (IloInt i = 0; i < nbNodes; ++i)
   rru_init[i] = IloIntArray(env, nbNodes);
ifstream rruInit("rru_init.dat");
rruInit >> rru_init;
rruInit.close();

IloIntArray2 wavs_init(env, nbNodes);
for (IloInt i = 0; i < nbNodes; ++i)
   wavs_init[i] = IloIntArray(env, nbLinks);
ifstream wavsInit("wavs_init.dat");
wavsInit >> wavs_init;
wavsInit.close();
```

In this way, a number of initialization assignments equal to the number of nodes is created, characterized by the values of the parameters a_{ij}, b_{ij} and w_{ij}.

Differently from the previous implementation, the model needs to be dynamically extended as new columns are generated, therefore it needs to be populated in a different way. The syntax for creating the model, the decision variables, the objective function and the constraints reads as follows:

```
IloIntArray rhs_bbu(env, nbNodes);
for (IloInt i = 0; i < nbNodes; ++i)
   rhs_bbu[i] = 1;
IloIntArray rhs_rru(env, nbNodes);
for (IloInt i = 0; i < nbNodes; ++i)
```

```
   rhs_rru[i] = 2;
IloIntArray rhs_wavs(env, nbLinks);
for (IloInt l = 0; l < nbLinks; ++l)
   rhs_wavs[l] = maxWavs;

IloModel master(env);

IloObjective activeHotel = IloAdd(master, IloMinimize(env));
IloRangeArray bbuConstraints = IloAdd(master,
   IloRangeArray(env, 0, rhs_bbu));
IloRangeArray rruConstraints = IloAdd(master,
   IloRangeArray(env, rhs_rru, IloInfinity));
IloRangeArray wavsConstraints = IloAdd(master,
   IloRangeArray(env, 0, rhs_wavs));

IloNumVarArray assign(env);
```

In this way, an empty objective function and empty constraints are created, which are going to be progressively populated during the Column Generation process. In particular, for the objective function it is sufficient to specify that it is a minimization problem. For the constraints, lower bounds and upper bounds are specified according to the mathematical model that has been defined. Finally, array "assign" will hold the decision variables for each assignment.

The syntax for the initialization of the Master problem reads as follows:

```
for (IloInt i = 0; i < nbNodes; ++i)
   assign.add(IloNumVar(activeHotel(1) +
      bbuConstraints(bbu_init[i]) + rruConstraints(rru_init[i])
      + wavsConstraints(wavs_init[i])));
```

The one-liner within the loop performs several actions. With the "add" method, a new variable is added to the "assign" array. As arguments for the definition of the new variables, the coefficients in the objective function and in the constraints are specified. In the above syntax, "activeHotel(1)" imposes that the coefficients of the new variable (which is one of the x_i variables) in the objective function is equal to 1, whereas for the constraints it specifies the initialization coefficients in the constraint matrix a_{ij}, b_{ij} and w_{ij}, which were imported from the initialization data file.

As a next step, an IloCplex object acting as a solver for the Master problem is created, along with data structures for holding the dual prices and the coefficients for the new assignment generated via Column Generation. The syntax is the following:

```
IloCplex masterSolver(master);

IloNumArray price_bbu(env, nbNodes);
IloNumArray price_rru(env, nbNodes);
IloNumArray price_wavs(env, nbLinks);
```

```
IloNumArray new_bbu(env, nbNodes);
IloNumArray new_rru(env, nbNodes);
IloNumArray new_wavs(env, nbLinks);
```

At this point, the Column Generation loop is started. In this particular implementation, the Column Generation subproblem model is instantiated, solved and destroyed within the loop. Had one implemented the first method for the Column Generation subproblem, in which it needs to be solved only once, the model could have been defined outside the loop as for the Master problem. In this case, where $|S|$ problems needs to be solved per loop iteration, it would have been impractical to define $|S|$ models outside the loop, considering also the fact that for large network it could cause a non-negligible memory overhead. Besides, in Column Generation based heuristics, exact methods are called only in case of a negative response from the pricing heuristic algorithm. Therefore, it is more convenient, also for future extensions, to instantiate and destroy the subproblem within the loop.

The loop can be declared as follows:

```
IloInt check = 1;
while (check) {
   // Column Generation Procedure
}
```

This means that the loop will run until a certain check is satisfied. The check will return false when no assignment with negative reduced cost is returned from solving the Column Generation subproblem.

As a first step within the Column Generation loop, the Master problem is solved and the values of the dual variables are stored in the data structures created before, querying them from the IloCplex masterSolver object via the "getDual" method. The syntax reads as follows:

```
masterSolver.solve();

for (IloInt i = 0; i < nbNodes; ++i)
   price_bbu[i] = masterSolver.getDual(bbuConstraints[i]);

for (IloInt i = 0; i < nbNodes; ++i)
   price_rru[i] = -masterSolver.getDual(rruConstraints[i]);

for (IloInt l = 0; l < nbLinks; ++l) {
   price_wavs[l] = -masterSolver.getDual(wavsConstraints[l]);
```

At this point, a for loop solving the Column Generation subproblem is declared:

```
for (IloInt node = 0; node < nbNodes; ++node) {
// Subproblem building and solving
// If (reduced cost < 0) add column to the Master problem
// else break
}
```

In the following, until otherwise specified, it will be assumed that the code is running within the for loop. The Column Generation subproblem model can be built as follows:

```
IloEnv env2;

IloModel colGen(env2);

IloObjective reducedCost = IloAdd(colGen, IloMinimize(env2, 1 -
   price_bbu[node]));
IloNumVarArray rru(env2, nbNodes, 0, 1, ILOINT);
IloNumVarArray wavs(env2, nbLinks, 0, maxWavs, ILOINT);

// Constraints 4.15
for (IloInt j = 0; j < nbNodes; ++j)
   colGen.add(rru[j] * costMatrix[node][j] <= maxDistance);

// Constraints 4.16
for (IloInt l = 0; l < nbLinks; ++l) {
   IloExpr v(env2);
   for (IloInt j = 0; j < nbNodes; ++j) {
      v += rru[j] * delta[l][node][j] * rrusAtNode[j];
   }
   colGen.add(wavs[l] == v);
   v.end();
}

reducedCost.setLinearCoefs(rru, price_rru);
reducedCost.setLinearCoefs(wavs, price_wavs);

IloCplex colSolver(colGen);
```

A new optimization environment is instantiated, in which variables and constraints are declared. Particular care must be put when working with multiple optimization environments, as operations between objects pertaining to different environments will raise an exception. Furthermore, since the instantiation of these object is within the Column Generation loop, which is expected to run for a large number of iterations, care must be taken in freeing correctly the allocated memory, otherwise severe memory bloating will arise.

The objective function initialized with the constant terms (1 and the price related to the selected active BBU Hotel), whereas for variables α and λ the prices are set via the method "setLinearCoefs" of the IloObjective reducedCost object, taking as arguments a vector of variables and their linear coefficients in the objective functions. Constraints are added via the "add" method, in the same way the models in the previous chapters were built. Finally, an IloCplex object "colSolver" is created to be used as a solver for the Column Generation subproblem.

The model is then solved, and if the reduced cost turns out to be negative a new column is added to the Master problem, otherwise the current loop iteration is ended. The syntax is the following:

```
colSolver.solve();

// Check if reduced cost is above tolerance
if (colSolver.getValue(reducedCost) >= -TOL) {
   env2.end(); // free memory!
   continue;
}

check = 1;

for (IloInt i = 0; i < nbNodes; ++i) {
   if (i == node)
      new_bbu[i] = 1;
   else
      new_bbu[i] = 0;
}

for (IloInt i = 0; i < nbNodes; ++i) {
   if (colSolver.isExtracted(rru[i]))
      new_rru[i] = colSolver.getValue(rru[i]);
   else
      new_rru[i] = 0;
}

for (IloInt l = 0; l < nbLinks; ++l)
   if (colSolver.isExtracted(wavs[l]))
      new_wavs[l] = colSolver.getValue(wavs[l]);
   else
      new_wavs[l] = 0;

env2.end(); // free memory!

assign.add(IloNumVar(activeHotel(1) + bbuConstraints(new_bbu) +
   rruConstraints(new_rru) + wavsConstraints(new_wavs)));
```

The first if statement checks whether the objective function value, that is the reduced cost of the column, is above a very small tolerance (e.g. 10^{-6}). In case of a reduced cost above the tolerance, the new optimization environment is destroyed and the current loop iteration is ended, moving on to the subproblem for the next active BBU Hotel. In case of a negative reduced cost, the new generated column is added to the data structures that were created previously. Before querying the model for the values of the variables, one must check whether or not these variables have been extracted, otherwise an exception might be thrown. After storing the values of the column coefficients in the appropriate data structures, the new optimization environment is destroyed. Finally, the new generated column is added to the Master problem using the same syntax that was shown during the initialization phase, specifying the coefficients for the new variable in the Master problem objective function and constraints.

At this point in the code both the inner loop (the one cycling all active BBU Hotels) and the outer loop (the Column Generation loop) are terminated. At this point, the optimal solution of the LP relaxation of the optimal BBU Hotel placement problem, hence a lower bound, is found. A simple heuristic for finding an integer solution can be the following:

```
master.add(IloConversion(env, assign, ILOINT));
masterSolver.exportModel("D:/PNET/model.lp");
masterSolver.solve();
cout << "Solution status: " << masterSolver.getStatus() << endl;
cout << "Solution value: " << masterSolver.getObjValue() <<
   endl;
```

The variables for the assignments are converted from float to integer variables, and the model is solved using the generated columns. Another possible heuristic framework could be to heuristically fix one column at a time based on some criteria (e.g. the one that in the LP relaxation solution is closer to 1, the last column added, and so on) and repeat the Column Generation procedure in order to generate additional columns until the integer solution is found.

4.4 Numerical Evaluations

In this section, the quality of the lower bound obtained for a 100 nodes network instance via the developed Column Generation algorithm is discussed.

4.4.1 Problem Statement

The considered problem is the first step of the Lexicographic method in the static traffic scenario, that is, the minimization of the number of active BBU Hotels, under maximum distance, maximum number of wavelengths per link and redundancy constraints. The evaluations were carried out for the 100 nodes Lattice network, since it is the instance for which the model developed in 2.2.2 did not find an optimal solution within the time limit.

4.4.2 Results

As a first result, it was observed that the approach for which the subproblem is solved $|S|$ times is much more efficient with respect to the approach in which it is solved only once. In particular, the first approach takes 82 seconds to converge, whereas the second approach takes more than an hour to converge. This can be traced back to the fact that more diversified columns are generated, at most one per potential active node. Furthermore, the ILP model in the approach where the subproblem is solved only once presents Big M constraints in the formulation, which are a well known cause of weak lower bounds, notably slowing down optimization as the number of iterations increases.

Regarding the obtained lower bound, it was found that the Column Generation model is capable of finding a lower bound of 7 active BBU Hotels. This is

better than the lower bound found in the 200 seconds time limit by solving the Step 1 presented in 2.2.2. By plugging this lower bound within the first step of the Lexicographic method unfortunately the optimal solution was not found in the residual time, however the same solution previously found (8 active BBU Hotel) was found in the residual time, with a much better optimality gap thanks to the computed lower bound (11.7% versus 25.5%), with an absolute difference of only 1 active BBU Hotel.

4.5 Conclusions

A Column Generation model for solving the linear relaxation of the optimal BBU Hotel placement problem was developed for the first step of the previously developed Lexicographic method in the static traffic scenario. For 100 nodes network instance for which the optimal solution was not found, it was demonstrated that the Column Generation model is capable of achieving a better lower bound in reasonable computing times, improving the optimality gap by 13.8%, with an absolute difference of only 1 BBU Hotel from the best integer solution found. Development of a heuristic pricing algorithm for the subproblem, calling the exact algorithm only when the heuristic fails to return a negative reduced cost, can reduce the computing times. Finally, this baseline can be extended to be part of a Column Generation based heuristic or an exact Branch-and-Price algorithm.

Chapter 5

Conclusions and Future Work

In this thesis work two major problems in C-RAN optimization are considered, and exact algorithmic frameworks are developed.

For the first problem, with reference to a static traffic scenario, a lexicographic method for optimal BBU placement is developed, obtaining scalable and efficient optimization for networks with up to 100 nodes, and the implementation details are outlined. In particular, it is shown that for the two most important objectives, optimality can be guaranteed for networks with up to several tens of nodes in few minutes of computing time, whereas for the largest network of 100 nodes a solution of much better quality is achieved with respect to an aggregate approach. The methodological contribution is also underlined, since in many works in telecommunications engineering literature multi-objective ILP models are solved via an aggregate method: on the contrary, leveraging expert domain knowledge and the application scenario, the lexicographic ordering of the objectives can lead to significant time savings and to gaining more degrees of freedoms in tackling the different subproblems.

For the second problem, with reference to a dynamic traffic scenario, an algorithm for optimal BBU placement with costs for virtual network function migrations is developed, and the implementation details are outlined. Leveraging the lexicographic method previously developed, computing times under few tens of seconds are obtained, allowing for fast reconfiguration of the network. As far as my knowledge goes, this is the first time in literature that such problem is considered along with latency constraints and redundant assignment for reliability purposes. It is demonstrated that this model obtains solutions negligibly worse with respect to the ones obtained by the model for the static traffic scenario, with up to 82% savings in the number of migrations. The time granularity with which the algorithm should be applied is also discussed. In case of a trend of increasing traffic it is best to run the algorithm with a fine time granularity (e.g. every minute) in order to follow the traffic pattern and hastily reconfigure the network, avoiding potential blockage scenarios. In case of a trend of decreasing traffic, a coarser time granularity (e.g. thirty minutes) can be envisaged, due to the fact that blockage scenarios are unlikely and virtual resources need only to be released.

Finally, a Column Generation approach for the static traffic scenario is developed. The aim is to provide a baseline that can be extended for Column Gen-

eration based heuristics or for exact Branch-and-Price algorithms. A Column Generation model formulation for the first step of the lexicographic method is developed, and the implementation details are outlined.

A possible extension of this work can be, starting from the Column Generation baseline algorithm, to develop a full-fledged Column Generation based heuristic or an exact Branch-and-Price algorithm, which have the potential to perform quite efficiently given the promising lower bounds achieved. The model can be further extended in order to take into account all possible paths from RRUs to BBU Hotels, and not only the shortest path. In fact, a model considering all possible paths would be better be solved via a Column Generation algorithm, given the large number of paths to be considered. Finally, the model can be further extended by inserting additional reliability requirements, such as for protection against single or multiple link failures.

Bibliography

[1] 3GPP. *Release description; Release 16*. Technical Specification (TS) 21.916. Version 2.0.0. 3rd Generation Partnership Project (3GPP), June 2021.

[2] Mamta Agiwal, Abhishek Roy, and Navrati Saxena. "Next Generation 5G Wireless Networks: A Comprehensive Survey". In: *IEEE Communications Surveys & Tutorials* 18.3 (2016), pp. 1617–1655. DOI: 10.1109/COMST.2016.2532458.

[3] Md Shipon Ali et al. "Downlink Power Allocation for CoMP-NOMA in Multi-Cell Networks". In: *IEEE Transactions on Communications* 66.9 (2018), pp. 3982–3998. DOI: 10.1109/TCOMM.2018.2831206.

[4] Aleksandra Checko et al. "Cloud RAN for Mobile Networks—A Technology Overview". In: *IEEE Communications Surveys & Tutorials* 17.1 (2015), pp. 405–426. DOI: 10.1109/COMST.2014.2355255.

[5] K. Chen et al. *C-RAN: The Road Towards Green RAN*. White Paper, Ver. 3.0. Mobile China Research Institute, Dec. 2013.

[6] Longbiao Chen et al. "Deep mobile traffic forecast and complementary base station clustering for C-RAN optimization". In: *Journal of Network and Computer Applications* 121 (2018), pp. 59–69. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2018.07.015. URL: https://www.sciencedirect.com/science/article/pii/S1084804518302455.

[7] Nicola Di Cicco, Valentina Cacchiani, and Carla Raffaelli. *Adaptive Resource Optimization in Reliable Cloud Radio Access Networks with Dynamic Traffic*. Submitted at ONS GLOBECOM 2021.

[8] Nicola Di Cicco, Valentina Cacchiani, and Carla Raffaelli. "Scalable Multi-objective Optimization of Reliable Latency-constrained Optical Transport Networks". In: *DRCN 2021 - Design of Reliable Communication Networks; 17th International Conference*. 2021, pp. 1–6.

[9] Matthias Ehrgott. *Multicriteria Optimization*. Springer Science & Business Media, 2005.

[10] Teresa Gomes et al. "Maximally node and SRLG-disjoint path pair of min-sum cost in GMPLS networks: a lexicographic approach". In: *Photonic Network Communications* 31 (June 2015). DOI: 10.1007/s11107-015-0524-0.

[11] Chih-lin I et al. "Rethink fronthaul for soft RAN". In: *IEEE Communications Magazine* 53.9 (2015), pp. 82–88. DOI: 10.1109/MCOM.2015.7263350.

[12] Bahare Masood Khorsandi, Federico Tonini, and Carla Raffaelli. "Centralized vs. Distributed Algorithms for Resilient 5G Access Networks". In: *Springer Photonics Networks Communications* 37.3 (2019), pp. 376–387.

[13] Bahare Masood Khorsandi et al. "Survivable BBU Hotel placement in a C-RAN with an Optical WDM Transport". In: *DRCN 2017 - Design of Reliable Communication Networks; 13th International Conference*. 2017, pp. 1–6.

[14] Rashid Mijumbi et al. "Network Function Virtualization: State-of-the-Art and Research Challenges". In: *IEEE Communications Surveys Tutorials* 18.1 (2016), pp. 236–262. DOI: 10.1109/COMST.2015.2477041.

[15] Francesco Musumeci et al. "Optimal BBU Placement for 5G C-RAN Deployment Over WDM Aggregation Networks". In: *Journal of Lightwave Technology* 34.8 (2016), pp. 1963–1970. DOI: 10.1109/JLT.2015.2513101.

[16] Daniele Pinchera, Stefano Perna, and Marco Donald Migliore. "A Lexicographic Approach for Multi-Objective Optimization in Antenna Array Design". In: *Progress In Electromagnetics Research M* 59 (2017), pp. 85–102. DOI: 10.2528/PIERM17042106.

[17] Navrati Saxena, Abhishek Roy, and HanSeok Kim. "Traffic-Aware Cloud RAN: A Key for Green 5G Networks". In: *IEEE Journal on Selected Areas in Communications* 34.4 (2016), pp. 1010–1021. DOI: 10.1109/JSAC.2016.2549438.

[18] Mohamed Shehata et al. "Survivable BBU Placement for C-RAN over Optical Aggregation Networks". In: *2018 20th International Conference on Transparent Optical Networks (ICTON)*. 2018, pp. 1–4. DOI: 10.1109/ICTON.2018.8473920.

[19] Rodrigo Izidoro Tinini et al. "Energy-Efficient vBBU Migration and Wavelength Reassignment in Cloud-Fog RAN". In: *IEEE Transactions on Green Communications and Networking* 5.1 (2021), pp. 18–28. DOI: 10.1109/TGCN.2020.3035546.

[20] Massimo Tornatore et al. "On the complexity of routing and spectrum assignment in flexible-grid ring networks [Invited]". In: *IEEE/OSA Journal of Optical Communications and Networking* 7.2 (2015), A256–A267. DOI: 10.1364/JOCN.7.00A256.

[21] Elaine Wong et al. "Enhancing the survivability and power savings of 5G transport networks based on DWDM rings". In: *IEEE/OSA Journal of Optical Communications and Networking* 9.9 (2017), pp. D74–D85. DOI: 10.1364/JOCN.9.000D74.

[22] Hao Yu et al. "Dynamic 5G RAN slice adjustment and migration based on traffic prediction in WDM metro-aggregation networks". In: *IEEE/OSA Journal of Optical Communications and Networking* 12.12 (2020), pp. 403–413. DOI: 10.1364/JOCN.403829.

[23] Jiawei Zhang et al. "Energy-efficient traffic grooming in sliceable-transponder-equipped IP-over-elastic optical networks [invited]". In: *IEEE/OSA Journal of Optical Communications and Networking* 7.1 (2015), A142–A152. DOI: 10.1364/JOCN.7.00A142.