UNIVERSITÀ DI PISA

Dipartimento di Ingegneria dell'Energia dei Sistemi,

del Territorio e delle Costruzioni

Relazione per il conseguimento della

Laurea Magistrale in Ingegneria Gestionale

Models and Algorithms for Antenna Coordination

RELATORI

Dott.ssa Laura Galli
Dipartimento di Informatica

Prof. Giovanni Stea
Dipartimento di Ingegneria dell'Informazione

IL CANDIDATO

Emanuele Guerrazzi
emaguerra@hotmail.it

Sessione di Laurea del 30/11/2016

Anno Accademico: 2015/2016

Consultazione NON consentita

Contents

List of Figures

List of Tables

Sommario

Questa tesi è il risultato di un lavoro durato circa sei mesi svolto al Dipartimento di Ingegneria dell'Informazione dell'Università di Pisa.

Lo scopo di questa tesi è di studiare algoritmi in grado di minimizzare l'interferenza tra antenne appartenenti a sistemi LTE.

Questo è un problema "decisionale" che può essere modellato come un problema di ottimizzazione matematica, più precisamente come un problema di programmazione lineare intera.

In questa tesi vengono presentati algoritmi esatti e mate-euristici, il cui tempo di esecuzione dipende dal numero di antenne, quindi viene presentata una estensiva valutazione delle prestazioni su istanze piccole e grandi.

Tutti i risultati sono stati ottenuti utilizzando il software di ottimizzazione matematica "IBM ILOG CPLEX Optimization Studio" © (CPLEX) e tutti i programmi sono stati implementati in linguaggio C-C++.

I risultati sono promettenti, infatti viene mostrato che una buona coordinazione delle antenne porta ad una minore interferenza nel sistema.

Abstract

This thesis is the result of a work of about six months at the Department of Computer Science Engineering of the University of Pisa.

Scope of this thesis is to study algorithms to minimize the inter-cell interference in LTE systems.

This is a "decision" problem that can be modelled as a mathematical optimization problem, more precisely as an Integer Linear Program.

We present exact and math-heuristic algorithms, whose running time depends on the number of antennas, therefore we present an extensive performance evaluation on small and large scale instances.

All the results have been obtained using the mathematical optimization tool "IBM ILOG CPLEX Optimization Studio" © (CPLEX) and all programs were implemented in C-C++ language.

The results are promising, indeed we show that a good coordination of antennas leads to smaller system interference.

Chapter 1

Introduction

Mobile communications have become an everyday commodity. In the last decades, they have evolved from being an expensive technology for a few selected individuals to today's ubiquitous systems used by a majority of the world's population. So, in the recent years, the number of mobile users is considerably increased. The use of smartphones, notebooks, tablets, and other mobile devices has spread significantly for a whole series of applications: voice calls, video streaming, web browsing and many other applications that require access to the Internet.

In this context, the broadband Internet access demand is constantly increasing with the growth of customers' needs and the birth of new services. So, mobile communications system needs to adapt to this dynamic context in order to support voice and data traffic at ever-higher speed.

Mobile communication technologies are often divided into *generations*, with 1G being the analog mobile radio systems of the 1980s, 2G the first digital mobile systems, and 3G the first mobile systems handling broadband data.

In this thesis we deal with the "fourth generation" (4G), that we will call also with the name of **Long Term Evolution** (LTE) or **Long Term Evolution Advanced** (LTE-A). We won't get into the details of telecommunication technologies, but we will deal the problem of **inter-cell interference** in such systems.

We will see this details in the next chapters.

In this thesis, we face this problem using mathematical optimization techniques.

Operations Research is a branch of applied mathematics that deals with optimization problems. Thanks to the important findings in this field of the last two decades or so, now we have we have powerful optimization tools that can be used in many practical applications: from business processes to physical models.

Mathematical models are idealized representations expressed in terms of mathematical symbols and expressions. Thus, if there are n related quantifiable decisions to be made, they are represented as **decision variables** $x_1, x_2, ..., x_n$ whose respective values are to be determined. The appropriate measure of performance (e.g., profit) is then expressed as a mathematical function of these decision variables. We can see this performance as a "benefit". This function is called **objective function**. Any restrictions on the values that can be assigned to these decision variables are also expressed mathematically, typically by means of inequalities or equations.

Such mathematical expressions for the restrictions often are called **constraints**. The constants (namely, the coefficients and right-hand sides) in the constraints and the objective function are called the **parameters** of the model. The mathematical model might then say that the problem is to choose the values of the decision variables so as to maximize the objective function, subject to the specified constraints. Such a model, and minor variations of it, typifies the models used in OR [1].

So, we can say that *Operations Research* is a branch of applied maths that deals with *decisional problems*, where we have to take decisions about the use of available *resources*, that are limited in quantity, in order to satisfy a set of assigned conditions (that are the *constraints*) and maximizing the obtainable benefit from the use of these resources.

In this thesis, we will see how to model a LTE system, how to construct a good model of this system and which algorithms are suitable for which scenarios.

Organization of this Thesis

Chapter 2 presents the background of mobile networks and mathematical optimization, explaining some terms that can help the reader in the following chapters.

Chapter 3 presents the system model of LTE, showing which are the *resources* and the *constraints* involved. Then we translate in mathematical language the system model, and we introduce the class of optimization problem dealt with.

Chapter 4 contains the algorithms for small-scale coordination, used to coordinate from 3 to 21 antennas, describing each algorithm.

Chapter 5 instead is about algorithms for large-scale coordination, used to coordinate from 24 to 57 antennas.

Chapter 6 contains the evaluation of the performances of these algorithms. The performances are the *gap* compared to the optimum and the *execution time.*

Chapter 7 reports conclusions of this thesis.

Appendix reports info about CPLEX and more details about results.

Research Methodology

In table 1.1 we report goals and methods that have been used in this thesis in the various chapters:

Table 1.1: Research methodology

Chapters	**Goals**	**Methods**
Background	To describe the current state of art both of LTE systems and of mathematical optimization.	Search of related papers and books.
System and Mathematical Programming Model	To model the LTE system from an engineering point of view and to construct an efficient mathematical model.	Study of system's resources and constraints, use of math language and modelling techniques.
Small-scale coordination algorithms	To design algorithms for small-scale coordination.	Programming and verifying robustness of the algorithms
Large-scale coordination algorithms	To design algorithms for large-scale coordination.	Programming and verifying robustness of the algorithms
Performance evaluation	To evaluate the performance of the algorithms in various scenarios.	Use of typical performance measures of algorithms.

As this thesis required some computer science skills, it has been necessary to learn more in detail C, C++ and some script programming languages.

A preliminary training about CPLEX software has also been necessary.

The development of the various algorithms, that are the core part of this thesis, has been done through the text editor *Geany*, using Ubuntu as OS.

A large part of this thesis has been done at the Department of Computer Science Engineering of the University of Pisa.

Chapter 2

Background

In this chapter we speak about LTE more in detail and start to see its related problems. Then, we introduce mathematical optimization in its most common forms, explaining terms that can help the reader to understand the following chapters and showing the background about the main algorithms.

At the end of this chapter there are listed some solutions to the inter-cell interference in LTE systems.

2.1 The LTE technology

2.1.1 *Third Generation Partnership Project*

The **Third Generation Partnership Project (3GPP)** is a collaboration between groups of telecommunications associations, known as the Organizational Partners, which was established in December 1998.

3GPP standards are structured as *Releases*, each one including hundreds of individual standards documents (e.g. UMTS was firstly designed in *Release '99*). All these documents are available in the 3GPP site ([2]).

3GPP has continuously developed releases to support the evolution of mobile communications. All of these advances have provided a high degree of continuity in the evolving systems, allowing existing equipment to be prepared for future features and functionality, i.e. delivering higher data rates, quality of service and cost efficiencies.

Within this context, the ***Long Term Evolution*** **(LTE)** is often called "4G", but many also claim that LTE release 10, also referred to as *LTE-Advanced*, is the true 4G evolution step, with the first release of LTE (release 8) then being labeled as 3.9G. It must be pointed out that LTE and LTE-Advanced are the same technology, with the "Advanced" label primarily being added to highlight the relation between LTE release

10 (LTE-Advanced) and ITU/IMT-Advanced[1], as ITU decided to apply “4G” label also for LTE, we can refer either to 4G or to LTE [3].

As said in chapter 1, in this thesis we will use as synonyms 4G, LTE and LTE-A.

The main objectives of LTE are to minimize the system and User Equipment (UE) complexities, allow flexible spectrum deployment in existing or new frequency spectrum and to enable co-existence with other 3GPP Radio Access [4].

LTE offers high-quality services with very high performances in terms of:

- **Data rate**, the ever increasing demand for higher data rates for web browsing, streaming and file transfer pushes the peak data rates for mobile systems from kbit/s for 2G, to Mbit/s for 3G and getting close to Gbit/s for 4G.
- **Delay**, interactive services such as real-time gaming, but also web browsing and interactive file transfer, have requirements for very low delay, making it a primary design target.
- **System capacity**, from the mobile system operator’s point of view, the total data rate that can be provided on average from each deployed base station site is of importance. In the case of capacity shortage in a mobile system, the *Quality-of-Service* (QoS) for the individual end-users may be degraded.

2.1.2 Requirements

LTE is the successor of the High Speed Packet Access (HSPA) technology. The main requirements for LTE are the following:

High transmission rate , i.e. 300 Mbps in the downlink and 170 Mbps in the uplink [2] with 20 MHz bandwidth.

Spectrum flexibility starting with 1.4 MHz up to 20 MHz. Simpler architecture with respect to the UMTS Terrestrial Radio Access Network (UTRAN).

Advanced support to mobility

Low latency

Quality of Service (QoS) support, using an IP-based paradigm.

Interoperability with non-3GPP systems, like WiMAX and WiFi.

In order to achieve the above targets, LTE employs:

[1]IMT-Advanced is a set of requirements issued by the ITY-R for 4G systems.

[2]Downlink refers to the transmission path from a cell site to a terminal, whereas the uplink is the transmission path from a terminal to a cell site.

- New medium access systems:
  - *Orthogonal Frequency Division Multiple Access* (OFDMA) in the downlink.
  - *Single Carrier Frequency Division Multiple Access* (SC-FDMA) in the uplink.
- *Hybrid ARQ* [3] *with soft combining*, allowing the terminal to rapidly request retransmission of erroneously data. Incorrectly received data block is stored at the receiver rather than discarded, and when the retransmitted data block is received, the two blocks are combined.
- *Multi-antenna transmission* schemes:
  - Spatial Multiplexing, referred as *Multiple Input Multiple Output* (MIMO): the sender transmits, from different antennas, up to 4 different data streams in the same time-frequency resources. The receiver is able to separate streams and process the received signal.
  - Transmit Diversity: the sender transmit, from different antennas, a single data stream, thus improving the signal strength.

We will not enter details, for which we refer to 3GPP specifications [5].

2.1.3 High-level view of resource allocation in LTE

Every user has a *transmission rate* demand in Mbps, that is the demand necessary to satisfy mobile services. his demand is satisfied by a *base station*. We can refer to it either as "node" or *eNodeB* (eNB), that means "Evoleved Node B". It is the evolution of the element "Node B" in UMTS [6].

The antenna manages communications with the user equipments (UEs), as they will be defines below. An eNB decides who and how much will receive data. We call *downlink* the flux of data from eNB to UEs, and *uplink* the flux of data from UEs to eNB.

An eNB is assigned to each user. That eNB will translate the transmission rate demand in a **(Virtual) Resource Block** (RB) demand, where a RB is a virtual division of the spectrum of bandwidth in which a eNB can transmit data. In fact, a communication channel, or sub-carrier, is also known as RB.

Once the translation has been complete, the user receive from the assigned eNB enough bandwidth to satisfy the transmission rate demand.

[3] ARQ is the acronyme of *Automatic Repeat-reQuest*. It's a strategy that allows the telecommunication system to find an error, but it doesn't solve it.

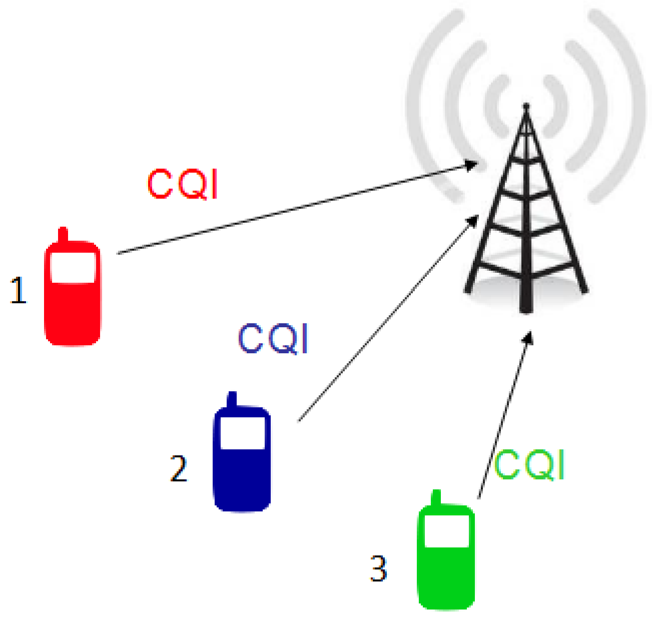


Figure 2.1: Uplink communication: CQI

In LTE, cell transmissions are arranged in time slots called **Transmission Time Intervals** (TTI), whose duration is 1 ms. The eNB has to satisfy a certain demand in 1 ms.

Each eNB has a vector of RBs available that can be allocated to the users. Each eNB can allocate a specific RB to only one user in each TTI, but the same user can be assigned to more than one RB in each TTI. So, algorithms work on this model knowing that resources are limited, and every bandwidth corresponds to a certain number of blocks that have to be allocated.

How many bits to fill each RB?

The quality of the wireless channel varies over both time and frequency.

Each RB carries a different amount of bits depending on the **Channel Quality Indicator** (CQI) reported by the UE it is addressed to through the uplink communication, as we see from figure 2.1.

If a UE communicates a high CQI, then the eNB will fill RBs with many bits. Vice-versa, if a UE has low CQI, it will receive few bits per RB, because it means that the quality of the channel is lacking.

CQI is periodically measured by UE, with a period that is *vendor specific*, however its order of magnitude is 10^-3 s.

CQI can vary from 1 to 15, while the number of bits whereby the eNB fills RBs can vary from 0 to 93 , but we won't get into the details.

The CQI is computed according to the measured **Signal to Noise and Interference Ratio** (SINR) according to some formula that is *vendor specific.*

The SINR between user u and antenna e is defined as:

$$SINR_u^r = \frac{P_{e,u}}{N_G + \sum_{x \neq e} P_{x,u} \Delta_{e,x} / n_e} \tag{2.1}$$

where:

- $P_{e,u}$ is the power allocated by node e to each RB for user u
- N_G is the so-called Gaussian noise. It is added to any noise that is intrinsic to the system and it has uniform power across bandwidth and normal distribution over time.
- $P_{x,u}$ is the power perceived from u from every other antenna that is not e. It depends on the distance and angle between them, the propagation model and the transmitting power of x.
- $\Delta_{e,x}$ is the number of RBs simultaneously allocated by nodes e and x. This is also the number of interfering RBs.
- n_e is the total number of RBs allocated by the node. The number of RBs that u receives is less or equal to n_e.

We have said that each user u is characterized by a data rate demand D_u in Mbps that must be satisfied.

The number of RBs required by user u from node e is given by:

$$RB_u = \frac{D_u}{f(SINR_u^e)} \tag{2.2}$$

where $f(SINR_u^e)$ is a non-decreasing function of $SINR_u^e$ that we won't detail.

2.2 The inter-cell interference problem

LTE/LTE-Advanced is designed to operate with a one-cell frequency reuse, implying that the same time–frequency resources can be used in neighbouring cells. From an overall system-efficiency point-of-view, having access to the entire available spectrum in each cell and operating with one-cell reuse is always beneficial. However, it may also lead to relatively large variations in the signal-to-interference ratio, and thus also in the achievable data rates, over the cell area with potentially only relatively low data rates being available at the cell border.

The overall performance of a network can be measured in terms of:

- **Fairness**: with this term we mean that all nodes of a network expect to gain the bandwidth fairly and also the *quality of service* (QoS). QoS in the field of telecommunications is defined as:

  a set of specific requirements provided by a network to users, which are necessary in order to achieve the required functionality of an application (service) [7]

  Specific measures of fairness are available in [8].

- **Cell throughput**, is the amount of data which a network or entity sends or receives data, or the amount of data processed in one determined time space. It has basic units of measures the bits per second (bit/s or bps). The throughput can be lower if there are losses and delays in the system. Throughput is a good measure of the channel capacity of a communications link.

- **Energy efficiency**, as computer networks have become a major contributor to electricity consumption all over the world [9], which has fostered a large amount of research on how to make them more energy-efficient.

  Energy consumption has also an other side: the battery saving at the user side, but we won't focus on this side.

  The problem of energy consumption arises especially in the infrastructure side, when energy consumption meets *OPerating EXpenditure* (OPEX).

  LTE power saving schemes exploit the fact that cellular coverage is overlapping, due to both reliability and (mostly) performance design issues, and that traffic load is highly variable. Coverage is designed to carry peak-hour loads, which normally occur during working hours in business areas, while nodes use very few resources during (long) off-peak periods, e.g. at night. Therefore, some nodes can switch off during off-peak hours, and nearby nodes will increase their transmission power accordingly to guarantee coverage. All these mechanisms are currently taken in account in many studies.

  The power p consumed by a node is a linear function of the number of allocated RBs to the users on each TTI, more precisely:

  $$p = P_{base} + \rho \cdot n \tag{2.3}$$

  where:

  - P_{base} = *baseline power*
  - n = number of allocated RBs. Of course, $n < m$.

– ρ = a given constant

As in LTE the same spectrum can be re-used in neighboring cells, *coordinated scheduling* is employed to improve the overall network performance [10].

The problem of *Antenna Coordination* is relevant on LTE systems because a good coordination of the antennas can reflect the performances written above and lead to higher CQI perceived from the users.

Indeed, the SINR of a user improves if fewer of the RBs that it uses are interfered upon by its neighbors.

Thus, optimal allocation of RBs can reduce interference, hence the number of required RBs and so the total consumed power.

2.3 Mathematical Optimization

2.3.1 Overview of the Operations Research modelling approach

When dealing with an optimization problem, we typically encounter the following phases [11]:

1. **Problem identification**: in this phase we identify what is our goal, which are the available resources and which are the constraints.

2. **AS-IS analysis and data collection**: in this phase we study the reality as it is and collect some data that can be used as input parameters to our problem. This phase is crucial as the solutions we will find are solution of the *model* and not necessarily of the real problem. A model will typically describe a limited portion of reality, especially in more complex problems, but it must represent with a reasonable accuracy every aspects involved in finding a solution to the decisional problem.

3. **Construction of the model**: in this phase we translate a “word-described” problem in a mathematical problem. The objective function/s and the constraints are defined and mathematically modelled as equality or inequality with the proper type (e.g. linear, quadratic, etc.). An objective function is always of type min/-max. We define all input parameters, the kind of solution’s variable (e.g. integer, binary, real, the

4. **Determining one or more algorithms**: in this phase we focus on which math strategy is better for the particular problem and choose or create algorithms. For example, a simple linear problem can be solved with the Simplex Algorithm if the number of variables is not “too much”.

5. **Computational Tests**: once the algorithm/s are written in some codes, we test the effectiveness and efficiency of the algorithm/s and basically compare the objective function's value and the execution time with our goals. For example, we could reach the optimum value but in times that are too big for our scopes.

   Let be P our optimization problem and $z(P)$ the *optimum solution* for P. Let be x^* a *feasible solution*, that is a solution that satisfies all the constraints and all the bounds. We can evaluate the *relative error* ε_r as:

$$\varepsilon_r = \frac{x^* - z(P)}{z(P)} \tag{2.4}$$

6. **Improving the mode**l: improvements to results can be obtained using one (or both) of the following approaches:

   - Improving the formulation of the model.
   - Improving the algorithms.

   Both of these approaches are object of study and operations research in general is vividly in evolution.

This approach is very similar to the P-D-C-A (*Plan-Do-Check-Act*) used in management engineering.

The previous steps are not intended to be strictly sequential, because each step is highly correlated to the previous and the following ones. We may collect data and already have a (rough) idea about the mathematical model that will be built, and also the formulation of the model could preliminary consider the execution time.

So, Operations Research have many application fields that range from telecommunications, traffic regulation, production scheduling, organizational problems, economics and many other, but in every problem we have limited resources and a limited time (that may vary largely on different kinds of problems).

This aspects are crucially important when we construct a model and choose an algorithm. Many models are untreatable at the state of the art of nowadays.

The "quality" of a model is given balancing two opposing necessities:

1. To keep every element necessary to a complete and correct description of out phenomenon

2. To have a "sufficiently easy" model to deal with so we can obtain one ore more solutions in reasonable times.

Both in modelling and in choosing algorithm phase we will have to keep this in mind.

2.3.2 Mathematical Problems

Operations Research deals with many types of mathematical problems. The most frequent types of problems dealt with in practical applications are:

Linear Programming (LP): this type of problem has a linear *max* objective function, linear constraints and real variables. A general LP has the following form:

$$\max c^T x$$

s.t.

$$\begin{aligned} &Ax \leq b \\ &x \in \mathbb{R}^n \end{aligned}$$

This form is also called as *standard* form. Of course, a problem can also have a *min* objective function, with other constraints. We will see this in section 2.5.1.

Clearly, $c^T x$ is the *objective function* we want to minimize/maximize, in particular $c^T, c \in \mathbb{R}^n$ is the transposed vector of the coefficient of the variables in the objective function.

A is a $m \times n$ such that $A \in \mathbb{R}^{m \times n}$ that represents the coefficients of the variables in the m constraints. So, element $a_{ij} \in A$ represent the coefficient of the j-th variable in i-th constraint.

Integer Linear Programming (ILP): this type of problem is one of the most important problem for a managerial engineering, because with integer variables we can often model reality and many problems of scheduling, production planning and so on. It has the same form of a LP, but has $x \in \mathbb{Z}^n$. Many ILPs has $x \in \mathbb{Z}^n_+$

Mixed-Integer Linear Programming (MILP): another important type of problem is the family of MILPs, that are problems with two different sets of variables, one integer and the other real. A general MILP has the following form:

$$\max c^T x + h^T y$$

s.t.

$$\begin{aligned} &Ax + Gy \leq b \\ &x \in \mathbb{R}^n \\ &y \in \mathbb{Z}^p \end{aligned}$$

with $c \in \mathbb{R}^n, h \in \mathbb{Z}^p, A \in \mathbb{R}^{m \times n}, G \in \mathbb{Z}^{m \times p}$.

Non-Linear Programming : in this family of problems, each element has a non-linear objective function. We can find types of problems such as:

- *Quadratic Problem (QP)*: this class of problem has a quadratic objective function and linear constraint. A general QP has the following form:

$$\max \frac{1}{2}x^T Qx + c^T x$$

s.t.

$$Ax + Gy \leq b$$
$$x \in \mathbb{R}^n$$

with $Q \in \mathbb{R}^{n\times n}$ is symmetric.

- *Quadratically Constrained Quadratic Problem (QCQP)*: this class of problem is a generalization of QP, because it includes quadratic constraints. A general QCQP has the following form:

$$\max \frac{1}{2}x^T Qx + c^T x$$

s.t.

$$\frac{1}{2}x^T Px + q^T x + r \leq 0$$
$$x \in \mathbb{R}^n$$

with $P \in \mathbb{R}^{n\times n}$, $q \in \mathbb{R}^n$ and $r \in \mathbb{R}$.

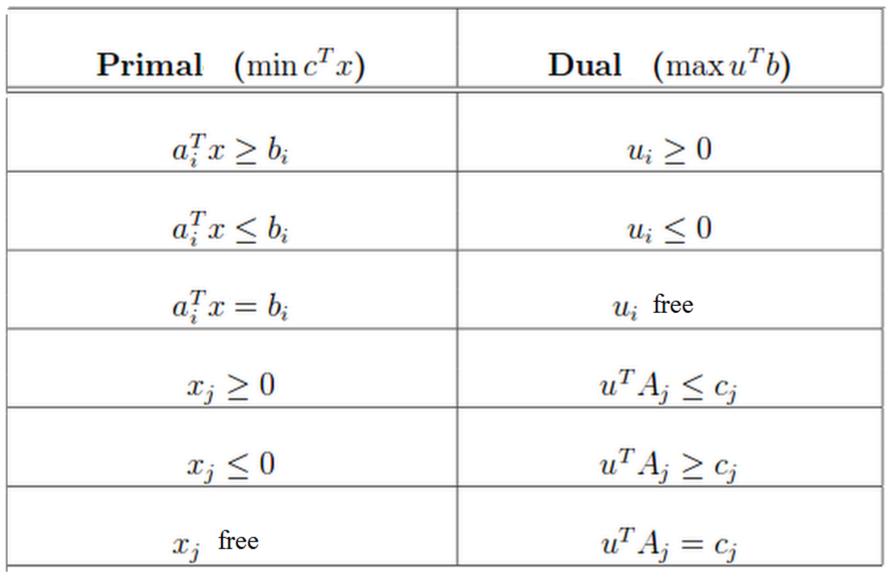
Of course, there are many sub-class of problems and others. For example, we may have a QP with $x \in \mathbb{Z}$, or a MILP with a $Ax + Gy \geq b$ constraint.

2.4 LP and ILP Algorithms

In this section we present a typical classification of LP and ILP algorithms.

Different algorithms are studied for different problems for many reasons. Of course, it is possible that one algorithm is feasible to a certain type of algorithm and not for another one due to mathematical reasons, but also because of *complexity.*

We will see how algorithms can be divided, studied and developed for mathematical optimization purpose.



Primal $(\min c^T x)$	**Dual** $(\max u^T b)$
$a_i^T x \geq b_i$	$u_i \geq 0$
$a_i^T x \leq b_i$	$u_i \leq 0$
$a_i^T x = b_i$	u_i free
$x_j \geq 0$	$u^T A_j \leq c_j$
$x_j \leq 0$	$u^T A_j \geq c_j$
x_j free	$u^T A_j = c_j$

Figure 2.2: Dual conversion table

2.4.1 Duality theory

One of the most important discoveries in the early development of linear programming was the concept of duality and its many important ramifications. This discovery revealed that every linear programming problem (we call it as **primal**, with the meaning of "original") has associated with it another linear programming problem called the **dual**.

Let's call P the primal problem in its standard form, using the same equations of a LP, as defined in the previous section.

Then its dual problem D is defined as:

$$\min\ b^T y$$

s.t.

$$Ay \geq c$$
$$y \geq 0$$

This two forms of the same problem are so closely related that the optimal solution of one form automatically provides the optimal solution to the other.

The fundamental rules to construct the dual form starting from a minimization primal problem can be summarize in figure 2.2, that provides a fast "conversion table" (it can be used in the two ways):

We refer to any book of Operations Research for further details.

2.4.2 Complexity Theory

Computational Complexity Theory is a branch of theoretical computer science that provides a method of quantifying *problem difficulty* in absolute sense, so that it's possible to compare the relative difficulty of two different problems or algorithms.

The elements of this theory are basically:

- **Dimension of the instance of a problem**: formally is the number of bits necessary to codify the instance. In practical, it is how much complex are the parameters we are dealing with in the particular problem. For example, the number of variables of a LP, or the number of constraints.

- **Algorithm complexity**: we know that a (deterministic) *algorithm* is defined a set of operations that must be executed to solve a problem. We can measure the execution time of an algorithm as the number of elementary operations executed by the algorithm. In general, we will consider always the *worst* case and we will say that an algorithm $\mathcal{A}$ has computational complexity $\mathcal{O}(f(n))$ if exists a constant $c > 0$ and a index $\bar{n} > 0$ such that the execution time of $\mathcal{A}$ to solve an instance of dimension n is $\geq c \cdot f(n)$ for each $n > \bar{n}$.

  An algorithm has *polynomial* complexity if its complexity is $\mathcal{O}(n^k)$ for a certain $k \in \mathbb{N}$. We will call *efficient* such algorithms.

  Note that typically 1 elementary operation take 1 $ns = 10^-9$ s. [12].

- $\mathcal{P}$ **class**: is a class of decisional problem that is solvable by a deterministic algorithm of polynomial complexity. Some examples are the *minimum path*, solvable by Dijkstra algorithm, or *generalized linear assignment*, or *any other LP*.

- $\mathcal{NP}$ **class**: is the class of decisional problems that are solvable by a non-deterministic polynomial algorithm. A non-deterministic polynomial algorithm is an algorithm that solves a decisional problem following 2 steps:

  1. Hypothesize a certain solution to the particular instance and that there is the possibility to verify this hypothesis.
  2. Check the hypothesis.

  We mention that $\mathcal{P} \subseteq \mathcal{NP}$, but no one still knows if $\mathcal{P} = \mathcal{NP}$

When we can say if a problem P_1 is more difficult than a problem P_2?

A decisional problem P_1 is *polynomially reducible* to another problem P_2 (we will write $P_1 \propto P_2$) if, for each instance I_1 of P_1 we can obtain in a polynomial time an instance I_2 of P_2 such that from the optimal solution of I_2 we obtain in a polynomial time the optimal solution of I_1.

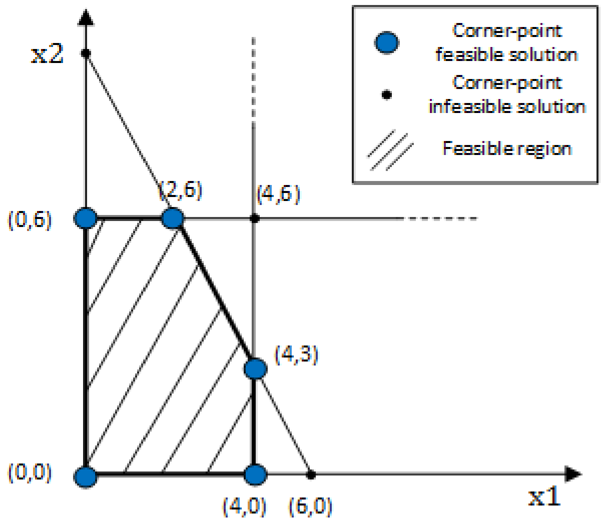

x2
x1
(0,6)
(2,6)
(4,6)
(4,3)
(0,0)
(4,0)
(6,0)
Corner-point feasible solution
Corner-point infeasible solution
Feasible region

Figure 2.3: Graphical representation of a *2*-dimensional LP

If this happens, than P_2 is *at least as difficult as* P_1.

Of course, if $P_1 \propto P_2$ and $P_2 \in \mathcal{P}$, then $P_1 \in \mathcal{P}$.

The most difficult problems are the so-called $\mathcal{NP}$-complete problems.

A problem P is $\mathcal{NP}$-complete if:

1. $P \in \mathcal{NP}$
2. $P_1 \propto P$ for each $P_1 \in \mathcal{NP}$

So, a $\mathcal{NP}$-complete problem is at least as difficult as any other $\mathcal{NP}$ problem.

Summing up, we say that problems of class $\mathcal{P}$ are *easy*, while problems of $\mathcal{NP}$-complete are *difficult*.

More details are available in [13].

2.4.3 Simplex algorithm

The *Simplex method* has been developed by George Dantzig in 1947.

From theory we know that if exists the optimal solution of a LP, then it is a *vertex*[4] of the *feasible region*. The feasible region is a part of the n-dimensional space (where n is the number of variables) such that any points belonging to this part is a feasible solution to the problem (that means it satisfies all constraints).

In fig 2.3 we can see a graphical representation of a LP.

It has proved that the *Simplex method* is a remarkably efficient method that is used routinely to solve huge problems on today's computers. The method relies on geometric concepts.

[4]A vertex is the point in common of two incident lines

The points of intersection are the **corner-point solutions** (CPS) of the problem.

Suppose our LP has a matrix $A \in \mathbb{R}^m \times n$, we call *basis* any indexes subset B such that the sub-matrix A_B has rank n. We can demonstrate that if a point $\bar{x}$ is a vertex, then it exists at least a basis B such that $\bar{x} = A_B^{-1} b_B$, that is the associated basis.

Vice-versa, given a certain basis we can associate to it a vertex using the same formula as above.

We denote with $I(\bar{x})$ the set of indexes of A that $\bar{x}$ satisfies with equality constraints, then $|I(\bar{x})| \geq n$; every subset with cardinality n will be a basis associated to $\bar{x}$.

we can have two situations:

- $|I(\bar{x})| = n$, then the vertex is called *not degenerate*
- $|I(\bar{x})| > n$, then the vertex is called *degenerate* and so there are more basis associated to it.

Supposing to start the algorithm with a vertex $\bar{x}$ and a basis associated to it, we iteratively follow these steps:

1. Check of optimality of the current vertex. This is done through the dual of the problem. We can stop here or we find a *growth direction* ξ. If the vertex is not degenerate, ξ is feasible, so we can improve our solution without getting out the feasible region. Vice-versa, the direction is not feasible.
2. Calculate the maximum shift in direction ξ. This is 0 if ξ is degenerate, ∞ if the problem is not limited or it can be finite and positive, so we find a CPS that improves the solution.
3. Update the basis with an *incoming* index k and a *outcoming* index h. This corresponds to a shift of indexes.

Details about this method are available in any OR's book.

Let's see now a numerical example.

Suppose we want to maximize

$$\max Z = 3x_1 + 5x_2 \tag{2.5}$$

s.t.

$$x_1 \leq 4 \tag{2.6}$$

$$2_2 \leq 12 \tag{2.7}$$

$$3x_1 + 2_2 \leq 18 \tag{2.8}$$

$$x \in \mathbb{R}^2_+ \tag{2.9}$$

Constraints (2.6), (2.7) and (2.8) are represented by the lines in fig 2.3.

The five that lie on the corners of the feasible region $(0,0),(0,6),(2,6),(4,3)$, and $(4,0)$ are the *corner-point feasible solutions* (CPF solutions). The other three $(0,9),(4,6)$ and $(6,0)$ are called *corner-point infeasible solutions.*

In this example, each corner-point solution lies at the intersection of two constraint boundaries, while if we have n decision variables, each CPS lies at the intersection of n constraint boundaries in a n-dimensional space.

For any linear programming problem with n decision variables, two CPF solutions are **adjacent** to each other if they share $n-1$ constraint boundaries. The two adjacent CPF solutions are connected by a line segment that lies on these same shared constraint boundaries. Such a line segment is referred to as an **edge** of the feasible region.

We can easily see in fig 2.3 which are the adjacent CPF solutions.

One reason for our interest in adjacent CPF solutions is the following general property about such solutions, which provides a very useful way of checking whether a CPF solution is an optimal solution.

Optimality test: Consider any linear programming problem that possesses at least one optimal solution. If a CPF solution has no adjacent CPF solutions that are better (as measured by Z), then it must be an optimal solution.

Thus, for the example, (2, 6) must be optimal simply because its Z = 36 is larger than Z = 30 for (0, 6) and Z = 27 for (4, 3). This optimality test is the one used by the simplex method for determining when an optimal solution has been reached.

2.4.4 ILP algorithms

Unlike linear programming problems, integer programming problems are very difficult to solve. In fact, no efficient general algorithm is known for their solution.

Optimum algorithms for ILP are:

Cutting planes , that is: given an $ILP = max\{c^T x | Ax \leq b, x \in \mathbb{Z}^n\}$ and its continuous relaxation $LP = max\{c^T x | Ax \leq b, x \in \mathbb{Z}^n\}$, a generic cutting plane algorithm does:

1. Solve the LP. Let be x^* an optimal solution.
2. If x^* is integer stop; x^* is an optimal solution to ILP.
3. If not, add a linear inequality constraint to LP that all integer solutions in ILP satisfy, but x^* does not, the go to # 2.

Branch & Bound: this method is also called "total enumeration" because the idea is to explore the total enumeration tree without exploring all the possible leaves. The *"Branch"* and *"Bound"* techniques let us do a so-called *implicit visit.*

- For the *"Bound"* techniques it's necessary to know the upper and lower evaluations of the optimal solution of an ILP. The most common techniques are:
  - Elimination of a constraint.
  - Continuous relaxation.
  - Sum of constraints.
  - Use of a feasible solution $\tilde{x}$.
  - Restriction (e.g. $x_i = 0$).

  If we are dealing of a maximization problem, these techniques give an *upper bound*, while if the problem is of minimization, they give a *lower bound.*
- For the *"Branch"*) techniques (that is the implicit visit), we describe the two most common rules. Let $v(P)$ be the upper bound of our problem, and $V_I(P)$ and $V_S(P)$ be the lower and upper bounds of the primal problem in its standard form. Let be P_i the sub-problems at i-th level in the enumeration tree. Then:
  - If $V_I(P_i) \geq V_S(P)$ then I do an implicit visit. Indeed, in the sub-tree of P_i there aren't better solutions than the current one.
  - If the optimum of the continuous relaxation of P_i is feasible then I do an implicit visit. I can update $V_S(P)$ with $V_I(P)$.

More details are available in any book dealing Operations Research, we cite [14].

Dynamic programming , it solves $ILPs$ sequentially basing on Bellman's *Optimality Principle*:

Regardless of the decisions taken to enter a particular state in a particular stage, the remaining decisions made for leaving that stage must constitute an optimal policy

General guidelines for constructing dynamic programming algorithms are:

1. View the choice of a feasible solution as a sequence of decisions occurring in stages, and so that the total cost is the sum of the costs of individual decisions.
2. Define the state as a summary of all relevant past decisions.
3. Determine which state transitions are possible. Let the cost of each state transition be the cost of the corresponding decision.
4. Define the objective function and write a recursion on the optimal cost from the origin state to a destination state.

Heuristic algorithms for ILP (and in general) are every algorithm able to give a *feasible* solutions typically in polynomial times. Heuristic algorithms can be classified in:

- *Constructive*: they start from an empty solution and iteratively add elements to the partial solution until some criteria are achieved. These criteria typically are expressed in terms of gap compared to the optimal solutions and/or time limits. These algorithms are typically *greedy*, based on implicit enumeration.
- *Local search*: they start from an initial feasible solution and try to modify it iteratively slightly modifying it. They stop when they reach a local optimum from where is no more possible to obtain improvements.
- *Math-heuristics*: they rely on math techniques and can be seen as an evolution of local search heuristics. They try to avoid local optima.

The quality of a heuristic solution typically improves from constructive heuristics to math-heuristics.

About relaxations The idea of the concept of *relaxation* is to take the original problem and making it easier to solve by modifying some constraints and eventually the objective function.

From theory we know that if we have an $IP = max\{c^T x \mid x \in X \subset \mathbb{Z}^n_+\}$ then we can find a *lower bound* $\underline{z}$ and an *upper bound* $\bar{z}$ respectively finding any feasible solutions x^* and solving the relaxation associated to the starting *IP*.

So the question is: which kind of relaxation should I use?

The most used relaxations are:

- *Continuous relaxation* (CR): this is simply the LP associated to the ILP in question, simply modifying the integrality constraint on the variables and setting the variables belonging to $\mathbb{R}$.
- *Combinatorial relaxation*: it is a easier combinatorial optimization problem. Nice combinatorial relaxations can be solved in polynomial time. Their structure varies on the particular kind of the original problem.
- *Relaxation by elimination*: it is simply based on deleting one or more constraint of the original problem in order to make it easier to be solved.
- *Lagrangian relaxation*: Let P be an ILP of this form:

$$\max c^T x \tag{2.10}$$

s.t.

$$Ax \leq b \tag{2.11}$$
$$Bx \leq d \tag{2.12}$$
$$x \in \mathbb{Z}^n \tag{2.13}$$

where (2.11) are "hard constraints", such that without them P would be easy to solve. Instead of deleting these constraints, the lagrangian relaxation penalizes the violation of these constraints adding them in objective function with an opportune coefficient.

We define the lagrangian relaxation RL_λ of P, as the following ILP:

$$\max c^T x + \lambda^T (b - Ax) \tag{2.14}$$

s.t.

$$Bx \leq d \tag{2.15}$$
$$x \in \mathbb{Z}^n \tag{2.16}$$

where $\lambda \geq 0$ is the so-called vector of *Lagrange's multipliers.*

Theorem of *weak duality* says that for every $\lambda \geq 0$, the optimum x^*_λ of RL_λ is an upper bound of P. In particular, if x^*_λ is feasible for P and it results that $\lambda^T(b - Ax) = 0$ (slack variables), then x^*_λ is optimal also for P.

How to choose the appropriate λ? Let be $v(RL_\lambda)$ the value of the objective function of (1.11).

We know that all λ must be ≥ 0, but we would like to minimize the optimal solution of RL_λ so that the gap with the optimum of P is reduced. So, we have to solve the so-called *lagrangian dual* D of P:

$$\min v(RL_\lambda) \tag{2.17}$$

s.t.

$$\lambda \geq 0 \tag{2.18}$$

To minimize such non-differentiable convex function, we could use the *subgradient's method*, that is a generalization of the mathematical concept of *gradient*.

More details about this method are available in many books that deal with Operations Research.

- *Surrogate relaxation*: a subset of constraints is substituted by one constraint that is a linear combination of those constraints. A multiplier vector u is also used. So we could reduce the number of the rows of the constraint matrix A in order to get a constraint of this form:

$$u^T(Ax) \leq u^T b \tag{2.19}$$

2.5 State of art

In LTE systems there aren't standards about coordination schemes, so that every operators can decide which algorithm has to be used. Practically, operators apply the First-fit algorithm or a Random approach. The first algorithm is deterministic and consists in allocating the first D blocks of the spectrum, where D is the block demand. The other is non-deterministic, and consists in a random allocation of D blocks. We will see this algorithms more in detail in chapter 4.

[15] reports a list of solutions that has been proposed by several authors. About this paper we report in particular:

- A large system analysis based on random matrix theory used for small-scale coordination.
- A study investigates the interference exposure of the macrocell tier on the small cells , showing that the interference is not maximised at the macrocell edge.
- A framework based on the *Belief Propagation* algorithm is developed to solve the inter-cell interference problem for heterogeneous networks. A factor graph

is designed to represent the inter-cell interference probabilistic relationship with various communication resources Based on a set of prior distributions, it calculates the posterior distributions of the scheduling variables algorithm achieves a good result in typically a couple of iterations.

. We refer to the paper for more details.

[16] shows a study on heterogeneous networks in LTE-A systems based on sharing spectrum and using game theory. It models LTE-A systems as a hierarchical structure with a macronode as “leader” and micronode as “followers”. In this structure the authors apply a Stackelberg game[5] and show that this approach leads to a better spectrum sharing among nodes.

Another study on heterogeneous networks in LTE-A systems has been done by [17], which advocates a reduction of inter-cell interference with the use of *Almost Blank Sub Frame* (ABSF) technique. This technique consists of a sub frame in which the interferer tier is not allowed to transmit data. The basic concept of ABSF is to mute some sub frames that causes interference and uses the RB in cell of lower level. The study is oriented in maximizing the throughput of the cells and shows that this goal is achieved.

Heterogeneous networks have been studied also in [18]. In this case there are proposed algorithms that deal with the problem of selecting which cells transmit in which RBs so as to mitigate the interference suffered by UEs. A layered approach is used to split the problem into *small-scale* coordination and *large-scale*. Small-scale coordination (SSC) arbitrates a small cluster of three cells, by partitioning the frame in interference logical subbands (ILSs). Each ILS defines the subset of cells that can transmit in the same RBs. SSC has been used as a basis for large-scale coordination (LSC), which was accomplished by defining the position of the ILSs in the frame, so as to minimize the interference among neighboring clusters. The authors modeled both SSC and LSC as optimization problems, and showed them to be too complex to be solved at optimality. The paper advocates fast heuristics that can be run at TTI timescale.

[5]In *game theory*, a *Stackelberg game* is a game in which the leader does the first move and the followers response according to the leader’s move.

Chapter 3

System and Mathematical Programming Model

3.1 System model

A **radio accesso network** (RAN) is a part of a mobile telecommunication system that interfaces individual devices with other parts of a network through some radio access technologies.

The RAN of a LTE system is called *Evolved Universal Terrestrial Radio Acces Network* (EUTRAN) and can be considered an evolution of the UMTS and HSDPA/HSUPA technologies specified in 3GPP release 5. More details are available at [19].

However, we will refer to EUTRAN with the term RAN.

The RAN of a LTE system consists of three main entities:

- **Antennas**
- **Communication channels** (i.e., OFDMA sub-carriers as previously seen).
- **Users**: We can refer to a user also as *User Equipment* (UE), because in fact the antenna communicates with the equipment of the users (e.g. Smartphone). WIth UE we mean any device used directly by an end-user to communicate, so UEs can be smartphones, notebooks, tablets and other mobile devices, as presented in Chapter 1.

A graphical representation of LTE System is given in fig 3.1.

We consider a large-scale multi-cell LTE network.

The network is logically divided into areas (called "hexagons"), each one under the coverage of a macro node, i.e., an antenna with high radiation capability. In fig 3.2,

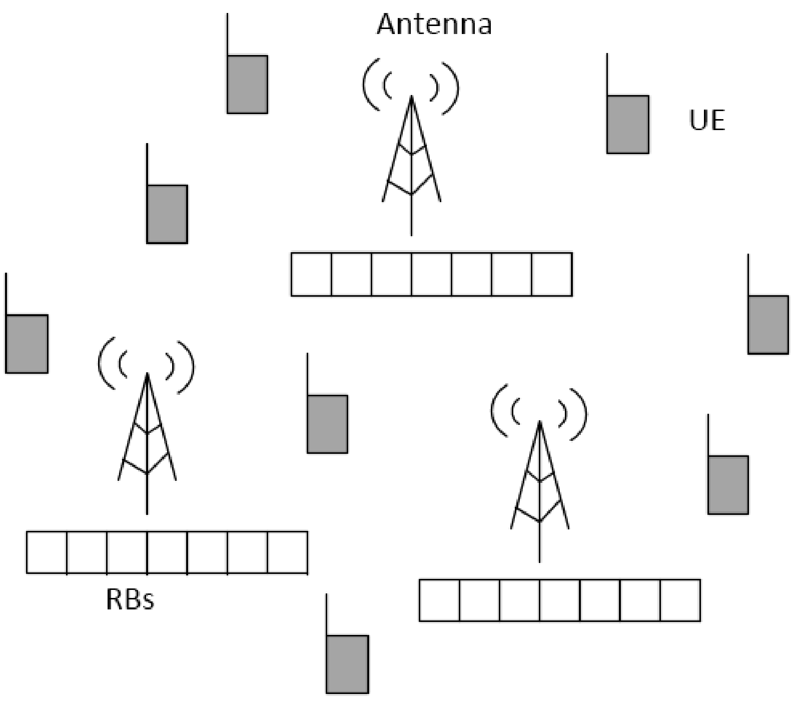


Figure 3.1: LTE System

we can see how a physical area of a territory (described from two x and y axis) can be tasselled using hexagons.

Each hexagon represents an area covered by three overlapping cells, and the little square set at the centre of each triad of hexagons represent the point in which are these cells.

We can assume that antennas are anisotropic, disposed and radiating at 120° from each other.

Each cluster can be identified by a *cluster ID* that is the number we can see in the square of fig 3.2.

The positioning of the 3 macro nodes is shown in fig 3.3.

We can give a *cell ID* to each antenna in order to identify each of them. In particular, if we give label 0 to the 180° antenna, label 1 to the 300° one and label 2 to the 60°, we can say that:

$$CellID = ClusterID \cdot 3 + i \tag{3.1}$$

where i is the label of the particular positioning.

In figure 3.4 we have a graphical representation of the inter-cell interference according to our system model.

A hexagon can also accommodate *micronodes*, usually located in hot-spots or at an hexagon edge (see fig 3.5).

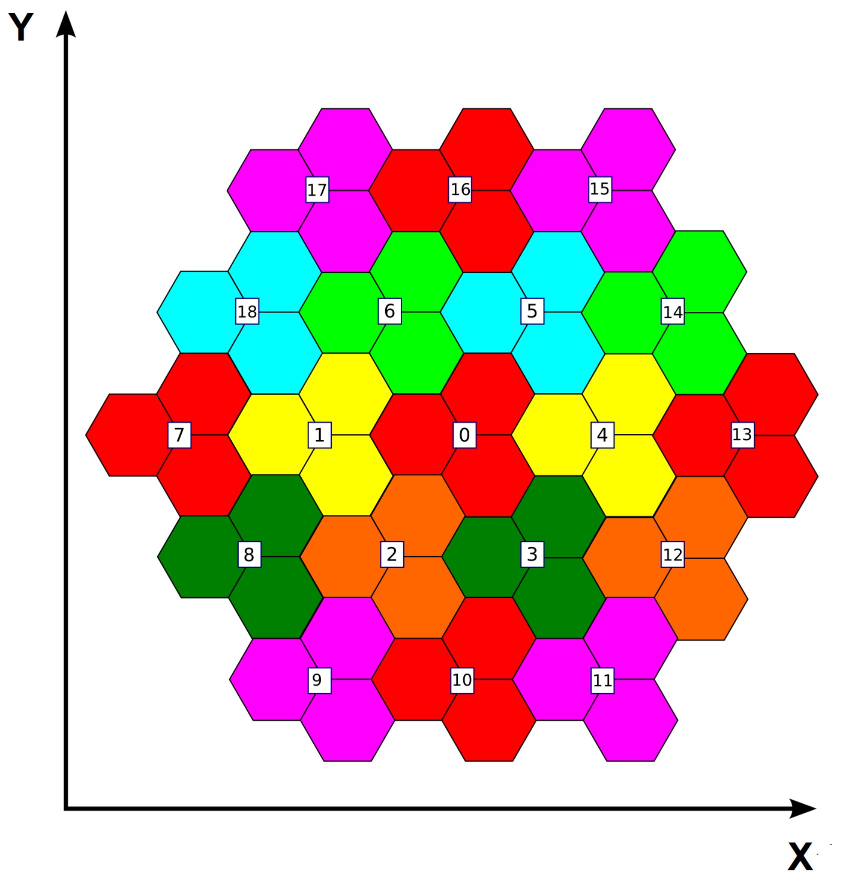


Figure 3.2: Hexagons in LTE

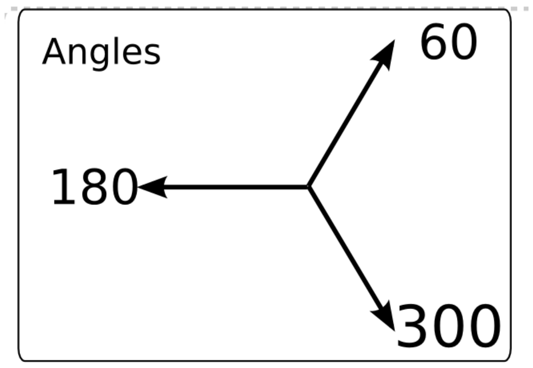


Figure 3.3: Positioning of the 3 macro nodes of a cluster

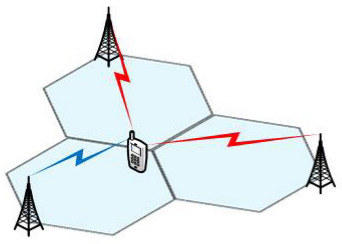


Figure 3.4: The inter-cell interference problem

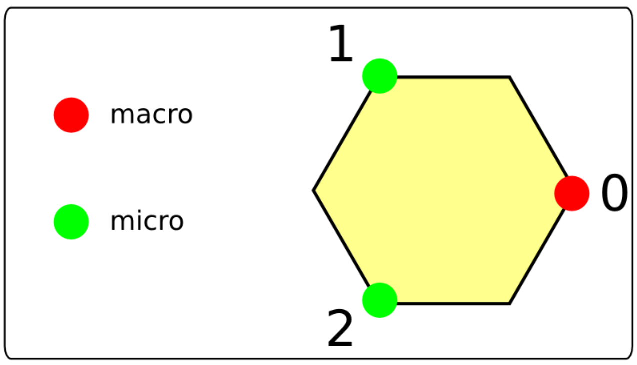


Figure 3.5: Location of *micro nodes*

The utility of micro nodes is to provide additional localized capacity at a lower power cost, for example in particular areas where there are lots of offices, shops etc.

About RBs demand

We assume that the operator possesses the following information: a per-hexagon historical load curve, detailing (at least) the overall bitrate requested in a hexagon over time in a sample day. There may be several such curves, of course, e.g., one describing working days, one for holidays, etc.

Moreover, the operator possesses a similar curve (or set thereof) detailing the number of users per cell over time, so that the average per-user bitrate can be inferred. Note that these curves do not contain information about the position of single users within a hexagon. The time resolution at which these curves are plotted is that of a sufficiently large interval (e.g., 15 minutes), called *snapshot* henceforth. We only consider the downlink direction, which is the most critical from the point of view of both carried load and infrastructure power consumption.

To infer a per-user requested data rate from the load curves we focus on a single snap- shot and a single hexagon.

Let L and H be the load and number of users for the hexagon in a given snapshot.

The average demand per user u in the given hexagon-snapshot is $D_u = L/H$. Also, for each pair of node-user $(e, u) \in E \times U$, we know the power $P_{e,u}$ allocated by node e to each RB for user u.

Note that, according to equation (2.2), RB_u may not be integer, but we consider it as an average value and the time span of a snapshot id large enough to allow a fluid approximation.

It's clear that the SINR of a user improves if fewer of the RBs that it uses are interfered upon by its neighbours. This means that antennas should not communicate using the same frequency, or at least should not do that if the interference between a pair is high. We will model this in the next section.

3.2 Mathematical programming model

In this section we'll see how we can model the problem of minimization of interference among antennas in LTE networks in mathematical terms.

We take the following parameters as input to the problem:

- N : number of antennas that have to be coordinated.

- m : maximum number of virtual blocks that can be used by every antenna. We refer to a virtual partition of bandwidth in which antennas transmit.

- U : set of users that have a demand to be satisfied.

Note that we want to maximize the *signal to interference and noise ratio* $SINR_u^e$ for every pair $(e, u), e \in N \ u \in U$ of antenna and user.

As we can see from equation (2.1), we have to **minimize the overall interference among antennas**. This because the lower the number of blocks simultaneously allocated, the higher the SINR.

In the following section we'll see two exact models that allow us to find the optimal solution, and we will evaluate their characteristics.

We remind that in Integer Programming the *choice of a formulation is crucial*, and a bad formulation could affect our ability to solve the problem.

For our purpose, we use a *pattern-based* model to solve the problem of antenna coordination.

For each $(i, j) \in N \times N$ we have a coefficient α_{ij} denoting the interference from the two antennas. We can populate a so-called *interference matrix*, where each diagonal element $\alpha_{ii} = 0$, as shown in the following matrix .

$$\alpha = \begin{pmatrix} 0 & \alpha_{ij} & ... & \alpha_{1N} \\ \alpha_{ji} & 0 & ... & ... \\ ... & ... & 0 & ... \\ \alpha_{N1} & ... & ... & 0 \end{pmatrix}$$

Note that $\alpha_{ij} \neq \alpha_{ji}$, because the radiation pattern of a cell may be isotropic.

The interference depends on the antenna pair but not on the blocks, so it isn't important to know which blocks are shared by a certain pair of antennas.

For each $i \in N$ we have a weight w_i denoting the importance of the i-th antenna and a block demand A_i.

The block demand is a translation of user's demand that is in Mbps, as seen in chapter 2.

Let us define a pattern p as a vector $[p_i]_{i \in N}$ of binary elements $p_i \in \{0, 1\}$ where:

$$p_i = \begin{cases} 1 & \text{if antenna i-th is active,} \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

Of course, we could have "tripper patterns" $p = e_i$ having only a 1 in position i; these correspond to blocks uniquely assigned to antenna i, and have cost $c_p = 0$.

The cost of a generic p-th pattern is:

$$c_p = \sum_{i \in N} \sum_{j \in N} w_i \alpha_{ij} p_i p_j \tag{3.3}$$

This formulation let the cost be 0 if a certain block is used by 1 antenna at maximum.

Let P be the set of all possible patterns. The cardinality of P is $2^{|N|}$.

The formulation of the problem is therefore:

$$\min \sum_{p \in P} c_p x_p \tag{3.4}$$

s.t.

$$\sum_{p \in P} p_i x_p = A_i \qquad i \in N \tag{3.5}$$

$$\sum_{p \in P} x_p \leq m \tag{3.6}$$

$$x_p \in \mathbb{N} \qquad\qquad p \in P \tag{3.7}$$

where:

- Expression (3.3) states that we have to minimize the sum of the costs associated with using x_i times the pattern p_i.

- Equation(3.4) states that we have to satisfy the blocks' demand of every antenna.
- Inequality (3.5) states that the total blocks we have at disposition is m.
- Expression (3.6) states that the variable x_p can have value in the range of integer positive number, while p must belong to P, that is the set of all possible patterns.

Scope of this model, is to minimize the cost of a certain set of coordinated cells.

Chapter 4

Algorithms for small-scale coordination

In this chapter we present some algorithms used for *small-scale* coordination. In fact, with these algorithms we are able to efficiently coordinate from 3 to 21 antennas.

We will discuss the performance of these algorithms and their limits in chapter 6.

4.1 Bruteforce

In this section we present the algorithm *Bruteforce.* This algorithm has been modified from the algorithm of Rendtl *et al* [20], firstly developed to solve *Max-cut* problem.

4.1.1 Columns Generation

From theory we know that when the number of variables is too “large” to deal with explicitly, it’s not a good idea to deal with all the variables at the same time, because the solver would have too many variables.

In our case, given the number N of antennas, we know that the variables we deal with are 2^N, that is exponential.

With this algorithm we want to use the mathematical technique of ***Columns Generation***. This process is also known as *variable pricing.*

The idea of this technique is very simple: instead of solving the problem with all its possible variables, we consider only a subset of the variables and populate a so-called *reduced master problem.* Then we dynamically add the columns (variables) that help us to find a better solution. We can recognize these columns because they have a reduced negative cost.

This approach follows the scheme reported in figure 4.1. The process can be also summarized with these sequential points:

1. Initialize the *reduce master problem* (RMP) with a "small" subset of variables. Let's call n the size of this subset, our coordination problem will have the same structure of (3.4)(3.5)(3.6), but with $p \in \tilde{P}$, where $|P| = n$. That means we give a small subset of variables to RMP. We apply an additional change to the problem changing constraint (3.5) to:

$$\sum_{p \in P} p_i x_p \geq A_i \qquad i \in N \tag{4.1}$$

   This because we populate the problem with a so-smaller set of variables (especially with the growing of N). This will make the problem easier to be solved than the original one, because we solve a *set covering inequality* instead of a *set partitioning inequality*[1], but the $\geq$ condition will make CPLEX tending to find the solution at the minimum cost. CPLEX could exceed A_i only when there aren't sufficient patterns in the problem. The last pattern of $\tilde{P}$ assure us that a feasible solution is always found, even if we pay a great cost in terms of interference.

2. Solve LP associated to the RMP and obtain primal solution x^*. Let's call λ_i the i-th dual variables associated with the i-th antenna's demand constraint (4.2) and μ^* the dual variable associated with the constraint on the max number of blocks (4.3).

3. If x^* is optimal for the original LP, i.e λ_i $i = 1, ..., N$ and μ^* are dual feasible, then stop. Otherwise, find some columns with negative reduced costs. These columns are those corresponding to violated dual constraints. So, we'll have to find those columns that violate dual constraints. This problem takes the name of **"Constraints separation problem"**.

4. Add columns found at step 3 and go to # 2.

Let's see these how to do these steps in practical.
First, we write the **dual problem** associated to our primal one.

[1]This because set partitioning has feasibility with $\mathcal{NP} - complete$ complexity, while set covering has an "easy" feasibility.

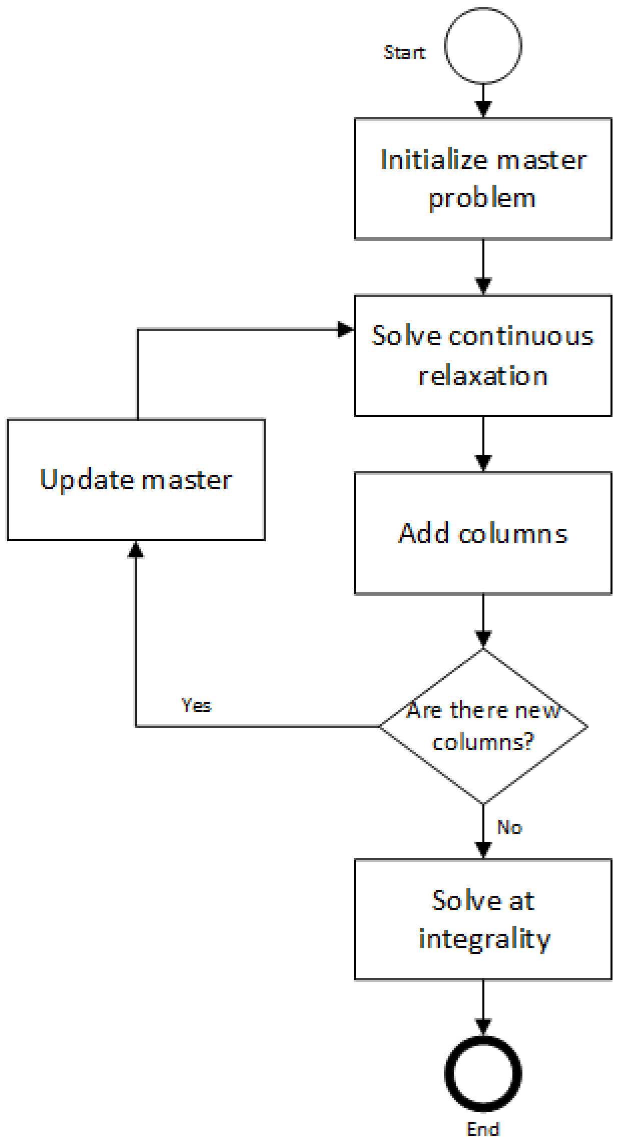


Figure 4.1: Columns generation scheme

Recalling equations (3.4) (4.1) (3.6) we can say that the dual problem associated with the primal one is:

$$\max \sum_{i \in N} \lambda_i A_i + \mu m \tag{4.2}$$

s.t.

$$\sum_{i \in N} p_i \lambda_i + \mu \leq C_p \qquad p \in P \tag{4.3}$$

$$\lambda_i \geq 0 \qquad i \in N \tag{4.4}$$

$$\mu \leq 0 \tag{4.5}$$

4.1.2 Steps of the algorithm

In *Bruteforce* algorithm, step 1 means: initialize our *RMP* with a reduced set of variables. The size of this set is $N + 1$. In particular, we give to the *RMP* the following pattern matrix:

$$\tilde{P} = \begin{pmatrix} 1 & 0 & 0 & ... & 0 \\ 0 & 1 & 0 & ... & \vdots \\ 0 & 0 & 1 & ... & \vdots \\ 0 & 0 & 0 & ... & 1 \\ 1 & 1 & 1 & ... & 1 \end{pmatrix}$$

that is a $(N + 1) \times N$ matrix.

Step 2 simply recall CPLEX to solve the continuous relaxation, and store the $N+1$ dual solutions associated.

Step 3 is the *pricing* phase. We check if there is some columns that violate constraint (4.7). In particular, we want to find a certain number K of columns to add. This K can be expressed as an absolute value $(100, 200, 500, 1000, \dots)$ or as a fraction of 2^N $(1‰, 2‰, 5‰, \dots)$. Appendix B reports the tuning of this parameter. In order to find those columns can write that constraint and do some maths:

So let's focus on equation (4.3).

We can write it as:

$$\sum_{i \in N} p_i \lambda_i - C_p \leq -\mu \qquad p \in P$$

and knowing that $C_p = p^T \omega \alpha p$ we can say:

$$\lambda^T p - p^T \omega \alpha p \leq -\mu \tag{4.6}$$

The constraint separation problem asks: does it exist a $p \in P$ such that:

$$\lambda^T p - p^T \omega \alpha p > -\mu$$

that is:

$$p^T \omega \alpha p - \lambda^T p < \mu \qquad p \in P \tag{4.7}$$

So, in order to generate columns, we have to find the following unconstrained optimization problem:

$$\min \; p^T \overline{\alpha} p - \lambda^T p \tag{4.8}$$

where

$$\overline{\alpha_{ij}} = \omega_i \cdot \alpha_{ij} \tag{4.9}$$

for every $\alpha_{ij} \in \overline{\alpha}$.

If we don't find any other column at negative reduced cost, we exit the loop and launch the *Branch & Bound* algorithm to find the integer solution.

If a certain numbers of columns has been found, step 4 is adding those columns to the RMP and go back to step 2. As we are dealing with a primal problem of minimization, this column generation will give us an upper-bound (UB), so that we'll have $UB \geq OPT$, where OPT is the optimum value of the problem.

4.1.3 Pricing problem

The problem of finding a good set of columns is to find those solutions that minimize the objective function, because the constraint that we want to violate is "<". We can't solve the constraint separation problem in polynomial-time, as it is $\mathcal{NP}$-hard

This problem is also called ***Unconstraied Boolean Quadratic Problem*** **(UBQP)**.

Indeed, it has no constraints, the variables are binaries (so they can be seen as boolean variables) and the objective function is quadratic. This is the *pricing problem* of determining the columns at reduced negative cost. We have to separate the dual problem presented in 4.1.1 .

The pricing problem is to find columns with reduced negative cost. Those columns will violate constraint (4.7). A violated constraint corresponds to a primal variable.

We separate the dual form in the following way: given a x^* that corresponds to the optimal solution of the continuous relaxation of my problem, we find an hyperplane that cut the solution from the remaining part of the polyhedron.

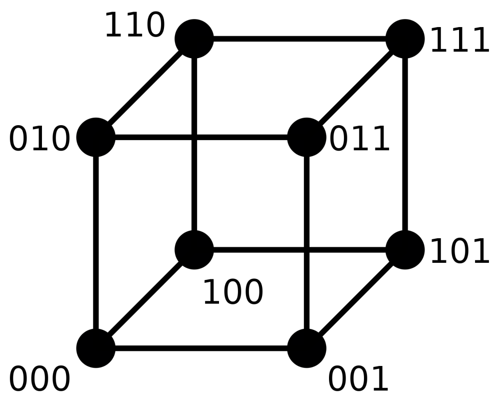


Figure 4.2: Binary representation of numbers in a hypercube of dimension 3

We know that, for what has been said in section 3.2, $\alpha_{ij} \neq \alpha_{ji}$, and $\omega_i \neq \omega_j$ for every $i, j \in N$. This means that in order to have $C_{ij} = C_{ji}$ <u>we must first symmetrize</u> $\overline{\alpha}$ matrix.

The symmetrization of $\overline{\alpha}$ can be simply done in the following way:

1. Start from $i = 1$.
2. For each $j = i, ..., N$ calculate a temporary arch cost $T_{ij} = \dfrac{\overline{\alpha ij} + \overline{\alpha ji}}{2}$
3. For each $j = i, ..., N$ set $\overline{\alpha ij} = T_{ij}$
4. For each $j = i, ..., N$ set $\overline{\alpha ji} = T_{ij}$
5. Increase i by 1 and go to # 2.

4.1.4 Exploring the N-dimensional hypercube

We know that every feasible solution of the UBQP can be found at the vertices of a N-dimensional hypercube, as we don't have any constraint. *Bruteforce* enumerates all the possible cuts of a complete graph and calculates their costs using a particular trick.

Exploiting the property of *Hamming's Distance*, we are able to enumerate all the possible patterns (that are columns) in a polynomial time.

Hamming's distance is calculated between two patterns of binary numbers and it's the number of positions in which the two patterns has different values. For example, pattern "1001" and "0101" has Hamming's distance = 2. Indeed, starting to count from right to left, those patterns have equal position 1 and position 2 but have different values in position 3 and 4.

We can see the Hamming's distance even using a N-dimensional hypercube (fig 4.3) , where N is the dimension of the patterns.

The enumeration of all the patterns can be done thanks to a particular function called *Graycode* developed by Giovanni Rinaldi[2]. The code allows to explore the hypercube passing through a hypercube's vertex *exactly* one time, starting from pattern $[00\ldots0]$ to pattern $[11\ldots1]$. This is done changing 1 bit per time, indeed *Graycode* returns an integer p that is the power of 2 that we have to add or to subtract to the previous number n in order to be the next bin-adjacent number. An integer s that can be -1 or 1 tells us if we have to add or subtract 2^p.

So, calling n' the next bin-adjacent number of n (that is the next vertex of a hypercube), we have:

$$n' = \begin{cases} n + 2^p & \text{if } s = 1 \\ n - 2^p & \text{if } s = \text{-1} \end{cases}$$

The complexity of this algorithm is $O(N)$.

Let be K the number of good columns that we want.

For every string of binary numbers that *Graycode* provides us, we can rapidly calculate the associated cut's cost C_{cut} and store the best K patterns in the following way:

1. Let be p the last bit changed from *Graycode*, Ω the cost of the last cut and Ω_{min} the minimum cost of all the cuts explored. When the algorithm start every bit of the initial pattern is setted to 0 and the $\Omega_{min} = \Omega = 0$.

2. Let be t the t-th position of the current pattern we're examining. For every pattern we find, t will vary on $1, \ldots, N$.

3. When $t < p$, check if both p and t are setted to 1.

    - If it is, add to Ω the quantity $2 \cdot C_{pt}$. Indeed, it means that in the previous cut p was setted to 0 and its cost with t wasn't included in Ω.
    - If $p = 0$ and $t = 1$, remove to Ω the quantity $2 \cdot C_{pt}$. Indeed it means that in the previous cut p was setted to 1 and its cost with t was included in Ω.
    - Otherwise, do nothing. It means that $p = 1$ and $t = 0$ or $p = 0$ and $t = 0$, but in every case the cost C_{pt} must not be included in Ω because node t is deactivated.

4. When $t = p$, check if $p = 1$.

    - If it is, add to to Ω the quantity $C_{pp} = -\lambda_p$. Indeed, it means that in the previous cut p was setted to 0 and its loop cost wasn't included in Ω.

[2]Research Director at the Institute of System Analysis and Computer Science "Antonio Ruberti".

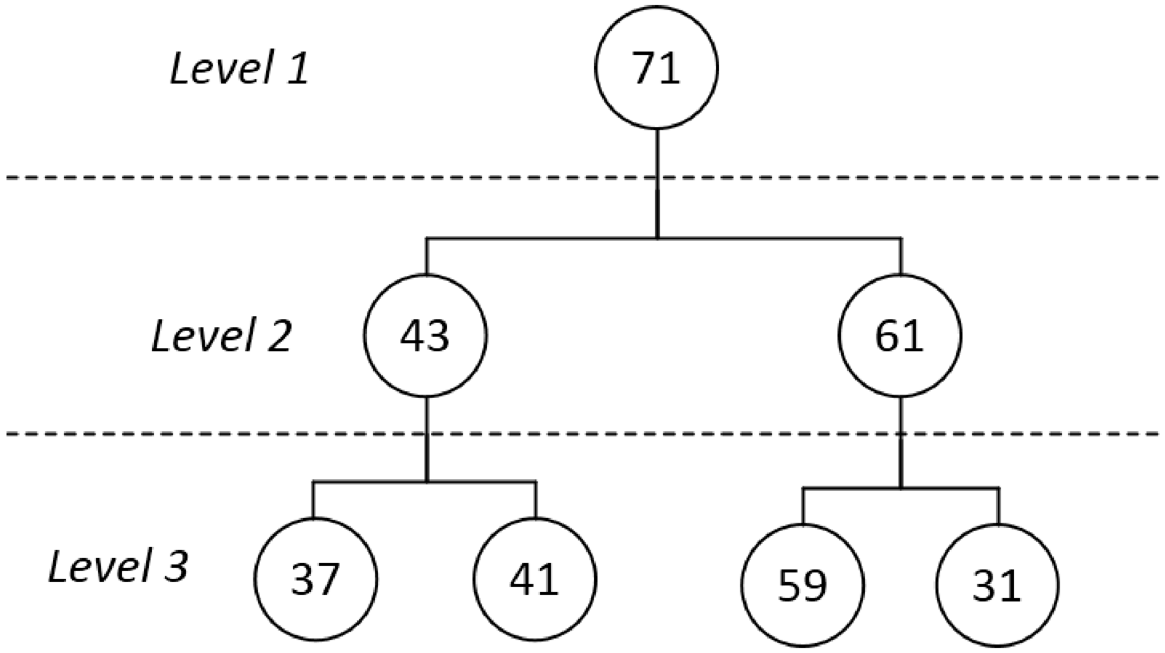


Figure 4.3: Graphical representation of a *max heap* tree

- If $p = 0$, remove to to Ω the quantity $C_{pp} = -\lambda_p$. Indeed, it means that in the previous cut p was setted to 1 and its loop cost was included in Ω.

5. When $p < t \leq N$, do the same check seen in # 3.

4.1.5 Storing K patterns

In order to manage all the patterns we can use a *vector V* structured as a particular *max heap* to store the best K patterns that we will add to the RMP.

A *max heap* tree is a binary tree in which:

- The content of a parent node is more than or equal to the content of each of its children.

- Each level is filled from left to right. Level $i + 1$ cannot be populated unless level i is completely full.

A graphical example of a *max heap* tree is given by figure 4.3.

In fact, our particular max heap stores the patterns that have the minimum cut's costs.

Let's see how can we do this.

Let be k a counter that indicates how many patterns have been stored in the heap. Of course, the maximum size of the heap is setted to K.

The *Bruteforce* algorithm goes on after a pattern has been found in the following way:

1. Let set $k = 0$ and let's start with empty V.
2. If $k < K$, add the current pattern to the V. Increase k by 1.
3. If $k = K$, create the heap on the V. Increase k by 1.
4. If $k > K$, check:
    - If $\Omega \neq 0 \ \wedge \ \Omega < \Omega_{min}$ then add the current pattern at the end of V, then swap the top's root with this leaf and set $\Omega_{min} = \Omega$. Then reheapify and delete the last element of V (that is the rightmost leaf at the lowest level). Note that the condition $\Omega \neq 0$ prevents the adding of pattern $00\ldots0$. that actually doesn't help our coordination problem as it corresponds to have every antenna turned off.
    - Otherwise do nothing

    At this time it's not necessary to increase k by 1.

Note about *heap*

A simple way to obtain the n smallest elements out of a set of n would be to sort the n elements of that set in increasing order and then select the k smallest. This has a computational cost $C_{minheap}$ equal to:

$$C_{minheap} = O(n \cdot log(n) + k) \tag{4.10}$$

where:

- $O(log(n))$ is the cost of heapification of n elements.
- $O(k)$ is the cost of selecting the first k elements

A slightly more efficient way to achieve the same goal is to use a maxheap. The maxheap will initially be filled with the first k elements of the set of n. Starting from the $k + 1$ th element e_j, e_j replaces the maxheap root node if it is smaller. This may require a reheapification, whose cost is however $O(log(k))$, since the maxheap includes k elements. This way, the computational complexity $C_{maxheap}$ of getting the same result is:

$$C_{maxheap} = O(n \cdot log(k) + k)) \tag{4.11}$$

As k can be expected to be considerably smaller than n, this saves some computations.

4.1.6 Pseudocode for *Bruteforce*

The pseudo-code for *Bruteforce* can be finally summarized as follows:

```
/****** main.cpp ******/
read.data();
crea.master();

while(get_new_cols) do{
bruteforce();
        for(i=0;i<K;i++{
                if(new_col[i] < mu*){
                        addcol.master()
                        solve.master()            // with CR
                        counter++;
                }
        }
        if(counter==0){
                get_new_cols=false;
                }
}
mipopt();                   // solve the MIP
delete.data();
```

We will refer to *Bruteforce* each when we are dealing of the algorithm just presented and when we are dealing of the specific function in the code. In the code, *Bruteforce()* function includes the *Graycode()* function that allows to explore the hypercube, as explained in paragraph 4.1.4.

4.2 *First-fit* RB allocation

As we have start to see in section 2.5, a (bad) way to satisfy UEs' demands is to use the *first-fit* (FF) algorithm.

For example, supposing to have $N = 3$ antennas, a number $m = 7$ of resource blocks (RB), and the blocks demands $A[1] = 5, A[2] = 2, A[3] = 4$, the FF uses the first n_i blocks of the spectrum for each i-th antenna, where $n_i = A[i]$.

We can see this in figure 4.4.

In this case, the number of interfering RBs between node i and node j is:

$$\Delta_{i,j} = min\{n_e, n_x\}$$

The remaining $m - min\{n_i, n_j\}$ will not interfere. This means that the number of allocated RBs that don't interfere is: $max\{0, |n_i - n_j|\}$

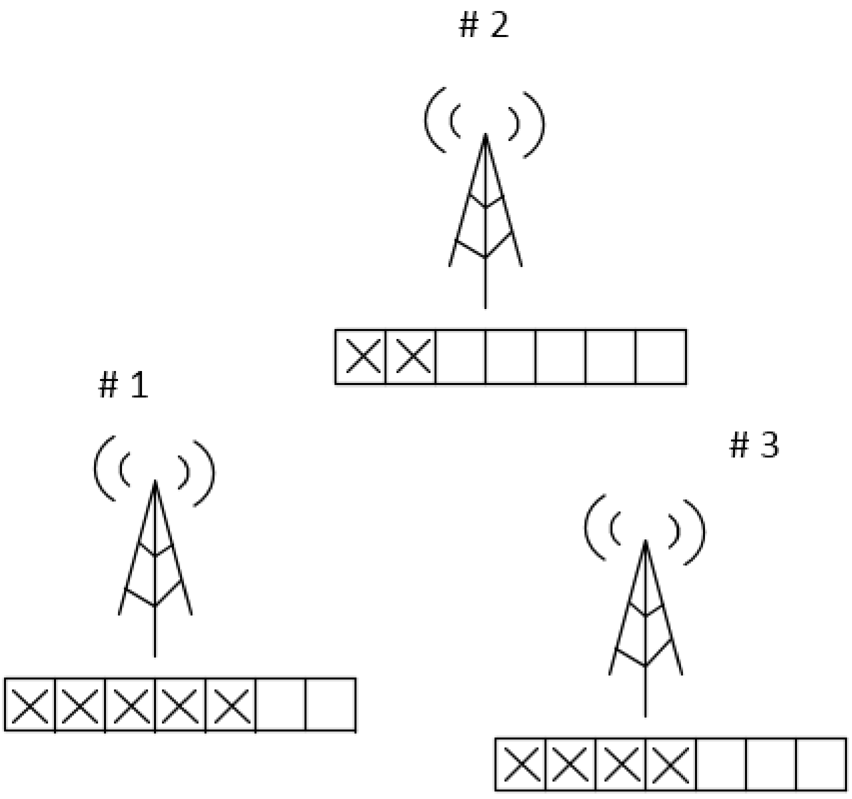


Figure 4.4: *First-fit* algorithm

Once the RBs have been allocated for each antenna, we can construct a *block spectre* matrix β of dimension $N \times m$ in which each row has the first n_i elements equal to 1, and the remaining $m - n_i$ setted to 0.

For the example of figure 4.4, where $N = 3$ and $m = 7$ the matrix β is:

$$\beta = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

The total cost of interference ι can be calculated (of course without CPLEX) as:

$$\iota = \sum_{i=1}^{N} \left(\sum_{j=1}^{N} \beta_{ij} \cdot \beta_{ji} \cdot \alpha_{ij}\right) \tag{4.12}$$

We remember that α_{ij} is the interference between antenna i and antenna j.

4.2.1 Pseudo-code for *First-fit*

The pseudo-code for *First-fit* can be summarized as follows:

```
/****** main.cpp ******/
read.data();

foreach(i in N) do{
        for(j=0;j<A[i];j++){
                block[i][j]=1;
        }
}

calculate.interference();

delete.data();
```

4.3 *Random* RB allocation

Another way to satisfy UE's demands is to use a *random* resource block (RRB) allocation algorithm.

We can think to choose randomly which RBs to use for each antenna. Of course, the number of RBs used must be equal to the blocks' demand of that antenna.

In this way, we will populate a $\beta^{N \times m}$ matrix that is the same of that one presented in the previous section, with the difference that in this case we randomly choose the blocks to set to 1.

4.3.1 A note about random generation

Ross [21] provides a method to randomly choose k elements from a set of n elements in such a manner that each of the $\binom{n}{k}$ subsets is equally likely to be the subset chosen (that is the *feasible* pattern in our case).

Let's suppose to have already generated such a subset. For each $j = 1, ..., n$ we set:

$$I_j = \begin{cases} 1 & \text{if element } j \text{ is in the subset} \\ 0 & \text{otherwise} \end{cases}$$

We discuss the *conditional distribution* of I_j, given $I_1, ..., I_{j-1}$.

Denoting with $P[I_j = 1]$ the probability that element j of I is equal to 1, we have:

$$P[I_1 = 1] = \frac{k}{n}$$

For element 2, we have:

$$P[I_2 = 1|I_1 = 1] = \frac{k-1}{n-1}$$

and

$$P[I_2 = 1|I_1 = 0] = \frac{k}{n-1}$$

because in the first case we have already allocated 1 of the k blocks.

We can infer that:

$$P[I_2 = 1|I_1] = \frac{k - I_1}{n-1}$$

This can be generalized as:

$$P[I_j = 1|I_1, ..., I_{j-1}] = \frac{k - \sum_{i=1}^{j-1} I_i}{n - j + 1} \tag{4.13}$$

where denominator is obtained as $n - (j - 1) = n - j + 1$.

Let's call U a *uniform (0,1) random variable*[3]. We remember that a uniform random variable defined in (a, b) is a random variable such that each possible value assumable is equally probable[4].

Since $PU < a = a, 0 \leq a \leq 1$, we can generate a sequence of (at most n) random numbers $U_1, U_2, ...$ and set:

[3]We can reproduce the uniform (0,1) distribution using the uniform random generator of Giovanni Rinaldi, Research Director at the Institute of Systems Analysis and Computer Science "Antonio Ruberti".

[4]In particular, the probability is $\frac{1}{b-a}$

$$I_1 = \begin{cases} 1 & \text{if } U_1 < \dfrac{k}{n} \\ 0 & \text{otherwise} \end{cases}$$

$$I_2 = \begin{cases} 1 & \text{if } U_2 < \dfrac{k - I_1}{n-1} \\ 0 & \text{otherwise} \end{cases}$$

$$\vdots$$

$$I_j = \begin{cases} 1 & \text{if } U_2 < \dfrac{k - \sum_{i=1}^{j-1} I_i}{n-j+1} \\ 0 & \text{otherwise} \end{cases}$$

The process stops when $I_1 + \cdots + I_j = k$ and the random subset (in our case *random pattern*) is the k elements whose I-value equals 1.

4.3.2 Pseudo code for RRB

The pseudo-code for *RRB* can be summarized as follows:

```
/****** main.cpp ******/
read.data();
srand(seed);

populate(block_spectre);

foreach(i in N) do{
        num= A[i];                  //numerator
        den  = m;                   //denominator
        for(j=0;j<m;j++){
                if(U[j]<num/den){
                        block_spectre[i][j]=1;
                        num--;
                }
                else{
                        block_spectre[i][j]=0;
                }
                den--;
        }
}

calculate.interference();

delete.data();
```

Chapter 5

Algorithms for large-scale coordination

In this chapter, we'll present some algorithms usable to coordinate a number of antennas varying up to 57.

We remember that the complexity of this combinatorial optimization problem is 2^N, where N is the number of antennas, so we won't be able to enumerate all the columns in a reasonable time, even if we do it linearly through *Bruteforce*. Even if *Bruteforce* uses a smart way to enumerate, it still enumerates and leads to unacceptable execution time.

We can still do optimization with *columns generation*, but we have to design new algorithms more oriented to *efficiency* (feasibility with a little execution time) rather than effectiveness (optimality, that is prohibitive for combinatorial issues).

We have to use some **Time-Limits** (TL) in running our algorithms, because it isn't possible to manage problems with 2^N variables with $N = 50$ or similar.

Note that there are two TLs that have to be studied:

1. A TL on the columns generation.

2. A TL on the total process.

In appendix C we will discuss the criteria to set the two time limits.

As we have to face a combinatorial problem with TL constraints, we have initialized all the algorithms with a *randomic* approach that is the same as presented in section 4.3.

From that feasible initial solution, the algorithms will polish the bound in order to get less interference.

5.1 UBQP

We have seen in section 2.3.2 what is a *Quadratic Programming* (QP) problem.

Recalling equation (4.8), we have to solve the following *constraints separation problem*:

$$\min \; p^T \overline{\alpha} p - \lambda^T p \tag{5.1}$$

This problem is the same that *Bruteforce* solves with enumeration.

The idea of this algorithm is to do columns generation solving with CPLEX this UBQP.

We could think to return a set of feasible columns at every cycle of the algorithm and so doing column generation.

Unfortunately, as the function (5.1) is in general not convex, CPLEX can't give us a set of feasible solutions but can return us only the optimum.

This means that we add only 1 column per time, and then we will solve a new constraint separation problem, as the constraint separation inequality (4.7) changes λ_i and μ at every cycle.

Of course, the column will be added if it has *reduced negative cost*, that is if its value is $< \mu^*$, where μ^* is the $N+1$-th dual solution associated to the max block constraint of the primal model.

As we use this algorithms for large-scale coordination, we expect not to stop the process for reduced cost reasons, but for time limits ones.

Note about solution pool As CPLEX doesn't provide a solution pool for problems with non-convex function, a very simple way to bypass this drawback is to iteratively add the following constraint for every constraint separation problem:

$$\sum_{i:\tilde{x}_i=0} x_i + \sum_{i:\tilde{x}_i=1} (1 - x_i) \geq 1 \tag{5.2}$$

where:

- $\tilde{x}$ is the dual optimum found at the previous run.
- x_i is the i-th component of the new dual optimum we want to find.

It's clear that (5.2) forces CPLEX to find another dual optimum that differs at least one element from all the $\tilde{x}$ found at previous runs.

This approach is theoretically good, but too much expensive in terms of execution time, so that many trivial tests shows that is better to solve the optimum primal model rather than passing through the dual.

5.1.1 Pseudocode of UBQP

The *pseudo-code* of UBQP algorithm is:

```
/****** main.cpp ******/
read.data();
crea.master();

while(get_new_cols && execution_time < time_limitCG) do{
solveUBQP();
                if(new_col < mu*){
                        addcol.master()
                        solve.master()
                        counter++;
                }
        if(counter==0){
                get_new_cols=false;
                }
        execution_time += calculate_time();
}
mipopt(time_limitMIP);          // solve the MIP with the assigned
    TL
delete.data();
```

5.2 UBQP flip

This version is very similar to the first one, but it generate a *flip pattern* at every cycle of CPLEX's quadratic solver.

A flip pattern is a pattern that is identical to another except for 1 bit that is different.

In this algorithm, we can generate a flip pattern in a very trivial way, as follows:

1. Let p^* be the local optimum of (5.1). We know that p^* has N components.
2. Let p_{flip} be the flipped pattern. For all $i \in 0, \ldots, N$, set $p_{flip}[i] = p^*[i]$
3. Choose randomly an integer $j \in 0, \ldots, N$.
4. If $p_{flip}[j] = 0$, set $p_{flip}[j] = 1$, and vice-versa.

Note that the triviality of this approach is due also because of TL reasons.

The flipped pattern will have a much more similar to the optimum cost as many are the antennas.

A simple example is reported in figure 5.1.

This approach is a heuristic of type *"local search"*, as it finds another pattern in proximity of the pattern that has been added to the primal problem.

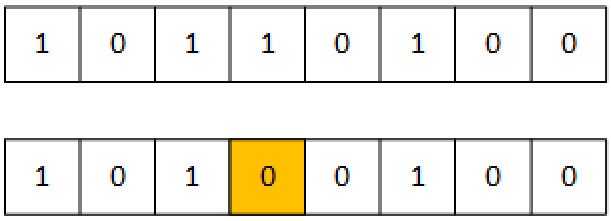


Figure 5.1: Example of flipped pattern

5.2.1 Pseudocode of UBQP flip

The pseudocode of UBQP flipped algorithm is:

```
/****** main.cpp ******/
read.data();
crea.master();

while(get_new_cols && execution_time < time_limitCG) do{
solveUBQP();
                if(new_col < mu*){
                        addcol.master()
                        solve.master()
                        counter++;
                }
        if(counter==0){
                get_new_cols=false;
                }
        execution_time += calculate_time();
}
mipopt(time_limitMIP);  // solve the MIP with the assigned TL
delete.data();
```

5.3 UBQP linearized

CPLEX has a function called *populate* that returns the optimal solution with other good feasible ones, according to some parameters such as:

- CPXPARAM_MIP_Pool_Intensity, controls the trade-off between the number of solutions generated for the solution pool and the amount of time or memory consumed.

- CPXPARAM_MIP_Pool_Capacity, sets the maximum number of solutions kept in the solution pool.

- CPXPARAM_MIP_Pool_RelGap, sets a relative tolerance on the objective value for the solutions in the solution pool. Solutions that are worse (either greater in the case of a minimization, or less in the case of a maximization) than the incumbent solution by this measure are not kept in the solution pool. For example, setting this parameter to 0.01 make solutions worse than the incumbent by 1% or more to be discarded.

- CPXPARAM_MIP_Pool_AbsGap, sets an absolute tolerance on the objective value. It works similar to the previous one.

All *populate* parameters are available in [22].

In order to exploit the *populate* function of CPLEX, that returns a certain sized pool of feasible (and good) solution, we can think to transform the quadratic problem into a linearized one.

We can linearize equation (5.1) in the following way:

$$\min \sum_{1 \le i < j \le N} 2\alpha_{ij} y_{ij} + \sum_{i=1}^{N} \lambda_i x_i \tag{5.3}$$

s.t.

$$y_{ij} \ge x_i + x_j - 1 \qquad \forall i, j : i < j \tag{5.4}$$

$$y_{ij} \le x_i \qquad \forall i, j : i < j \tag{5.5}$$

$$y_{ij} \le x_j \qquad \forall i, j : i < j \tag{5.6}$$

$$x \in \{0,1\}^N \tag{5.7}$$

$$y \in \{0,1\}^{\binom{N}{2}} \tag{5.8}$$

where:

- x_i is the i-th element of the pattern (that is the *column*) we are searching for.

- $y_{ij} = x_i x_j$, so that is 1 if both x_i and x_j are 1, and 0 otherwise.
- Expression (5.3) states that we must minimize the cost associated with
- Inequality (5.4) states that if both x_i and x_j are 1 then even y_{ij} must be 1.
- Inequalities (5.5) and (5.6) states that if x_i or x_j are 0, then even y_{ij} must be 0.

As we have $\binom{N}{2} = N \cdot (N-1)/2$ y_{ij} variables, and for each y_{ij} there are three constraints, we have in total $\frac{3}{2} \cdot N \cdot (N-1)$ constraints.

This number of constraints is larger compared to the original UBQP.

In CPLEX, we can ask to the *populate* function to store a maximum number K of feasible and good solutions. Note that this number may be not reached, depending on the parameters explained at the beginning of this section.

5.3.1 Pseudocode of UBQP linearized

The pseudocode of UBQP linearized is:

```
/****** main.cpp ******/
read.data();
crea.master();

while(get_new_cols && execution_time < time_limitCG) do{
solvelinzdUBQP();
populate();
                for(i=0;i<populate.size();i++){
                        if(new_col[i] < mu*){
                                addcol.master()
                                solve.master()
                                counter++;
                        }
                }
        if(counter==0){
                get_new_cols=false;
                }
        execution_time += calculate_time();
}
mipopt(time_limitMIP);  // solve the MIP with the assigned TL
delete.data();
```

Chapter 6

Performance evaluation

In this chapter, we'll present the results of the various algorithms.

The performances we will evaluate are:

- **Interference gap**, we will use the following formula:

$$GAP = \frac{Heur - Ref}{Heur}\% \tag{6.1}$$

  where $Heur$ is the value of the heuristic used and Ref is the algorithm of reference. When possible, we'll use the optimum algorithm (that is the CPLEX's mipopt on the complete model) as Ref, otherwise we will use the best algorithm available.

  Note that $Heur \gtreqless Ref$, so for example can be lower than the optimum (e.g. in the continuous relaxation) or greater (e.g. in the first-fit algorithm) or even equal (we will see it happens often in *Bruteforce* algorithm),

  Of course (6.1) underestimates the real gap, because $0 \leq GAP < 100$, and we can't have a measure of how much worse than the optimum is the heuristic. We decided to use (6.1) because of its fluency in comparison.

- **Execution time**, as our optimization problem must be solved iteratively in reality, the execution time is crucial. We remind that the transmission data occurs every TTI, that is 1 ms. It's important to use fast algorithm.

Note aboute test All the tests have been done on a PC with 8 Intel Core i7 CPUs at 3.60 GHz, 16 GB of RAM and Ubuntu 14.04 OS, using CPLEX 12.6.1.

The tests have been done through *scripts*, which launch the program we have written and simulate a certain configuration of a LTE system as described in chapter 3. A general script can be written as follows:

```
/****** script ******/
for ($nAnt=3;$nAnt<MAX_nAnt;$nAnt+=3){ do
        for ($nBlocks in 25 50 100){ do
                for ($param in param_valued){ do        # only if
                   the algorithm has additional parameters
                        for ($seed in 1:10){ do
                                ./main $nAnt $nBlocks $param $seed
                        done
                done
        done
done
```

Instead, the statistics associated to the data obtained from test are obtained using *perlscript*, a particular type of script a bit more elaborated that allows to rapidly have many statistics about the algorithms.

6.1 Scenarios

Tests have been conducted basing on 57 antennas disposed at the vertices of hexagons, as we can see in figure 3.2.

The parameters of our tests are:

- **Number of blocks**: LTE standards [23] refer to a bandwidth of 5, 10 or 20 MHz. As 1 RB = 180 kHz these bandwidths can be translated in 25, 50 and 100 resource blocks (some RBs are reserved). We call this parameter m.

- **Interference matrix**: Each element of the *inteference matrix* can be computed based on the position and radiation pattern of the antennas. In practice, it is the power α_{ij} received by antenna i from j, and it is expressed in Watt. The radiation pattern of a LTE anisotropic antenna has the typical "beam" form.

- **Blocks demand**: we can set 3 different scenarios:

  Low demand

$$A_i \in \begin{cases} [5-8] & \text{if } m = 25 \\ [10-15] & \text{if } m = 50 \\ [20-30] & \text{if } m = 100 \end{cases}$$

  Average demand

$$A_i \in \begin{cases} [12-15] & \text{if } m = 25 \\ [25-30] & \text{if } m = 50 \\ [50-60] & \text{if } m = 100 \end{cases}$$

High demand

$$A_i \in \begin{cases} [20 - 23] & \text{if } m = 25 \\ [40 - 45] & \text{if } m = 50 \\ [80 - 90] & \text{if } m = 100 \end{cases}$$

where A_i is the demand of i-th antenna.

The variability of A_i is given by the *rand()* function of C++ that provides a uniform distribution of random numbers between a given interval.

- **Weights**: as we refer to a homogeneous network composed only by *macro-cells*, we set to 1 every weight of every antenna. So, the cost of interference will depend only on the interference matrix. We have called this parameter ω.

.

6.2 Small-scale coordination

In this section we present the results of:

- **Optimum algorithm** (OPT): it's taking the whole $2^N \times N$ matrix of all possible patterns and give it to CPLEX in order to launch the integer B& B algorithm. Its *interference cost* will be the reference for the others algorithm. The command line to execute OPT is:

  ```
  ./main Nant Nblock Seed
  ```

  Every seed creates a particular demand scenario. For each N and m instance, we use 10 different seeds. As N varies on $3, 6, ..., 21$ and $m = 25, 50, 100$, the total number of tests for OPT is $21 \cdot 3 \cdot 10 = 630$

- **Continuous Relaxation** (RC): it's similar to the previous one but launch the simplex method on the $2^N \times N$ matrix, so it gives us a lower-bound. It has the same test characteristics of OPT. Theoretically, it may not return an integer result, so we have to use it as a reference for the lower bound. As the pattern-based model is strong, we may have an integer solution when running it.

- **Bruteforce** (BF): it works explained in chapter 4. The command line to execute OPT is:

  ```
  ./main Nant Nblock Permille Seed
  ```

For each N, m and K instance, we use 10 different seeds. As N varies on $3, 6, ..., 21$, $m = 25, 50, 100$ and $K = 1, 2, 3, 5, 10, 20, 50, 100, 200, 500, 1000$, the total number of tests for BF is $21 \cdot 3 \cdot 11 \cdot 10 = 6930$

- **First-fit allocation** (FF): it works as explained in chapter 4. It has the same test characteristics of OPT.
- **First-fit allocation** (RRB): it works as explained in chapter 4. The command line to execute OPT is:

```
./main Nant Nblock Seed Seed2
```

Seed2 has been added in order to have a more realistic view of how RRB works.

The seconds seed allows a different feasible random patterns generation for every demand configuration created by the first seed for every instance of N and m. The total number of tests for RRB is $21 \cdot 3 \cdot 10 \cdot 10 = 6300$.

As BF have an additional parameter K that is the max number of columns we add at every cycle, we have to do an additional study on the optimal choice of K. This study has been reported in Appendix B.

In these scenarios all weights of antennas have been setted to 1, in the hypothesis we are coordinating a group of macro-cells.

We use equation (6.1) to calculate the gaps of the algorithms compared to OPT.

For each scenario we will show a *speedup* comparison between BF and CR [1].

The *speedup* SU_{AB} between algorithm A and algorithm B is defined as:

$$SU_{AB} = \frac{Ex.Time_A}{Ex.Time_B} \tag{6.2}$$

Of course, A and B must solve the same problem in the same conditions (e.g. same machine).

If $SU_{AB} > 1$ then B is faster than A. For example, if $SU_{AB} = 2$, it means that B is twice faster than A. If $SU_{AB} < 1$ then B is slower than A. For example, if $SU_{AB} = 0.333$, it means that A is three times faster than B.

Trivially, if $SU_{AB} = 1$, the algorithms have the same execution time, but this case is more theoretical than real.

We remember that BF has a parameter K that is the maximum number of columns to be added at every cycle of the algorithm.

We can define K in absolute values (e.g. 100 columns for iteration) or in relative values. We prefer this last choice.

[1] We haven't compared FF and RRB in terms of execution time because it would have been trivial as both of them run under 1 ms, even if they have higher gaps

Table 6.1: Small coordination in average demand scenario, 50 Blocks: OPT, RC and BF. $K = 1$ ‰ at 21 antennas

	OPT	RC		BF	
N	Ex. time (ms)	Gap (%)	Ex. time (ms)	Gap (%)	Ex. time (ms)
3	1.8	0.0	0.3	0.0	1.8
6	2.3	0.0	0.4	0.0	2.2
9	6.1	0.0	1.0	0.0	5.2
12	45	0.0	8	0.0	10
15	538	0.0	103	0.0	36
18	6753	0.0	1364	0.0	222
21	86796	0.0	15298	0.0	1632

K is defined as reported in (B.1).

We search the optimal K that minimize the overall execution time of *Bruteforce* algorithm, and use it when compare the execution times of the algorithms.

The choice of the optimal K is reported in Appendix B and the value of the optimal K is reported in each interference gap table and speedup chart.

In small-scale coordination we are particularly in coordinating 21 antennas because with the system model we have done, such a coordination makes the central cluster of antennas to be "protected" from interference.

6.2.1 Average demand

Interference gap

Table 6.1 and 6.2 summarizes the results of "average demand" scenario in the 50 blocks scenario, reporting both execution times and gaps.

The K parameter used at 21 antennas is 1 ‰, while in the other cases the K parameter may differ. In all the cases, BF runs in acceptable times.

Configurations with 25 and 100 blocks give very similar results, so we don't report those tables.

Table 6.2: Small coordination in average demand scenario, 50 blocks: OPT, FF and RRB.

	OPT	FF		RRB	
N	Ex. time (ms)	Gap (%)	Ex. time (ms)	Gap (%)	Ex. time (ms)
3	1.8	60.3	0.05	29.3	0.06
6	2.3	61.2	0.07	32.0	0.08
9	6.1	60.1	0.08	30.1	0.10
12	45	59.2	0.11	28.2	0.12
15	538	58.9	0.15	27.5	0.16
18	6753	58.7	0.20	27.4	0.21
21	86796	57.9	0.23	25.8	0.26

Table 6.3: Small coordination in low demand scenario, 50 Blocks: OPT, RC and BF.

	OPT	RC		BF	
N	Ex. time (ms)	Gap (%)	Ex. time (ms)	Gap (%)	Ex. time (ms)
3	1.2	0.0	0.28	0.0	1.6
6	2.3	0.0	0.37	0.0	2.5
9	6.0	0.0	0.99	0.0	2.8
12	44	0.0	8	0.0	6
15	449	0.0	103	0.0	23
18	5485	0.0	1345	0.0	158
21	68231	0.0	14185	0.0	1247

Speedup comparison

Figure 6.1 shows the speedup comparison between the continuous relaxation and *Brute-force* when antennas have 50 RB at their disposal. Of course, we don't compare these algorithms with the first-fit or with the random one because this latter have considerably smaller execution time.

6.2.2 Low demand

Interference gap

Table 6.3 and 6.4 summarizes the results of "low demand" scenario in the 50 blocks scenario, reporting both execution times and gaps.

The K parameter used at 21 antennas is 3 ‰, while in the other cases the K parameter may differ. In all the cases, BF runs in acceptable times.

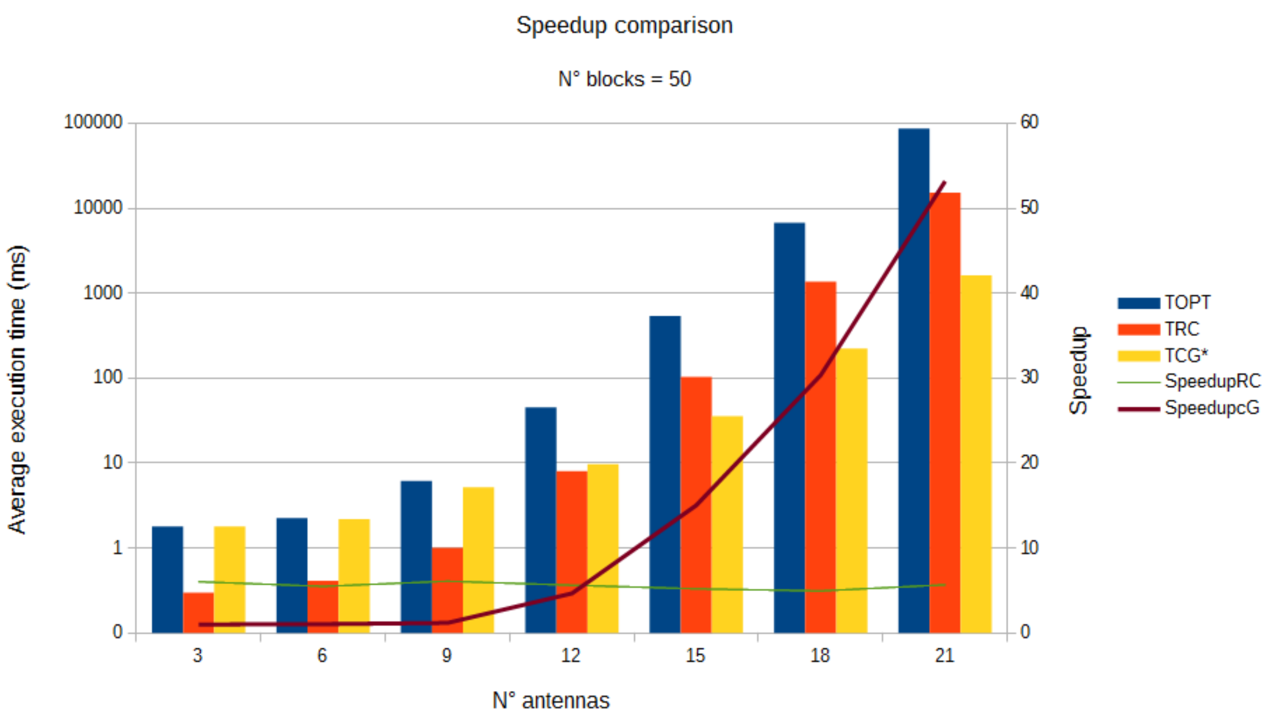


Figure 6.1: Average demand scenario: speedup comparison for 50 blocks

Table 6.4: Small coordination in low demand scenario, 50 blocks: OPT, FF and RRB.

	OPT	FF		RRB	
N	Ex. time (ms)	Gap (%)	Ex. time (ms)	Gap (%)	Ex. time (ms)
3	1.2	100	0.05	100	0.06
6	2.3	99.5	0.07	98.4	0.07
9	6.0	98.9	0.09	95.9	0.09
12	44	98.2	0.11	93.4	0.13
15	449	97.8	0.15	91.8	0.17
18	5485	97.4	0.20	90.4	0.20
21	68231	96.9	0.23	88.7	0.25

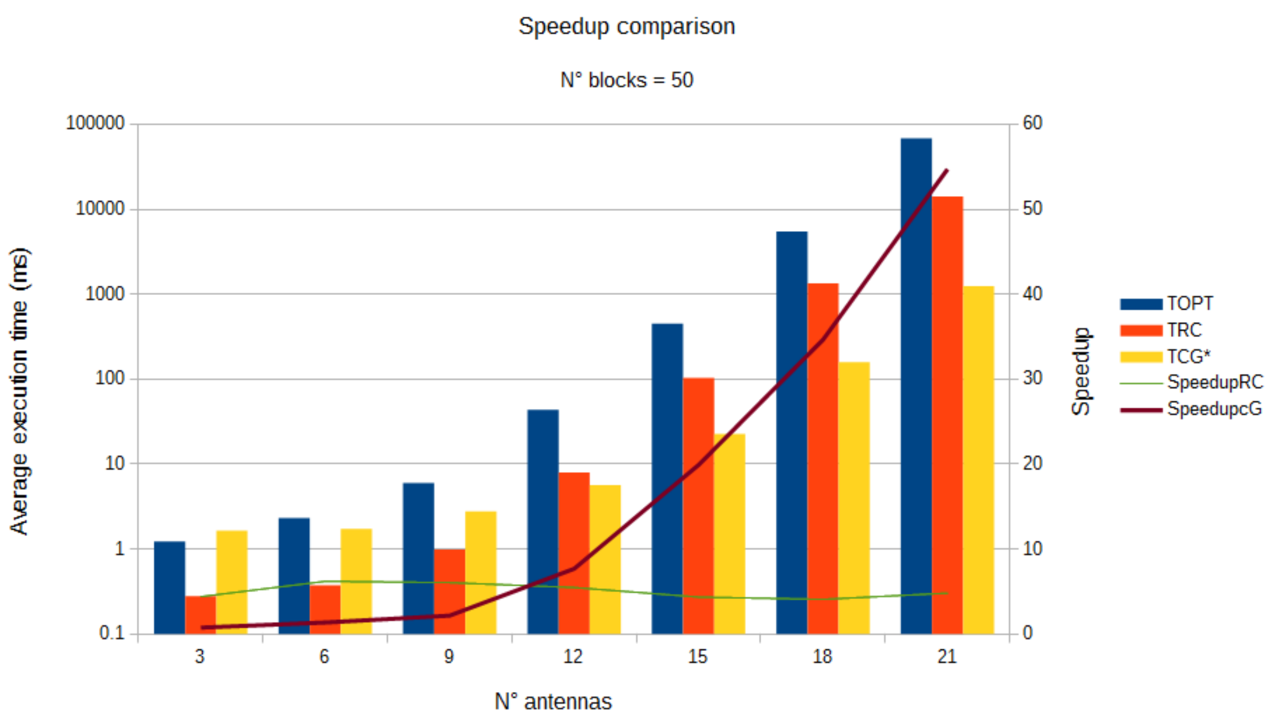


Figure 6.2: Low demand scenario: speedup comparison for 50 blocks

Speedup comparison

Fig 6.2 shows the speedup comparison between the continuous relaxation and *Brute-force* when antennas have 50 RB at their disposal. Of course, we don't compare these algorithms with the first-fit or with the random one because this latter have considerably smaller execution time.

Even in this case BF is faster than RC from 12 antennas.

6.2.3 High demand

Interference gap

Table 6.5 and 6.6 summarizes the results of "High demand" scenario using $m = 50$.

The K parameter used at 21 antennas is 1 ‰, while in the other cases the K parameter may differ. In all the cases, BF runs in acceptable times.

Speedup comparison

Figure 6.3 shows the speedup comparison between the continuous relaxation and *Brute-force* when antennas have 50 RB at their disposal. Of course, we don't compare these

Table 6.5: Small coordination in high demand scenario, 50 Blocks: OPT, RC and BF.

	OPT	RC		BF	
N	Ex. time (ms)	Gap (%)	Ex. time (ms)	Gap (%)	Ex. time (ms)
3	1.8	0.0	0.31	0.0	1.9
6	2.3	0.0	0.38	0.0	2.1
9	6.8	0.0	1.1	0.0	2.3
12	62	0.0	11	0.0	4
15	717	0.0	107	0.0	22
18	8298	0.0	1424	0.0	166
21	116985	0.0	14238	0.0	1167

Table 6.6: Small coordination in high demand scenario, 50 blocks: OPT, FF and RRB

	OPT	FF		RRB	
N	Ex. time (ms)	Gap (%)	Ex. time (ms)	Gap (%)	Ex. time (ms)
3	1.8	17.1	0.05	3.5	0.06
6	2.3	16.0	0.07	3.3	0.08
9	6.8	16.1	0.09	3.3	0.10
12	62	15.9	0.11	3.2	0.12
15	717	15.8	0.15	3.2	0.18
18	8298	16.0	0.19	3.2	0.20
21	116985	15.8	0.25	3.1	0.25

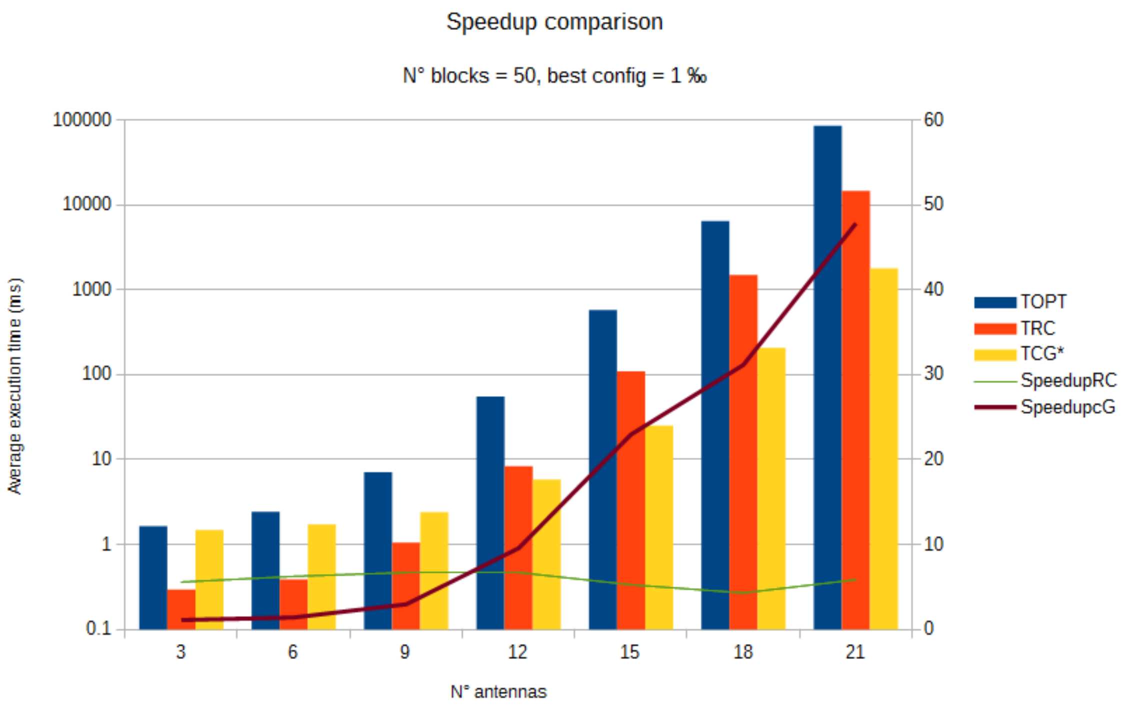


Figure 6.3: High demand scenario: speedup comparison for 50 blocks

algorithms with the first-fit or with the random one because this latter have considerably smaller execution time.

6.3 Evaluation of small-scale algorithms

We have seen from the previous sections we have seen that our *Bruteforce* (BF) algorithm reaches the optimal solution in very less time compared to CPLEX's branch-and-bound and to the continuous relaxation. This latter still reaches the optimal solution, but employs too much time, even if it's considerably less than the optimal solution.

First-fit (FF) and random resource block allocation (RRB) has high gaps in general, even if the RRB is always better than FF.

From tables of section 6.2, we can easily see that the average gap of this two algorithms decrease as the demand increases.

Indeed, let's call $\rho_i = A_i/m$ the *saturation ratio* of i-th antenna. We remember that $A_i \leq m$ for every i. So, if A_i increases, as it happens from "low" to "high" scenario, $\rho_i \to 1$.

If this happens, then the RBs at disposal of the i-th antenna are very few, so the possibility of optimization decreases and so every algorithm tends to return the same result (or similar).

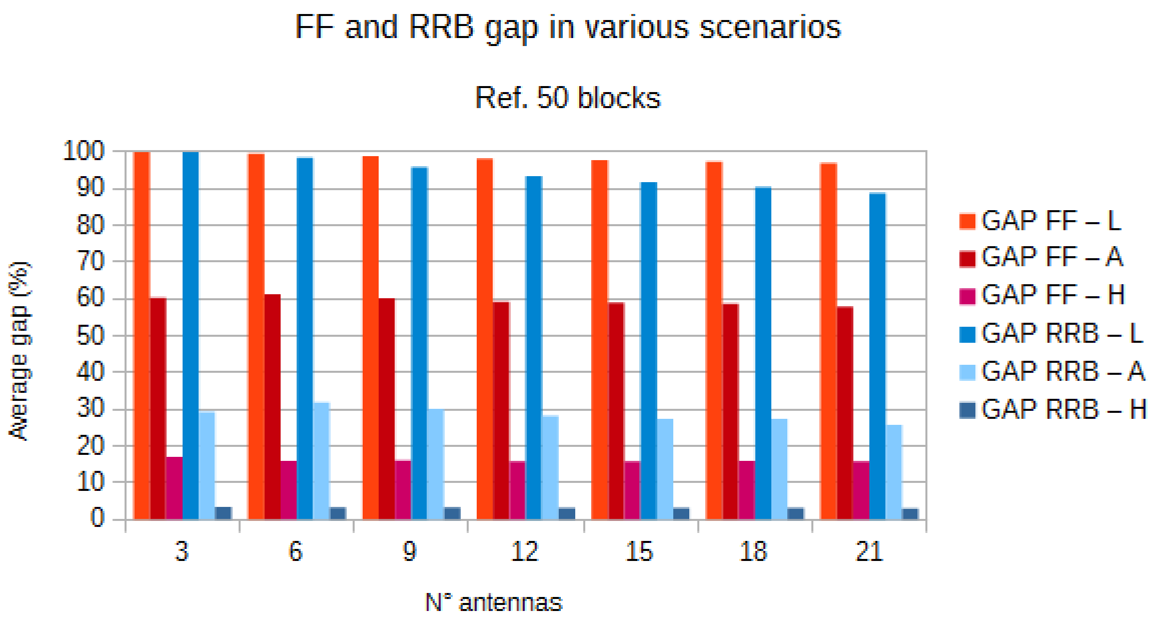


Figure 6.4: FF and RRB gaps over various scenarios

Instead, if A_i is very little compared to the m blocks at disposal from i-th antenna, then it's very important how we allocate these blocks. In this case, when $\rho \to 0$, we can see huge difference between the mathematical algorithms and FF and RRB.

In particular, RRB always gives a better bound than FF.

This can be graphically view in figure 6.4.

Instead RC and BF are very robust at the changing of the demand and their gaps still remain 0.

About execution time

Bruteforce is very suitable for small-scale coordination as it returns the optimum in a very less time compared to CPLEX branch-and-bound and RC, as we can see from the speedup charts.

Figure 6.5 shows how it varies the execution time of OPT, RC and BF in the various scenarios.

It seems that the execution time of OPT increases with the demand while RC is stable and BF prefer to deal with high demands, but this doesn't affect the result too much.

A comprehension of the semi-constant speedup of RC is now clearly visible, as the execution time both of OPT and RC grows of 1 order of magnitude for every more 3 antennas to coordinate.

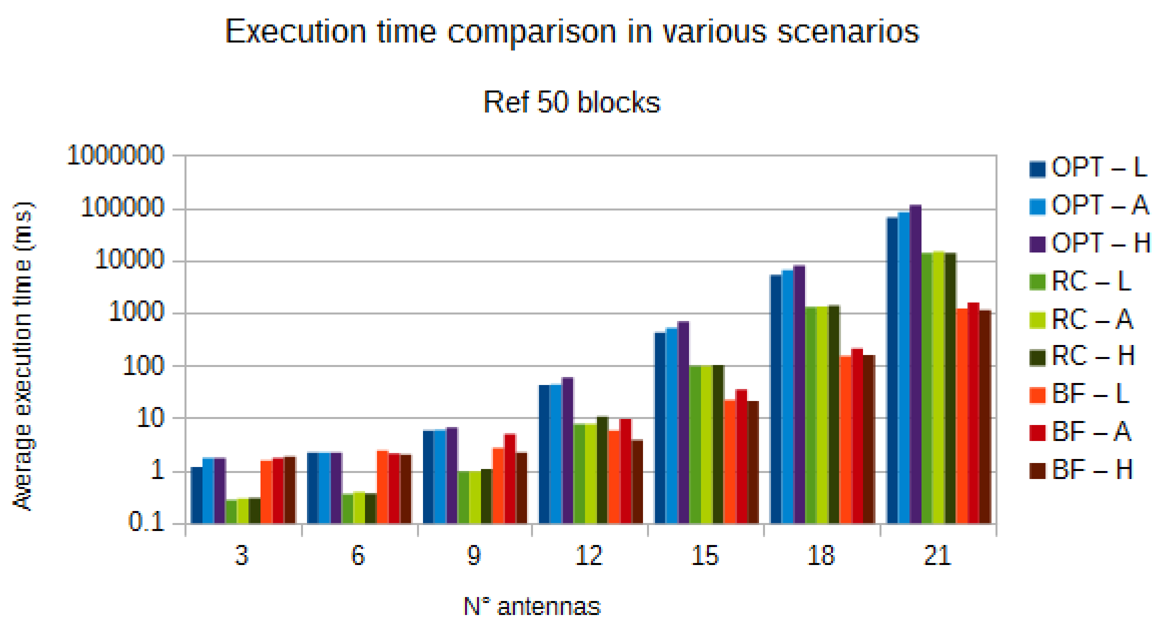


Figure 6.5: OPT, RC and BF execution times over various scenarios

However, we can see that the execution time of BF tends to *double* for every more antenna. This is due to the *linear enumeration* of *Graycode*. Indeed, calling T_{N-1} the execution time necessary to enumerate 2^{N-1} patterns, knowing that $2^N = 2^{N-1} \cdot 2$ we have that the execution time T_N necessary to enumerate 2^N pattern is:

$$T_N = T_{N-1} \cdot 2 \tag{6.3}$$

This effect is clearly visible from about 15 antennas. This is also the reasons for which it's not reasonable to use BF for large-scale coordination.

FF and RRB instead are very fast, but they huge gap make them not efficient in order to minimize interference.

About fairness

We would prefer if every antenna we coordinate contributes in the same quantity to the overall interference. This would lead to the same Quality of Service perceived by each user of the system.

Fairness is a measure of this fact. In reality, a situation of perfect fairness practically is never reached.

A way to quantify how *unfair* our system is to quantify how variable the data are. In this case, we can plot a **Lorenz curve**. In this case, we can define our Lorenz curve

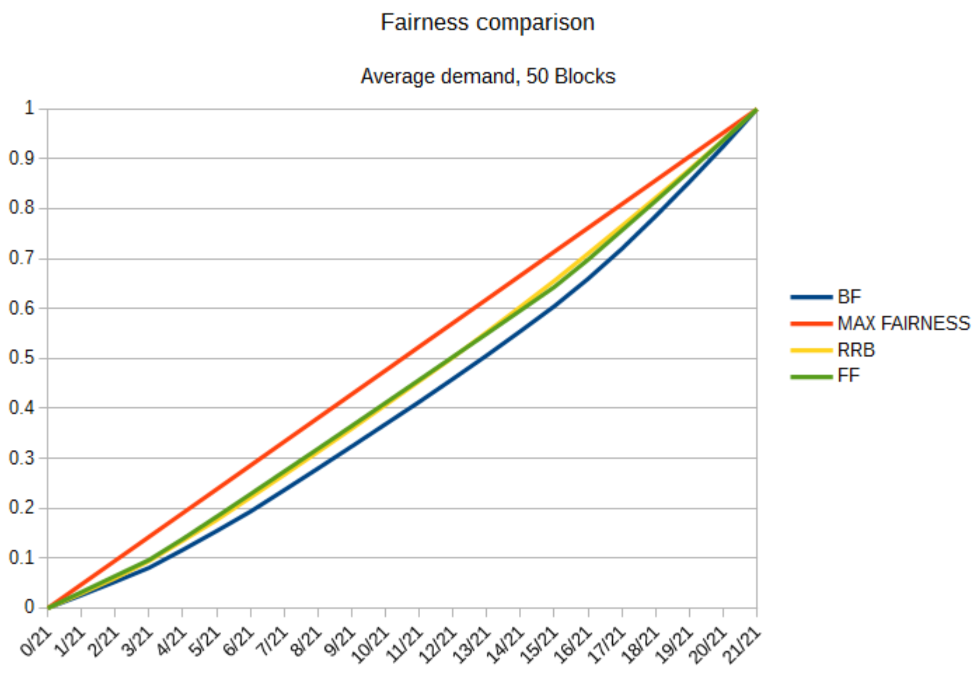


Figure 6.6: Fairness comparison in average demand scenario

as:

1. Sort the single cost of interference ϕ_i from each antenna and get the ordered statistics $\phi^i_{(1)} \leq \phi^j_{(2)} \dots \phi^k_{(N)}$ where i, j, k are different antennas and $1, 2, \dots, N$ are the indexes for them.

2. Compute $\Phi = \sum_{i=1}^{N} \phi_i$, that is the total cost of interference.

3. Plot points $(1/N, \phi^i_{(1)}/\Phi), (2/N, (\phi^j_{(1)} + \phi^i_{(2)})/\Phi, \dots, (k/N, \sum_{i=1}^{N} \phi_i/T)$

4. Interpolate, assuming (0,0) as first point of the curve.

Maximum fairness is represented by the bisector of the first quadrant. This is called the line of maximum fairness, and can be plotted for reference. The situation of minimum fairness (or maximum unfairness)is instead the horizontal axis and the vertical line passing through (1,0).

In the middle of these two references there are the other common cases.

In figure 6.6 is reported the fairness comparison of our small-scale algorithms [2] in the average scenario.

As we can see, all the algorithms have a good fairness, even if BF is more unfair than FF and RRB.

[2]Of course, OPT and RC are not reported as BF also reach the optimal solution.

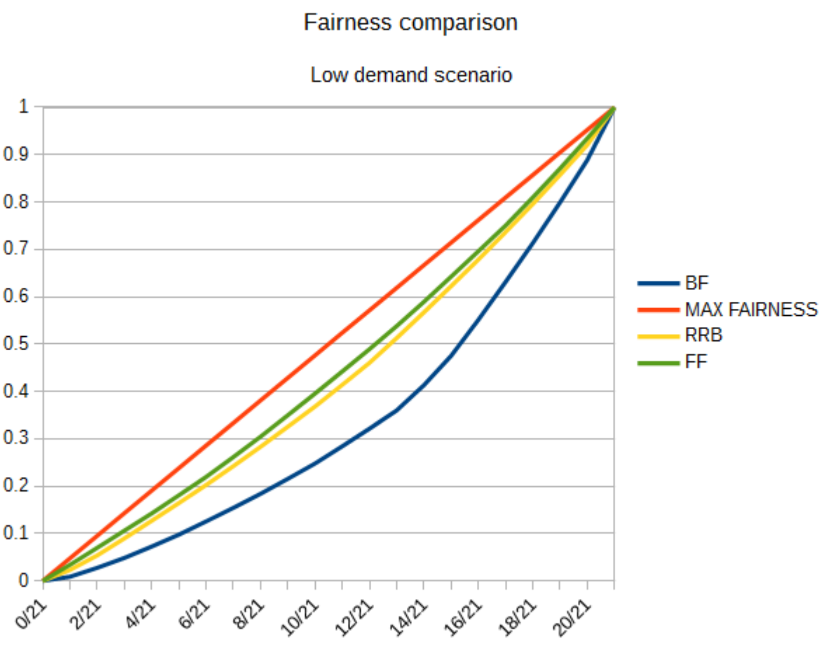


Figure 6.7: Fairness comparison in low demand scenario

This is due to an optimization issue, as optimization aims to minimize the overall interference. In fact we have to remind that the total interference with the use of BF is less than FF and RRB, so that even the more penalized antennas coordinated by BF causes less interference than FF and RRB.

From figure 6.7 and 6.8 we can also see that fairness increases with the demand. For an high demand scenario, the fairnesses of the three algorithms are practically overlapped.

Even in this cases, we have to remind that the increase of the demand makes more interference among the antennas we coordinate.

We can conclude that *Bruteforce* is the best small-scale algorithm among all those seen.

6.4 Large-scale coordination

In large-scale coordination is prohibitive to refer to the optimal solution due to the exponential execution time.

So, we can do a comparison based on the total *interference cost* [W], including for each algorithm the two TLs discussed in chapter 5.

As we have a global TL of 5 s on the total execution time, we won't compare the

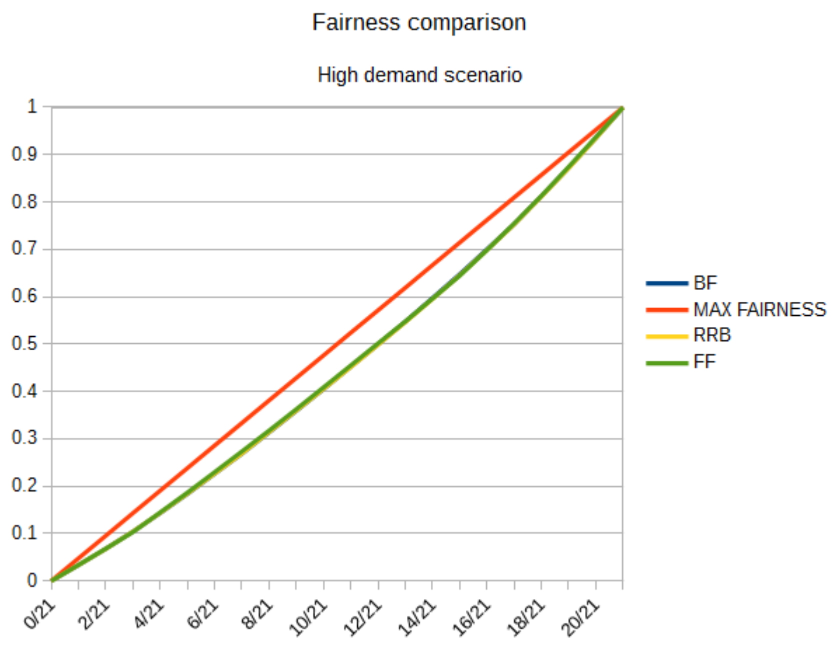


Figure 6.8: Fairness comparison in high demand scenario

actual execution time[3].

The algorithms we will compare are:

- **First-fit**
- **Random allocation**
- **UBQP**, that is the columns generation through CPLEX's QP solver
- **UBQP flip**, that is a version of UBQP in which every time CPLEX gives us a pattern we randomly flip a bit of that pattern in order to generate another pattern similar to the previous one
- **UBQP linearised**, that is the linearisation of the UBQP problem, which can exploit *populate* function of CPLEX.

From preliminary tests that the total execution time of these large-scale algorithms is due practically only to column generation and to final mipopt.

Trivially, if we increase the TL on the column generation, we'll generate more variables and get a better solution, while if we increase the TL on the mipopt, we'll find a better bound with the variables at disposal.

[3]The actual execution time may be greater than 5 s because the execution time is not controlled in real time

Table 6.7: Large coordination in average demand scenario: gap comparison until 21 antennas

N	UBQP	UBQP flip	UBQP linzd
3	0.0	0.0	0.0
6	0.0	0.0	0.0
9	0.0	0.0	0.0
12	0.0	0.0	0.0
15	0.0	0.0	0.0
18	0.6	0.4	2.7
21	3.0	2.3	2.4

So, we have firstly tuned the algorithms splitting the overall TL in its two components *column generation* and *mipopt* in order to get the best solution at disposal.

We refer to appendix C for details about the choice of optimal partition.

All the following results have been averaged from a set of 10 runs, with the exception of RRB, that has been averaged from a set of 100 runs, as it has been ran with 10 different seeds for each configuration.

6.4.1 Average demand

We can compare our algorithms with the optimal solution until 21 antennas.

Table 6.7 and summarizes the results of "average demand" scenario in the 50 blocks scenario, reporting the gaps compared to the optimal solution.

We can see that these algorithm are very good also in small-scale coordination, even if they have a bigger execution time than *Bruteforce*.

Figure 6.9 gives us an overall vision of the 3 algorithms.

We can see clearly that the FF is very distant from the other algorithms.

Figure 6.10 shows in detail the interference comparison among UBQP, UBQP flip, UBQP linzd and RRB.

6.4.2 Low demand

Table 6.8 and summarizes the results of "low demand" scenario in the 50 blocks scenario, reporting the gaps compared to the optimal solution.

Figure 6.11 gives us an overall vision of the 3 algorithms.

In this case, we can see a quite large gap between our heuristics and RRB, with FF that is very far from the other algorithms.

Figure 6.12 shows in detail the interference comparison among UBQP, UBQP flip, UBQP linzd and RRB.

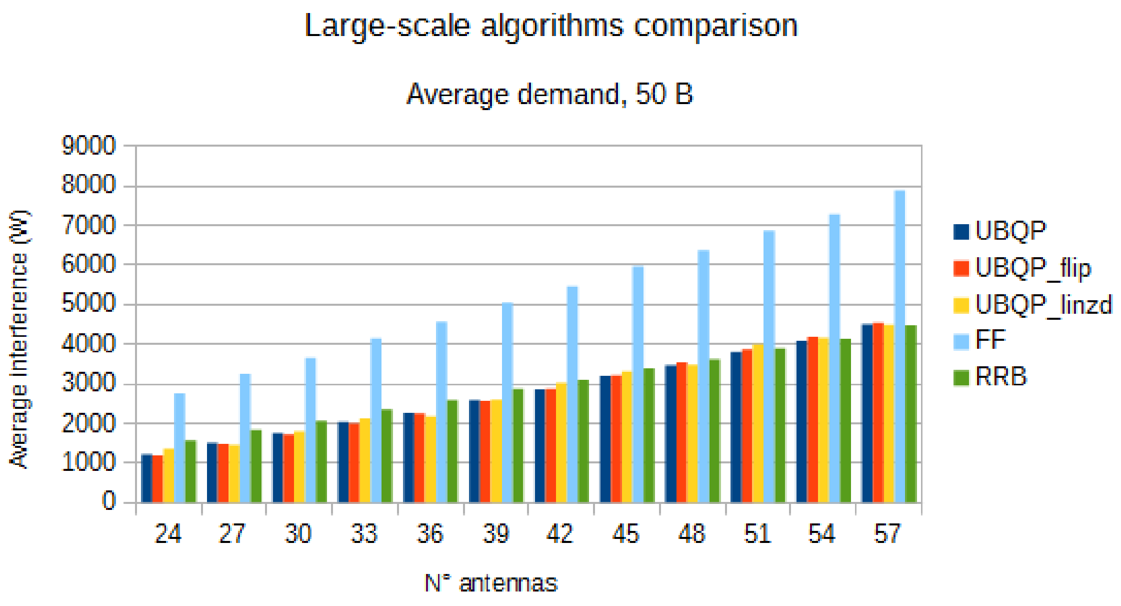
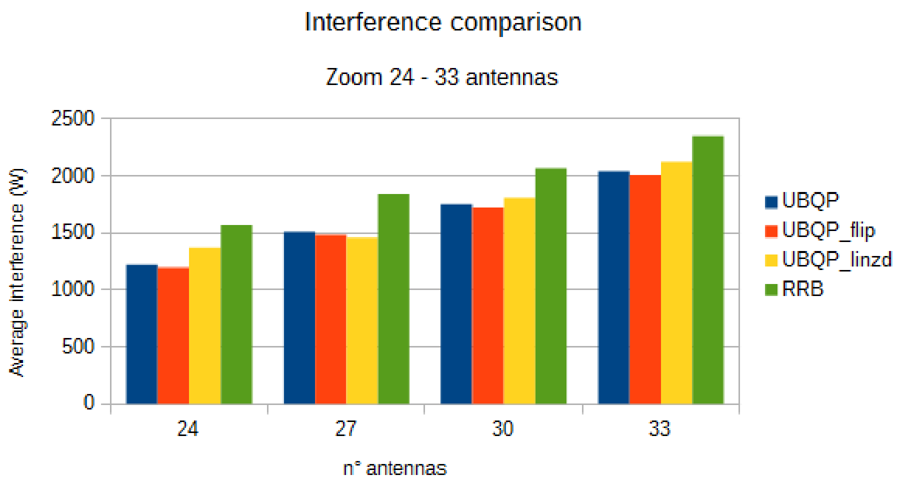


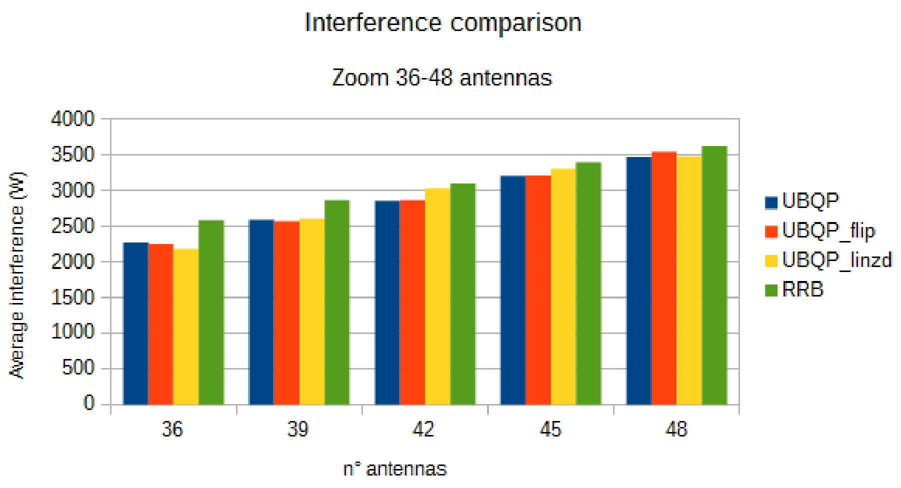
Figure 6.9: Interference comparison in average demand scenario

Table 6.8: Large coordination in low demand scenario: gap comparison until 21 antennas

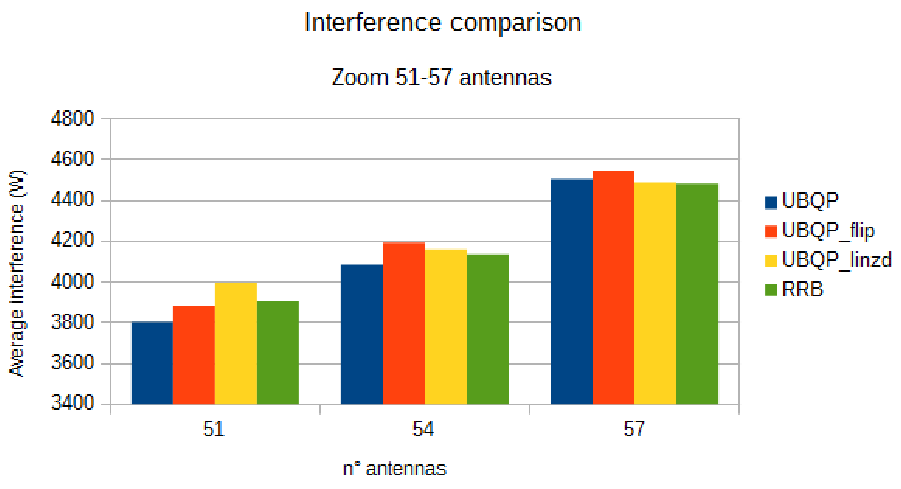
N	UBQP	UBQP flip	UBQP linzd
3	0.0	0.0	0.0
6	0.0	0.0	0.0
9	0.0	0.0	0.0
12	0.0	0.0	0.0
15	0.0	0.0	0.0
18	0.0	0.0	0.0
21	0.3	0.1	1.0



(a) 24-33 antennas.



(b) 36-48 antennas.



(c) 51-57 antennas.

Figure 6.10: Interference comparison in average demand scenario: zoom on UBQP, UBQP flip, UBQP linzd and RRB

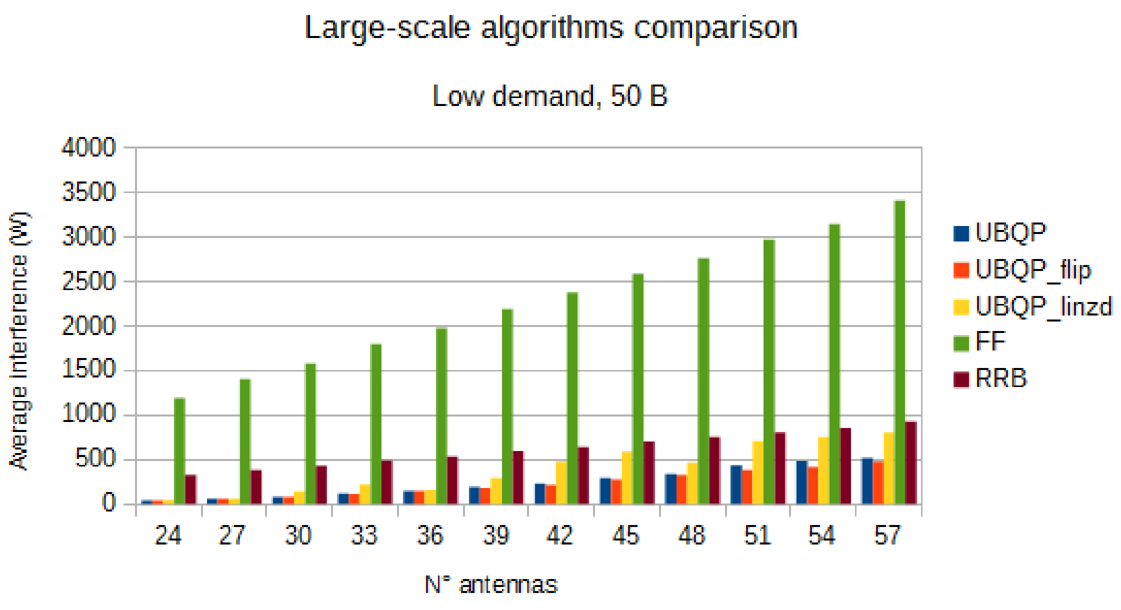


Figure 6.11: Interference comparison in low demand scenario

Table 6.9: Large coordination in high demand scenario: gap comparison until 21 antennas

N	UBQP	UBQP flip	UBQP linzd
3	0.0	0.0	0.0
6	0.0	0.0	0.0
9	0.0	0.0	0.0
12	0.0	0.0	0.0
15	0.0	0.0	0.0
18	0.0	0.0	0.0
21	0.0	0.0	0.0

We can clearly see that the differences between the algorithms are more pronounced in this case.

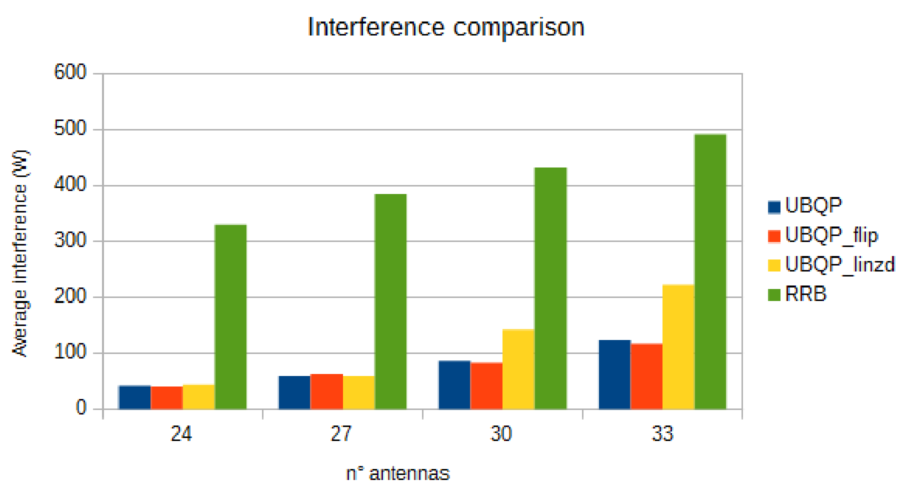
For example, we can coordinate 33 antennas with an overall interference of about 100 W instead of 500 W as obtained from RRB.

Even at 57 antennas, with the “UBQP flip” algorithm we can save about 400 W of interference.

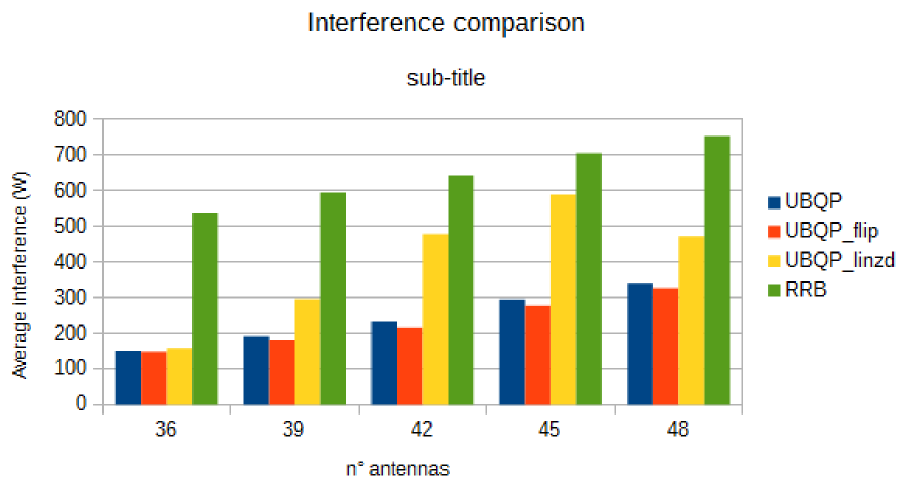
6.4.3 High demand

Table 6.9 and summarizes the results of “high demand” scenario in the 50 blocks scenario, reporting the gaps compared to the optimal solution.

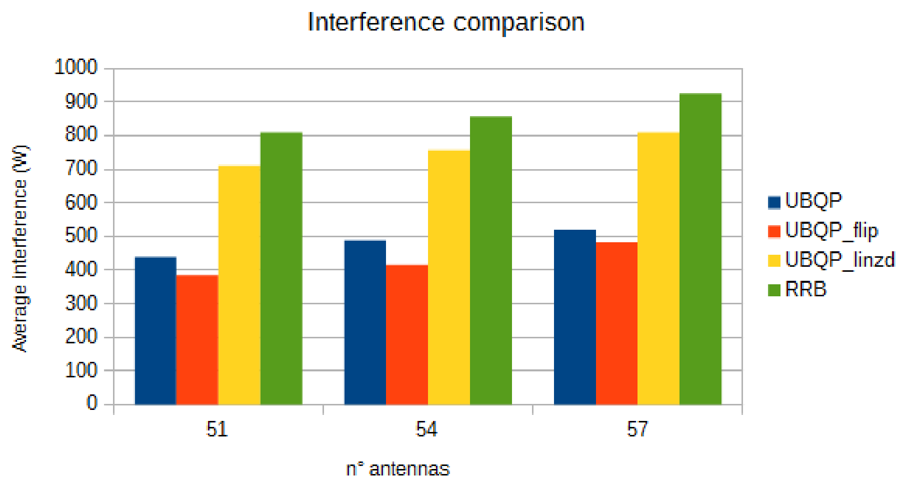
As expectable, the heuristics reach the optimal solution in a high demand scenario.



(a) 24-33 antennas.



(b) 36-48 antennas.



(c) 51-57 antennas.

Figure 6.12: Interference comparison in low demand scenario: zoom on UBQP, UBQP flip, UBQP linzd and RRB

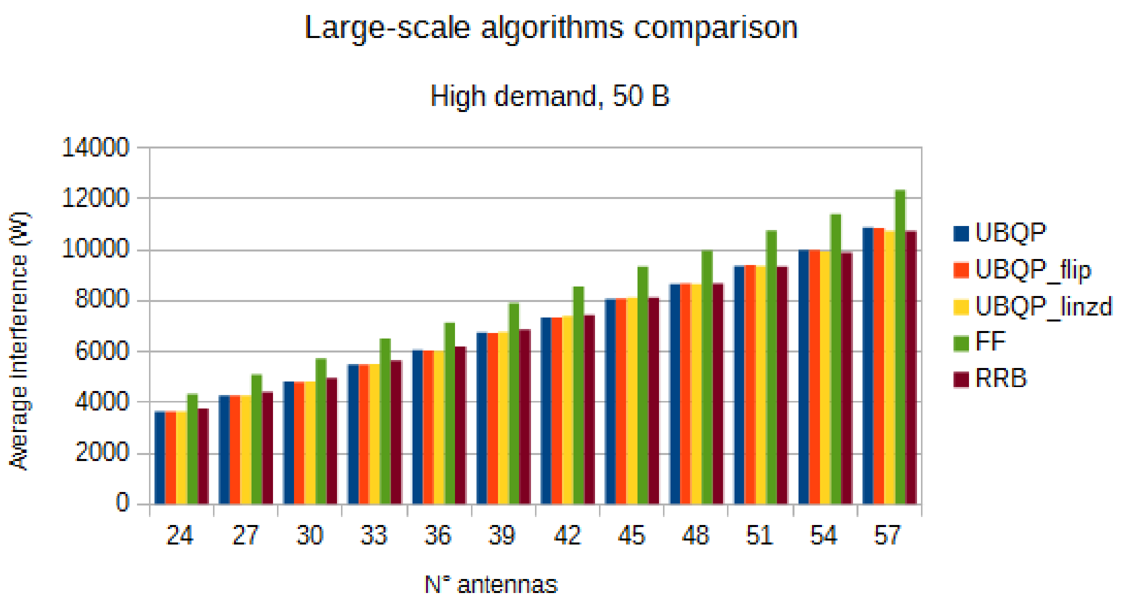


Figure 6.13: Interference comparison in high demand scenario

Figure 6.13 gives us an overall vision of the 3 algorithms.

In this case, we can see that the gap among our heuristics and RRB is very small, and also FF doesn't differ too much.

Figure 6.14 shows in detail the interference comparison among UBQP, UBQP flip, UBQP linzd and RRB.

We can clearly see that UBQP, UBQP flip, UBQP linzd and RRB practically return the same result from 24 antennas to 57.

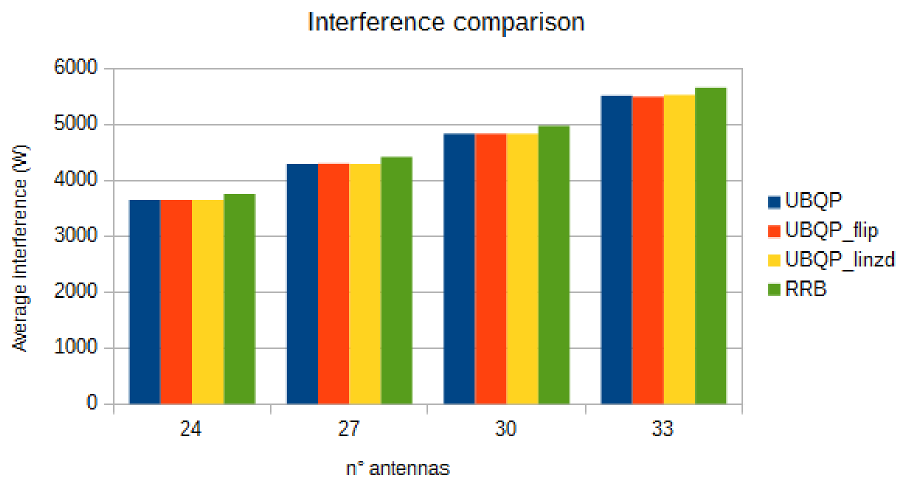
6.5 Evaluation of large-scale algorithms

We have seen that with the increase of N, the gaps between algorithms decrease compared to the small-scale coordination. This is due to the increase of complexity of the problem with a TL of 5 s.

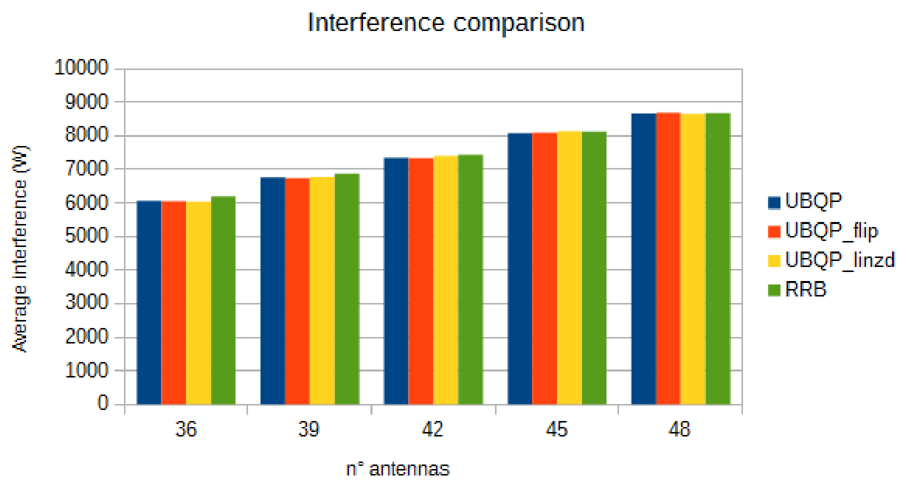
We can also see that in a low demand scenario a coordinated scheduling leads to observable better results, while in the high demand scenario the algorithms return practically the same result.

In every scenarios, when $N \to 57$, all the algorithms tend to give the same results.

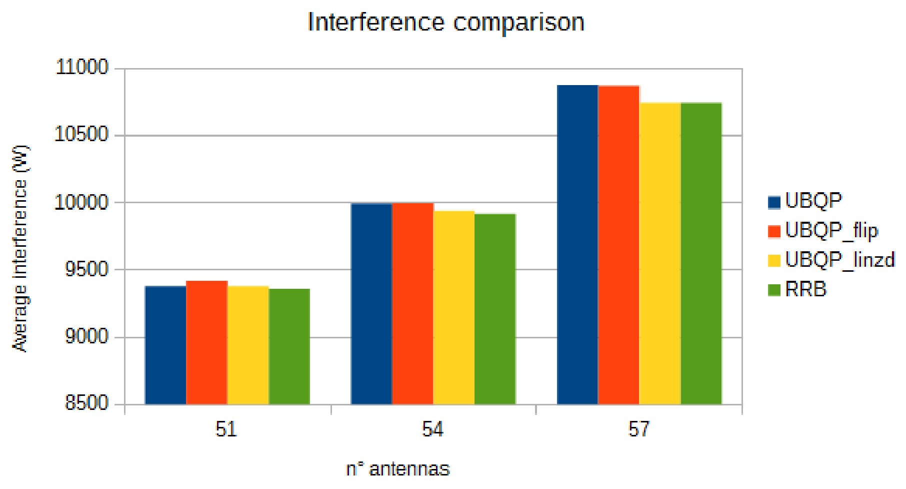
Note that at 57 antenna sometimes (e.g. in the high scenario) the RRB can do better than some of our heuristics. The reason is that when CPLEX can't return (for time-limit reasons) a solution, the algorithm takes the initial one, that is random. On the other side, RRB is ran 10 times for every configuration, so it's possible that



(a) 24-33 antennas.



(b) 36-48 antennas.



(c) 51-57 antennas.

Figure 6.14: Interference comparison in high demand scenario: zoom on UBQP, UBQP flip, UBQP linzd and RRB

sometimes the solution of RRB is better than the random one returned by a heuristic when CPLEX can't return a solution.

From the previous charts, we can say that UBQP flip is a slight better version than UBQP, and so UBQP flip is our best algorithm at disposal for a large-scale coordination.

This algorithm can significantly reduce interference even for a large number of antennas in a low scenario.

Chapter 7

Conclusions

In this thesis, the inter-cell interference problem in mobile networks has been addressed. Since the interference represents a major limiting factor of LTE, a description of solution based on mathematical techniques has been provided.

In chapter 2 we have shown a quite exhausting background on LTE and on mathematical optimization.

In chapter 3 we have propose a system model based on hexagons and on three main entities: antenna, communication channel and user. Then, a mathematical model has been also proposed to translate the system in a mathematical language.

Chapters 4 describes the algorithm *Bruteforce*, that does a smart enumeration of columns in order to do the *column generation* that helps the problem to find a solution without using all the possible variables, that are exponential.

In chapters 5 we have defined a time limit of 5 s for the coordination problem, and we have seen new algorithms still based on the column generation technique but not on enumeration.

Evaluating the performances of our algorithms, we have seen:

- In small-scale coordination *Bruteforce* reaches the optimal solution in the small-scale coordination, running in acceptable times (about 1 s).
- In large-scale coordination, where Bruteforce is no more suitable, we can use the UBQP or UBQP flip algorithms in order to obtain less interference in a fixed time of 5 s.

At every scale of coordination, we observe that the reduction of interference becomes more pronounced as the block demand becomes lower. This because there are more possibilities to optimize when the demand is low, and vice versa.

Future works can use the results of this thesis to test how this algorithms improves the network performance in dynamic coordination, where we iteratively coordinate a certain set of antennas over the time.

Appendix A

CPLEX

CPLEX is the abbreviation of **IBM ILOG CPLEX Optimization Studio** ©. It is a software developed in order to solve mathematical programming problems.

CPLEX can solve many types of mathematical programming such as:

- Linear programming (LP)
- Integer linear programming (ILP)
- Mixed integer linear programming (MILP)
- Quadratically constrained quadratic programming (QCQP)

and many others.

CPLEX can work in 2 modalities:

1. **Interactive modality**: based on launching CPLEX as a normal program and inserting commands from the terminal. It can be useful for problems with few variables and constraints, and it is easier to use.
2. **Callable library**: based on including CPLEX library (*cplex.h*) in a C or C++ program and writing a code. It is more flexible than the previous one and it allows to iteratively add variables and/or constraints to the problem.

The second modality is of course that one which more suits our aims.

Appendix B

Tuning of *Bruteforce*

In this chapter we show how K has been tuned for *Bruteforce algorithm.*

In this appendix we report the tuning only in the case of average demand. The tuning of K wants to minimize the overall execution time of *Bruteforce* at 21 antennas.

We are particularly interested in coordinating 21 antennas because, as we can see from fig 3.2, we are able to "protect" from interference the central cluster.

The total execution time of algorithm *Bruteforce* (BF) can be divided in:

- **Enumeration's execution time** (ENU) : amount of time taken to generate the dual problem, solve it enumerating through *Graycode* and give back the new patterns.

- ***Primopt* time** (PRIM) : amount of time taken by CPLEX to solve the relaxed problem with the new patterns.

- ***Mipopt* time** (MIP) : execution time of CPLEX's B&B in order to solve the problem at integrality.

- **Overhead time** (OH): this time can be defined as:

  $OH = TOT - ENU - PRIM - MIP$, where TOT is the total execution time measured by the machine.

The following charts show the so-called *profiling* of the algorithm *Bruteforce.* This is helpful as it shows strengths and weaknesses of the algorithm, and it permit us to individuate the best K.

Instead of varying K on absolute values (e.g. 1000 columns per time) we have set K as:

$$K = k \cdot \frac{2^N}{1000} \tag{B.1}$$

with k that is a "permille" parameter that varies on a logarithmic scale.

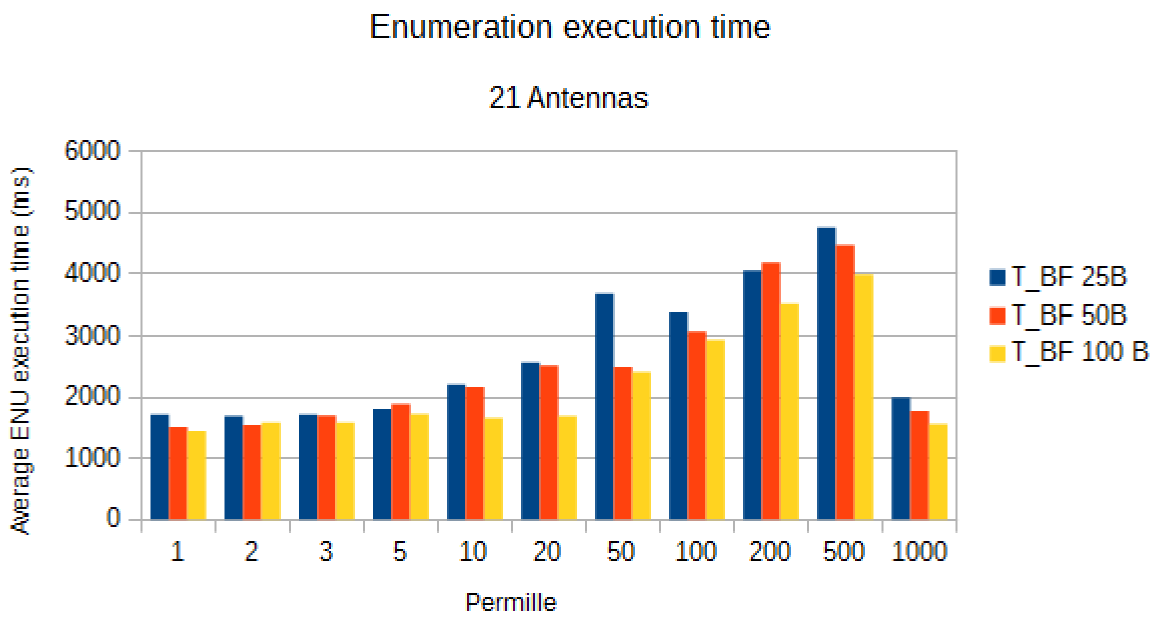


Figure B.1: Average demand scenario: enumeration's execution time for 21 antennas

Table B.1: Best K for every set of antennas

Number of antennas	K
3	1000
6	1000
9	20
12	20
15	2
18	2
21	1

The values of k we have tested are: $1, 2, 3, 5, 10, 20, 50, 100, 200, 500, 1000$.

These values let us see the trend of the algorithm without stressing tests and machines.

Note that some k is not available for $N < 10$, for example $k = 50$ doesn't exist for $N = 3$, as the 5% of $2^3 = 8$ is rounded to 0. This have been taken in account in the code we have written, so that only non-trivial k have been taken in account in the simulation.

Table B.1 summarizes the studies on the optimal K for every set of coordinated antennas for the average scenario.

This table can be used in *Bruteforce*'s code to make it as fast as possible for every set of antennas. We can clearly note that K tends to decrease with the increase of N.

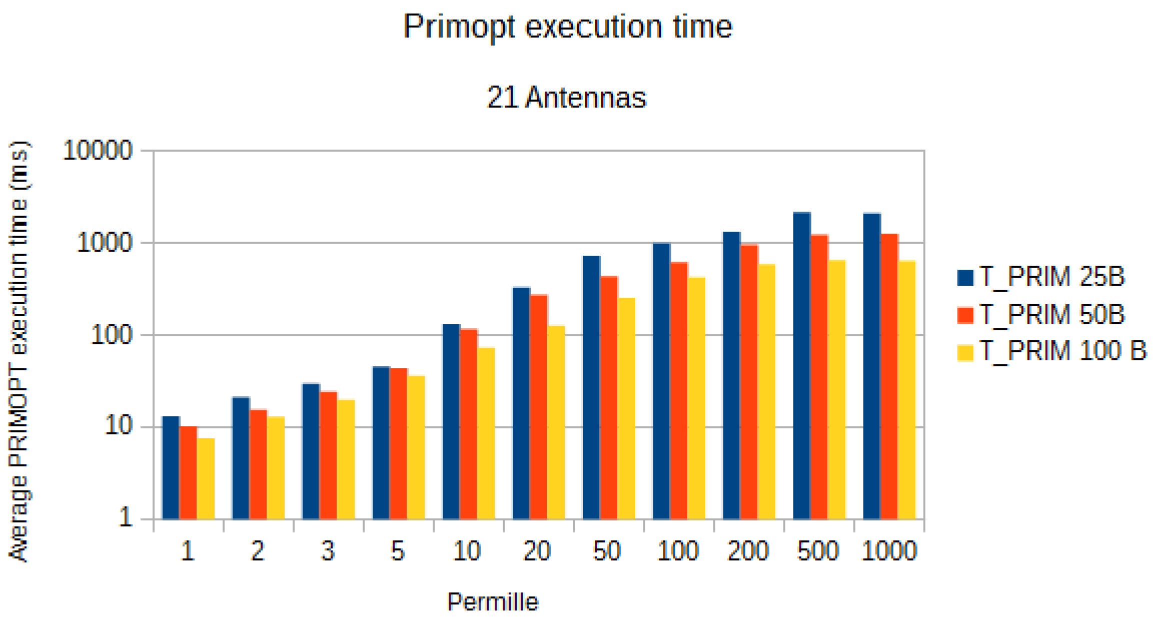


Figure B.2: Average demand scenario: *Primopt* execution time for 21 antennas

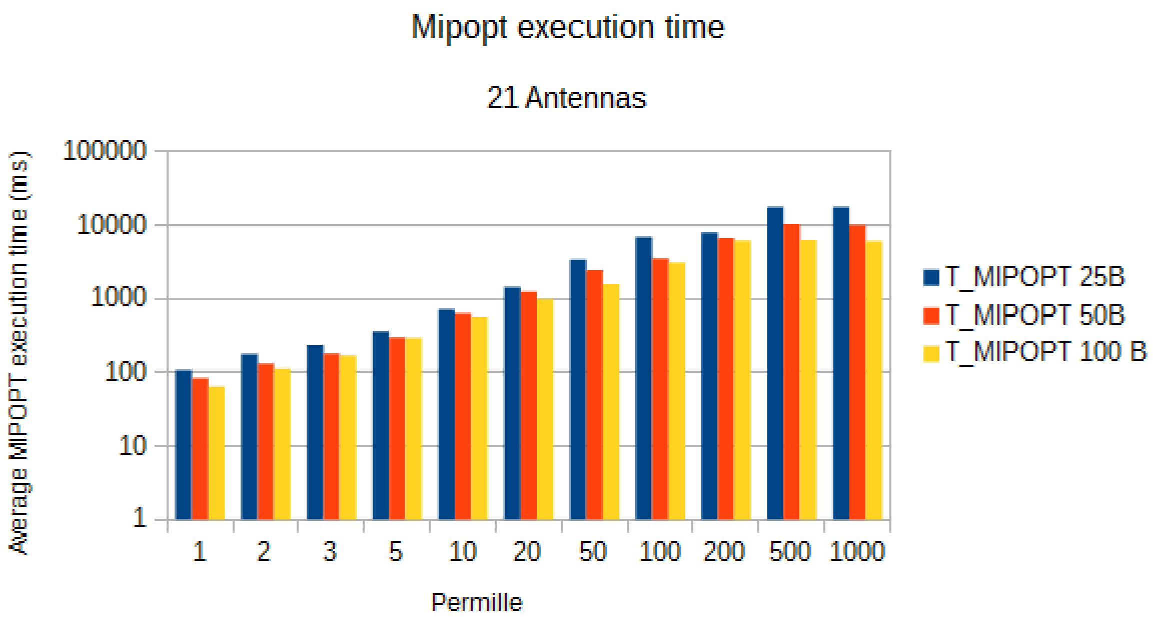


Figure B.3: Average demand scenario: *Mipopt* execution time for 21 antennas

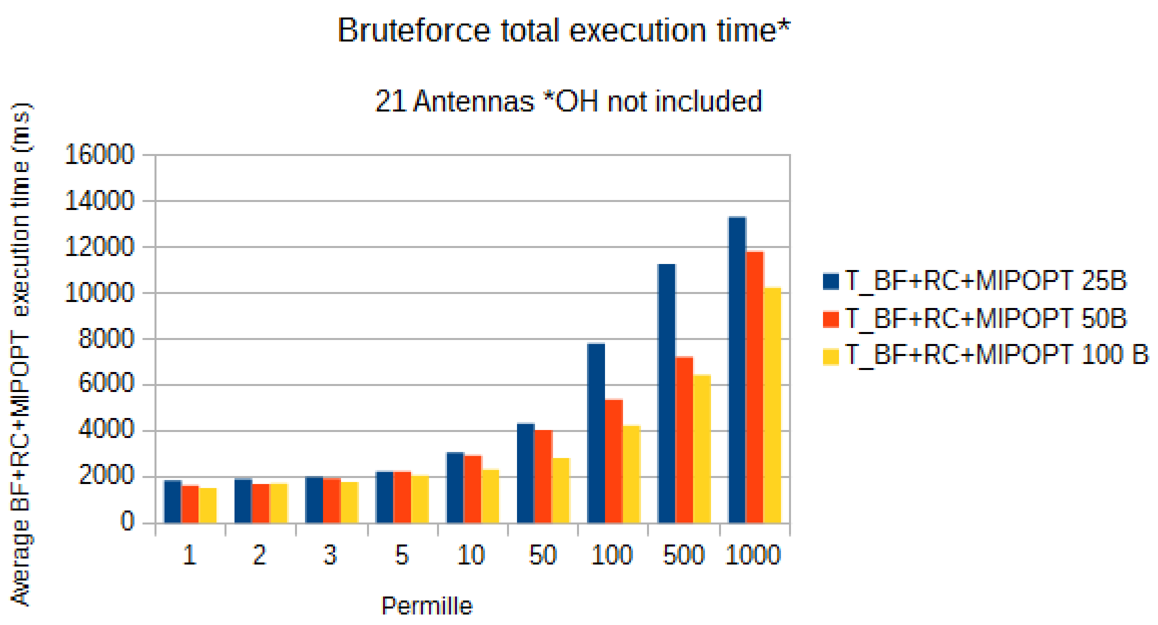


Figure B.4: Average demand scenario: Total active execution time for 21 antennas

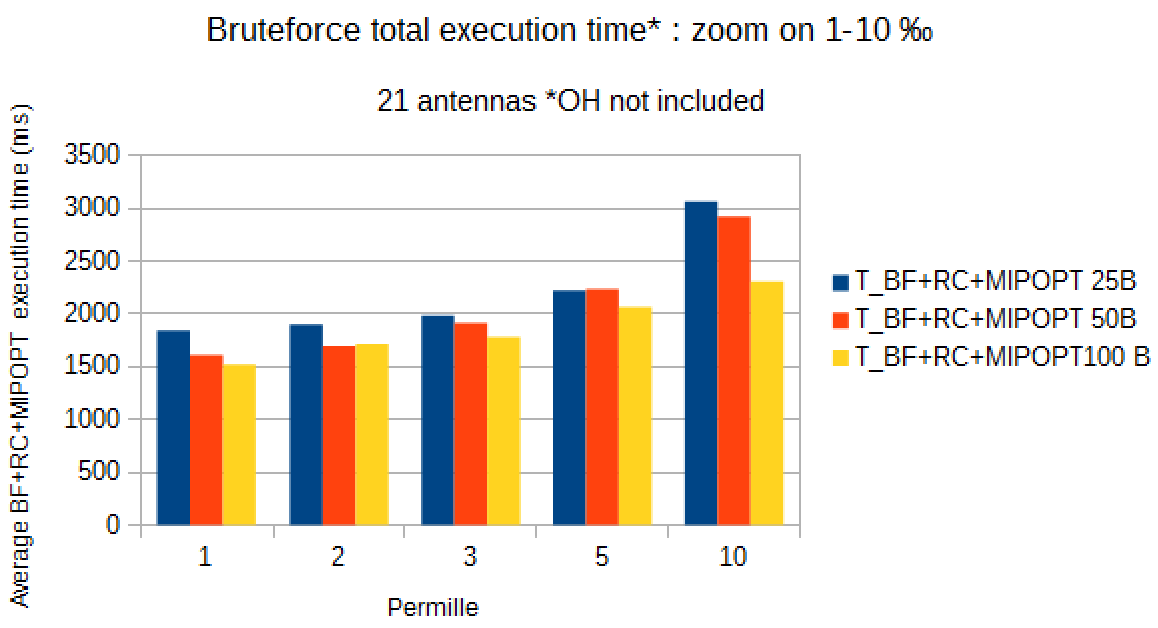


Figure B.5: Average demand scenario: Total active execution time for 21 antennas, zoom on 1-10 ‰

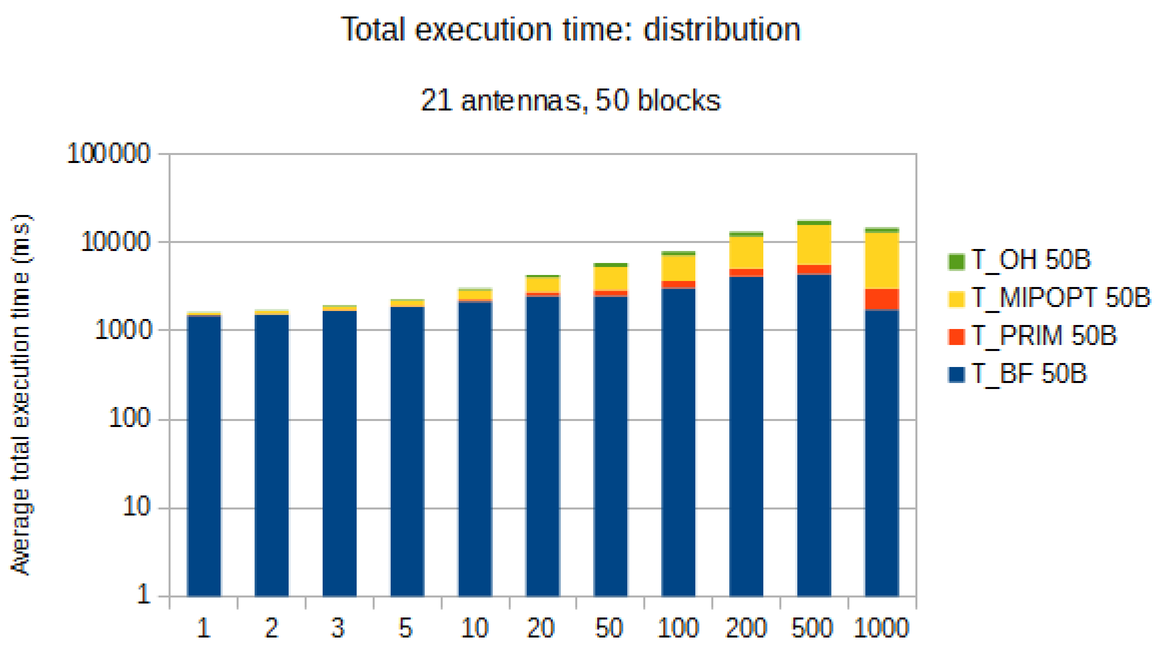


Figure B.6: Average demand scenario: Distribution of total execution time for 21 antennas

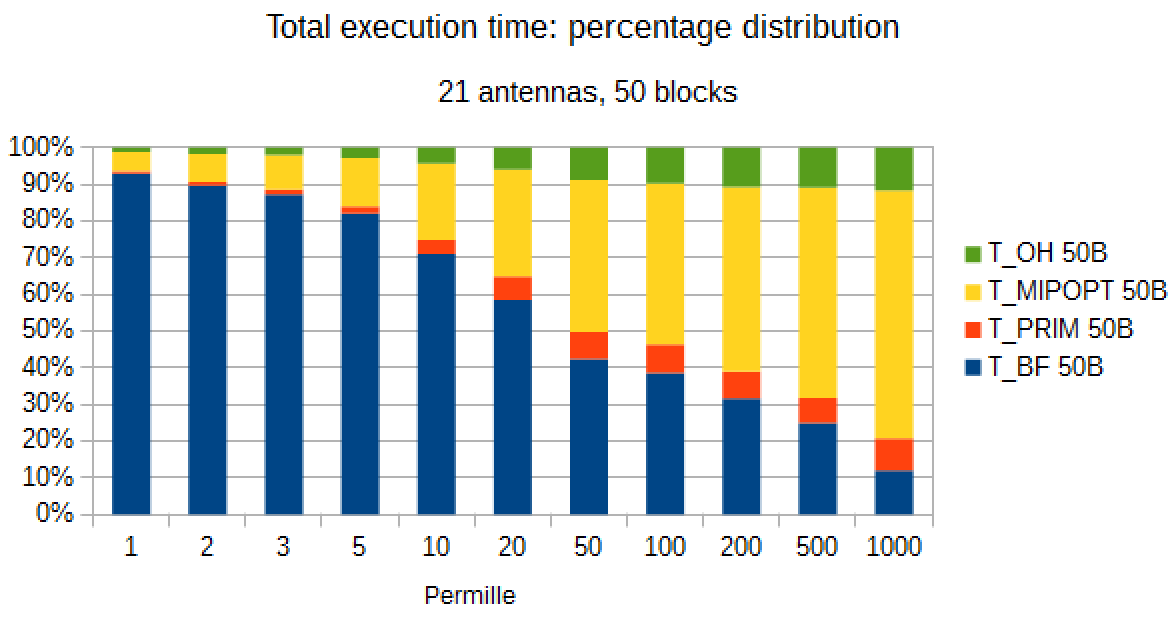


Figure B.7: Average demand scenario: Percentage distribution of execution time for 21 antennas

Comments

From the previous charts we can deduce that:

- The enumeration execution time implicitly has a trade-off: if K is very little, than the *graycode* and *bruteforce* function will run fast, but the whole algorithm needs more cycles before not finding a column with a reduced negative cost. Vice-versa if K is very big, the cycles would be repeated considerably less, but the above functions will employ much time. Another consideration is about the resolution of the test: we can note a swoop at 1000 ‰, but we don't know how the time scales between 500 and 1000 ‰.

  So, the enumeration execution time has a typical trade-off function shape.

- The *primopt* execution time, that is the execution time of CPLEX's Simplex method, grows not linearly with K. This is due to the bigger number of variables dealt with by CPLEX at every time it solves the master.

- Of course, also *mipopt* suffers the growth of the number of variables. Thus, as it is one of the major contributor of the overall execution time of *Bruteforce* algorithm, is not recommended to go to high values of permille parameter.

Appendix C

Tuning of *time limits*

In this chapter we deal with the tuning of the *time limits* (TLs) for large-scale algorithms.

We remind that we have a fixed TL of 5 s on the total execution time. The TL on the total execution time can be divided with very good degree of approximation in:

- TL on the *column generation*

- TL on the CPLEX's *branch-and-bound*.

Of course we want to split the total TL in order to achieve a minimum interference. We will consider the following partitions:

- **1-4** that is 1 s for the column generation and 4 s for the final branch-and-bound. This configuration should help the problem to find a good quality solution with the variables at disposal, but penalize may not provide the problem with a sufficient number of variables.

- **2-3** that is 1 s for the column generation and 4 s for the final branch-and-bound. This is a middle configuration.

- **3-2** that is 1 s for the column generation and 4 s for the final branch-and-bound. This is a middle configuration.

- **4-1** that is 1 s for the column generation and 4 s for the final branch-and-bound. This configuration should help the problem with a set of good variables but makes the solver do the branch-and-bound in a little time.

We considered only the optimal partition for average demand scenario and conjecture that partition is good also for the other scenarios.

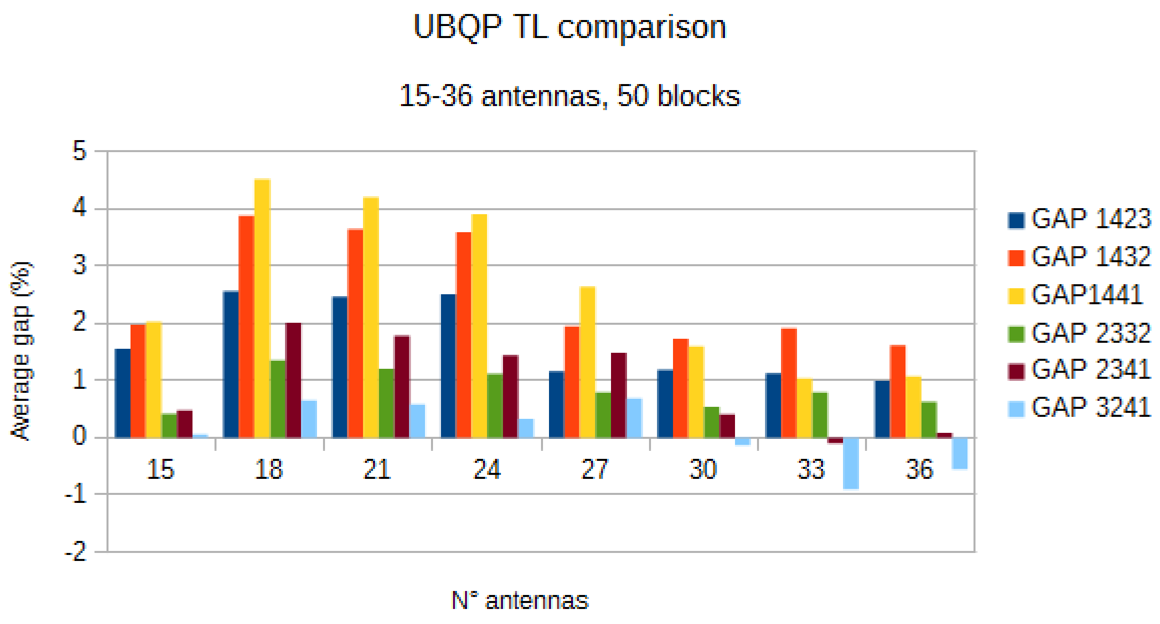


Figure C.1: TL's Gap comparison for UBQP algorithm: 15-36 antennas

C.1 Partition for UBQP

In the following figures we report the tuning of the two TLs. As we have no optimal reference, we use the *relative gap* between every partition.

We compute this gap with the following formula:

$$Gap_{i,j} = \frac{Val_i - Val_j}{Val_i} \tag{C.1}$$

where Val_i, Val_j is the value obtained by algorithm i, j.

If such a gap is **positive**, then algorithm j is better than algorithm i (as we are interested in minimizing the interference), and if it is **negative**, then algorithm i is better than algorithm j, because it means that $Val_i < Val_j$.

We have to keep this in mind when analysing the following charts.

For example, Gap_{14-23} is the gap between the UBQP algorithm ran with 1-4 partition and the one with 2-3 partition.

In that case the formula used is: $Gap_{14,23} = \dfrac{Val_{14} - Val_{23}}{Val_{14}}$.

Figures C.1 and C.2 show the gap comparison[1] for algorithm UBQP, referring to an average demand scenario and a bandwidth of 50 RBs.

From these charts we can characterize three zones:

[1]We don't report gaps for 3,6,9 and 12 antennas because they are 0. The effects of the partition occur from 15 antennas onwards.

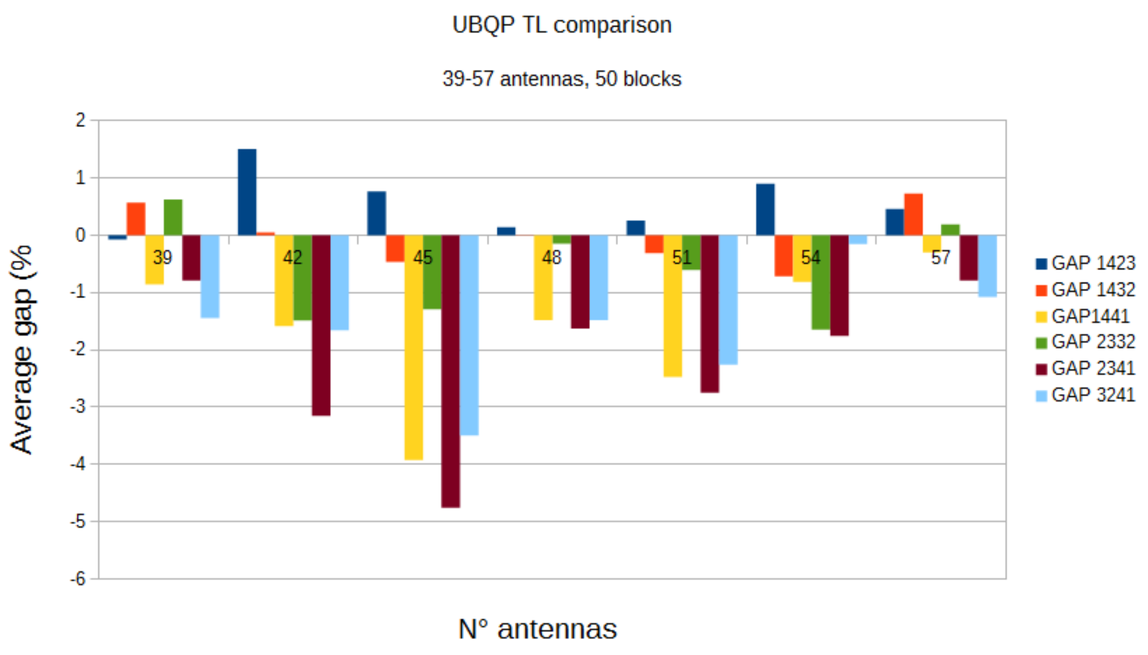


Figure C.2: TL's Gap comparison for UBQP algorithm: 39-57 antennas

3-27 antennas , in this zone is better to use the $4-1$ configuration. Indeed, all the gaps are relative, that is the $4-1$ partition is the best partition.

30-39 antennas , in this zone is better to use a $3-2$ configuration. Indeed, the only negative column is the $3-2$ one. This means that it is the best partition available.

42-57 antennas , in this zone is better to use a $2-3$ configuration. In fact, we can note that the gaps tend to be 0 in proximity of 57 antennas.

We can notice that with the growing of the number of antennas, the best partition tends to prefer to split its time for the branch-and-bound, because the problem becomes more difficult to solve. As said before, the gaps tend to 0 in proximity of 57 antennas. This is due to the complexity of the algorithm, that can't deal very well with a very great number of antennas.

C.2 Partition for UBQP flip

For this algorithm we can use the same considerations of the UBQP.

Figure **??** and **??** show the gap comparison for algorithm UBQP flip, referring to an average demand scenario and a bandwidth of 50 RBs.

We can characterize four zones:

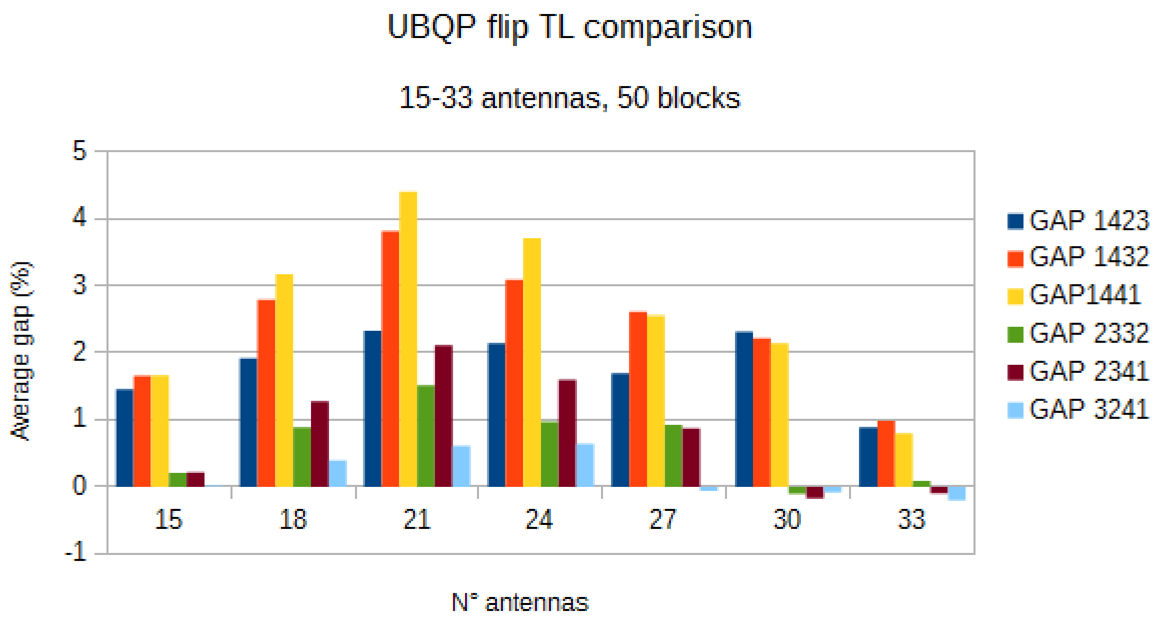


Figure C.3: TL's Gap comparison for UBQP flip algorithm: 15-36 antennas

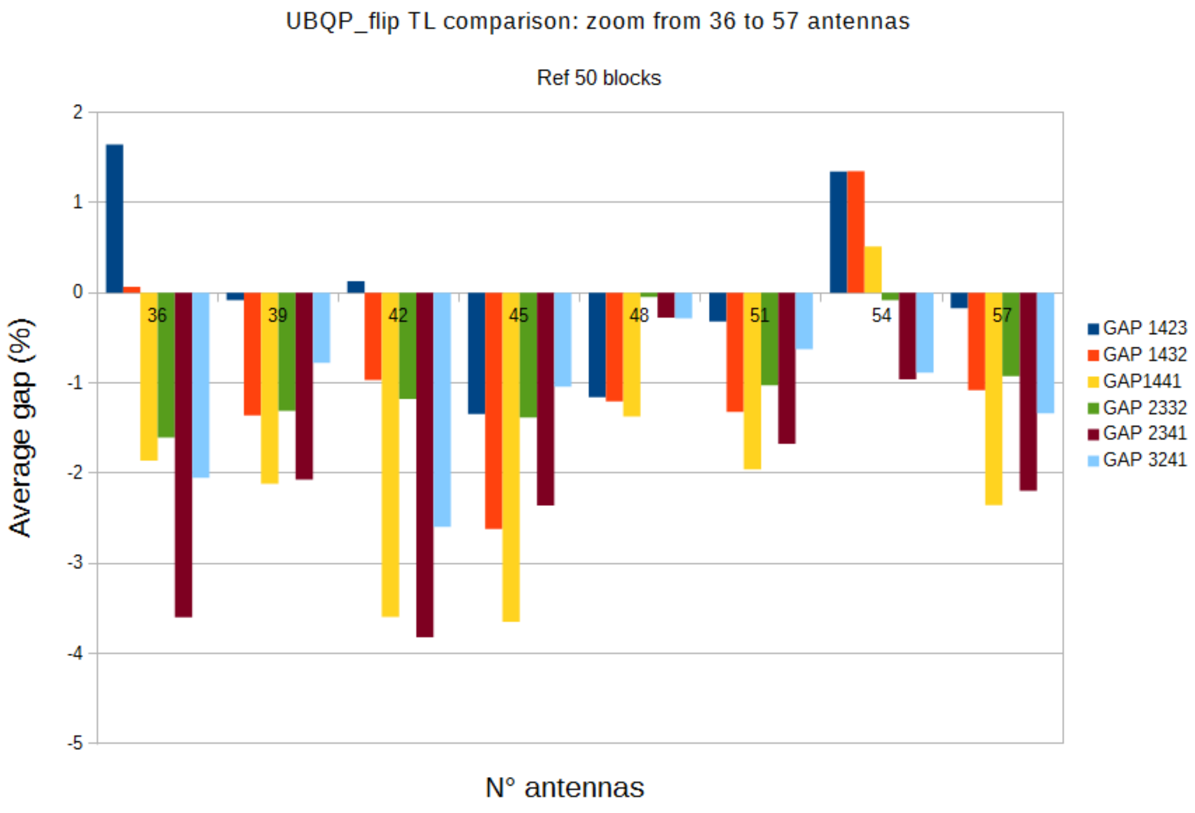


Figure C.4: TL's Gap comparison for UBQP flip algorithm: 39-57 antennas

3-24 antennas , in this zone is better to use the $4-1$ configuration.

27-33 antennas , in this zone is better to use a $3-2$ configuration.

36-42 antennas , in this zone is better to use a $2-3$ configuration.

45-57 antennas , in this zone is better to use a $1-4$ configuration.

We can observe a trend to use the time at disposal to the branch-and-bound with the growth of the number of antennas.

C.3 Partition and tuning of K for UBQP linzd

In this algorithm we have an additional parameter compared to the previous ones: K.

We have seen in chapter 7 that this parameter is related to the generation of the columns: the bigger the K, the bigger the number of columns generated.

In fact, there is no static relationship between K and the number of columns generated, due to many CPLEX parameters. The observable fact is that the number of columns generated can be considerably smaller than expected.

Recalling equation (B.1) we see how K is related to 2^N. As in large-scale coordination N can be 50 or similar, it's not manageable to give as parameter such K, so we have modified the code in order to make the theoretical number of columns manageable from the machine. When dealing with a number of antennas greater than 21 (e.g. 51), the k parameter has been reduced.

The tuning of this algorithm has been done focusing on certain K values and on certain values of N.

In particular, as high values of K don't return good results, we have focused on the following values: $1, 2, 3, 5, 10$.

The numbers of antennas we have studied are: $27, 36, 42, 48, 57$. These values have been chosen in order to have a sufficient resolution that allows us to characterize some zones.

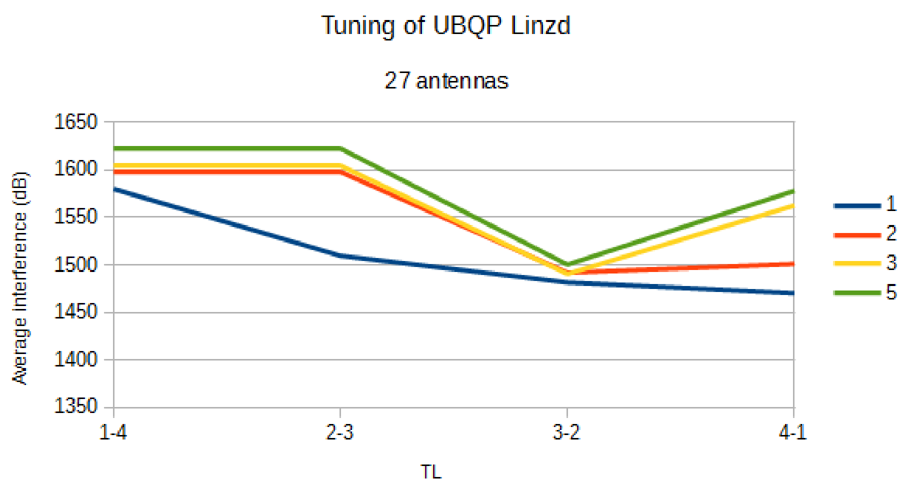
Even in this case the study has as reference the average demand scenario and 50 RBs at disposal from each antenna.

Figures C.5 and C.6 report the result of the study.

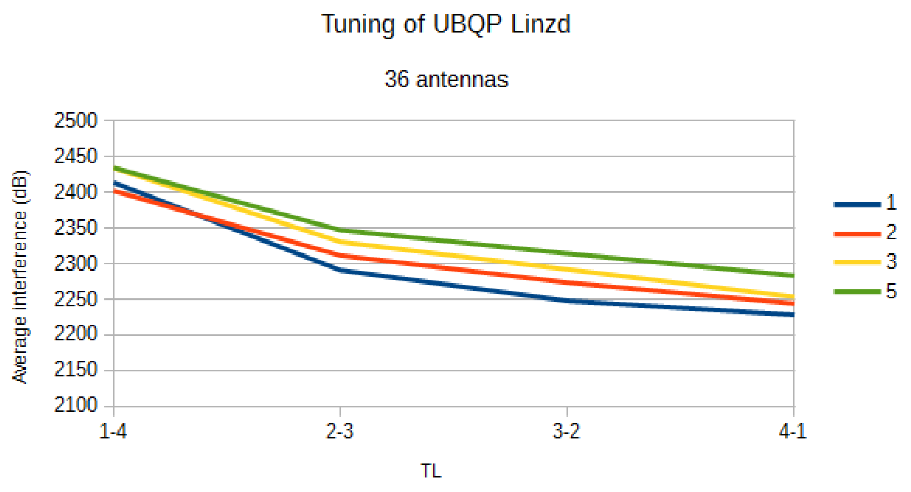
We observe that pratically the 1 configuration is the best, so we can tune $K = 1$ for every number of antenna. Furthermore, the choice of K becomes more irrelevant with the growing of N.

We can also characterize three zones:

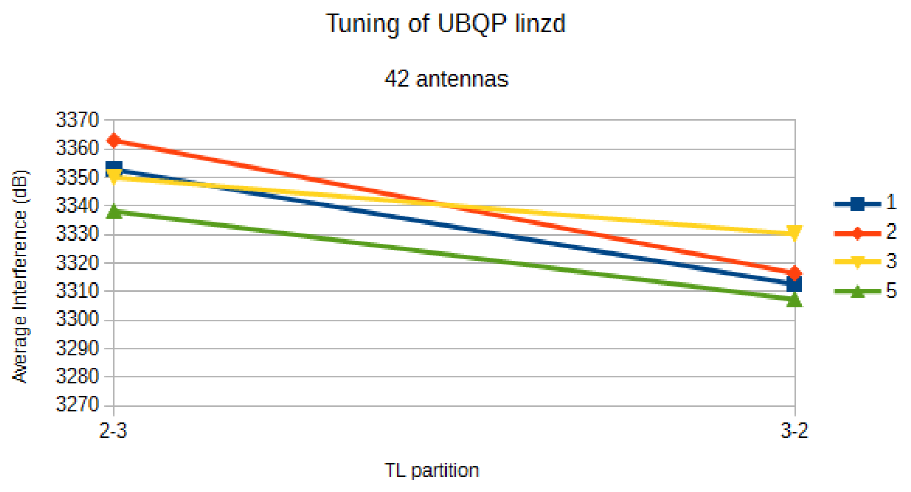
15-36 antennas in this zone is better to use the $4-1$ configuration.



(a) 27 antennas.

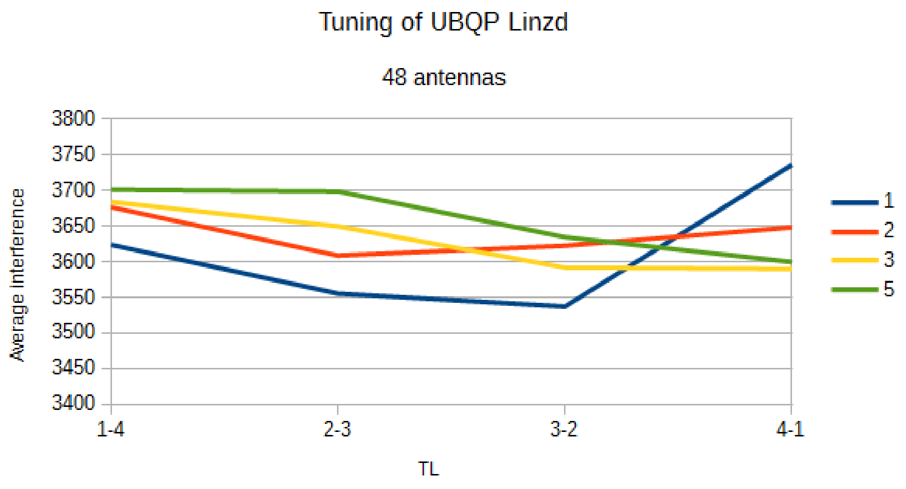


(b) 36 antennas.

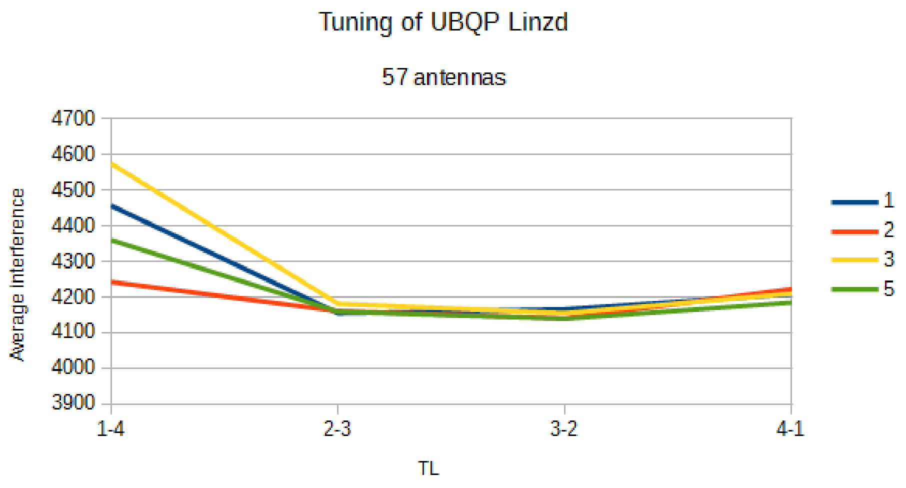


(c) 45 antennas.

Figure C.5: Tuning of UBQP linzd: 27-45 antennas.



(a) 48 antennas.



(b) 57 antennas

Figure C.6: Tuning of UBQP linzd: 48-57 antennas.

39-48 antennas in this zone is better to use a $3-2$ configuration.

51-57 antennas in this zone is better to use a $2-3$ configuration.

With the tuning of these algorithms, we have ran a test, whose results are presented in chapter 6.

Bibliography

[1] Hillier F., Lieberman G., 2001, *Introduction to operation research*, Mc Graw Hill, 11.

[2] http://www.3gpp.org/

[3] http://www.itu.int/net/pressoffice/press_releases/2010/48.aspx

[4] Motorola Inc.,*Long term evolution(LTE)*, Technical white paper, p. 2, 2007

[5] http://www.3gpp.org/technologies/keywords-acronyms/98-lte

[6] http://www.3gpp.org/technologies/keywords-acronyms/100-the-evolved-packet-core

[7] Carvalgo F., Magedanz T., *Quality of service in telecommunication networks*, Technical University of Berlin, Berlin, Germany.

[8] Huaizhou SHI, R. Venkatesha Prasad, Ertan Onur, I.G.M.M. Niemegeers, *Fairness in Wireless Networks - Issues, Measures and Challenges*, Delft University of Technology, Delft, Netherlands.

[9] S. Lambert et al., *Worldwide electricity consumption of communication networks*, Optics express, vol. 20, no. 26, Dec. 2012

[10] A. Virdis, G. Stea, D. Sabella, M. Caretti, *A framework for energy-efficient node activation in heterogeneous LTE networks.*

[11] www.di.unipi.it/optimize/Courses/ROIG/1617/Appunti1617.pdf

[12] B. Korte, J. Vygen, *Combinatorial Optimization*, Springer, 2000

[13] http://www.di.unipi.it/optimize/Courses/RO2IG/aa1617/Complexity_theory.pdf

[14] Pappalardo M., Passacantando M, *Lezioni di Ricerca Operativa*, Pisa University Press, 2010.

[15] Archana Bomnale, *A Survey on Inter-Cell Interference Reduction Techniques in LTE-A Heterogeneous Networks*, Electronics and Telecommunication Department, Mukesh Patel School of Technology Management and Engineering, NMIMS University, Mumbai, India.

[16] S.Indhumathi, R. Selvaraj, V.Elavarasi, *Minimizing interference by sharing spectrum in LTE-A using game theory*, International Journal of Modern Trends in Engineering and Research, 2349 - 9745.

[17] M.G. Shajan, P.V. Khushbu, *Minimizing Interference by Indoor-cell of 4G LTE Networks*, 2015 Fifth International Conference on Communication Systems and Network Technologies.

[18] G. Nardini, G. Stea, A. Virdis, D. Sabella, M. Caretti, *Practical large-scale coordinated scheduling in LTE-Advanced networks*, Springer Science+Business Media, New York, 2015.

[19] 3GPP, *Overview of 3GPP Release 5*, 3GPP TSG RAN #20 , Hämeenlinna, Finland, 4-6 June 2003

[20] Franz Rendl, Giovanni Rinaldi, and Angelika Wiegele. *Solving Max-Cut to optimality by intersecting semidefinite and polyhedral relaxations*. Math. Programming, 121(2):307, 2010.

[21] Sheldon M.Ross, *Introduction to Probability and Statistics for Engineers and Scientists*, 2009: 164-166

[22] IBM ILOG CPLEX Optimization Studio CPLEX User's Manual, Version 12 Release 6, 2014: 278-294.

[23] LTE-Advanced (3GPP Release 10 and beyond), 17 – 18 Dec. 2009, Beijing, China Nakamura, Takaharu / FUJITSU LIMITED. 3GPP TSG-RAN-WG4 Chairman

Acknowledgements

I would like to express my gratitude to my supervisor, Dr. Laura Galli for giving me the precious opportunity to deal with Operations Research in a concrete context. I would like also to thank my other supervisor, Prof. Giovanni Stea, that has introduced me into the world of computer science engineering.

Their guidance has been fundamental for allowing me to understand and appreciate the complex world of research.

My sincere thanks to Engineer Giovanni Nardini for the precious advices and support that he gave me, without which it would have been much more difficult to complete this work. He has been patient with me and he shared part of its great know-how.