4 Overview of radio-network optimization

Once a radio network is launched, an important part of its operation and maintenance deals with monitoring its quality characteristics. With the evolution of mobile communications, the complexity of network planning has grown along with its throughput capacity, thus making it practically impossible to plan modern radio networks with traditional methods. In this sense, an examination of coloured coverage maps in conjunction with some statistical analysis are no longer appropriate tools for troubleshooting a network. Moreover, since real-world radio networks are large and many of their configuration parameters are interdependent, an engineer is not able to cope with the level of complexity present in these systems. For this reason, the computer, along with specialized software, guides the engineer to the most appropriate configuration for the network. In the context of this thesis, this process is refered to as radio-network optimization.

Radio-network optimization may be divided into two fundamental phases: analysis and decision [110]. The analysis phase consists on the examination of the network performance, which mainly focuses on the definition and collection of Key-Performance Indicators (KPIs). KPIs are quantifiable measurements that reflect different network-quality factors. The second phase deals with the decision making, based on the analytical results collected in the previous phase, about changing a particular configuration or parameter setting. The process, a representation of which is depicted in Figure 4.1, is repeated until the achieved results are acceptable. Notice the similarity with the general, decision-making process that was presented in Chapter 1, Figure 1.1.

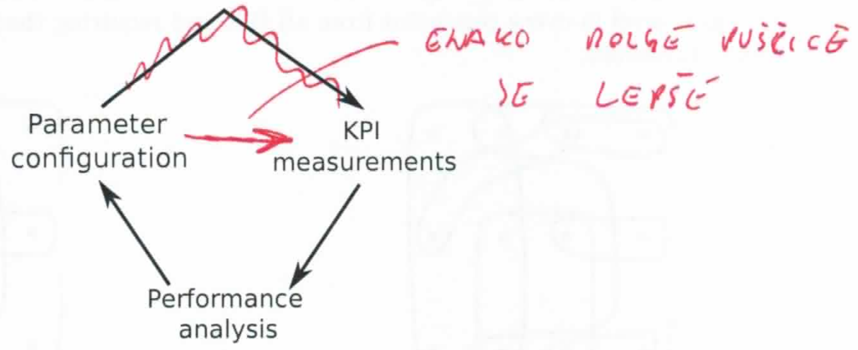

Figure 4.1: A typical optimization cycle for radio networks. This sequence is repeated until the achieved results are acceptable.

Since radio networks are increasingly more sophisticated, the need for optimization methods that are capable of copeing with greater complexity is far from declining. Indeed, several radio-network optimization problems were shown to be NP-hard, since the computational time grows non-polynomially with the problem size [7, 9, 68, 73, 97, 129, 147]. As described in [110], there are other reasons directly related with the growth of already deployed networks that also increase the need for optimization methods:

Network performance improvement more users receive service coverage with the same physical infrastructure, making parameter optimization the less expensive and only viable short-term approach.

Changes in user profile the introduction of new, faster services puts additional stress on the infrastructure, requiring additional optimization efforts.

Changes in the propagation conditions the allocation of different frequency bands for different systems, e.g., GSM, UMTS or LTE, requires the deployment of new BSs, the radio propagations of which behave differently, especially in urban areas.

Depending on the optimization problem being addressed, network operators define an optimization target that is represented by an objective function that maps possible configurations into a real value. Unfortunately, there is no universal objective function in the field of radio-network optimization [110]. However, it is possible to optimize for a different target at a time, such as service coverage, SHO balance or signal propagation. Particular optimization algorithms for solving these problems are presented in Chapters 7, 8 and 9, respectively. In all three cases, the introduced optimization approaches are performed "offline", meaning that the optimization software is not an active part of the target radio network. As the feedback information of each optimization target, the statistical data about the network functioning is used.

In the following, an overview of some well-known optimization problems for radio-mobile networks is given. Each section describes an optimization problem, and presents a short survey of recently proposed optimization methods for solving them.

4.1 Optimizing base-station locations

Some authors [74, 105] formulate the problem of locating BSs in terms of the minimum set-covering problem (see Figure 4.2). The set-covering problem is defined by considering the signal level in every test point from all BSs and requiring that at least one level is above a fixed threshold.

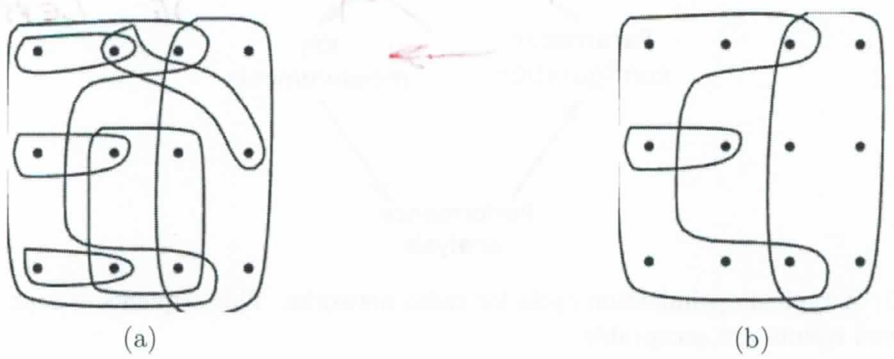

Figure 4.2: A graphical representation of the set-covering problem: (a) the problem input and (b) the solution.

A different formulation considers the BS-site location problem as a p-median problem [177], in which the BS location is the only decision variable considered. To each of the candidate solutions, an installation cost is also associated. The p-median problem constitutes seeking p different locations each time, regardless of how distant the sites are. The problem involves selecting one installation-candidate site from each region such that the traffic capacity and the size of the covered area of the network are maximized with the lowest installation cost.

4.1.1 Related work

Aydin et al. [13] proposed a solution to the *p*-median problem based on three metaheuristic algorithms: a genetic algorithm, SA, and tabu search. Their experimental study focused on the performance comparison between the three approaches.

In [177], the authors also used a simplified *p*-median problem as the model. They presented the results of extensive simulations to compare the performance of three different metaheuristic algorithms.

A solution to the set covering problem is proposed by Hao et al. [74]. An implementation of SA was developed to solve the formulated combinatorial problem. The presented results showed the feasibility of the proposed approach.

Mathar and Niessen [105] proposed a hibrid method that combines a linear-programming approach with SA. The SA algorithm substituted linear programming whenever an exact solution was out of reach because of the complexity of the problem instance.

Amaldi et al. [7] presented a discussion about the computational results of two different heuristics: greedy search and tabu search. The problem formulation was based on a set of candidate sites where the BSs could be installed, an estimation of the traffic distribution and a propagation description of the area to be covered. Some years later, the same authors [9] extended the problem formulation by adding the BS configuration and the hardware characteristics as additional constraints of the integer program. In both works, they proposed a mixed, integer-programming model to maximize the trade-off between covered area and installation costs.

Finally, Whitaker et al. [128] focused on providing a required service-coverage target at the lowest possible financial cost. Their work compared the performance of four different algorithms, namely SEAMO, SPEA2, NSGA-II and PESA, all targeting multiobjective optimization targets.

THIS IS NOT MULTIOBJECTIVE ?
DAJ POGLEJ PRAVO FORMULACIJO "NALOGE" MORA BITI VEČKRITERIJSKA

4.2 Optimizing antenna parameters

Since an antenna shapes the emitted energy, its configuration plays an important role in the coverage and interference of a radio network. The two most important parameters in this sense are the azimuth angle and the tilt (or elevation angle) of the antenna. The antenna azimuth, an example pattern of which is depicted in Figure 4.3, is the horizontal direction of the main antenna beam. The antenna tilt (see Figure 4.4) is defined as the angle of the main antenna beam relative to the horizontal plane. Both of these parameters have a great influence on network quality, although the antenna tilt usually requires less effort to implement, since most modern radio networks already support remote electrical tilt [12]. The adjustment of these two parameters optimize some important aspects of the network, e.g.:

- the path loss between the BS and the UE,
- the interference between neighboring cells, which leads to an overall capacity increase of the network.

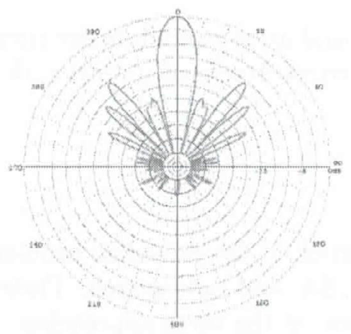

Figure 4.3: An example of an antenna-azimuth pattern.

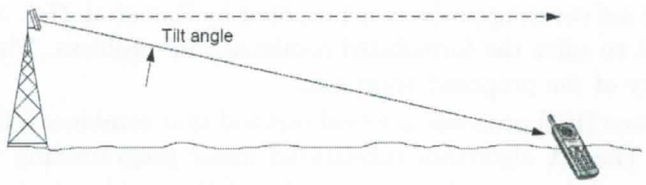

Figure 4.4: The antenna tilt, showing its angle with the horizontal plane.

4.2.1 Related work

One of the approaches proposed by Gerdenitsch et al. [59] involves an "ad-hoc" strategy for adjusting antenna azimuth and downtilt by analyzing the structure of the network. The objective of this optimization was to improve the number of served users in the target area.

Siomina and Yuan [144] proposed an approach for automated optimization of antenna azimuth and tilt, including support for both mechanical and electrical tilt. The implementation introduced a SA-based algorithm that searches the solution space of feasible antenna configurations. The goal of the optimization was to improve power sharing among different cell channels and ultimately improve the throughput of the network.

In [67], the authors presented a compound optimization method containing two loops: the inner one and the outer one. The inner loop concentrated on frequency planning while the outer loop focused on finding the optimal settings of antenna azimuth and tilt for the current solution delivered by the inner loop. This approach is interesting because of its flexibility, e.g., the inner loop could be replaced with some other optimization objective, like service coverage.

In [48], the authors proposed an autonomous optimization approach for the antenna tilts. Based on the gradient-ascent method, the presented heuristic showed the fast convergence needed for an online-optimization method to be effective.

Combining the power setting of the pilot signal with the antenna configuration is also a common practice. For example, Siomina et al. [141] present an optimization approach for maximizing the service coverage that combines both antenna parameters with the power of the pilot signal. Their SA-based algorithm searches the solution space of possible configurations in order to improve the performance of the target radio network by reducing the total interference. The simulation results show the algorithm is capable of tackling large network instances.

Two optimization algorithms for finding an optimal setting of antenna tilt and pilot-signal power were also introduced in the previously cited work by Gerdenitsch et al. [59].

The first algorithm is based on a rule-based approach, while the second one extends it by incorporating SA. The evaluation of both techniques showed that the second algorithm achieves better solutions.

In a different work, Gerdenitsch et al. [60] proposed a genetic algorithm to tackle the optimization of antenna tilt and pilot-signal power, the goal of which is to increase network capacity. The implementation involves a deterministic fitness selection scheme, a problem-specific recombination operator and an improved mutation operator. After the initial identification of the best promising individuals, a local-optimization technique is applied to improve their fitness.

4.3 Optimizing coverage

Coverage is arguably the most common optimization objective considered in radio-network optimization. A general objective function for coverage optimization can be defined as follows:

$$f_{\text{cov}} = \frac{A_{\text{covered}}}{A_{\text{total}}},$$

where A_{covered} represents the area covered by the network and A_{total} represents the total area under optimization. Thus, the expression f_{cov} represents the proportion of the total area that is actually under network coverage, the value of which ranges from 0 (no coverage) to 1 (total coverage).

The area under optimization is usually divided into squares (or pixels) of a certain size, creating a Regular Square Grid (RSG) of a certain resolution. A pixel is considered covered if a given QoS measure is above a defined threshold. It is also common to use a binary function, e.g., $cov(x, y)$, to check the coverage of a given pixel. The function returns 1 if the pixel located at (x, y) is covered, and 0 otherwise.

4.3.1 Related work

Siomina and Yuan [148] considered the problem of minimizing the pilot-signal power subject to a coverage constraint. Their approach consists of an iterative mathematical program, based on a linear-integer formulation of the problem. The simulation results showed the trade-off between service coverage and power consumption for different test networks.

Almaldi et al. [9] investigated several mathematical-programming models for supporting decisions on where to install new BSs and how to select their configuration in order to find a trade-off between coverage and costs. The overall model takes into account signal-quality constraints, in both uplink and downlink directions, as well as the pilot-signal power.

Connecting UMTS and LTE-network optimization from the coverage point of view, Sallent et al. [133] presented a framework to automatically identify a cell with sub-optimal coverage. The framework input is a collection of KPIs, including usage statistics and UE field measurements. The authors used the experimental results to provide a projection for the LTE self-optimization functionality.

In [162], the authors introduced a sparse-sampling algorithm to optimize the coverage of LTE networks by means of antenna tilt. By applying a reinforcement-learning technique, their approach optimizes coverage without prior knowledge about the target network.

Parkinnen et al. [167] presented a gradient-descent method to minimize an objective function that combines some KPIs with the service coverage. The power of the pilot signal of the cells is periodically updated in order to improve the afore-mentioned function.

4.4 Discussion

Regarding the optimization methods presented in the previous sections, three distinctive groups emerge: genetic algorithms, linear programming and other search methods.

Genetic algorithms These algorithms work on a population of solutions, allowing a more comprehensive search for optimal solutions. As a direct consequence, an increase in running time is commonly observed. The implementation effort of genetic algorithms is to some degree higher than for simpler search methods, e.g., local search, but their inherent structure makes parallel implementations rather simple. Additionally, genetic algorithms are less likely of being trapped in local optima, since they are a population-based metaheuristic.

Linear programming Linear-integer or mixed-integer programming are widely used in different optimization areas and there are many good software packages to solve such problems. Consequently, if a problem can be modeled as a continuous linear problem, there is usually no difficulty in finding optimality. In the context of this survey, linear programming has proven useful for BS-location optimization in early network planning stages.

Other search methods Other search methods, e.g., local search, SA, tabu search or gradient descent, usually represent a compromise between running time and quality of results. Their effectiveness relies on evaluating a great number of alternative solutions. The number of parameters taken into account, as well as the evaluation precision, directly influence their running time. These methods don't excel in full simulation scenarios. Moreover, some of them are easily trapped in local minima.

4.5 Summary

The variety of optimization problems described in the previous sections differ in many aspect, like implementation, running time and solution quality. Picking the right method for a given situation depends on the optimization task and the desired results. Since computational time is usually an important restriction, simpler and faster methods may be preferable.

Beside the convenience of a literature overview as presented here, it is very important to develop a feeling for the properties, advantages and drawbacks of the respective methods. Moreover, the recommendations of experts regarding the interpretation of the solutions and the feedback from everyday network operation are an essential input for establishing high-quality optimization methods.

Additionally, as noted for some of the cited works, hybrid methods, i.e., approaches that combine different optimization and/or search techniques, usually give competitive solutions to complex optimization problems. Therefore, a simple optimization algorithm can quickly find a subset of reasonable solutions, whereas a more complex method could be applied afterwards, to refine the search. Sometimes it may also be useful to apply a local search method at the end to find better solutions in the proximity of a current one.

5 Principles of GPU programming

GENERAL PURPOSE ?!!

During the past few years, the computing industry has been delivering extra processing power in the form of parallel computing, i.e., more cores instead of higher-frequency rates. Concurrently to this situation, new hardware architectures have appeared. Among them is the programmable Graphics Processing Unit (GPU), the computing capacity of which is already showing a faster progress compared to CPUs. The peak performance of the latest GPU is several times greater than the latest CPU, but even more important is the trend [40].

The GPU market was almost exclusively driven by the games industry. This fact makes this commodity hardware cheaper than other alternatives for High-Performance Computing (HPC), e.g., the classic 'mainframe' servers. Moreover, the performance-per-watt of GPUs is an extra benefit over their raw performance [40].

In February 2007, the first real opportunity for using GPUs for general programming and scientific computing came about with the release of the Compute Unified Device Architecture (CUDA).{ }.

In the following, the focus on the CUDA architecture. In particular, similar hardware produced by other vendors, e.g., AMD and Intel, as well as other programming frameworks like OpenCL{ } are based on the same technology paradigm of massively-parallel processors. For this reason, the principles presented in this section can be applied to current GPU technologies, in a vendor-independent manner.

5.1 CUDA

A GPU is effectively a large set of processor cores with the ability to directly address into a global memory. This makes it easier for developers to implement data-parallel kernels. The CUDA programming model [117] was created for developing applications for the GPU platform. A system within this model consists of a host, i.e., a traditional CPU, and one or more compute devices that are massively data-parallel coprocessors, i.e., a GPU. The host code transfers data to and from the global memory of the GPU using CUDA-function calls. Each processor of a CUDA device supports the Single-Program Multiple-Data (SPMD) model [11], in which all concurrent threads are based on the same code, although they may not follow exactly the same path of execution.

A CUDA program consists of multiple parts that are executed on either the CPU or the GPU. For this reason, the CUDA programming model may be viewed as an environment to isolate program parts that are rich in data parallelism and thus should be executed on the GPU. The parallel parts of a CUDA program are implemented as kernels, i.e., functions written in the C language that describe the work of a single thread of execution. Several restrictions apply on kernel functions: there must be no recursion, no static variable declarations, and a non-variable number of arguments [131]. At run time, the kernel invocation is typically done on thousands of threads, which are organized within developer-defined groups called thread blocks. The threads can share data and synchronize their execution only within the block they belong to. Thread blocks are, in turn, organized in a higher-hierarchy level called grid. All threads within a given grid execute the same kernel function.

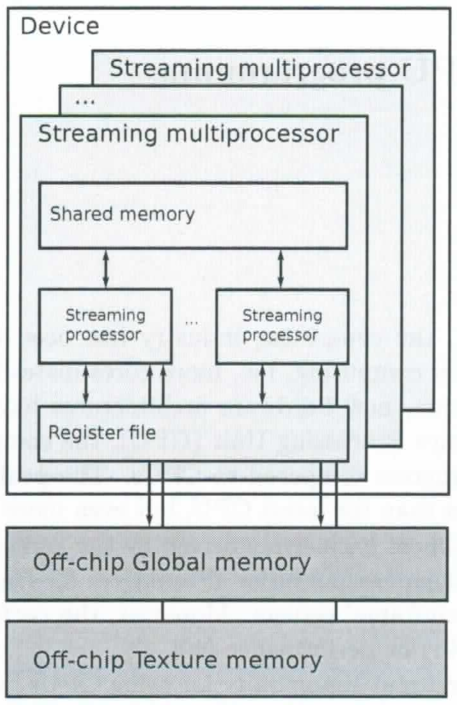

Figure 5.1: A simplified diagram of the architecture of a modern GPU. Adapted from [131].

VEDNO?!? ALI JE TO VARIABILNO

During execution, continous sections of threads within a block are grouped into warps of 32 parallel threads, which represent the multi-threading scheduling unit [131]. A Streaming Multiprocessor (SM) executes one instruction at a time for all the 32 threads in the warp. Consequently, if a thread block is not evenly divisible by the warp size, any remaining thread slots are wasted. Another undesirable situation appears when threads in a warp take different control paths, since the execution performance of the entire warp is reduced.

KAJ JE TO! LAHKO BI GA ŽE PREJ VPELJAL!!!

The architecture of a typical GPU is depicted in Figure 5.1. It consists of n SMs, each containing m Streaming Processors (SPs). Each SP executes a single instruction of a thread in a Single-Instruction, Multiple-Thread (SIMT) manner [40]. The registers of each SM are dynamically partitioned among the threads running on it.

Table 5.1 lists the different types of memory on a GPU, along with some of their properties. Specifically, the location of the memory, its hit-latency in terms of clock cycles, whether it is read-only or not, and the program scope it may be accessed from. Due to the different memory system a GPU has, the actual throughput an application can achieve depends on issues related to the access to memory, in particular the slow accesses to global memory from the GPU chip, and the use of shared memory in the SPs to mitigate the high-latency effects [40]. Since variables in the source code can be declared to reside in global, shared, local, or constant memory, a programmer has the means to organize the code in such a way that the application throughput can be maximized.

Clearly, there are hard limits to the memories, threads, and total bandwidth available to an application running on a GPU. Managing these limits ~~is~~ ARE critical when optimizing applications, but applying strategies for avoiding one limit can easily cause other limits to be hit. Additionally, managing the behavior of threads so that those in the same warp follow the same control paths and load contiguous values from global memory can improve the execution performance [131].

A detailed discussion of the CUDA programming model can be found in [51].

Table 5.1: Properties of different memory types on a GeFor~~g~~ce 8800 GPU. Adapted from [131].

Memory	Location	Latency	Read-only	Program scope
Global	off-chip	200-300 cycles	no	global
Texture	on-chip cache	~100 cycles	yes	global
Shared	on-chip	~register ?!!	1 cycle no	function
Register	on-chip	~1 cycle	no	function

Table 5.2: Naming-convention translation between OpenCL and CUDA. Adapted from [90].

OpenCL	**CUDA**
Grid	Grid
Work group	Block
Work item	Thread
__kernel	__global__
__global	__device__
__local	__shared__
__private	__local__
image*n*d_t	texture<type,*n*,...>
barrier(L|M|F)	__syncthreads( )
get_local_id(0|1|2)	threadIdx.x|y|z
get_group_id(0|1|2)	blockIdx.x|y|z
get_global_id(0|1|2)	*(not implemented)*

5.2 OpenCL

The Open computing language (OpenCL) [151] is an open parallel computing API designed to enable GPUs and other co-processors to work together with the CPU, providing additional computing power. As a standard, OpenCL 1.0 was released in 2008, by The Khronos Group, an independent standards consortium [109].

The main advantage of OpenCL over CUDA relies on the fact that its source code can be compiled to run on a variety of hardware, including multicore CPUs and GPUs from different vendors. This provides a complete framework that is capable of exploiting the parallel features of different hardware without the need of changing the implementation. Moreover, being an open standard, its users are not tight to the decision of only one vendor. As it was mentioned before, the details described in the previous sections may be equally applied on CUDA and OpenCL.

One unfortunate consequence of the vendor variety is that CUDA and OpenCL have each introduced its own naming conventions. For the sake of consistency, in Table 5.2, a short "translation dictionary" between both platforms is presented.

In the remaining of this thesis, the naming convention introduced by the CUDA platform is used.

5.3 Summary

This chapter gave an overview of the basic concepts, the potential and the limitations of the GPUs when used as parallel procesors for general programming. In particular, emphasis has been given to the details that differenciate this platform from the traditional serial architecture on CPUs. In this sense, it is important to recognize which applications can

NI JASNO!

KAJ JE ZATE TRADICIONALNA ZAPOREDNA ARHITEKTURA ?!?

benefit from using a GPU and which not, also taking into consideration the effort required at the implementation time.