Computing and Informatics, Vol. 33, 2014, 1025–1046

A GPU-BASED PARALLEL-AGENT OPTIMIZATION APPROACH FOR THE SERVICE COVERAGE PROBLEM IN UMTS NETWORKS

Lucas Benedičič, Mitja Štular

Telekom Slovenije, d.d.
Cigaletova 15
SI-1000 Ljubljana, Slovenia
e-mail: {lucas.benedicic, mitja.stular}@telekom.si

Peter Korošec

Computer Systems Department
Jožef Stefan Institute
Jamova cesta 39
SI-1000 Ljubljana, Slovenia
e-mail: peter.korosec@ijs.si

Abstract. In the context of coverage planning and control, the power of the common pilot channel signal determines the coverage area of a network cell. It also impacts the network capacity and thus the quality of service. We consider the problem of minimizing the total amount of pilot power subject to a full coverage constraint. Our optimization approach, based on parallel autonomous agents, gives very good solutions within an acceptable amount of time. The parallel implementation takes full advantage of GPU hardware in order to achieve impressive speed-up. We report the results of our experiments for three UMTS networks of different sizes based on a real network currently deployed in Slovenia.

Keywords: UMTS, mobile network, coverage, optimization, pilot power, parallel, agents, GPU

1 INTRODUCTION

The coverage problem in third-generation (3G) mobile networks has received a great deal of attention in the last years. Its complexity demands the confluence of different skills in areas such as propagation of radio signals, telecommunications, and information systems, among others.

One of the reasons behind the problem complexity relies in the conflicting measures generally used to compare different proposed solutions. These aspects include network capacity, quality of service, service coverage, and recently, issues related with human exposure to electromagnetic fields generated by base station antennas [6]. Public opinion has been extremely sensitive regarding this issue, and thus many countries have already imposed safety standards to limit the electromagnetic field levels.

It is clear that, even almost 10 years after the launch of the first commercial UMTS network, service coverage planning remains a key problem that all mobile operators have to deal with. Its intricacy arises from the wide range of different combinations of configuration parameters and their evaluation-time complexity. One crucial parameter, which is mainly subject of adjustment, is the transmit power of the common pilot channel (CPICH). The CPICH transmit power is common to many different planning and optimization problems in UMTS networks [13].

The CPICH transmits in the downlink of a UMTS cell system. The transmit power is usually between 5 % and 10 % of the total power available at the base station [9]. The capacity of a cell is limited by the amount of available power at the base station and the interference level at the mobile terminal. The coverage area of any cell is controlled by changing its pilot power, which consequently modifies the service area of the network.

From the network perspective, lowering pilot power usage leaves more power available for increased network capacity. This is especially important if the traffic and other channels are configured relative to CPICH [9]. Moreover, as the demand for mobile internet access and data services increases [5], so does the pressure on existing network infrastructure, making parameter optimization a viable short-term solution [13].

There are different approaches in the literature tackling the coverage problem [13, 18]. Some of them even claim to achieve near-optimal solutions [19]. As a matter of fact, such formulations have proven useful only for small network instances and often fail when challenged with real-world networks.

The idea of using autonomous agents for optimization is not new [4]. It has proven to be a solid optimization approach for solving different types of problems, not only within the area of mobile networks [3, 6], but also in other fields [23, 25]. Nevertheless, we have found no reference in the literature of any similar optimization method for solving the service coverage problem in mobile networks. Moreover, this is, to the best of our knowledge, the first work to experiment with optimization of a real UMTS network on a fully-enabled GPU environment.

Our optimization approach is based on a state-of-the-art mathematical model, that has been previously used to solve a comparable problem [2]. We tackle the problem of computational-time complexity when dealing with big problem instances by implementing a parallel version of our agent-based algorithm entirely on GPU. This approach minimizes overhead when deploying a larger number of agents working in parallel over the service area, limited only by the amount of available memory.

We have tested our algorithm on subsets of a real UMTS network deployed in Slovenia by Telekom Slovenije, d.d. The results show that the solutions found, and more importantly their quality, are greatly improved when compared to other common planning techniques.

We begin our discussion by introducing previous work and a description of the coverage problem, where we formally introduce some of its key elements. We then introduce the parallel-agent approach in detail, as well as the strategies used for result comparison. Having described the parallel-agent approach, we move on to discuss the GPU implementation. This is followed by the simulations and experimentation, performed on three sub-networks of a real mobile network deployed in Slovenia. We conclude with an overview of the achieved results, from the optimization as well as the implementation points of view, and discuss future research directions.

2 PREVIOUS WORK

In [19], Siomina and Yuan considered the problem of minimizing the total amount of pilot power subject to a full coverage constraint. They tackled the problem with an iterative linear programming approach, reporting very good results for some small-sized test networks. The authors also noted that bigger problem instances could not be solved because of hardware constraints on the target platform.

In a different work [1], we tackled the full-coverage problem of the service area under optimization. Reported experimentation on the same problem instances as in [19] showed improved quality at the cost of longer running time. Moreover, our approach was not limited by any hardware restrictions, since it successfully tackled bigger problem instances, finding high quality solutions even for large networks. Such solutions were found in a reduced amount of time by increasing the number of agents deployed during optimization without compromising solution quality.

The algorithm in [1] is the basis for the optimization algorithm presented in this paper, with some essential improvements, namely:

- the implementation is no longer completely CPU-based,
- it does not use a black-box coverage evaluator and
- the algorithm presented here contains improvements in the agent's behavior during optimization, including the introduction of the so-called "special" agents and fine-tuned step sets.

3 PROBLEM DESCRIPTION

In the problem of optimization of pilot powers for service coverage, the objective is to find a set of pilot power settings for all cells in the network, such that the total pilot power used is minimized, and a given service coverage criterion is fulfilled. We consider the pilot power minimization problem subject to a full coverage constraint of the service area.

Because the mathematical model of the problem is not of primary interest here, we will just outline it, so that all problem elements are formally defined and represented. For additional information regarding mathematical models of comparable problems see [13].

3.1 Problem Elements

We start by considering a UMTS network of m cells and use C to denote the set of cells, i.e. $C = \{1, \ldots, m\}$. A pixel grid of a given resolution represents the service area for which the signal propagation predictions are known. Let n denote the total number of pixels in the service area and let S denote the pixel set, i.e. $S = \{1, \ldots, n\}$. We also denote $att_{c,s}$, $0 \leq att_{c,s} \leq 1$, as the attenuation factor between a cell c and a pixel s, which is calculated by performing signal propagation predictions for every pair of $c \in C$ and $s \in S$.

For every $c \in C$, we define p_c^T as the total transmission power available in cell c. This power is shared among all channels in the cell (i.e. CPICH, other common channels, and dedicated traffic channels). We define p_c as the amount of power allocated to the pilot signal of cell c, where p_c may adopt any value from a finite set of possible pilot power levels, $P_c = \{p_c^1, p_c^2, \ldots, p_c^T\}$. Consequently, the received pilot power of cell c in pixel s is $p_c att_{c,s}$.

Considering the full coverage constraint, each pixel in the service area should have at least one cell covering it. We assume that a pixel s is under coverage of a cell c if its signal-to-interference ratio, SIR, at pixel s is not lower than a given threshold, γ_c, i.e.

$$SIR(c, s) = \frac{p_c att_{c,s}}{\sum_{i \in C} p_i^T att_{i,s} + \tau_0} \geq \gamma_c, \tag{1}$$

where τ_0 is the thermal noise.

Since we are solving the coverage problem for an interference-limited radio network, as UMTS networks are, we are assuming that all cells in the network operate at full power (note p_i^T in (1)). In terms of the level of interference present in the network, this assumption represents the worst case scenario, whereas in terms of the service coverage solution, it represents a lower bound [16]. The same assumption has also been used in [2, 19].

Specifically, the interference levels are significantly lower during normal operation conditions, i.e. when an expectedly normal traffic load is served by the network. In such case, the coverage of every cell increases in comparison to heavy traffic conditions. The principle behind this behavior is called cell breathing [9]. In a nutshell,

cell breathing allows cells with high user traffic to offload some of it to neighboring cells. They do so by adapting the geographic size of their service area. Consequently, highly loaded cells decrease in coverage size, while neighboring cells increase their covered service area to compensate.

The optimization problem corresponds to finding the pilot power levels p_c, for all cells $c \in C$, such that coverage of at least b pixels is guaranteed, while the total amount of pilot power used is minimized. Since we are considering full coverage, we denote $b = n$.

3.2 Problem Complexity

It has been proven that the problem of pilot power optimization for full coverage of the service area is NP-hard, since it can be reduced to the set covering problems [24]. Consequently, as long as $P \neq NP$, it is unfeasible that a polynomial-time algorithm exists, which is able to find an exact solution to this problem.

3.3 Optimization Objective and Constraints

The optimization objective is defined as follows

$$P^* = \min \sum_{c \in C} p_c; \tag{2}$$

subject to

$$\frac{\sum_{s \in S} cov(s)}{b} = 1, \tag{3}$$

where

$$cov(s) = \begin{cases} 1, & \text{if and only if } \exists c \mid SIR(c, s) \geq \gamma_c \\ 0, & \text{otherwise} \end{cases}. \tag{4}$$

The definition of (4) provides us with a simple way of asserting the coverage of a given pixel, s. It follows that if the pilot signal of at least one cell c satisfies the imposed SIR threshold γ_c the pixel is covered and hence $cov(s) = 1$.

4 OPTIMIZATION APPROACHES

Since the problem instances we will be analyzing are part of a real mobile network deployed in Slovenia by Telekom Slovenije, d.d., there are no references in the literature of other optimization techniques dealing with exactly the same data set. For this reason, we will introduce two different strategies for setting the pilot power, that shall enable us to compare the experimentation results. The first strategy is attenuation-based pilot power, presented in [18], in which a pixel of the service area is always covered by the cell with the highest attenuation-factor value. The second strategy is our parallel-agent approach, based on ideas inspired by two-dimensional

cellular automata [15] and metaheuristics [21]. Similar criteria for result comparison have also been used in [19]. A detailed description is given in Section 4.2.

4.1 Attenuation-Based Pilot Power

The first heuristic for setting the pilot power of all cells in the network is known as attenuation-based, since it relies on the attenuation factor $att_{c,s}$. Following this heuristic, a pixel of the service area s is always covered by the cell exhibiting the maximum $att_{c,s}$. We use $c(s)$ to denote the cell that has, among all cells, the maximum attenuation factor over the pixel s as

$$c(s) = \max_{c \in C} att_{c,s}. \tag{5}$$

Assuming the maximum available power p_c^T is the same for all the cells in the network, it follows that for any pixel s, the cell with the maximum attenuation factor requires the minimum power level. Consequently, we introduce an equivalent definition for $c(s)$ as

$$c(s) = \min_{c \in C} p_{c,s}, \tag{6}$$

where $p_{c,s}$ is the pilot power of cell c such that the pixel s is covered.

Picking the cells conforming to (6) and setting the pilot powers accordingly, we achieve full coverage of the service area, where the pilot power of cell c is as follows:

$$p_c^{\mathrm{Att}} = \max_{s \in S : c(s) = c} p_{c,s}. \tag{7}$$

In this case, the solution exhibits a total pilot power of

$$P^{\mathrm{Att}} = \sum_{c \in C} p_c^{\mathrm{Att}}. \tag{8}$$

The last part of the procedure consists of finding, for every pixel, the maximum attenuation factor among all cells, i.e. $att_{c(s),s}$. This value is then used to sort the pixels in descending order. The solution is thus established by the first b pixels of the resulting sorted sequence.

4.2 Parallel-Agent Approach

In the parallel-agent approach, a set of autonomous worker agents explore the geographic area, targeted to be under mobile network coverage, in order to optimize the pilot power consumption of the network. Each agent randomly moves over the service area as it dictates different changes to the pilot power of the cells. An evaluator performs radio propagation predictions based on each agent's proposed change.

The search process during optimization is strictly random. However, several physical properties that are exclusive to the problem being solved are being exploited

during exploration of the search space. Additionally, whenever the current solution breaks any of the given constraints, the optimization process is guided back to the space of valid solutions, providing a mechanism for improving exploration and escaping from local optima.

Because of the independent nature in agent's behavior, a parallel implementation is fairly straightforward to achieve. The first hardware restriction we have to overcome is the amount of memory on the GPU device. Consequently, careful memory utilization and organization are critical to successfully accommodate all involved problem elements on the GPU. Figure 1 gives an overview of the optimization system architecture. Within this GPU-only architecture, agents work in a parallel and autonomous manner, while the evaluator reacts to agent changes.

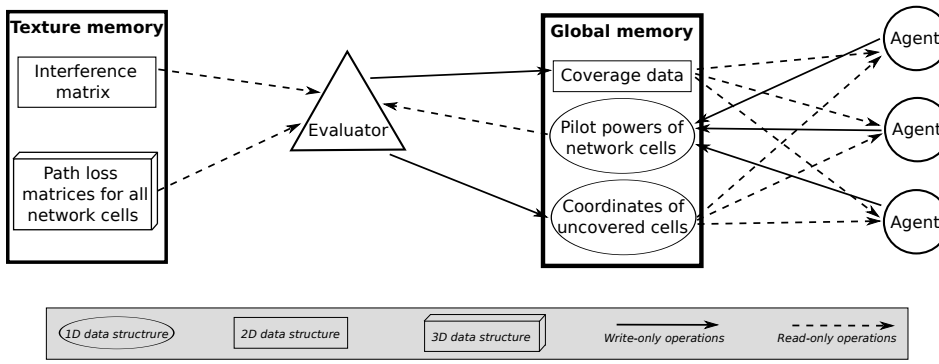


Figure 1. Architecture of the optimization system on GPU

4.2.1 The Agents

The agents apply the pilot power changes based exclusively on local information. Each of them encapsulates a set of steps that is consistently applied as it randomly moves through the service area of the network. Whenever an agent arrives at a pixel s, it identifies the set of cells covering the current pixel, namely

$$B(s) = \{c \in C \mid SIR(c, s) \geq \gamma_c\}. \tag{9}$$

The step set applied from this point on directly depends on the cardinality of $B(s)$, while the agent's movement over the service area is determined by the cardinality of set U, $U \subset S$, which is defined as

$$U = \{s \in S \mid \forall c \in C : SIR(c, s) < \gamma_c\}. \tag{10}$$

The agent's behavior is dictated by the pseudo-code shown in Table 1. An example diagram is depicted in Figure 2. Steps 1 to 4 are responsible for guiding the agent's movement. The coordinates are selected randomly from two sets. The first,

S, is the set of all pixels in the service area. The other one, U, is the set of pixels that are currently not being covered by the network. Only "special" agents may select pixel coordinates of the set U. It follows that an agent's movement depends on its "specialty" and the number of pixels not covered by the current solution. During steps 5 to 8, the agent applies step sets SS_0 and SS_1 based on the number of cells in $B(s)$.

Step	
	repeat
1	**if** Special agent() **and** $\|U\| > 0$ **then**
2	s = random element from U
	else
3	s = random element from S
	end if
4	move to s
5	**if** $\|B(s)\| = 0$ **then**
6	apply SS_0 // increase power
7	**else if** $\|B(s)\| \geq 1$ **then**
8	apply SS_1 // decrease power
	end if
	while not (stopping criteria)

Table 1. Pseudo-code of the agent's behavior

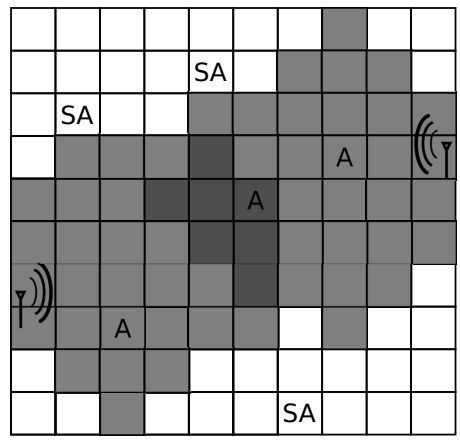


Figure 2. Example of coverage optimization of a two-antenna network by six agents (A), three of which are "special" agents (SA). The depicted area is divided into pixels that define the set S. The set U is represented by white pixels, which are not under coverage. Grayed pixels indicate the area under service coverage, whereas darker pixels indicate coverage by more than one network cell.

Step	
	repeat
1	c = next cell with maximum$att(s)$
2	p_c = Adjust pilot(c, increase rate)
3	**while** ($p_c \notin P_c$) **and** (num of tries $< |B(s)|$)

Table 2. Pseudo-code of step set SS_0

If the agent's current location, at pixel s, is not covered by any cell (i.e. $|B(s)| = 0$), the step set SS_0 (shown in Table 2) is applied. It starts by selecting the cell with the maximum attenuation factor at pixel s (step 1). If many cells have the same value, one is randomly picked out of them. Once c is uniquely identified, the agent changes its pilot power by increase rate dB (step 2). By using a higher increase rate, the network shall potentially cover s, as well as some neighboring pixels. Areas without coverage usually contain many uncovered pixels grouped together, forming irregular uncovered islands.

Step	
	repeat
1	c = next random cell($B(s)$)
2	p_c = Adjust pilot(c, decrease rate)
3	**while** ($p_c \notin P_c$) **and** (num of tries $< |B(s)|$)

Table 3. Pseudo-code of step set SS_1

The step set SS_1 in Table 3 is applied whenever the agent's current location, at pixel s, is covered by one or more cells (i.e. $|B(s)| \geq 1$). The first step randomly selects a cell from $B(s)$. The agent shall decrease the pilot power of c in step 2. This practice keeps the coverage constraint valid over s, although it might potentially break it on other pixels. Ideally, every pixel would be covered by exactly one network cell, although this is just a representation of a perfect solution that is almost entirely unreachable, because of the irregularity in network topology and terrain.

In both step sets, SS_0 and SS_1, the agent makes sure that the new pilot power setting, i.e. after applying the change, is an element of P_c. If this is not the case, cell c is discarded and another cell is selected at the first step of SS_0 and SS_1, adjusting its pilot power accordingly. This is repeated for a maximum of $|B(s)|$ tries to avoid getting trapped in an endless loop.

The values *increase rate* and *decrease rate* are configurable parameters that should be set before starting the optimization process. They indicate the dB adjustment proposed to the pilot power of cell c and are based on the physical properties of the problem being solved. Namely, lowering the pilot power of a cell decreases the interference at pixel s. Moreover, the target *SIR* value at pixel s is reduced under lower interference; thus coverage of this pixel may be achieved with lower pilot power. On the other hand, by increasing the pilot power of a cell with the maximum att_s, we improve coverage by evenly distributing the power among differ-

ent network cells, since the selected cell, c, is, on average, the nearest to the present location.

4.2.2 The Evaluator

The evaluator represents an important component of the optimization system, since it reacts to agent changes by recalculating the value of the objective function, i.e. coverage of the service area and the total pilot power used by all the cells in the network being optimized.

The path-loss and interference data were calculated in advance, using the commercial radio planning tool TEMS^{TM} CellPlanner, which has been generously provided by the Radio Network Department at Telekom Slovenije, d.d. These predictions already include losses and gains from cabling, hardware, and user equipment.

After a short initialization, during which the pre-calculated path-loss and interference matrices for the whole area are loaded, the evaluator computes the service area coverage, based on the pilot powers supplied as the initial solution from which the search process begins. Initial solutions are randomly generated sets, containing valid pilot power settings that fulfill the coverage constraint. The evaluator then waits for agent changes to arrive and incrementally calculates subsequent solutions accordingly.

It is the responsibility of the evaluator to maintain a special part of memory, intended for keeping track of the uncovered pixels in the service area, constantly updated. Whenever the current solution is not valid because of uncovered pixels, i.e. Equation (3) does not hold, "special" agents take on correcting the coverage. They achieve this by randomly selecting an uncovered pixel coordinate from this portion of memory (one at a time) so that a valid solution may be reached again. It should be noted that these "special" agents shall only apply step set SS_0 for as long as the solution is not valid. The portion of "special" agents that may work in correcting the current solution is an optimization parameter.

As it has been mentioned before, this constraint-repairing strategy enhances certain properties of the search process performed, namely:

- increases exploration of the search space, as different regions are also being inspected, and
- enables the algorithm to escape from local optima, leading the search to other areas containing potentially good solutions.

It is worth mentioning that the evaluator itself has no influence on the optimization process from the search point-of-view. Its task is to provide feedback and updated information to the agents exploring the service area. From the performance point-of-view, the importance of the evaluator is significant, as will be shown in the next sections.

5 IMPLEMENTATION

We have chosen the Open Computing Language (OpenCL) [20] as the implementation platform. OpenCL is an open parallel computing API designed to enable GPUs and other co-processors to work together with the CPU and so providing additional computing power. As a standard, OpenCL 1.0 was released in 2008, by The Khronos Group, an independent standards consortium [12]. For additional information about the OpenCL standard and API, we refer the reader to the numerous guides available online.

Our choice in using OpenCL was greatly influenced by the fact that its bitcode runs on a variety of hardware, including multicore CPUs and GPUs from different vendors. This provides a complete framework capable of comparing execution speed-up on different hardware without the need of changing the implementation.

One unfortunate consequence of the vendor variety is that NVIDIA's CUDA [14] and OpenCL documentation present disparate naming conventions for some key components. For the sake of consistency we present a short "translation dictionary" between them as shown in Table 4.

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*)

Table 4. Terminology translation between OpenCL and CUDA [10]

Despite the use of OpenCL as the target platform for our implementation, the details described in the next sections may be equally applied on CUDA.

The evaluator was completely implemented on GPU, because its performance has a great impact on the speed of the optimization system as a whole. The implementation of the agents is also based on GPU, which drastically reduces the number of data transfers between CPU and GPU, since all problem elements are available on the GPU during the optimization process. Therefore, it is a challenging task to accommodate all the needed elements on GPU memory, which is notably smaller than the RAM memory usually available on modern computers.

5.1 Evaluator on GPU

The evaluator implementation is one of the biggest challenges we faced. The great amount of data needed to evaluate agent changes of pilot power was the first restriction we run into, as there was not enough memory on the GPU for all of them. Moreover, it imposes a hard constraint in terms of the scalability of our optimization approach. Generally speaking, we intend to take advantage of the GPU in terms of parallel speed, while the size of the data sets used should not limit its use. As a solution, we propose to change the internal representation of the following data structures:

- the path-loss matrices, one for each of the cells of the network being optimized and
- the interference matrix, which holds data for the whole service area.

The path-loss matrices are potentially as big as the underlying data used to calculate the radio propagation predictions over the service area of the network. In our case, we have at our disposal terrain data covering more than $78\,000\,\mathrm{km}^2$, including a digital elevation model of $100\,\mathrm{m}^2$ resolution. The data include the whole country and some parts of the neighboring countries, ensuring availability even at the country borders. It is not possible to calculate the path losses of more than 100 transmitters over the afore-mentioned area, as the matrix elements would require more memory than is currently available on most modern GPU hardware. Such situations would drastically reduce the scalability of our approach, especially for large real-world networks. For this reason, we have decided to change the representation unit of the path-loss matrix elements from the linear scale to a logarithmic one, namely decibels (dB). After consultation with experts in the radio-telecommunications field, the decision was that the additional error introduced by using integer decibels instead of a real-numbered linear scale is negligible, since the digital elevation model, which has a resolution of $100\,\mathrm{m}^2$ per pixel, presents a bigger rounding problem. Therefore, the path-loss data-type was changed from *float* to *unsigned char*, consequently saving three bytes per data element. It follows that the path-loss between a network cell and any point on the service area should never exceed 255 dB. This scale is large enough for problem representation, since service discovery by the mobile terminal is still successful with a RSCP of around $-115\,\mathrm{dB}$ [9]. Moreover, since the path loss is, by definition, a positive value, we neglect the sign from the internal representation without losing information.

In terms of scalability, another improvement is introduced for further reducing the amount of memory needed for network coverage calculation, namely the calculation radius around each network cell. Taking, for example, a 10 km calculation radius around a cell is generally enough for coverage-calculation purposes over the 2 GHz spectrum of UMTS [9]. Clearly, the calculation radius drastically lowers the memory requirement on the GPU, especially when comparing a potential area of $400\,\mathrm{km}^2$ with the initial one, covering more than $78\,000\,\mathrm{km}^2$. To correctly locate

the reduced areas, delimited by the calculation radius around each network cell, we additionally supply the offset of the upper-left corner for each of the path-loss sub-matrices. The memory layout for the path-loss matrices is shown in Figure 3. It is worth pointing out that the introduced memory-efficiency methods do not have a significant impact on the quality of the solutions, which are generated by the algorithm. This will be further elaborated in Section 6.4.

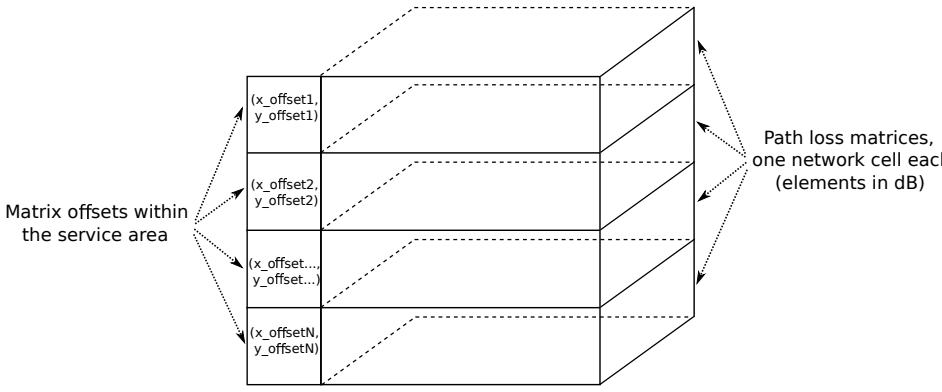


Figure 3. Texture memory organization of the path-loss matrices on the GPU. Each layer represents the path loss for one cell of the network under optimization

Because the content of the path-loss matrices is constant throughout the optimization process, they are kept in read-only texture memory to take advantage of the fast access time provided by the hardware. Additional speed-up is achieved by incrementally recalculating the service area coverage as the agent changes arrive. Specifically, the network cells containing new settings that have not yet been evaluated have their pilot powers saved in negative form, serving as a flag to indicate that coverage re-evaluation is needed. Since pilot powers are positive numbers, there is no possibility for confusion.

5.2 Parallel Agents on GPU

The autonomous and programmable nature of the agents makes them ideal for parallel and GPU-based implementations. By lowering the complexity of the step sets applied at each time, we were able to tackle larger optimization problem instances with good results in very short time.

For the kernel implementing agent behavior, only one work group is launched. It contains as many work items as there are agents deployed during the optimization, organized in a 1D grid. Each work item randomly generates a coordinate within the service area, using the system time in milliseconds as a random seed. Because OpenCL lacks functions for random-number generation, we implemented a simplified version of Marsaglia's generator [11]. Afterwards, each work item analyzes the

received signals at the current coordinate by applying step sets SS_0 or SS_1, as it has been explained in Section 4.2. The outcome of the analysis is saved in its local-memory position by each of the work items. It contains the *id* of the network cell (at position $2 \times local\_id$), and the pilot power setting (at position $2 \times local\_id+1$). The new pilot power is calculated as the dB difference from the previous one, based on the values of *increase rate* and *decrease rate*. Since both the cell *id* and the pilot power are of type *unsigned short*, there is enough space in a 16 Kb local memory block to allocate up to 4,096 independent agents. Therefore, this number is not bounded by size of local memory, since most modern GPUs have a hard limit in the number of work items per work group, being 256 or 512. The last step involves saving the new pilot powers back to their containing vector in global memory. This is done by only one of the work items within the work group, to avoid memory-access conflicts. At this moment, the sign of updated pilot powers is changed to indicate that coverage re-calculation is needed. Clearly, the vector containing pilot powers in global memory is of type *int*, as it must allow signed-value storage. Nevertheless, a single pilot power setting never exceeds 65 535 mW. In case there is more than one new setting for a specific network cell, the median among all proposed settings is calculated and applied as the new pilot power for that cell.

Even though the agent kernel does not achieve coalesced access to the GPU global memory, its sole implementation provides enhanced performance, since most of the data accessed during the optimization process is already available on the GPU. This significantly reduces the number of data transfers between the CPU and GPU, consequently improving the speed of the optimization process. Moreover, the kernel also produces truly parallel behavior of the agents, as they all explore the service area at the same time.

6 SIMULATIONS

6.1 Test Networks

All the test networks, Net_1, Net_2, and Net_3 are subsets of a real UMTS network deployed by Telekom Slovenije, d.d., in Slovenia. The path-loss predictions are calculated using the Okumura-Hata model [8], using a digital elevation model of $100\,\mathrm{m}^2$ resolution as input data and a receiver height of 1.5 m above ground level. The requirements for *SIR* coverage were provided by experts of the Radio Network Department at Telekom Slovenije, d.d.

Net_1 is deployed over a densely populated urban area. For this reason, the *SIR* coverage threshold is lower, since network capacity is the dominating factor, whereas coverage is flexible because of a higher cell density, i.e. more base stations per surface unit. Net_2 represents a network deployed over a dominant rural area, meaning that network capacity may be reduced at the cost of better coverage, since each cell must cover greater area. The last network, Net_3, represents a suburban area with a highly-dense populated, but relatively small, downtown center, where compromise between network capacity and coverage has to be achieved.

We have produced network configuration settings based on the attenuation-based approach. These settings represent what could be an initial network setup by common planning standards [9]. Moreover, such settings are also very straightforward to calculate by a network planner. Table 5 shows some configurations of the test networks used. The parameter values used during experimentation are shown in Table 6.

	Cells [m]	Area [km^2]
Net_1	77	100.00
Net_2	23	306.25
Net_3	129	405.00

Table 5. Network configurations

Parameter	Net_1	Net_2	Net_3
p_c^T	15.00 W	19.95 W	15.00 W
τ_0	$1.55 \cdot 10^{-14}$ W	$1.55 \cdot 10^{-14}$ W	$1.55 \cdot 10^{-14}$ W
γ_c	0.010	0.020	0.015

Table 6. Network parameters

6.2 Algorithm Parameter Settings

After short experimentation, we determined the parameter settings for the optimization algorithm. There was no fine tuning of parameters for each problem instance. Nevertheless, we gained valuable information regarding the agent's behavior that we used to set the following parameter values:

- increase rate was set to 0.2 dB,
- decrease rate was set to -0.1 dB,
- number of agents was set to 16 and
- 10 000 changes per agent were allowed.

6.3 Experimental Environment

All experiments were done on Intel i7 2.67 GHz computer with 6 GB of RAM running 64-bit Linux operating system. The GPU hardware was ATI HD5570 with 1 GB DDR3 RAM. The implementation language used was C, with OpenCL and OpenMPI extensions. Experimentation with MPI-based agents was carried out on a CPU-only cluster, managed by the XenServer virtualization tool. The virtual server exposed 12 CPUs and 16 GB of RAM, also running Linux.

6.4 Optimization Results

Table 7 shows that results achieved by our optimization approach significantly improved solution quality. This was expected, since we have already shown that our previous approach, which was used as basis, outperformed the previously best approach. Results show that we reduced pilot power usage in all networks and kept the service area under full coverage. Moreover, we may see that the solution for Net_1 improved the attenuation-based setting by more than 300 %. For Net_2, the improvement observed is around 232 %, with an improvement of more than 170 % for Net_3. This means that network capacity has been significantly increased in all three problem instances. Therefore, more users should be able to access services provided by the mobile network, since coverage is assured at lower power usage. Moreover, an increased speed in data services may also be achieved [9].

	Attenuation-based		Parallel agents	
	Total power [W]	Average power [W]	Total power [W]	Average power [W]
Net_1	419.292	5.445	137.064	1.780
Net_2	78.297	3.404	33.344	1.450
Net_3	1 014.113	7.861	582.954	4.519

Table 7. Optimization results after applying two different approaches, e.g. attenuation-based and parallel agents

After collecting data from ten independent runs, we generated convergence graphs, shown in Figures 4–6. The graphs contain feasible solutions only, i.e. solutions that meet the full-coverage constraint. Unfeasible solutions were marked with a value of inferior quality than the worst solution found by the algorithm in all ten runs. In case of Net_1, the value was set to 428, for Net_2 the value was set to 129 and for Net_3 the value was set to 1,435.

The analysis of convergence graphs of Net_1 and Net_2 shows that the algorithm quickly converges at the beginning, followed by a steady improvement of intermediate solutions. In Net_1 we notice an additional improvement of the solutions found, even towards the end. This fact suggests that in this case longer runs would potentially find even better solutions. For the instance Net_3, we observe a slower initial convergence, with steady improvement of intermediate solutions and no significant solution enhancement during the last 1 000 iterations. This fact, together with the aforementioned results, suggest that this problem instance presents a more difficult optimization case than Net_1 or Net_2. Nevertheless, the improvement observed is, on average, around 100 %.

6.5 Implementation Results

After measuring the quality of the solutions given by our parallel-agent approach, we present the experimental results regarding efficiency of different implementations. For this purpose, only execution times and speed-up factors are presented. The

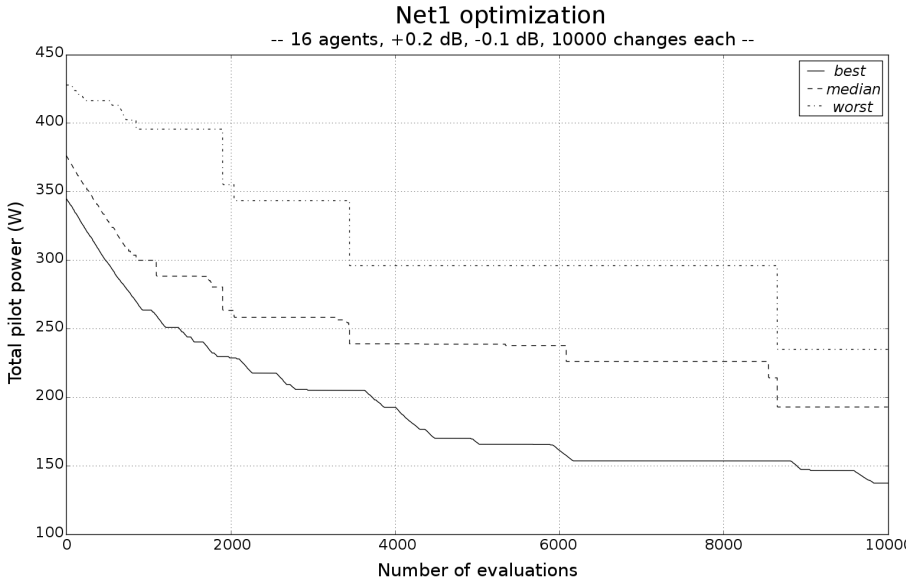


Figure 4. Convergence results for the optimization of network Net_1

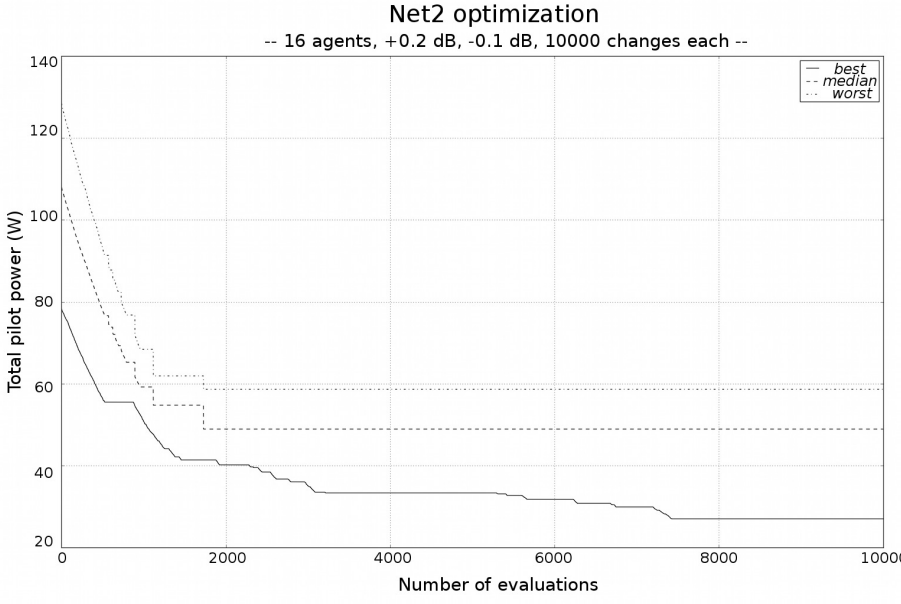


Figure 5. Convergence results for the optimization of network Net_2

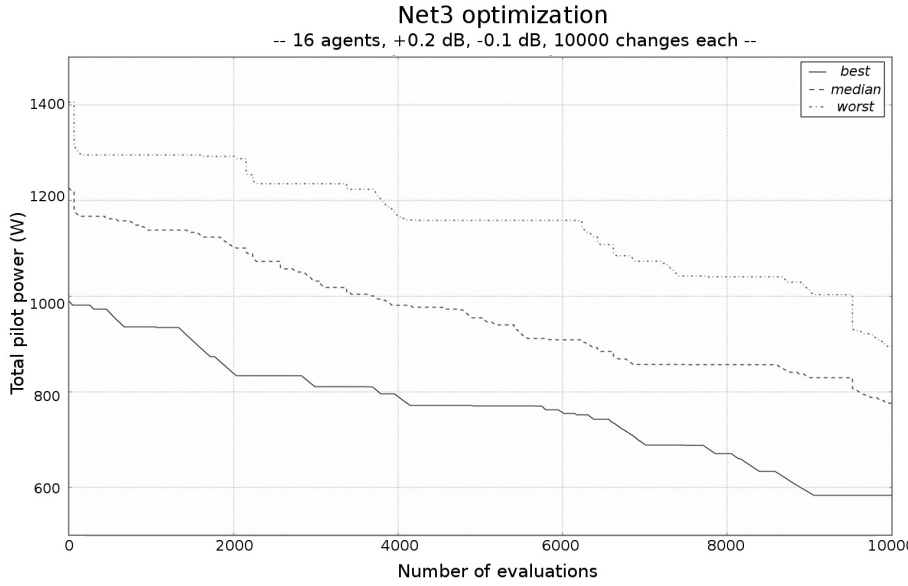


Figure 6. Convergence results for the optimization of network Net_3

running time was measured for ten independent runs on each platform. The number of changes per agent was limited to 100, while all other algorithm parameters were kept at the same values as previously stated in Section 6.2.

The results, shown in Table 8, are reported for different sizes of problem instances, i.e. Net_1, Net_2, and Net_3. These networks provide different combinations of network cells and service area size. Results are presented for:

- CPU-only implementation, including evaluator on CPU and MPI-based agents,
- GPU evaluator with MPI-based agents and
- GPU evaluator with agents on the same GPU.

The implementation combining CPU-based evaluator and MPI-based agents is the basis for the speed-up calculation.

	CPU eval. + MPI agents	GPU eval. + MPI agents		GPU eval. + GPU agents	
	Avg. time	Avg. time	Speed-up	Avg. time	Speed-up
Net_1	105 455	346	305x	67	1 574x
Net_2	33 700	195	173x	46	733x
Net_3	191 900	506	379x	117	927x

Table 8. Implementation-efficiency measures with average times [s] and speed-ups

It should be noted that the MPI implementation of agents, used for the first and second measured setups, is not fully parallel, since internal synchronization at network level is performed, so agent changes arrive in a serial fashion to the evaluator, before being evaluated.

The GPU-based evaluator, communicating with agents over MPI, provides the second measured setup. The implementation takes advantage of local memory for work item collaboration within a work group, and texture memory for constant elements, as has been explained in Section 5.1. Still, the speed-up is considerable, but could be further improved, since numerous data transfers between CPU and GPU are needed for the agents to access optimization-related information.

The last result set presents measurements for the complete GPU implementation, including evaluator and agents on the same device. The substantial speed-up delivered by this combination highlights the great impact that CPU-to-GPU memory transfers have on overall system performance. This fact is supported by the speed-up between the second and third measured setups, which exhibit, on average, an improvement of more than 400 %.

7 CONCLUSION

In this paper, we have addressed the problem of providing full coverage to a service area of a UMTS network by using a minimum amount of pilot power. We have put emphasis on the confluence of a real-world problem, with live data from a deployed mobile network, with state-of-the-art parallel GPU hardware and implementations that, to the best of our knowledge, have never been dealt-with before.

We have presented a parallel-agent approach, which is aimed at giving good solutions to big problem instances in an acceptable amount of time. The experimental results show that our approach is able to find competitive solutions when compared to other common radio-planning methods [9]. The presented results also demonstrate that our algorithm is able to find high quality solutions even for large networks, that contain many cells over a large service area. This fact suggests that our approach could be successfully applied to even bigger problem instances.

GPU architectures not only allow implementation of parallel heuristics in a natural way, they also substantially improve the performance of the optimization process. We have reported and validated the great performance gains by experimentation on problem instances of different sizes.

After successfully implementing the evaluator on GPU, we realized that the efficiency of this approach was limited by the CPU-to-GPU data transfers. Nevertheless, even with such implementation, we have already obtained substantial speed-up.

To deal with the CPU-to-GPU data transfer issue, we implemented a fully-enabled GPU optimization system that achieved impressive speed-up. Still, we had to consider different data representation schemes for the problem elements, so to avoid memory limitations on the GPU device. Comparison of our experimental

results with other algorithms dealing with the same and similar problems would be useful. However, this task is not straightforward, since the results of several works, e.g. [7, 22], depend on black-box evaluations, making experimental association very difficult, if possible at all.

Although not being currently used in everyday planning tasks, the work presented in this paper is currently being extended in Telekom Slovenije, d.d., for future use. Namely, with the imminent roll out of LTE, an implementation of the fourth generation mobile networks, the solving of the service coverage problem will receive a great deal of attention in the near future.

All in all, we consider that the present work provides a robust foundation for further research on grid-based metaheuristics with computationally expensive evaluation functions. In future work, we will further analyze our parallel-agent approach, including experimentation with different parameters, in order to gain better understanding of the dynamics leading the algorithm during the search process. Multi-GPU environments present an interesting possibility, where evaluator and worker agents could be deployed over separate GPU devices.

Acknowledgments

This project was co-financed by the European Union, through the European Social Fund.

REFERENCES

[1] Benedičič, L.—Štular, M.—Korošec, P.: Pilot Power Optimization in UMTS: A Multi-Agent Approach. Proceedings of the 13th International Multiconference Information Society, 2010, pp. 7–10.

[2] Chen, L.—Yuan, D.: Automated Planning of CPICH Power for Enhancing HSDPA Performance at Cell Edges with Preserved Control of R99 Soft Handover. IEEE International Conference on Communications, 2008, pp. 2936–2940.

[3] Cheung, G.—Tan, W.—Yoshimura, T.: Real-Time Video Transport Optimization Using Streaming Agent Over 3G Wireless Networks. IEEE Transactions on Multimedia, Vol. 7, 2005, No. 4, pp. 777–785.

[4] Chira, C.—Dumitrescu, D.—Pintea, C.-M.: Learning Sensitive Stigmergic Agents for Solving Complex Problems. Computing and Informatics, Vol. 29, 2010, No. 3, pp. 337–356.

[5] Cunningham, B.—Alexander, P.—Candeub, A.: Network Growth: Theory and Evidence from the Mobile Telephone Industry. Information Economics and Policy, Vol. 22, 2010, No. 1, pp. 91–102.

[6] Esposito, A.—Tarricone, L.—Luceri, S.—Zappatore, M.: Genetic Optimization for Optimum 3G Network Planning: An Agent-Based Parallel Implementation. Novel Algorithms and Techniques in Telecommunications and Networking, 2010, pp. 189–194.

[7] Gerdenitsch, A.: System Capacity Optimization of UMTS FDD Networks. Ph. D. dissertation, Technische Universität Wien 2004.

[8] Hata, M.: Empirical Formula for Propagation Loss in Land Mobile Radio Services. IEEE Transactions on Vehicular Technology, Vol. 29, August 1980, No. 3.

[9] Holma, H.—Toskala, A.: WCDMA for UMTS: Radio Access for Third Generation Mobile Communications. Third Edition. John Wiley & Sons, New York 2005.

[10] Kloeckner, A.: GPU Programming with PyOpenCL and PyCUDA. Available on `http://www.bu.edu/pasi/materials/` [Online; accessed 5-April-2011].

[11] Marsaglia, G.: Seeds for Random Number Generators. Communications of the ACM, Vol. 46, 2003, No. 5, pp. 90–93.

[12] Munshi, A.: The OpenCL Specification Version 1.0. Khronos OpenCL Working Group 2009.

[13] Nawrocki, M.—Aghvami, H.—Dohler, M.: Understanding UMTS Radio Network Modelling, Planning and Automated Optimisation: Theory and Practice. John Wiley & Sons 2006.

[14] NVIDIA CUDA Compute Unified Device Architecture – Programming Guide. NVIDIA 2008.

[15] Sarkar, P.: A Brief History of Cellular Automata. ACM Computing Surveys (CSUR), Vol. 32, 2000, No. 1, pp. 80–107.

[16] Schueller, J.—Begain, K.—Ermel, M.—Mueller, T.—Schweigel, M.: Performance Analysis of a Single UMTS Cell. Proceedings of the European Wireless Communications Conference 2000.

[17] Sarkar, P.: A Brief History of Cellular Automata. ACM Computing Surveys (CSUR), Vol. 32, 2000, No. 1, pp. 80–107.

[18] Siomina, I.—Yuan, D.: Pilot Power Optimization in WCDMA Networks. Proceedings of WiOpt '04: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, 2004, pp. 191–199.

[19] Siomina, I.—Yuan, D.: Minimum Pilot Power for Service Coverage in WCDMA Networks. Wireless Networks, Vol. 14, 2007, No. 3, pp. 393–402.

[20] Stone, J.—Gohara, D.—Shi, G.: OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. Computing in Science and Engineering, Vol. 12, 2010, pp. 66–73.

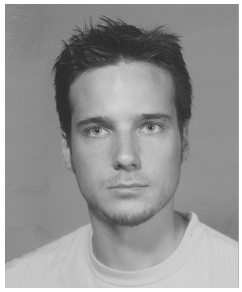
[21] Talbi, E.: Metaheuristics: From Design to Implementation. Wiley 2009.

[22] Türke, U.—Koonert, M.: Advanced Site Configuration Techniques for Automatic UMTS Radio Network Design. IEEE Vehicular Technology Conference, 2005, pp. 1960–1694.

[23] Valcarce, A.—De La Roche, G.—Jüttner, Á.—López-Pérez, D.—Zhang, J.: Applying FDTD to the Coverage Prediction of WiMAX Femtocells. EURASIP Journal on Wireless Communications and Networking, 2009, pp. 1–13.

[24] Värbrand, P.—Yuan, D.: A Mathematical Programming Approach for Pilot Power Optimization in WCDMA Networks. Australian Telecommunications, Networks and Applications Conference (ATNAC), 2003.

[25] Vasile, M.—Locatelli, M.: A Hybrid Multiagent Approach for Global Trajectory Optimization. Journal of Global Optimization, Vol. 44, 2009, No. 4, pp. 461–479.



Lucas Benedičič received his M. Sc. in computer sciences from the University of Primorska, Koper, Slovenia, in 2009 and the Ph. D. from the Jožef Stefan International Postgraduate School in 2014. He is currently a member of the Research and Development Department at Telekom Slovenije, d.d., in Slovenia. His research interests include parallel evolutionary algorithms, high-performance computing, GPGPU, and combinatorial optimization applied to radio mobile networks.

Mitja Štular is the Head of Research at Mobitel. He has been in telecommunications for the last 17 years. He has rich experience in research, teaching and management on various positions at the University of Ljubljana and Mobitel: Teaching Assistant, Researcher, UMTS Project Director, Chief Technology Officer, and Senior Strategy Adviser to CEO. He serves as a member of councils and boards of several institutions in the field of telecommunications and public health. He is one of the leading Slovene experts in mobile networks and applications.

Peter Korošec received his Ph. D. from the Jožef Stefan International Postgraduate School, Ljubljana, Slovenia, in 2006. Since early 2002 he has been a researcher at the Jožef Stefan Institute, Ljubljana, Slovenia. He is presently a researcher at the Computer Systems Department and an Associate Professor at the University of Primorska, Faculty of Mathematics, Science and Informational Technologies Koper, Koper, Slovenia. His research interests include combinatorial/numerical optimization with modern metaheuristics in parallel/distributed computing.