Manuscript Number: COR-D-18-00208R2

Title: On the overlap area of a disk and a piecewise circular domain

Article Type: Research Article

Keywords: Decision support systems; facility placement; service coverage; continuous space; graphics processing unit (GPU)

Corresponding Author: Dr. Marta Fort,

Corresponding Author's Institution: Universitat de Girona

First Author: Narcis Coll, Dr.

Order of Authors: Narcis Coll, Dr. ; Marta Fort; J.Antoni Sellarès, Dr.

Abstract: We provide an approach to solve the problem of locating a disk so that its overlap area with a piecewise circular domain is near-optimal when considering partial converage. To this purpose, we introduce the concept of overlap area map. Both, overlap area maps and near-optimal locations, provide useful tools for solving problems related to the placement, for example, of emergency warning sirens, cellular towers or radio receiving stations, that are commonly encountered in the location science field.
We present parallel algorithms on graphics processing units (GPUs) for computing an overlap area map and obtaining a set of near-optimal locations from the overlap area map. {These algorithms have as a key element the overlap area computation process to which we pay special attention.} In addition, we describe a way of visualizing the obtained solutions. Integration of computation and visualization facilitates decision makers, with an iterative what-if-analysis process, to gain more information in order to facilitate the selection of an appropriate location. Finally, we also provide and discuss experimental results showing the efficiency and scalability of our approach.

Highlights for "*On the overlap area of a disk and a piecewise circular domain*"

Narcís Coll, Marta Fort and J. Antoni Sellarès

- We provide optimal and near-optimal locations for a disk according to its overlap area with a piece-wise circular domain
- We provide a regular and a non-regular approach which uses a refinement process
- We describe and theoretically analyze a sequential and a parallel algorithm to solve the problem by using CUDA architecture
- We visualize the obtained information highlighting the optimal locations and analyze the experimental results obtained with the implementation of the proposed algorithms
- We compare the provided strategies in terms of efficiency theoretically and experimentally

On the overlap area of a disk and a piecewise circular domain

Abstract

We provide an approach to solve the problem of locating a disk so that its overlap area with a piecewise circular domain is near-optimal when considering partial converage. To this purpose, we introduce the concept of overlap area map. Both, overlap area maps and near-optimal locations, provide useful tools for solving problems related to the placement, for example, of emergency warning sirens, cellular towers or radio receiving stations, that are commonly encountered in the location science field. We present parallel algorithms on graphics processing units (GPUs) for computing an overlap area map and obtaining a set of near-optimal locations from the overlap area map. These algorithms have as a key element the overlap area computation process to which we pay special attention. In addition, we describe a way of visualizing the obtained solutions. Integration of computation and visualization facilitates decision makers, with an iterative what-if-analysis process, to gain more information in order to facilitate the selection of an appropriate location. Finally, we also provide and discuss experimental results showing the efficiency and scalability of our approach.

Keywords: Decision support systems, facility placement, service coverage, continuous space, graphics processing unit (GPU)

1. Introduction

Location science is a part of operations research and management science concerned with the placement of a limited set of facilities in order to optimize (minimize or maximize) at least one objective function: coverage, cost, travel distance, etc.

A large number of problems locating facilities use the concept of coverage. A demand is covered by a facility if the distance or travel time between the demand and the facility is less than a certain predetermined value called the coverage radius. For example, a house is covered by a fire station if the time of travel from the station to the house is less than ten minutes. The problems of coverage are of great applicability when planning the location of facilities for both public and private sectors.

Often, complete coverage of all demand in a region is not possible due to budgetary limitations on the number of facilities that can be sited. Thus, limited resources must be efficiently managed and regional demand should be covered to the greatest extent possible.

1.1. Background

The maximal covering location problem (MCLP), introduced by Church and ReVelle [8], seeks to identify the locations for a specific number of facilities in such a way that the coverage is maximum within a desired service distance. Both the demand and the location of the facilities are finite sets of points, and a demand point is considered *covered* if it lies inside the disk centered on a facility with radius equal to the specified service distance. Exacts algorithms for solving the MCLP are provided in [8, 14]. Since the MCLP is NP-Hard [28], exact methods may be inefficient for practical use. Thus, various heuristic methods that allow faster solution times with near-optimal results have been proposed [8, 13, 19, 29, 17, 30, 22]. Many different adaptations and extensions of MCLP have been introduced [4, 32, 18]. Non-trivial extensions include, for example, gradual coverage in which the basic coverage yes/no covering function is replaced by more general decreasing functions in the distance separating the user from the facility, thereby modeling partial coverage [9, 2, 3, 16, 21]. When facilities and demand are discrete and point-based, the MCLP based approach is appropriate, but in many practical cases the demand and/or candidate facility locations are continuously distributed over space, thus new approaches have been developed that help to improve the accuracy of the obtained solution.

The planar maximal covering location problem (PMCLP) under the Euclidean distance, originally defined by Church in [6] considers siting facilities in continuous space while representing demand as discrete points. In order to solve the PMCLP, properties that allow the discretization of continuous regions of the plane where to locate facilities are used. It turns out that a discrete and finite set of potential locations can be proven to contain an optimal coverage solution assuming binary coverage, that is, a demand object is either covered or not by a facility. This set of candidate locations is formed by finding the intersection of all circles of radius the specified service distance centered at each discrete demand location. These points are referred to as the circle intersection point set (CIPS). Then, the CIPS are used with MCLP to efficiently solve the PMCLP. Nevertheless, representing the demand in a discrete way introduces inaccuracies in the obtained solution [15].

In most real cases, the location of the demand and the candidate locations are continuously distributed over a region of the plane [26]. As an example, when locating emergency warning sirens, it is important for sirens to be audible anywhere in a given region. Another example is when locating cellular towers, calls could be placed and/or received at any location in a region. In a similar way, emergency warning sirens and cellular equipment can effectively be sited almost anywhere since these types of facilities can be mounted on posts or existing structures. Murray and Tong in [25] present the extended planar maximal covering problem assuming Euclidean distance (EPMCE). They extend the work of [6] by modeling the demand as a finite union of spatial objects (points, lines or polygons). They derive a finite set of potential facility sites from continuous space, referred to as polygon intersection point set (PIPS), and prove that the PIPS contains an optimal solution to the EPMCE. The EPMCE is solved as a discrete MCLP using the PIPS. However, this model often introduces significant errors in the obtained solution since, as in the case of the PMCLP, it does not maximize coverage because partial coverage is not taken into account in the objective function [34]. Partial coverage on rectangular demand and rectangular service zones (PMCLP-PCR) is studied in [1] by Bansal and Kianfar. The PMCLP-PCR problem has important applications in multi camera view-frame selection to maximize the partial coverage of rectangular regions. In [33], in the context of earth-observing satellites, was studied the single camera version of this problem by taking into account not only the covered area but also the quality of the obtained images. Murray et al. [27] introduce the continuous maximal coverage problem (CMCP) under the Euclidean distance, in which the demand is considered present everywhere within the region and the facilities can be located anywhere in the plane. In [23], Matisziw and Murray solve the CMCP for a single facility, with the assumption of uniformly distributed demand and a disk-like service area for the facility. In both papers, computationally prohibitive geometry-based approaches are used that only guarantee optimality for the one single facility case [34]. Solving exactly the CMCP, even for locating a single facility, is difficult and computationally very expensive because an infinite number of locations must be considered, both for the demand of service and for the installation of the facilities. Therefore, standard optimization techniques for solving discrete location models are not applicable to the CMCP, and new efficient methods are required to solve it.

Furthermore, there has been also some related work on the problem that we study in this paper in the computational geometry field. Given two simple polygons P and Q with n and m vertices, respectively, Mount et al. [24] gave an algorithm to compute their maximum overlap under translation in $O(n^2m^2)$ time. Cheong et al. [5] proposed an algorithm to approximate the maximum overlap using random sampling techniques. Given $\varepsilon > 0$, with high probability the additive error is $\varepsilon \cdot \min\{\text{area}(P), \text{area}(Q)\}$ and the running time is $O(n + (m^2\varepsilon^{-4} \log_2 m))$. More recently, Cheng and Lam [7] presented an algorithm to approximate the maximum overlap of two polygons P and Q, built upon the framework of Cheong et al. [5]. Polygons P and Q may have multiple holes. If n denotes the total number of vertices in P and Q, the running time of the algorithm is $O(n^2\varepsilon^{-3} \log^{1.5} n \log(n/\varepsilon))$. If one of the two polygons is convex, the additive error with high probability is $\varepsilon \cdot \text{area}(P)$ and the running time can be improved to $O(n \log n + \varepsilon^{-3} \log^{2.5} n \log((\log n)/\varepsilon))$.

Therefore, the inefficiency of the existent methods, the large volume of data to be processed and the nature of the problem turn the parallel algorithms into a solution to explore in order to solve this problem. Moreover, the programmability and high computational rates of graphics processing units (GPU) make them a powerful platform for computationally demanding tasks where it is needed to process a large amount of data or perform a lot of operations. Parallel processing capability of the GPU allows you to split complex computing tasks into thousands of smaller tasks that can be run simultaneously. This capability allows for the solution of many problems with the GPU in a fraction of the time required by the CPU. The GPUs have quickly become an standard industry that powers millions of PCs, notebooks, workstations and supercomputers around the world. In particular, the general purpose computing on GPUs (GPGPU), as a way to reduce execution times, is being used by many researchers in several computational fields ranging from numerical computing operations and physical simulations to knowledge discovery, data mining and bioinformatics geometry processing [20, 12].

1.2. Problem formalization

In this paper we focus on the one-facility case of the CMCP with the assumption of uniformly distributed demand, and a disk-like service area for the facility. The facility can be located anywhere on the plane, its service area is not necessarily completely contained within the piecewise circular domain and partial coverage is taken into account. See Figure 1 for a motivational example.

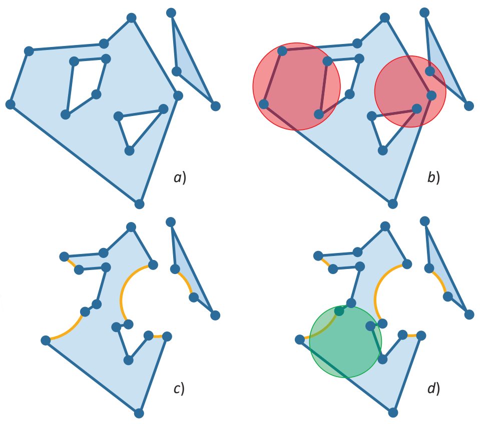

Figure 1: a) Polygonal domain to be partially covered by facilities with circular service coverage; b) Domain partially covered by two facilities; c) Piecewise circular domain not yet covered; d) New facility partially covering the piecewise circular domain.

Considering partial coverage forbids us to use a pre-computed set of area values of several predefined regions, each one of which could be considered either covered or not covered in its totality. On the contrary, we have to compute a huge number of intersection areas. If these areas were computed in the way used by Geographical Information System (GIS) commercial software, that is, by first obtaining the part of the polygon included in the disk, and second computing the area of such part, the intersection between the polygon and thousands of disks would have to be computed and, consequently, a huge memory space would be needed. Hence we have the aim to obtain a fast and robust enough way to exactly compute the overlap area of a disk and piecewise circular domain.

Moreover, an inherent limitation of the one-facility CMCP is that only one optimal location is returned as the answer, which in many cases is very expensive to compute and in general is also too restrictive. However, near-optimal solutions that could be considered by experts as even more appropriate than that of the optimal one, possibly exist. Thus, our goal is to approximately solve the problem by providing a set of suitable near-optimal locations, that is, locations that cover an area that differs by a prefixed constant from the area covered by an optimal location.

A *piecewise circular curve, pwc-curve*, is a finite ordered list of connected circular arcs and line segments (that can be considered as circular arcs with infinite radius). The arcs and line segments are the edges, and the points where these edges intersect are the vertices of the pwc-curve. A pwc-curve is closed if its first and last vertices coincide, and weakly simple if some pair of non-adjacent edges may intersect but the edges do not cross. A *piecewise circular region, pwc-region*, is a set whose boundary is a closed weakly simple pwc-curve. A *pwc-region with holes* is a pwc-region from which the union of the interiors of a finite number of enclosed pwc-regions, has been removed. The enclosing pwc-region is named *outer component* and the enclosed pwc-regions *holes*. The boundaries of the outer component and the holes are pairwise disjoint, and the holes are empty. A *piecewise circular domain, pwc-domain*, is the union of a finite collection of non overlapping pwc-regions with holes.

Let $D_r(q)$ be the disk of center $q \in \mathbb{R}^2$ and radius $r > 0$. From now on, consider that r is constant. Given a pwc-domain P, we denote $A(q)$ the overlap area of $D_r(q)$ with P. If there were a location q satisfying $A(q) = \pi r^2$, q would be an *ideal* solution of the covering problem. However, in the general case the covering problem does not have ideal solutions. Let $\overline{A}$ be the maximum value of $A(q)$ and consider the set of *optimal locations*

$$OL = \{q \in \mathbb{R}^2 \,|\, A(q) = \overline{A}\}\,.$$

Since the set OL can be just a single point, we are interested in computing a larger set of near-optimal locations. Given $\varepsilon > 0$, we define the set OL_ε of *ε-optimal locations* by:

$$OL_\varepsilon = \{q \in \mathbb{R}^2 \,|\, \overline{A} - A(q) \leq \varepsilon\}\,.$$

Our principal goal is to efficiently compute an approximation of the set OL_ε.

1.3. Our contributions

In this paper we develop an approach to solve the CMCP for a single facility under the Euclidean distance, with the assumption of uniformly distributed demand and taking into account partial coverage. The prohibitive running times of the existing approximation algorithms for solving the problem, motivated us to design a GPU parallel approach to efficiently find an approximate solution. Initial versions of this paper, one dealing with polygonal domains and the other with pwc-domains, were presented in two informal workshops [10, 11].

We start presenting the strategy we designed to efficiently and robustly compute the exact overlap area between a disk and a pwc-domain. To the best of our knowledge, this is the first paper describing how to compute this area in an efficient and exact way.

Next, we propose grid-based algorithms to compute a discrete overlap area map and obtain a discrete set of ε-optimal locations from the discrete overlap area map. Note that, since each ε-optimal location represents not only a point but also all the points contained in the grid cell centered on that point, we could say that we obtain a continuous solution defined by a set of discrete points. On the other hand, as it is usually done, the space occupied by a facility is represented by a single point, and hence we deal with a simple approximation of the facility. Moreover, slightly moving the location of the facility does not affect much on the final coverage area. Hence, it is reasonable to use an approach that, like ours, provides approximate solutions. In addition, to improve the understanding of the problem, we have designed an interface to visualize the obtained overlap areas hightlighting the ε-optimal locations. This integration between computation and visualization facilitates, to decision makers, the obtention of an approximately optimal location for a new facility. Indeed, they can use an iterative what-if-analysis process in which the allowed absolute error ε and the discretization size can be changed. Each iteration is intended to provide additional detail to ensure a better understanding of the problem in order to be able to obtain a better solution. The short response times for computing and visualizing the results facilitate the interaction with the user.

Finally, we provide and discuss experimental results obtained with the implementation of our algorithms that show the efficiency and scalability of our approach. We do not compare our results with the previously existent papers because this is the first time that this problem is solved in the way it is setted in this paper. As it is explained in the previous work, the existent papers do not consider partial coverage and most of them place more than one facility. Hence, comparing our algorithm with the previous once would be completely unfair.

1.4. Organization of the paper

The remainder of the paper is organized as follows. In section 2, we describe the efficient way we designed to compute the area of the overlap between a disk and a pwc-domain. Section 3 provides a grid-based approach to compute a discrete overlap area map from which we obtain a discrete set of ε-optimal locations. In section 4, we sketch a sequential and present a parallel algorithm for computing a discrete overlap area map and a set of ε-optimal locations, based on the grid-based approach provided in the previous section, together with their space and time complexity analysis. In section 5, we present an discuss experimental results, including the visualization of the solutions, and the analysis of the running times. We end the paper with the conclusions in section 6.

2. Overlap area computation

We need an efficient way to compute the area of the overlap between a disk $D_r(q)$ and a pwc-domain. The strategy we present here is based on Green's theorem [31], often used to compute areas in calculus, and on several geometrical observations that allows analyzing several specific and well defined cases.

To start with, note that the overlap area can be computed as the area of the overlap between the disk and the outer components of the domain minus the area of overlap between the disk and each one of the holes. Along this section we provide a way to exactly and efficiently compute the area $A(q) = \text{area}(D_r(q) \bigcap R)$ of the overlap of $D_r(q)$ with a pwc-region R without holes. This area $A(q)$ can be computed in time proportional to the number n of the vertices of R as follows. By placing the origin O of the coordinate system at q, the area $A(q)$ is equal to the area of overlap between $D_r(O)$ and R. Given a point p lying on the boundary B of R, we consider the point $\overline{p}$ defined as follows: if p is contained in $D_r(O)$ then $\overline{p} = p$, otherwise $\overline{p}$ equals to the radial projection of p onto $D_r(O)$. Consider now the curve $\overline{B}$ defined by

$$\overline{B} = \{\overline{p}/p \in B\}\,.$$

Observe (see Figure 2) that the poly-curve $\overline{B}$: 1) is weakly simple; 2) can be continuously approximated by simple closed curves; 3) encloses a region whose area equals $A(q)$. Thus, the area $A(q)$ can be computed by using Greens's theorem as follows:

$$A(q) = \frac{1}{2}\int_{\overline{B}} -ydx + xdy\,.$$

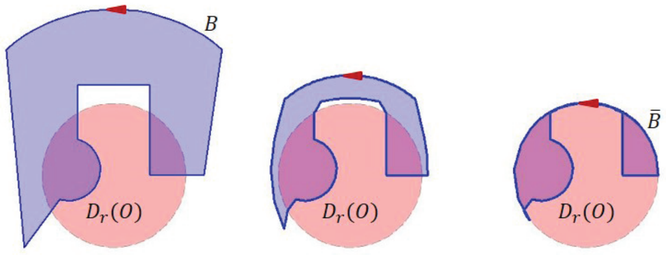

Figure 2: Obtaining $\overline{B}$ from B.

Moreover, taking into account that a circular arc or a segment intersects a circle in at most two points, the curve B can be expressed as a closed piecewise curve $B = \bigcup_{i=0}^{m-1} B_i$ ($m \leq 3n$) where each curve B_i with endpoints p_i and p_{i+1} corresponds to one of the next cases:

Case 1: B_i is a segment contained in $D_r(O)$.

Case 2: B_i is a circular arc contained in $D_r(O)$.

Case 3: B_i is a pwc-curve exterior to $D_r(O)$ satisfying that the intersection between B_i and the half-line with origin O and direction vector $\overrightarrow{Op_i}$ equals p_i (see Figure 3).

Consequently, it holds:

$$A(q) = \frac{1}{2}\int_{\overline{B}} -ydx + xdy = \frac{1}{2}\sum_{i=0}^{m-1} I_i\,,$$

with

$$I_i = \int_{\overline{B}_i} -ydx + xdy = \int_a^b \begin{vmatrix} x(t) & x'(t) \\ y(t) & y'(t) \end{vmatrix} dt\,,$$

where $(x(t), y(t)), t \in [a, b]$, is a parametrization of the curve $\overline{B}_i$. Next, we discuss on the parametrization of each B_i and how to compute I_i depending on the case to which the curve corresponds.

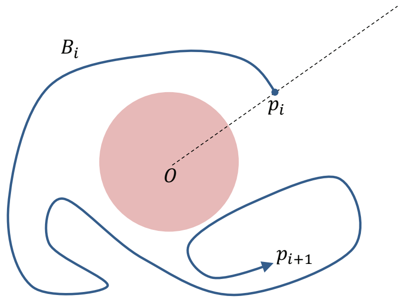

Figure 3: B_i is exterior to $D_r(O)$.

Case 1: B_i is a segment contained in $D_r(O)$. The segment $\overline{B}_i = B_i$ can be parameterized by:

$$(x(t), y(t)) = p_i + t(p_{i+1} - p_i), \quad t \in [0, 1] .$$

Then, it holds:

$$(x'(t), y'(t)) = p_{i+1} - p_i$$

and

$$I_i = \int_0^1 \det(p_i + t(p_{i+1} - p_i), p_{i+1} - p_i)dt = \det(p_i, p_{i+1}) .$$

Case 2: B_i is a circular arc contained in $D_r(O)$. The arc $\overline{B}_i = B_i$ can be parameterized by:

$$(x(t), y(t)) = (c_{i1} + R\cos(\theta_i + t), c_{i2} + R\sin(\theta_i + t)), \quad t \in [0, \beta_i] ,$$

where $c_i = (c_{i1}, c_{i2})$ is the center of the arc B_i, R is its radius, θ_i is the oriented angle between the x axis and the vector $\overrightarrow{c_i p_i}$, and β_i is the oriented angle between the vectors $\overrightarrow{c_i p_i}$ and $\overrightarrow{c_i p_{i+1}}$. Then, it holds:

$$(x'(t), y'(t)) = (-R\sin(\theta_i + t), R\cos(\theta_i + t)) ,$$

and

$$\begin{aligned} I_i &= \int_0^{\beta_i} c_{i1}R\cos(\theta_i + t) + c_{i2}R\sin(\theta_i + t) + R^2 dt = \\ &= c_{i1}R(\sin(\theta_i + \beta_i) - \sin(\theta_i)) - c_{i2}R(\cos(\theta_i + \beta_i) - \cos(\theta_i)) + \\ &+ R^2\beta_i = \det(c_i, p_{i+1} - p_i) + R^2\beta_i . \end{aligned} \tag{1}$$

Case 3: B_i is a pwc-curve exterior to $D_r(O)$ with endpoints p_i and p_{i+1}. Let p^0_{i+1}, p^1_{i+1}, ..., $p^k_{i+1} = p_{i+1}$ be the intersection points between the curve B_i and the half-line with origin O and direction vector $\overrightarrow{Op_{i+1}}$. The curve B_i can be divided into two parts B^1_i and B^2_i. The part B^1_i connects p_i with p^0_{i+1}, while the part B^2_i connects p^0_{i+1} with $p^k_{i+1} = p_{i+1}$. The second part B^2_i can be also divided into k pieces according to the points p^0_{i+1}, p^1_{i+1}, ..., $p^k_{i+1} = p_{i+1}$ (see Figure 4).

Let $B^{2,j}_i$ be the piece of B^2_i that connects p^j_{i+1} with p^{j+1}_{i+1}. Then, we have:

$$I_i = \int_{\overline{B}_i} -ydx + xdy = \int_{\overline{B}^1_i} -ydx + xdy + \int_{\overline{B}^2_i} -ydx + xdy = \int_{\overline{B}^1_i} -ydx + xdy + \sum_{j=0}^{k-1} \int_{\overline{B}^{2,j}_i} -ydx + xdy. \tag{2}$$

Consider now the simple closed curve determined by the union between $B^{2,j}_i$ and the oriented segment $s_{i,j} = p^{j+1}_{i+1} p^j_{i+1}$. Since this curve is external to the disk $D_r(O)$ and the projection $\overline{s}_{i,j}$ onto $D_r(O)$ of each $s_{i,j}$ vanishes to the point $\overline{p}_{i+1}$, for each j it holds that

$$0 = \int_{\overline{B}^{2,j}_i} -ydx + xdy + \int_{\overline{s}_{i,j}} -ydx + xdy = \int_{\overline{B}^{2,j}_i} -ydx + xdy + 0. \tag{3}$$

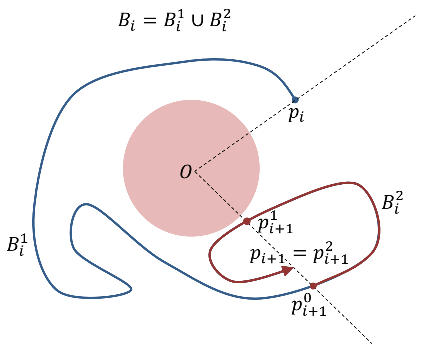

Figure 4: B_i subdivision process.

Consequently, from (2) and (3) we have:

$$I_i = \int_{\overline{B}_i} -ydx + xdy = \int_{\overline{B}_i^1} -ydx + xdy.$$

Let A_i be the shortest oriented arc on $D_r(O)$ that connects the point $\overline{p}_{i+1}$ with the point $\overline{p}_i$ and let $B_{i,i}$ be the closed curve determined by $B_i^1 \cup p_{i+1}^0\overline{p}_{i+1} \cup A_i \cup \overline{p}_i p_i$ (see Figure 5). Since B_i^1 does not cross the half-line with origin O and direction vector $\overrightarrow{Op_i}$, it holds that $B_{i,i}$ is simple. Then, I_i can be computed by:

$$I_i = \int_{\overline{B}_{i,i}} -ydx + xdy - \int_{A_i} -ydx + xdy\,,$$

since the radial projection onto $D_r(O)$ of the segments $p_{i+1}^0\overline{p}_{i+1}$ and $\overline{p}_i p_i$ are, respectively, the points $\overline{p}_{i+1}$ and $\overline{p}_i$.

The line integral over A_i is a particular case of (1) when $c_i = (0,0)$, $R = r$ and β_i is the oriented angle α_i between the vectors $\overrightarrow{Op_{i+1}}$ and $\overrightarrow{Op_i}$. Then, it holds:

$$\int_{A_i} -ydx + xdy = r^2\alpha_i\,.$$

The result of the line integral over $\overline{B}_{i,i}$ depends on the orientation of $B_{i,i}$ and on whether the disk $D_r(O)$ is interior or exterior to $B_{i,i}$. Let n_i be the number of intersections between B_i and the half-line h with origin O in the direction of the vector $-\overrightarrow{Op_i}$. According to n_i and α_i, there are three cases to consider (see Figure 5):

Case 3a: n_i is odd and $\alpha_i > 0$. Then, $D_r(O)$ is interior to $B_{i,i}$ and $B_{i,i}$ is oriented counterclockwise. Consequently,

$$\int_{\overline{B}_{i,i}} -ydx + xdy = 2\pi r^2 \qquad \text{and} \qquad I_i = r^2(2\pi - \alpha_i)\,.$$

Case 3b: n_i is odd and $\alpha_i < 0$. Then, $D_r(O)$ is interior to $B_{i,i}$ and $B_{i,i}$ is oriented clockwise. Consequently,

$$\int_{\overline{B}_{i,i}} -ydx + xdy = -2\pi r^2 \qquad \text{and} \qquad I_i = -r^2(2\pi + \alpha_i)\,.$$

Case 3c: n_i is even. Then, $D_r(O)$ is exterior to $B_{i,i}$. Consequently,

$$\int_{\overline{B}_{i,i}} -ydx + xdy = 0 \qquad \text{and} \qquad I_i = -r^2\alpha_i\,.$$

Thus, taking into account all the above, we compute the area $A(q)$ as follows. We initialize $A(q)$ to 0 and an exterior curve Γ to empty, and then the boundary edges, segments or arcs, of the pwc-region are sequentially processed. Given an edge e, first the change of origin of coordinate system is applied to its points. Next:

- if e is a segment exterior to $D_r(O)$ or e is an arc whose supporting circle is exterior to $D_r(O)$ then:

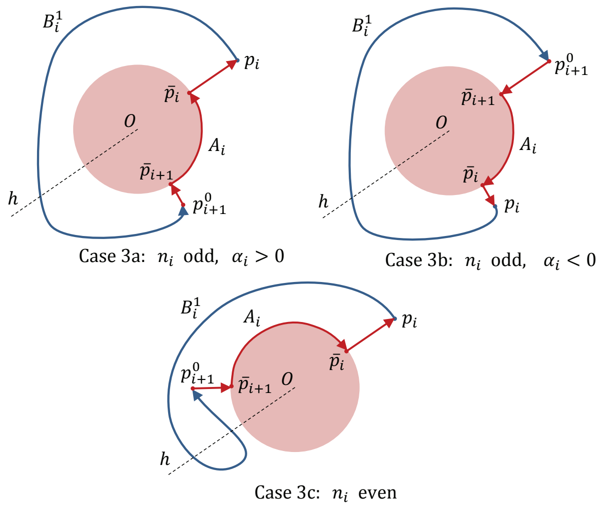

Figure 5: Exterior chain cases.

  - if Γ is empty or e does not intersect the half-line with origin at O that passes through the initial point of Γ, e is joined to Γ,
  - otherwise, the integral over Γ is added to $A(q)$ and Γ is initialised with e,
- otherwise, the integral over Γ is added to $A(q)$, Γ is initialised empty, e is split according to the intersection points with the boundary of $D_r(O)$, and the integral over each part of e is added to $A(q)$.

Finally, $A(q)$ is divided by 2.

3. Computing a set of ε-optimal locations

In this section we describe our approach to obtain an approximation of the set OL_ε of ε-optimal locations, which relies on area maps. We define the *area map* as the function that maps every point $q \in \mathbb{R}^2$ to the overlap area $A(q)$. Thanks to following lemma the search for ε-optimal locations can be reduced to the minimum bounding box of P. Consequently, the computation of an approximation of the area map can be restricted to this bounding box.

Lemma 1. *At least a location optimizing $A(q)$ belongs to the region delimited by the convex hull $CH(P)$ of the pwc-domain P. Consequently, it also belongs to the minimum bounding box of P.*

Proof. Assume that q_0 is an optimal location such that $q_0 \notin CH(P)$, and denote $\widetilde{q_0}$ the closest point to q_0 among all the points in $CH(P)$ (see Figure 6). Let L be the line through $\widetilde{q_0}$ which is orthogonal to the segment $\widetilde{q_0}q_0$. The line L subdivides the plane in two half-planes H_1 and H_2, being H_1 the half-plane that contains q_0. It holds that no point of $CH(P)$ is included in H_1. Suppose the contrary and let q be a point of $CH(P)$ included in H_1. Then, the segment $q\widetilde{q_0}$, which is included in $CH(P)$ because $CH(P)$ is convex, would intersect the disk centered at q_0 with radius $||q_0\widetilde{q_0}||$ which contradicts the fact that $\widetilde{q_0}$ is the point of $CH(P)$ closest to q_0. Consequently, we have that:

$$D_r(q_0) \cap CH(P) \subseteq D_r(q_0) \cap H_2 \subseteq D_r(\widetilde{q_0})\,. \tag{4}$$

Taking into account that

$$D_r(q_0) \cap P \subseteq D_r(q_0) \cap CH(P)\,, \quad D_r(q_0) \cap P \subseteq P$$

and (4) we also have that:

$$D_r(q_0) \cap P \subseteq D_r(\widetilde{q_0}) \cap P\,. \tag{5}$$

From (5) we conclude that $A(\widetilde{q_0}) \geq A(q_0)$. □

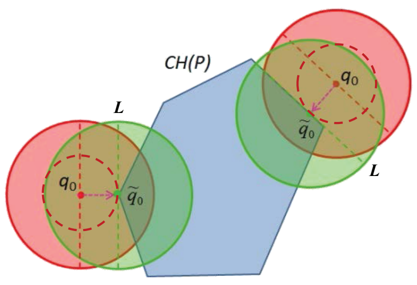

Figure 6: Observe that $D_r(q_0) \cap CH(P) \subseteq D_r(\widetilde{q_0})$.

In the following sections two different approximations of the area map depending on ε are obtained. First, a basic regular approach that uses a uniform grid provides a *uniform ε-area map* (section 3.1), then, a *nonuniform ε-area map* is obtained by locally refining an initial grid (section 3.2).

3.1. Uniform ε-area map

The minimum bounding box of the pwc-domain P is sampled with a uniform grid G_{d_ε} composed of square cells of side length d_ε. The length d_ε is chosen according to Theorem 1, which uses Lemma 2. The set of centers of the cells of G_{d_ε}, denoted by C_{d_ε}, is used to obtain the discrete graph defining the uniform area map

$$UAM_\varepsilon = \{(c, A(c)) \,|\, c \in C_{d_\varepsilon}\}\,.$$

Let $\overline{A}_\varepsilon$ be the maximal area over the centers of C_{d_ε}, and

$$OC_\varepsilon = \{c \in C_{d_\varepsilon} \,|\, A(c) = \overline{A}_\varepsilon\}\,.$$

By Theorem 1, if $c \in OC_\varepsilon$ the absolute error $A(c) - \overline{A}$, the difference between the value $A(c)$ and the optimal overlap area value assuming that the center can be placed anywhere, is smaller or equal than $\varepsilon > 0$. That is, all the centers in OC_ε are ε-optimal locations.

Lemma 2. *Given two points q_1 and q_2 satisfying $\|q_1q_2\| \leq 2r$, then $|A(q_1) - A(q_2)| \leq 2r\|q_1q_2\|$.*

Proof. If $\|q_1q_2\| \leq 2r$, we have that $D_r(q_2) \cap P$ is included in the union of $D_r(q_1) \cap P$ and the lune $L(q_2, q_1) = D_r(q_2) \setminus D_r(q_1)$ (see Figure 7). Thus,

$$A(q_2) \leq A(q_1) + \text{area}(L(q_2, q_1))\,.$$

Moreover, taking $h = \|q_1q_2\|$ and by using Taylor expansion, we have:

$$\text{area}(L(q_2, q_1)) = 2r^2 \arcsin\left(\frac{h}{2r}\right) + \frac{h}{2}\sqrt{4r^2 - h^2} \leq 2rh\,.$$

Consequently $A(q_2) \leq A(q_1) + 2r\|q_1q_2\|$ and, by interchanging q_1 and q_2, $A(q_1) \leq A(q_2) + 2r\|q_1q_2\|$. □

Theorem 1. *Given $\varepsilon > 0$, if the grid side length d_ε satisfies*

$$d_\varepsilon \leq \min\left\{2\sqrt{2}r, \frac{\varepsilon}{\sqrt{2}r}\right\},$$

then the absolute error $\overline{A} - A(c) \leq \varepsilon$, where $c \in OC_\varepsilon$.

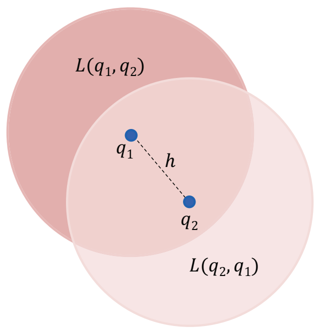

Figure 7: Lunes $L(q_1, q_2)$ and $L(q_2, q_1)$.

Proof.

Given a point q in a cell of center c and side length d, we have that $\|qc\| \leq d/\sqrt{2}$. Then, if d is taken satisfying $d \leq 2\sqrt{2}r$, accordingly to Lemma 2 we have:

$$A(q) \leq A(c) + 2r\|qc\| \leq A(c) + \sqrt{2}dr\,. \tag{6}$$

Consequently, fixed $\varepsilon > 0$, and according to (6), an ε absolute error can be guaranteed by choosing the side length d_ε of a square grid cell satisfying:

$$d_\varepsilon \leq \min\left\{2\sqrt{2}r, \frac{\varepsilon}{\sqrt{2}r}\right\},$$

because, in such a case:

$$\overline{A} = A(q) \leq A(c_q) + \sqrt{2}d_\varepsilon r \leq \overline{A}_\varepsilon + \varepsilon = A(c) + \varepsilon\,,$$

where q is an optimal location, c_q is the center of the cell containing q and $c \in OC_\varepsilon$.

□

3.2. Nonuniform ε-area map

Using the grid G_{d_ε} on the entire bounding box of P wastes many grid cells in areas where are not necessary. Moreover, it may be computationally demanding in time and memory requirements. Using a global refinement method would have the same problems, thus, we propose a local grid refinement method that starts with a coarser grid of side length $d_0 > d_\varepsilon$ and refines only some specific cells until the required d_ε precision is achieved.

The local grid refinement strategy identifies the cells to be refined, named *parent cells*, according to lemma 3 and lemma 4 which hold for $d \leq \sqrt{2}r$. They provide a filtering criterium to quickly detect the *terminal cells* that should not be refined because their points: i) can not be ε-optimal locations (Lemma 3) or ii) are all ε-optimal locations for being ideal locations, i.e. their overlap area achieves the πr^2 upper bound (Lemma 4).

Lemma 3. *Let $\overline{c}_0$ be a cell center that along the refinement process provisionally maximizes the overlap area. If $d_\varepsilon \leq d \leq 2\sqrt{2}r$ and $A(c) + \sqrt{2}dr < A(\overline{c}_0) - \sqrt{2}dr$ for the center c of a grid cell g, then no point of g is an ε-optimal location.*

Proof. From (6) we know that given a point $q \in g$ we have:

$$A(q) \leq A(c) + \sqrt{2}dr\,,$$

and consequently:

$$A(q) < A(\overline{c}_0) - \sqrt{2}dr \leq \overline{A} - \sqrt{2}dr\,.$$

Then, since d fulfills that $d_\varepsilon \leq d \leq 2\sqrt{2}r$, it holds:

$$\overline{A} - A(q) > \sqrt{2}dr \geq \sqrt{2}d_\varepsilon r \geq \varepsilon\,.$$

Consequently, no point $q \in g$ can be an ε-optimal location. □

Lemma 4. *Assume that $D_r(c)$ covers the grid cell g, i.e. $d \leq \sqrt{2}r$, and that c is an ideal location, i.e. $A(c) = area(D_r(c) \cap P) = area(D_r(c)) = \pi r^2$. Then, if each center c' of each one of the eight cells adjacent to cell g is also an ideal location, any point of the cell g is an ideal location.*

Proof. The union F of the set of all disks of radius r whose center belongs to the cell g is the r-offset of the cell g. Observe in Figure 8 that, when $d \leq \sqrt{2}r$, the union of the eight disks of radius r centered at the center of the grid cells adjacent to cell g together with the disk $D_r(c)$ is a simply connected set that covers the r-offset F. Consequently, since the union of the eight disks is included in P, the r-offset F is also included in P and any point of the cell g is an ideal location. □

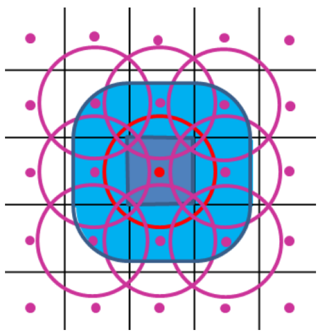
Figure 8: The union of the nine disks covers the cell offset.

To obtain the *nonuniform ε-area map $NUAM_\varepsilon$* several refinement steps are used. An initial coarse grid G_{d_0} with $d_\varepsilon < d_0 \leq \sqrt{2}r$ is considered, its cells centers define the set of possible locations C_{d_0}. The overlap area associated to these points is computed and the terminal and parent cells of G_{d_0} are identified according to lemmas 3 and 4. Next the i-refinement step, in this case for $i = 1$, starts. A new length d_i with $d_\varepsilon \leq d_i < d_{i-1}$ is determined and a *child grid* G_{d_i} with cells of side length d_i is placed on each parent cell. To be able to use Lemma 4 and classify the cells of G_{d_i}, the grid is enlarged adding two auxiliary rows and columns surrounding it. We consider C_{d_i} the set of all the centers providing from the grids G_{d_i} and $\widetilde{C}_{d_i}$ the enlarged set of centers containing also those providing from the auxiliary cells. The overlap area of the disks centered at the points in $\widetilde{C}_{d_i}$ are computed. Meanwhile, the terminal and parent cells for the next $(i+1)-$refinement step are selected from the centers in C_{d_i}. The refinement process ends at the $S-$refinement step when $d_S \leq d_\varepsilon$ in which case an ε-absolute error is guaranteed. The graph defined by the terminal cell centers detected at the i-refinement step $(0 \leq i \leq S)$ and their overlap areas is denoted by $NUAM_i$. Similarly, the set of terminal optimal locations obtained until step i is denoted OC_i. At the end of the process $NUAM_S$ is the $NUAM_\varepsilon$ and OC_S is the desired OC_ε.

3.3. Approximating the set of ε-optimal locations and visualizing the ε-area map

The set of ε-optimal locations is approximated by the union of the cells falling in one of the next two cases:

Case 1: The center c of the cell is a ε-optimal location, i.e. $A(c) = \overline{A}_\varepsilon$.

Case 2: The center c of the cell is not a ε-optimal location, but the cell may contain ε-optimal locations, i.e.

$$A(c) + \varepsilon \geq \overline{A}_\varepsilon - \varepsilon\,.$$

Furthermore, to improve the understanding of the obtained solutions, we visualize the obtained area map approximation (Figure 9) by painting:

- the centers in Case 1 in red if they are ideal solutions and in orange otherwise.
- the centers in Case 2 in green.

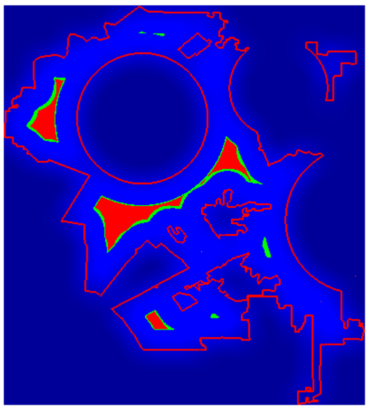

Figure 9: Image of a uniform ε-area map. The set OC_ε contains ideal solutions and is painted in red. The cells that may contain ε-optimal locations are painted in green, the rest of analyzed cells in a blue.

- the rest in a blue color gradation according to the overlap area of each location, the darker the blue a point is colored the smaller its overlap area is.

The visualization of the area map approximation facilitates the task of the decision makers. It provides locations for a new facility that according to ε are considered good enough. This allows them to can take into account additional criteria (availability, suitability of the environment or installation costs) to choose the best one. The approach permits using an iterative what-if-analysis process in which the absolute error ε or the number of refinement steps used to obtain the $NUAM_\varepsilon$ can be changed.

In fact, the differences between the area map obtained from the UAM_ε and the $NUAM_\varepsilon$ can also be easily seen visualizing them (Figure 10). It intuitively shows how the refinement process achieves the same accuracy requiring less memory space and execution time (which is also theoretically and experimentally proved in Section 4.3 and Section 5).

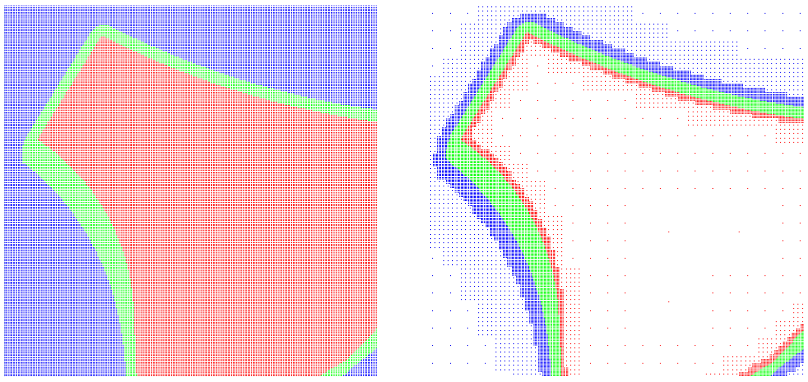

Figure 10: Uniform area map (left) versus nonuniform area map (right).

4. Algorithms description

In this section we describe a sequential and a parallel algorithm to obtain the ε-overlap area maps and their ε-optimal locations, together with an algorithm to visualize them. We first sketch the sequential algorithm and then describe in detail the GPU parallel one, we also provide their complexity analysis and their theoretical comparison.

For simplicity in the writing, we sometimes identify a cell with its center. Hence even though the sets C_{d_i} and $\widetilde{C}_{d_i}$ contain cell centers we may talk about *the cells in* C_{d_i} or $\widetilde{C}_{d_i}$ when referring to the cells whose centers are contained in C_{d_i} or $\widetilde{C}_{d_i}$.

4.1. Sequential implementation

The sequential algorithm proceeds, according to the results of Sections 2 and 3, as follows (for further details, specially about lines **2)**, **3)**, **6)**, **7)** and **9)** see Section 4.2):

Algorithm 1: *Sequential algorithm*

```
1)  store $P$ and compute its bounding box
2)  determine $d_0$ with $d_\varepsilon \leq d_0 \leq \sqrt{2}r$
3)  obtain the initial grid $G_{d_0}$ and the cell centers $C_{d_0}$
4)  compute $A(c)$, the overlap area of $P$ and the disks centered at $c$, for each $c \in C_{d_0}$
5)  while $d_i > d_\varepsilon$ with $0 \leq i \leq S$ do
6)      determine $d_{i+1} < d_i$
7)      classify the cells in $C_{d_i}$
8)      identify the parent cells of $C_{d_i}$
9)      obtain $C_{d_{i+1}}$ and $\widetilde{C}_{d_{i+1}}$
10)     compute $A(c)$ for each $c \in \widetilde{C}_{d_{i+1}}$
11)     increment $i$
12) endwhile
13) visualize the obtained results
```

The bottleneck of this sequential algorithm is the computation of the overlap area of P with the considered disks. To exactly compute the overlap area of the pwc-domain P and a given disk it is necessary to determine which parts of the disk covers P, and this is not possible by only finding the intersection points of P with the disk. Thus, we have to traverse P and, consequently, is not possible to use indexed structures, like regular grids or quad-trees, to avoid considering some parts of P. However, the computation of these overlap areas can be fully parallelized, as well as the classification of the analyzed cells and the obtention of the child cell centers. Thus, we next present the parallel version of this sequential algorithm.

4.2. Parallel implementation

The parallel version of the algorithm also uses the results presented in Sections 2 and 3. The GPU-parallel algorithm is sketched in the following pseudocode, where the steps shared with the sequential version are marked in italics, and explained in detail next.

Algorithm 2: *Parallel GPU algorithm*

```
1)  store P and compute its bounding box
2)  transfer $P$ to the GPU
3)  determine $d_0$ with $d_\varepsilon \leq d_0 \leq \sqrt{2}r$
4)  obtain the initial grid $G_{c_0}$ and the cell centers $C_{d_0}$
5)  transfer $C_{d_0}$ to the GPU
6)  compute $A(c)$ for each $c \in C_{d_0}$ (parallel)
7)  transfer the area information to the CPU
8)  delete $C_{d_0}$ from the GPU
9)  while $d_i > d_\varepsilon$ with $0 \leq i \leq S$ do
10)     determine $d_{i+1} < d_i$
```

```
11)     classify the cells in $C_{d_i}$ (parallel)
12)     identify the parent cells of $C_{d_i}$ (CPU)
13)     transfer the parent cells centers to the GPU
14)     obtain $C_{d_{i+1}}$ and $\widetilde{C}_{d_{i+1}}$ (parallel)
15)     compute $A(c)$ for each $c \in \widetilde{C}_{d_{i+1}}$ (parallel)
16)     transfer the computed areas and centers to the CPU
17)     delete $\widetilde{C}_{d_{i+1}}$ from the GPU
18)     increment i
19) endwhile
20) delete $P$ from the GPU
21) visualize the obtained results
```

To describe the algorithm in detail it is subdivided in the following steps:

A) *Pwc-domain P storage*: (lines **1)**, **2)**);
B) *G_{d_0} and C_{d_0} determination* (lines **3)-5)**);
C) *Parallel overlap areas computation* (lines **6)-8)**, **15)-17)**);
D) *d_{i+1} determination* (line **10)**);
E) *Parent cells identification* (lines **11)-13)**);
F) *C_{d_i} and $\widetilde{C}_{d_i}$ determination ($i > 0$)* (line **14)**);
G) *Results visualization* (line **21)**).

Next we remark the main issues of each of these steps.

A) Pwc-domain P storage

The vertices of P are stored in pwc-curve order (see Figure 11). Two consecutive vertices are either adjacent in P or correspond to two different pwc-curves. We first store the vertices defining the outer regions and then those defining the holes. Two consecutive vertices of the same pwc-curve define an edge, and the first vertex of each pwc-curve is also repeated as its last vertex. We also store, in vectors, the position of the first vertex of each pwc-curve and for each circular arc: the angle it defines and the radius and center of its supporting circumference. To know whether an edge is a line segment o a circular arc we use an array of integers. If v_i is the starting point of a line segment edge or the ending vertex of a pwc-curve it stores a -1. An integer value $j \geq 0$ indicating the position, in the corresponding vectors, of the information associated to the circular arc going from v_i to v_{i+1} is stored otherwise.

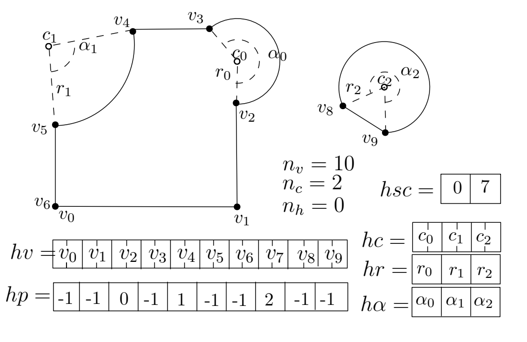

Figure 11: Pwc-domain representation

To solve the problem in the GPU we transfer the pwc-domain P there. We use: *hv* the array of real values storing the vertex coordinates; *hsc* the integer array storing the starting position of each pwc-curve; n_v, n_c and n_h, three integer values with the number of vertices, enclosed pwc-curves and holes, respectively; *hp* the integer array to store the integers encoding whether the edge from v_i to v_{i+1} is a line or a circular segment; and three extra real arrays *hr*, *hc* and

$h\alpha$, with the radii, centers and angles of the circular arcs, respectively. The center of the supporting circumference of a circular arc from v_i to v_{i+1} is $(hc[2j], hc[2j+1])$, its radius $hr[j]$ and the angle defined by its arc is $h\alpha[j]$, wherever $j = hp[i] \geq 0$. Let $\mathcal{H}_\mathcal{P}$ denote all these GPU-arrays and integers needed to represent and store P in the GPU.

B) G_{d_0} and C_{d_0} determination

Let $k_a \times k_b$ be the size of the initial coarse grid. The integers k_a and k_b are at least 2 and are determined so that the grid has squared cells with side length $d_0 \leq \sqrt{2}r$. We can start with $d_0 = \sqrt{2}r$. Since the grid should cover the $a \times b$ minimum bounding box of P, we take $k_a = \lceil a/d_0 \rceil$ and $k_b = \lceil b/d_0 \rceil$. We center the pwc-domain in the grid and determine the grid cell centers defining C_{d_0} (see Figure 12). The centers coordinates are stored in an array, traversing the grid in a row first fashion. After obtaining them in the CPU, they are transferred to the GPU, where are stored in a real vector hq of size $nq = 2k_ak_b$.

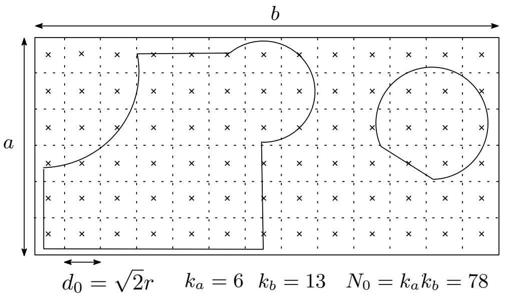

Figure 12: G_{d_0} centered at P, the centers of C_{d_0} marked in crosses.

C) Parallel overlap areas computation

This computation is required twice along the algorithm (lines **6)** and **15)**). The aim is to compute the areas $A(q) = D_r(q) \cap P$, with $q \in C_{d_0}$ in line **6)** and $q \in \widetilde{C}_{d_i}$ for $i \geq 1$ in line **15)**. Let Q denote the corresponding set of centers and hq of size n_q the GPU-vector storing Q.

The areas are parallelly computed using a thread per point $q \in Q$ and a kernel that has as input hq, nq, r and $\mathcal{H}_\mathcal{P}$, and as output two real arrays ha and ha_M of size nq and 1, respectively. Array ha contains the computed areas and ha_M the obtained optimal value. The thread with global index id_g is associated to the point $q \in Q$ with coordinates $(hq[2id_g], hq[2id_g+1])$. To compute $A(q)$ the thread traverses P as is explained in Section 2. When all the edges of P have been considered, the obtained overlap area is stored in $ha[id_g]$. Then, $ha_M[0]$ is updated with an atomic operation whenever the already obtained optimal area has not reached the ideal value πr^2.

Since each thread analyzes all the vertices of P, it is worth to use shared memory to let the threads cooperate to transfer $\mathcal{H}_\mathcal{P}$ from the global to the shared memory. Being B the block size used, storing $\mathcal{H}_\mathcal{P}$ in the shared memory requires $6B$ real and B integer values of shared memory per block. The number of pwc-curves defining P, hc, is usually smaller than B. Hence, the first hc threads of the block, those whose local index id_l fulfills $0 \leq id_l < hc$, store hsc in the shared memory. Then, to start with, each thread transferees a vertex to shared memory. It stores $hv[id_l]$ and $hp[id_l]$, and if $j = hp[id_l] \neq -1$ also $hc[j]$, $hr[j]$ and $h\alpha[j]$ to the id_l position of the corresponding vectors in the shared memory. When all these vertices of P have been analyzed by all the threads, the information of the next B vertices of P is transferred. It obviously requires two synchronization points, one after transferring the vertices to the shared memory and the other after processing the transferred vertices.

When the kernel ends, hq is deleted from the GPU and ha and ha_M are transferred to the CPU but not deleted from the GPU because they are used later.

D) d_{i+1} determination

If $d_i > d_\varepsilon$ a new refinement step is required and d_{i+1} has to be determined, which is equivalent to choose the size the child grid $G_{d_{i+1}}$. Since the cells to refine are squared cells, we use child grids of size $k_{i+1} \times k_{i+1}$ with $2 \leq k_{i+1} \leq k_{GPU}$, k_{GPU} is a GPU-depending constant discussed in step *E)* and in Section 5.

The value of k_{i+1} can be provided by the user or computed according to the refinement needs, if possible, a prefixed number S of as regular as possible refinement steps. In the latter case, k_{i+1} should be $\lceil \sqrt[S]{d_0/d_\varepsilon} \rceil$ truncated to the interval $[2, k_{GPU}]$. Hence, the value d_{i+1} is set to be d_i/k_{i+1}. However, when d_{i+1} becomes smaller than d_ε, i.e. $i + 1 = S$, we take $k_S = \lceil d_{S-1}/d_\varepsilon \rceil$ and $d_S = d_{S-1}/k_S$.

E) Parent cells identification

When the $(i + 1)$-refinement step is required, the parent cells of C_{d_i} need to be identified. They are obtained after classifying all the cells of C_{d_i} into terminal or parent cells by using the results of Section 3.2.

Accordingly, the classification uses all the areas computed at i-refinement step which are those corresponding to C_{d_0} for $i = 0$ and to $\widetilde{C}_{d_i} \supset C_{d_i}$ for $i > 0$. Two kernels proceeding slightly different are used depending on whether $i = 0$ or $i > 0$. However, both classify the cells in C_{d_i} in parallel. Each thread considers a cell of C_{d_i} and classifies it by storing, in the integer array hr of size $|C_{d_i}|$, a 0, 2 or 1 depending on whether the cell can not contain any ε-optimal location (Lemma 3), is ideal (Lemma 4) or should be refined (see Figure 13). Both kernels have as output hr and as input the computed areas ha, the size of ha (i.e. $|C_{d_0}|$ or $|\widetilde{C}_{d_i}|$ $i > 0$), the dimensions of G_{d_i} and two real values a_D and a_I. The values a_D and a_I are the threshold values of Lemmas 3 and 4, i.e. $a_D = \overline{A}_i - 2\sqrt{2}d_i r$, where $\overline{A}_i$ is the optimal area obtained at the i-refinement step, and $a_I = \pi r^2$.

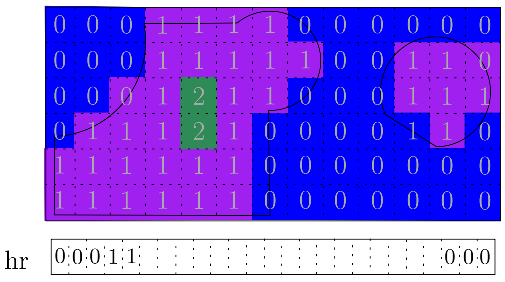

Figure 13: Cells classification into terminal (discarded/ideal) cells (labeled with a 0 or 2) or parent cells (labeled with a 1).

The kernel used in the first refinement step, $i = 0$, which analyzes the cells of the initial $k_a \times k_b$ grid G_{d_0}, is called associated to two dimensional CUDA-grid of threads. A two-dimensional CUDA-grid is used so that the threads in a block share as much areas as possible when checking Lemma 4 (see Figure 14). The thread with global

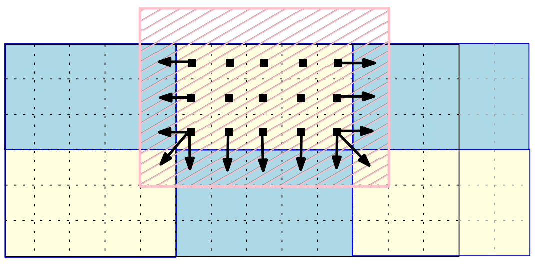

Figure 14: Cells classification kernel for i=0. A 6×15 two-dimensional CUDA-grid with 3×5 CUDA-blocks. The areas transferred to shared memory by the threads (squares) of a block dashed.

identifier (idx_g, idy_g), within the two-dimensional CUDA-grid, considers the cell (idx_g, idy_g) of the $k_a \times k_b$ grid G_{d_0} and classifies it. Its area is stored in $h_a[pos]$ with $pos = idx_g k_b + idy_g$. Hence, the cell can not contain optimal points when $h_a[pos] < a_D$; it is ideal if $h_a[pos] = a_I$ and its 8 neighbors too; or a parent cell otherwise. When $\overline{A}_i$ achieves the ideal value, Lemma 4 is used and the areas are transferred from the global to the shared memory. If $B' \times B''$ is the size of the two-dimensional block used, the existent $(B' + 2) \times (B'' + 2)$ potentially used areas in a block are transferred to the shared memory (dashed region of Figure 14). Each thread transferees the area of its cell and the threads of the block boundary also transferee the areas of its existing exterior neighbors (represented in arrows in Figure 14). Obviously, a synchronization point after transferring the area is needed.

The kernel used in the rest of the i-refinement steps ($i \geq 1$) is called with a one-dimensional CUDA-grid of $\widetilde{C}_{d_i}$ threads. Each block becomes responsible of $\mu \geq 1$ complete child grids (see Figure 15), this is the reason why k is upperbounded by the GPU-depending constant k_{GPU}. Each thread transferees an area value to shared memory. After

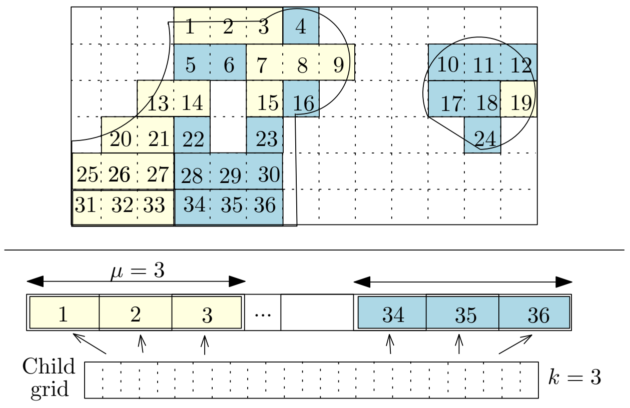

Figure 15: Cells classification kernel for $i = 1 > 0$. The parent cells of G_{d_0} enumerated above and grouped per blocks ($\mu = 3$). Below the representation of *ha* in the GPU, $k = 3$.

transferring the areas, only the threads whose cell belongs to C_{d_i} classify the cell and store the corresponding 0, 1 or 2 in *hm*.

Once the kernel has classified all the cells, *ha* is deleted from the GPU and *hm* is transferred to the CPU where the resultant integer vector is traversed, the parent cells are extracted and their centers are stored in an array. The parent cells centers are used in the next step where the child grids are build and the classification of the cells in C_{d_i} is reused later in the visualization process.

F) C_{d_i} and $\widetilde{C}_{d_i}$ determination ($i > 0$)

The centers of the already identified parent are transferred to the GPU where are used to center, in each of them, a new child grid of size $k_{i+1} \times k_{i+1}$. In the case that the ideal area has been reached, i.e. $\overline{A}_i = \pi r^2$, the grid G_{d_i} is enlarged adding the extra rows and columns mentioned in Section 3.2 (see Figure 16). Thus, a $k \times k$ grid with $k \in \{k_{i+1}, k_{i+1} + 2\}$

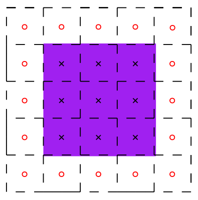

Figure 16: A child grid of size $k = 3$, with its real (black crosses) and auxiliary (red circles) cell centers.

is centered in each of the R_i parent cells centers. It is done with $R_i k^2$ threads, and hence, each thread determines a point of $\widetilde{C}_{d_{i+1}}$. The first k^2 threads build the child grid centered at the first parent cell, the next k^2 those centered at the second one, and so one. The child centers are stored in the resized array *hq* which now has size $R_i k^2$. The cells centers of the same child are stored in consecutive positions by traversing the child grid in a row first fashion (see Figure 15). The set $\widetilde{C}_{d_{i+1}}$ is maintained in the GPU-array *hq*, but also transferred to the CPU where is also needed.

G) Results visualization

To visualize the solution, we paint the center of the terminal cells obtained along the process. Hence, not all the analyzed points (Figure 17 a) are visualized (Figure 17 b).

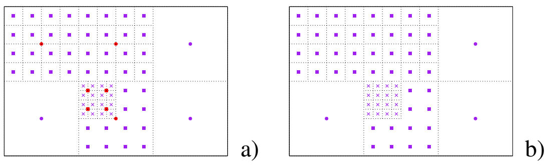
Figure 17: Centers of the analyzed: a) real cells, b) terminal cells.

4.3. *Complexity analysis*

In this section, we provide the complexity analysis of the sequential and the parallel strategy concerning both the worst case time and the space complexity analysis. Finally, we theoretically compare the time and space complexity of the parallel and the sequential algorithm which is, by now, the best known sequential algorithm to solve this problem. Since the details have been provided for the parallel version of the algorithm, we start by analyzing in detail the complexity of the parallel algorithm and next use the observations done to give the complexity of the sequential one.

Parallel algorithm complexity analysis

First each kernel and refinement step are analyzed separately and then the whole algorithm analysis is provided. Let N_i denote $|C_{d_i}|$ for $i \geq 0$, hence, $N_0 = k_a k_b$ and $N_i = R_i k_i^2$ for $i > 0$. Note that when $\overline{A}_{i-1}$ is the ideal area, $4(k_i+1)R_i$ extra auxiliary cells are used in the area computation, however the number of considered cells is in $O(N_i)$.
Time complexity analysis: The time complexity analysis of a parallel algorithm usually analyzes the *worst case total work* done by the algorithm, which is the number of executed operations over all the processors or threads in the worst case, and the *execution time of the algorithm* when p threads are used.

We start analyzing the worst case total work done at the refinement step i.

The parallel work involves that done by the kernels used in the: i) *Parallel overlap areas computation*. It is executed by as many threads as points has to analyze. Each thread computes an area $A(c)$ analyzing the n_v edges of P in $O(1)$ time per edge (note that P has as many edges as vertices) doing $O(n_v)$ work. Since $O(N_i)$ points are analyzed, the total work done by this kernel is of $O(n_v N_i)$. Its execution time when p_i threads are used is of $O(n_v N_i/p_i)$ where $p_i = O(N_i)$. However, when $\overline{A}_i$ is not the ideal area $O(N_i)$ atomic operations are done. ii) *Parent cells identification* which classifies the cells in $C_{d_{i-1}}$ with $O(N_{d_{i-1}})$ threads. Each thread considers one grid cell and does at most $9 = O(1)$ checks. Hence, the kernel does $O(N_{i-1})$ total work using N_{i-1} threads doing $O(1)$ work each. Its execution time when p_{i-1} threads are used is $O(N_{i-1}/p_{i-1})$ with $p_{i-1} = O(N_{i-1})$. iii) C_{d_i} *and* $\widetilde{C}_{d_i}$ *determination*. It $\widetilde{C}_{d_i}$ by using one thread per point in $\widetilde{C}_{d_i}$ doing $O(1)$ work each. Again its execution time when $p_i = O(N_i)$ threads are used is of $O(N_i/p_i)$.

The sequential work consists of: i) determining d_i and k_i which is done in $O(1)$ and ii) selecting the parent cells using the obtained classification which takes $O(N_{i-1})$ time.

According to this analysis we can state that the:

Total work done in parallel to handle the *ith* refinement step is $w_0 = O(n_v N_0)$ and $w_i = O(N_{i-1} + n_v N_i)$ for $i > 0$. Meanwhile $O(N_{i-1})$ work is done in the CPU. Thus, $w_0 = O(n_v N_o)$ and $w_i = O(N_{i-1} + n_v N_i)$ for $i > 0$. Considering the whole refinement process which has S refinement steps, the total work is of

$$w = \sum_{i=0}^{S} w_i = O(n_v N_0) + \sum_{i=1}^{S} O(N_{i-1} + n_v N_i) =$$

$$= \sum_{i=0}^{S} O(n_v N_i) + \sum_{i=0}^{S-1} O(N_i) = \sum_{i=0}^{S} O(n_v N_i) = O(n_v N).$$

Obtaining an accurate upper-bound of N_i is really difficult. Denoting by d_i the length of the grid cell considered at step i, we can state that $N_i = O(\lceil(\lceil a/d_i\rceil)(\lceil b/d_i\rceil)$. But, as it is shown in Section 5, in real applications it is much smaller. Thus, the total work done by the whole algorithm is $w = O(n_v N)$.

Execution time of the parallel part of our algorithm is $t_0 = O(n_v N_0/p_0)$ and $t_i = O(N_{i-1}/p_{i-1} + n_v N_i/p_i)$ for $i > 0$ with $p_i = O(N_i)\ \forall i \geq 0$. Consequently, in the best case it is $O(n_v)$. Thus, considering the S refinement steps, the execution time of the parallel part of our algorithm becomes

$$t_p = O\left(n_v \sum_{i=0}^{S} \frac{N_i}{p_i} + \sum_{i=0}^{S-1} \frac{N_i}{p_i}\right) = O\left(n_v \sum_{i=0}^{S} \frac{N_i}{p_i}\right).$$

In the best case, when $p_i = O(N_i)$, it becomes $O(n_v S)$.

When the part solved in the CPU is also considered the execution time becomes $O(N_0 + n_v N_0/p_0)$ for the first step and $O(N_{i-1} + N_{i-1}/p_{i-1} + n_v N_i/p_i)$ for $i > 0$.
Hence, we finally obtain that the execution time of the whole algorithm is

$$t_p = O\left(\sum_{i=0}^{S} N_i + n_v \sum_{i=0}^{S} \frac{N_i}{p_i}\right) = O\left(N + n_v \sum_{i=0}^{S} \frac{N_i}{p_i}\right)$$

which in the best case becomes $O(N + n_v S)$.

Space complexity analysis: Storing the domain requires $2n_v + 4n_c$ real and $n_v + n_c + 3$ integer values in the global memory, where n_c is the number of pwc-curves. The first kernel also uses $6B$ real and B integer values in the shared memory per block and $3N_i$ real values to store the centers coordinates and the computed overlap areas. Thus, it has a space complexity of $O(n_v + n_c + N_i) = O(n_v + N_i)$. Considering the whole refinement process, the maximal amount of memory used in the first kernel is $O(n_v + N_{max})$, where $N_{max} = \max\limits_{i=0\cdots S} N_i$.

The second kernel does not use P nor the centers coordinates, but, since the second kernel is called whenever a new refinement step is needed we do not delete P from the global memory. Thus, P is transferred only once to the GPU and used when it is needed. On the other hand, the center coordinates are deleted from the global memory as soon as the first kernel ends because they are not used again. Hence, in the second kernel we need the already $O(N_{i-1})$ computed areas and N_{i-1} integer values to classify the cell centers. Thus the amount of memory needed is $O(N_{i-1})$. It also uses shared memory, the amount of shared memory needed in a block with B threads is $O(B)$. During the whole refinement process it requires $O(N_{max} + B)$ memory.

The third kernel needs the R_i parent cell centers obtained from the N_{i-1} analyzed cells and computes the new $O(N_i)$ cell centers. Thus, since $N_i = O(k_i^2 R_i)$, the amount of memory needed at time step i is $O(R_i + N_i) = O(N_i)$. During the whole refinement process it requires $O(N_{max})$ memory.

Concerning the information stored in the CPU, we maintain P using the same amount of global memory used in the GPU, $O(n_v)$, and the coordinates of all the analyzed $O(N)$ cell centers with their area and classification (when it is obtained). It represents $O(N)$ extra information.

Sequential algorithm complexity analysis

After analyzing the complexity of the parallel algorithm, we can easily obtain the time complexity of the sequential version. As before, the time complexity of the first refinement step is $O(n_v N_0)$ and for $i > 0$ it is $O(N_{i-1} + n_v N_i)$. When all the refinement process is considered the time complexity becomes again $t = O(n_v N)$. Note that since the areas are computed exactly both in the parallel and the sequential algorithm the refinement process will lead to the same analyzed N centers.

Theoretical comparison between the parallel and the sequential algorithm

The standard measures to theoretically compare a parallel and a sequential algorithm are the parallel speedup and the work efficiency of the parallel algorithm. These two measures are referred to the time complexity t of the best known sequential algorithm to solve the problem and the execution time t_p and the total work w of the parallel algorithm, respectively.

These measures are used to compare our parallel and sequential algorithms. Note that the proposed sequential algorithm is the best known sequential algorithm to solve the problem and thus we are fairly comparing the parallel algorithm versus the sequential one.

Work efficiency: If w is the total work of the parallel algorithm, the work efficiency is defined as the ratio $\frac{w}{t}$. In our case, it is

$$\frac{w}{t} = O(n_v N/n_v N) = O(1).$$

It means that all the work done in the parallel strategy is also done in the sequential one which is the optimal work efficiency.

Theoretical parallel speedup: The theoretical parallel speedup of a parallel algorithm with execution time complexity t_p is the ratio $\frac{t}{t_p}$. In our case

$$\frac{t}{t_p} = O\left(\frac{n_v N}{\frac{n_v N}{p} + N}\right) = O\left(\frac{p n_v}{n_v + p}\right).$$

Note that this ratio is due to the fact that each refinement step is partially solved sequentially.

The parallel speedup of the part that is solved in parallel within each refinement step i is $\frac{t_i}{t_{p_i}} = O\left(\frac{nN_i}{n_v N_i/p_i}\right) = O(p_i)$. If we consider the best parallel case were the number of threads is maximal, then $p_i = N_i$ and $\frac{t_i}{t_{p_i}} = O\left(\frac{n_v N_i}{n_v}\right) = O(N_i)$. Which means that the speedup is the number of analyzed centers because all the work is done in parallel.

Space complexity comparison: The space complexity of both the sequential and parallel algorithm is $O(n_v + N)$. Thus, the space complexity of the parallel algorithm is again optimal.

5. Experimental results

This section starts with the experimental setting, continues with the interaction process with the user and the results visualization, and ends analyzing the provided algorithms in terms of running times. The presented running times show that: the provided parallel algorithm is fast and robust and much better than the best sequential algorithm; using a refinement process instead of starting with a finer grid of size d_ε makes sense; that using the algorithm of Section 2 to compute the areas is worth. We do not compare our results with those of the previous existent algorithms, because, as we said, it is the first algorithm designed to solve this problem. Existent studies do not consider partial coverage and most of them are designed to place more than one facility. Hence the comparison of our algorithm with the existent ones would be completely unfair.

5.1. Experimental setting

The algorithms have been implemented in C++, for the parallel part Cuda C has been used and the visualization is done by using OpenGL. The running times presented in this section have been obtained using a Inter(R) Core(TM) i7-4790CPU with a Tesla k40 active GPU.

We have considered several ε, r, S and f values and a pwc-domain. The used domain is extracted from a polygonal chain representing Los Angeles city the second-largest city in the United States[1]. This city is embedded in a bounding box of $47km$ of width and $71km$ of height and its perimeter is of about $550km$. The original polygon was defined by 4943 edges determining one single component with 8 holes, see Figure 18 a). We have assumed that the city has been partially covered by four disks of different radii which have been extracted from the domain. It has yield to the pwc-domain of Figure 18 b) which is the one we consider and refer as 'LA domain' along this section. This pwc-domain is defined by 3569 edges and has 8 pwc-curves, two are outer components defining two different regions and the rest define six holes of the bigger component. The bonding box containing 'LA domain' measures $40.28km$ of width and $54.64km$ of height and its perimeter is of $496km$.

[1]Obtained from *OpenStreetMap!* https://www.openstreetmap.org

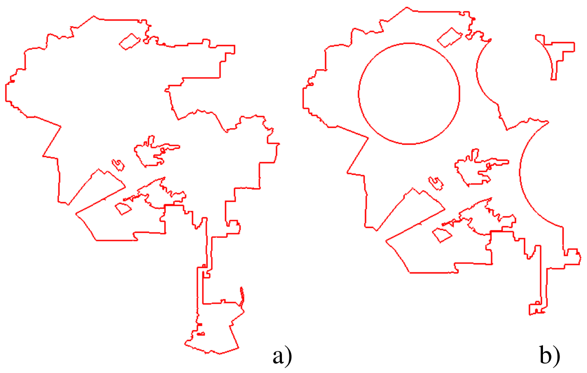

Figure 18: Original polygonal domain b) pwc-domain after partial covering the original one with 4 discs.

5.2. Interaction and visualization process

As we mentioned at the beginning of the paper, computing exact optimal locations is in many cases very expensive and, in general, is also too restrictive because sub-optimal solutions could be considered by experts as even more appropriate than an optimal one. Thus, we compute and visualize the overlap area map which facilitates the work of the decision makers. The fast running times of our algorithms propitiate the interaction with the user in an iterative what-if-analysis process in which the user can easily change several parameters, such as the radius, the initial grid, the error tolerance to obtain suitable locations. Moreover, we do not only provided the ε-optimal locations, but visualize all the obtained information which far from providing too much information allows the user to easily visualize what happens on the whole domain.

In Figure 19, we can see the influence of the radius of the considered disk on the obtained ε-optimal locations. The ε-optimal locations are painted in green and the rest in a blue gradation (the darker the smaller the area). Note that, as expected, the smaller the radius the more ε-optimal locations exist. In the figure the smallest considered disk has a radius of $2.5km$, the others of $3.3km$, $12.5m$ and the largest, which covers the whole domain, has a radius of $29.2km$. For the first two considered radii ideal cells have been found, thus the optimal obtained area is πr^2, and the

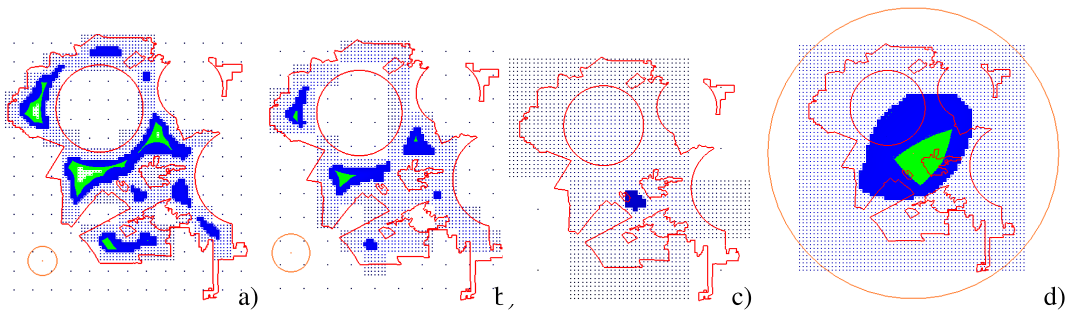

Figure 19: Overlap area map for a disk or radius: a) $2.5km$, b) $3.3km$, c) $12.5km$ and d) $29.2km$.

disk is fully contained in the domain. Meanwhile, the optimal area obtained for the larger disk corresponds to the region area because the domain is fully contained in the disk. The tolerance used in these images is of $\varepsilon = 0.07km^2$ and four refinement steps are used in each of them. Obtaining the last result is quite slower because we refine almost all the blue and all the green area until the desired tolerance is reached.

Let us also see the influence of ε on the obtained results. Figure 19 is obtained with $\varepsilon = 0.07km^2$. In Figure 20 a) and b) $r = 3.3km$ and ε is of $0.35km^2$ and $0.0035km^2$, respectively. Figure 20 c) is a zoom of an interesting region of Figure 20 b) that allows a more in deep visualization of the refinement process. In Figure 20 d) and e) it is $r = 29.59km$, $\varepsilon = 3.5km^2$ and $0.35km^2$, respectively. In both cases the interesting area can be fully identified.

5.3. Running times

Finally, several running times are presented. In them we can see the influence of the radius r, the allowed error ε and the number of refinement steps S. The considered radius are of 2.5, 7.5, 10 and 12.5 km, the allowed error varies

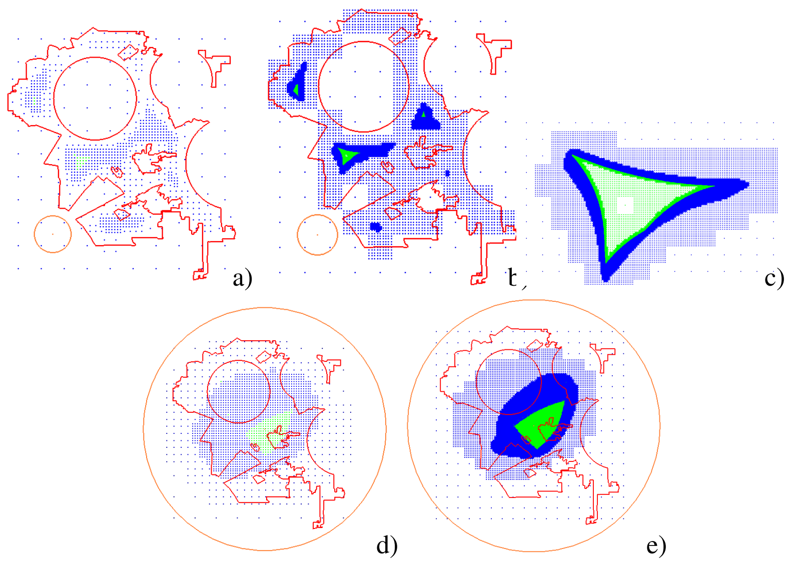

Figure 20: Overlap area map with different r and ε values.

from 0.0035 and 0.7 km^2, and the number of refinement steps goes from 1 to 6. The running time decreases increasing r and ε and using the refinement process. However the optimal number of refinement steps is not always 6, it depends on ε and r. In fact $S = 4$ tends to provide fast running times and thus it is a good number of refinement steps.

Figure 21 provides the experimental results related to the uniform ε-area map. We can see the running times of the sequential-CPU and the parallel-GPU strategies together with the number of analyzed cells, i.e. of areas computed, when using a uniform grid. The figure presents results associated to ε-errors of 0.035, 0.07, 0.35 and 0.7 km^2. The two smallest ε values are not presented because the obtained running times are much bigger, and if they are represented, only those two running times can be guessed from the chart.

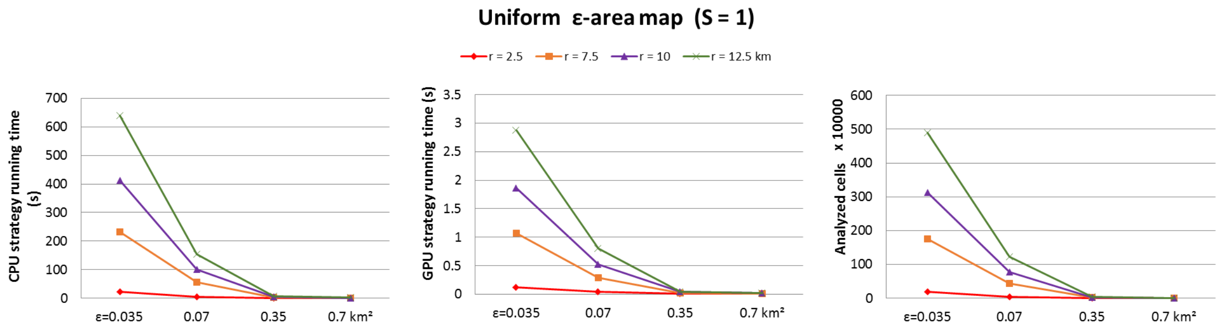

Figure 21: Uniform ε-area map running times

Figure 22 presents the running times of the non-uniform strategy in the first and second columns, and the number of cells analyzed in each case in the third one. In order to obtain the results for each specific number of refinement steps S, we tried to proceed as explained in Section 4.2. However, sometimes starting with that initial grid implies needing more refinement steps to reach d_ε. In this case, we reduce the initial side length by a factor of 0.8 as many times as needed until S steps suffice to reach to the desired d_ε side length.

The first column shows the running times of the sequential version of the algorithm which is, again, always slower than the parallel one, whose running times are presented in the second column. It is not surprising, because the areas computation, which is the bottleneck of the algorithm, is completely paralellized within each refinement step. Hence, parallelizing the algorithm makes sense. Comparing Figures 21 with the last four values of each char of Figure 22 placed on the right of the grey vertical line, we can see how the refinement is worth. The refinement divides by 10, approximately, the running time in most of the cases. Hence, from the running times we deduce that the refinement

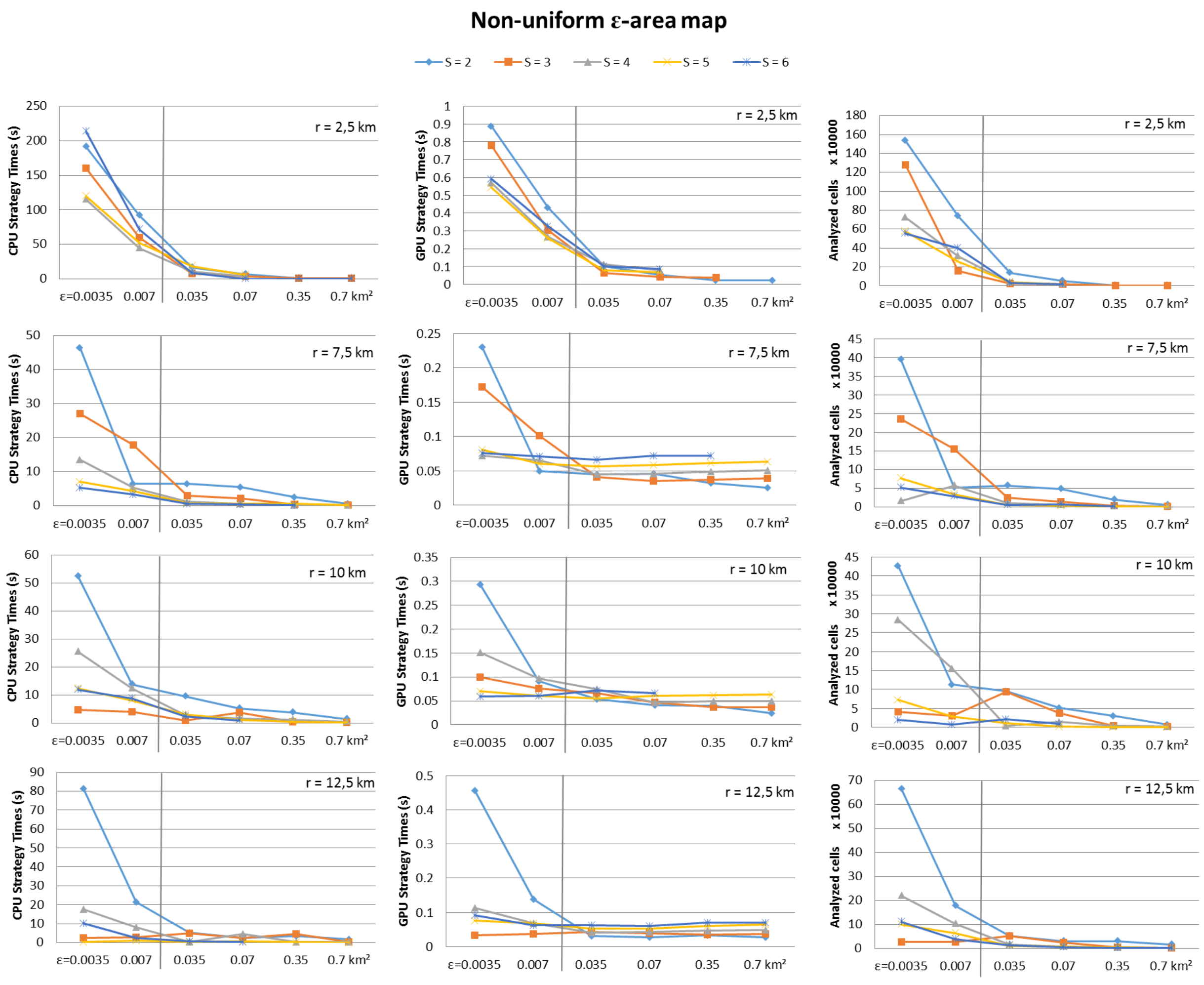

Figure 22: Non-uniform ε-area map running times

speeds up the process. However, the third column corroborates it, as we mentioned in Section 4.3, the refinement process saves work because the number of analyzed cells in the uniform ε-area map decreases notoriously with the refinement.

6. Conclusions and future work

In this paper we solve the problem of locating a disk so that its overlap area with a piecewise circular domain is near-optimal when considering partial coverage. The proposed solutions, which exactly computes the covered area, involves computing many times the overlap area of a piecewise circular domain and a disk. Even though these computations are the bottleneck of our approach, since they are independent one from the other, we parallelize them. Moreover, we design and provide an efficient strategy, that can be run either in the CPU or GPU, to exactly compute the overlap area of a disk and a pwc-domain which uses Green's theorem and takes into account several geometric properties.

We propose two grid-based approaches for computing a discrete set of ε-optimal locations obtained from discrete ε-area maps. We start presenting a way to compute a uniform ε-area map. Next it is improved by using a refinement process which leads to a non-uniform ε-area map. The obtained non-uniform ε-area map depends on the chosen number of refinement steps and on the inital coarse grid, however, all the existent ε-optimal location are always obtained regardless of the initial grid used. According to the mathematical basis, the obtained ε-optimal locations do not represent only the center of the grid cell but all the points contained in the cell. Hence, we somehow provide a continuous solution defined by a discrete set of points that define regions that contain ε-optimal locations. In fact, if someone considers that the ε-guaranteed error is not sufficient and wants to keep on approximating the solutions to the optimal ones, the ε-optimal solutions could be used as seeds for a typical gradient algorithm used in optimization problems.

In the paper we sketch a sequential and a parallel algorithm to be run on a CPU and a GPU, respectively, for computing the ε-area maps and the set of ε-optimal locations. Both strategies were implemented and the experimental results demonstrated that the parallel version is robust and much faster than the sequential one. Moreover, using the refinement process is worth. We observed that using four refinement steps and considering $\sqrt{2}r$, the provided thresold, for defining the coarse grid tends to provide good results. We also presented the theoretical space and time complexity analysis of the algorithms that, not only, corroborated the obtained experimental running times, but also proved that the GPU-parallel algorithm is theoretically optimal.

Finally, in order to improve the understanding of the problem, we described and implemented a way of visualizing the obtained overlap area maps by highlighting the near-optimal locations. The short response times for computing and visualizing the results allow the user to solve the problem through a practical interactive what-if-analysis process.

As future work we are interested in considering signal blockage due to obstacles and circular sector service areas instead of perfect circular coverage for the service area.

7. Acknowledgments

Work partially funded by the Ajut per la millora de la productivitat científica/UdG/3/Ref. MPCUdG2016-031UdG-52211-C2-2-R. We also acknowledge NVIDIA Corporation for the donation of the Tesla K40 GPU.

[1] M. Bansal and K. Kianfar, Planar maximum coverage location problem with partial coverage and rectangular demand and service zones, INFORMS Journal on Computing, 29 (2017), pp. 152-169.
[2] Berman, O. and D. Krass: The Generalized Maximal Covering Location Problem, Computers & Operations Research 29, 563581 (2002).
[3] Berman, O., D. Krass and Z. Drezner: The Gradual Covering Decay Location Problem on a Network, European Journal of Operational Research 151, 474480 (2003).
[4] O. Berman, Z. Drezner and D. Krass: Generalized coverage: New developments in covering location models, Computers & Operations Research 37, 16751687 (2010).
[5] O. Cheong, A. Efrat, S. Har-Peled, Finding a guard that sees most and a shop that sells most, *Discrete & Computational Geometry* **37(4)** (2007) 545-563.
[6] R.L. Church, The planar maximal covering location problem, *Journal of Regional Science* **24(2)** (1984) 185–201.
[7] S.W. Cheng, C.K. Lam, Shape matching under rigid motion, *Comput. Geom. Theory Appl.* **46(6)** (2013) 591–603.
[8] R.L. Church, C. ReVelle, The Maximal Covering Location Problem, *Papers of the Regional Science Association* 32 (1974) 101–118.
[9] R. L. Church and K. L. Roberts, Generalized coverage models and public facility location, in Papers of the Regional Science Association, 1983, pp. 117-135.
[10] N. Coll, M. Fort, J.A. Sellarès, Computing the maximum overlap of a disk and a polygon with holes under translation, *XVI Encuentros de Geometría Computacional* (2015) 57–60.
[11] N. Coll, M. Fort, J.A. Sellarès, Computing the maximum overlap of a disk and a piecewise circular domain under translation, *European Workshop on Computational Geometry (EuroCG)* (2016) 223–226.
[12] Y. Cai, S. See (Eds.), *GPU Computing and Applications*, Springer Singapore, (2015).
[13] Daskin, M. S., Network and Discrete Location: Models, Algorithms and Applications, John. Wiley and Sons, Inc., New York, 1995.
[14] B. T. Downs, Jeff Camm, An exact algorithm for the maximal covering problem, Naval Research Logistics 43(3): 435 461, 1996.
[15] T. Drezner, Z. Drezner, Replacing continuous demand with discrete demand in a competitive location model, *Naval Research Logistics* **44** (1997) 81-95.
[16] Z. Drezner, G. O. Wesolowsky, and T. Drezner, The gradual covering problem, Naval Research Logistics (NRL), vol. 51, pp. 841-855, 2004.
[17] L.G.A. Espejo, R.D. Galvão, B. Boffey, Dual-based heuristics for a hierarchical covering location problem, Computers & Operations Research, 30 (2) (2003), pp. 165-180.
[18] R.Z.Farahani, N. Asgari, N. Heidari, M. Hosseininia, M. Goh, Covering problems in facility location: A review, Computers & Industrial Engineering, 62 (2012), pp. 368-407.
[19] R.D. Galvão, Charles ReVelle, A Lagrangean heuristic for the maximal covering location problema, European Journal of Operational Research 88 (1996) 114-123.
[20] W.W. Hwu (Ed.), *GPU Computing Gems Emerald Edition*, Morgan Kaufmann, (2011).
[21] O. Karasakal and E. K. Karasakal, "A maximal covering location model in the presence of partial coverage, Computers & Operations Research, vol. 31, pp. 1515-1526, 2004.
[22] J.M. Lee, Y.H. Lee, Tabu based heuristics for the generalized hierarchical covering location problema, Computers & Industrial Engineering, 58 (4) (2010), pp. 638-645.

[23] T.C. Matisziw, A. T. Murray, Siting a facility in continuous space to maximize coverage of a region, *Socio-Economic Planning Sciences* **43** (2009) 131-139.

[24] D.M. Mount, R. Silverman, A.Y. Wu, On the area of overlap of translated polygons, *Computer Vision and Image Understanding* **64(1)** (1996) 53–61.

[25] A.T. Murray, D. Tong, Coverage Optimization in Continuous Space Facility Siting, *Int. J. Geogr. Inf. Sci. Vol.* **21(7)** (2007) 757–776.

[26] A.T. Murray, Geography in coverage modeling: exploiting spatial structure to address complementary partial service of areas, *Ann Assoc Am Geogr* **95(4)** (2005) 761–772.

[27] A.T. Murray, T.C. Matisziw, H. Wei, D. Tong, A Geocomputational Heuristic for Coverage Maximization in Service Facility Siting, *Transactions in GIS* **12(6)** (2008) 757–773.

[28] N. Megiddo, E. Zemel, S.L. Hakimi, The maximum coverage location problem, *SIAM Journal of Algebraic and Discrete Methods* **4(2)** (1983) 253–261.

[29] Resende, M.G.C.: Computing Approximate Solutions of the Maximum Covering Problem with GRASP, Journal of Heuristics 4, 161177 (1998).

[30] ReVelle, C., M. Scholssberg and J. Williams: Solving the maximal covering location problem with heuristic concentration, Computers & Operations Research 35, 427435 (2008).

[31] K.F. Riley, M.P. Hobson, S.J. Bence, Mathematical methods for physics and engineering, Cambridge University Press, 2010, ISBN 978-0-521-86153-3

[32] L.V. Snyder, Covering problems, *Foundations of Location Analysis* H. A. Eiselt and V. Marianov (Eds.), Springer-Verlag, Chapter 6, (2011) 109–135.

[33] D. Song, A. van der Stappen, and K. Goldberg, Exact algorithms for single frame selection on multiaxis satellites, IEEE Transactions on Automation Science and Engineering 3, 16-28 (2006).

[34] R. Wei, A.T. Murray, Continuous space maximal coverage: Insights, advances and challenges, *Computers and Operations Research* **62** (2014) 325-336.