# Nearest and farthest spatial skyline queries under multiplicative weighted Euclidean distances

Marta Fort [a,b] J. Antoni Sellarès [a] Nacho Valladares

[a] *Graphics and Imaging Laboratory, Universitat de Girona, 17003 Girona, Catalonia, Spain*
[b] *Ciències de la Computació, Universitat Politècnica de Catalunya, 08034 Barcelona, Catalonia, Spain*

**Abstract**

Consider two point sets in the plane, a set of points of interest and a set of query points that is used to establish distance restrictions with respect to the set of points of interest. A nearest/farthest spatial skyline query retrieves the subset of desirable or relevant points of interest, called skyline points, such that no other point of interest is simultaneously closer to/farther from all the query points. The nearest/farthest top-$k$ spatial skylines, are the best $k$ nearest/farthest spatial skylines among the existent ones. All these queries find applications in decision-making support systems, facility location, crisis management and in trips or events planning. To take into account that each point of interest has a different importance, a weight is assigned to each of them and multiplicative weighted Euclidean distances are used. In this paper, we study, for the first time, the nearest and farthest spatial skyline queries when multiplicative weighted Euclidean distances are considered. We prove that most of the properties of the traditional non weighted nearest and farthest spatial skyline queries are no longer true under the weighted Euclidean distance and, consequently, the strategies used for solving non weighted spatial skyline queries are not usable in the weighted case. We present a sequential and a parallel algorithm, to be run on the CPU and on a Graphics Processing Unit, respectively, for solving nearest/farthest weighted spatial skyline queries and to extract the nearest/farthest top-$k$ spatial skylines. We provide the time and space complexity analysis of both algorithms together with their theoretical comparison. We also have developed a simple interface to deal with weighted spatial skyline queries which allows to visualize and store in a file the obtained spatial skylines. Finally, we present and discuss experimental results obtained with the implementation of the proposed sequential and parallel algorithms.

*Key words:* Computer science; Decision-making support System; Nearest and farthest spatial skyline query; Weighted Euclidean distance; Graphics Processing Unit (GPU)

## 1. Introduction

In this paper, we study nearest and farthest spatial skyline queries when multiplicative weighted Euclidean distances are considered. This problem has applications in facility location, crisis management or in trips or events planning. For instance, when planning a trip, a nearest spatial skyline query helps to select the set of desirable hotels regarding their distance to museums, historical buildings or beaches. By the other

*Email addresses:* mfort@imae.udg.edu (Marta Fort), sellares@imae.udg.edu (J. Antoni Sellarès), nacho.valladares@gmail.com (Nacho Valladares).

side, when we plan the construction of a holiday center, a farthest spatial skyline query helps in identifying suitable locations far away from garbage dumps, noisy places or chemical plants.

The two main elements of a spatial skyline query are a set of points of interest and a set of query points used to establish distance restrictions with respect to the set of points of interest. A point of interest is relevant or desirable if it is better than any other point of interest. From now on, these points will be called *skyline* points.

Since the resulting set of skyline points can be too large and, in practice, many times it is preferable to identify a small number of points among the skyline points, we specify a ranking function that reflects the relevance of each point in the skyline. This ranking or scoring function allows to retrieve the subset of the best $k$ lowest or highest scored skyline points which are called the top-$k$ spatial skylines.

By analyzing the applications of the problem, we notice that the Euclidean distance may not be able to simulate a realistic scenery. In practice, there exist points of interest (e.g., hotels, shops, garbage dumps) with different attraction or repulsion capabilities, say with different importance. To take into account the importance of the points of interest, we assign a weight to each of them. Experts take available information of the points of interest (prestige, price, magnitude, services, etc) and then aggregate these factors to obtain their weights [8,9]. In this way, we reflect that relevance depends on distance and importance. This is done according to the model presented in [4] which is widely used in location analysis [12–14,27]. We associate to each facility a multiplicative weighted Euclidean distance defined by the product of the Euclidean distance and the inverse of the facility weight, hence the higher the weight, the smaller the weighted distance with respect to the Euclidean distance. The formal definition can be found in Section 3.1.

To the best of our knowledge, nearest and farthest spatial skyline queries considering multiplicative weighted Euclidean distances have not received attention so far, although they have practical applications as we show in the next examples

### 1.1. *Motivational examples*

As we briefly mentioned before, solving spatial skyline queries can help, for example, to select desirable hotels of a city taking into account, but not exclusively, their proximity to a set of query points. The traditional spatial skyline points chose the desirable hotels taking into account, only, their distance to a set of query points. But this scenery is not realistic. When users chose a hotel, they obviously take into account the distance of the hotel to a set of query points they want to visit, but they also take into account several other aspects related to the hotel, such as the price or the opinion of the previous users. These other aspects are used to assign an attraction value to the hotel which is used to weight the distance to that hotel. The bigger the weight associated to a hotel, the higher its attraction power and the less important becomes the distance when the weighted Euclidean distance is used. The Euclidean distance is divided by the weight of the hotel simulating the hotel attraction power of that hotel.

The spatial skyline points are those points that are not dominated by any other point. A point $p$ dominates point $p'$ if $p$ is better placed than $p'$ with respect to all the query points, in terms of proximity. Hence, what determines whether a point dominates another is their relative position when they are sorted according to the proximity to the query points. Assigning weights to the hotels and using them to compute the proximity produces alterations in this relative order and consequently produces changes in the obtained skyline points. This will not happen if we assign weights to the queries $Q$. Adding weights to the queries will produce changes in the distance values but not in their sorting. Hence assigning weights to the queries may have sense but is useless because it has no effects in the obtained output.

Next, we provide two specific examples showing the existent differences between taking and not taking into account the weights to compute the spatial skyline points.

*Example 1.* Consider the set of hotels and the set of tourist points of interest of a given area of Barcelona. A potential tourist can use the results of a nearest spatial skyline query to decide which hotel to book with the goal of being close to most of the city's tourist attractions. In Figure 1 we can see the results obtained

when multiplicative non-weighted (a) and weighted (b) Euclidean distances are considered. The weights associated to the hotels correspond to the real rating from 0 to 5 given by the users mapped to an integer rating from 0 to 10. Its value is used to gradate the red color when the restaurant is painted, the darker the color, the smaller the weight.

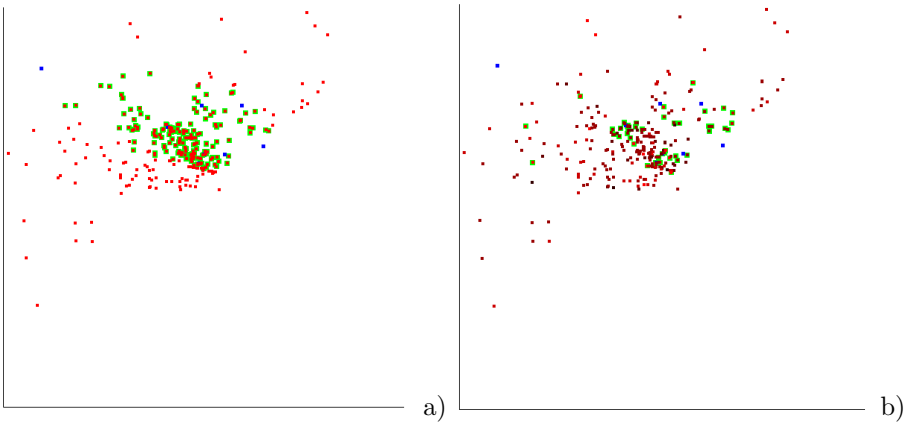


Fig. 1. Desirable hotels (in green) of Barcelona to visit the blue tourist attractions considering a) non weighted b) weighted distance

Note that the results obtained differ significantly depending on whether or not the weight of the hotels is considered. Not all the points in the convex hull of $Q$ are weighted spatial skyline points, meanwhile all are unweighted spatial skyline points. Furthermore, some points quite far from the convex-hull of $Q$ can become weighted spatial skyline points, but they will rarely be unweighted spatial skylines. Finally, in general, the number of skylines tends to diminish when weights are considered.

*Example 2.* Consider now the group of hotels and a set of noisy nightlife places such as Mercabarna a wholesale market or tipycal places where people go to have fun in Barcelona, such as: Plaç del Sol, Plaça Reial, Plaça de la Mercè, Passeig Colom or Nova Mar Bella Beach. A potential tourist can use the results of a farthest spatial skyline query to decide which hotel to book with the aim of avoiding crowds, noise, etc. In Figure 2 we can see, again, the results obtained when multiplicative non-weighted and weighted Euclidean distances are considered and a bit closer to $Q$.

Notice, once again, that the results obtained differ significantly depending on whether or not the weight of the hotels is considered. In this case, we find farthest weighted skyline points much closer to the points of $Q$, the unweighted spatial skyline points tend to always be on the boundary of $P$.

Many of the properties used to solve non weighed spatial skyline queries are not further true in the weighted case. For example, neither the convex hull nor the Voronoi diagram can be used to accelerate the weighted skylines computation. In the non weighted case, efficient index structure with a regular space partition (e.g., Q+Tree [37]) are used to reduce the number of dominance tests, but the properties allowing their use mainly rely on the triangular inequality that is no longer true when weights are considered. Hence, most of the results and strategies based on geometric properties that allow avoiding computations, such as those used in the unweighted case, are not generalizable to the weighted case. Consequently, we use robust and simple methods that do not rely on the triangular inequality. One of these methods is the brute force algorithm that requires scanning the set of points of interest once for each point. This brute force method can be parallelized by determining, independently, if each point of interest is a skyline point.

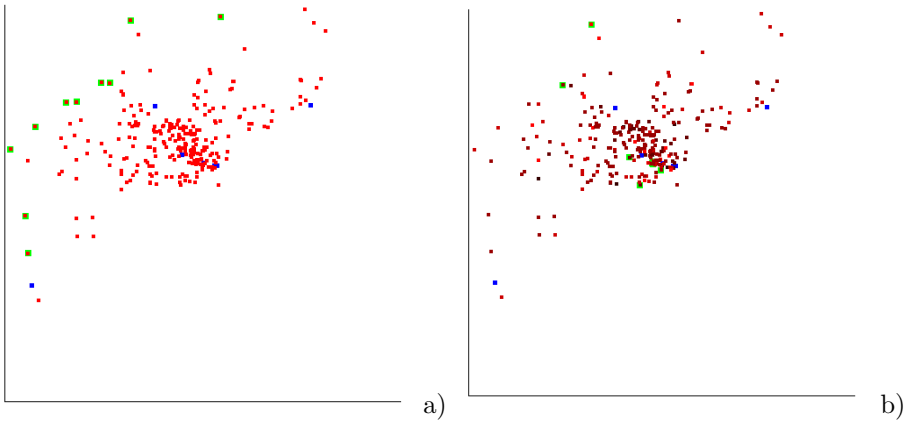


Fig. 2. Desirable hotels of Barcelona (in green) to avoid the blue frequented places considering a) non weighted b) weighted distance

### 1.2. *Our contributions*

In this paper, we study, for the first time, *nearest and farthest weighted spatial skyline queries*, i.e. nearest and farthest spatial skyline queries considering multiplicative weighted Euclidean distances.

Computing the similarity with a not metric function makes that most of the strategies used for solving non weighted spatial skyline queries are not usable in the weighted case. Moreover, we can not take benefit of the data structures usually used to avoid tests. However, we can take advantage of the fact that, the weighted spatial skyline queries can be solved in a highly parallelizable brute force algorithm which allows us to obtain a fast approach to solve the problem with the GPU.

Our main contributions are:

- We summarize the geometric properties used to solve the spatial skyline queries under the Euclidean distance and analyze them under the weighted Euclidean distance. In fact, we show that most of them are not valid when considering multiplicative weighted Euclidean distances, mainly because we are not using a metric function.

- We solve, in working towards practical solutions, the nearest and farthest spatial weighted skyline queries with a sequential algorithm, but also with a GPU-parallel algorithm, under CUDA architecture. We theoretically and experimentally analyze both approaches and show the efficiency, robustness and fast running times of the parallel one. We also present several images to visualize the desired spatial skylines or top-k spatial skylines.

### 1.3. *Organization of the paper*

After presenting the main idea of the spatial skyline problems, we organize the remainder of the paper as follows. We present related work, including the traditional skyline problems, in Section 2. In Section 3, we study the properties of the spatial skyline problem under the weighted Euclidean distance and compare them with the Euclidean spatial skyline problem properties. In Section 4, we present the sequential and the parallel algorithm to solve the weighted spatial skyline problem and theoretically analyze and compare them. In Section 5, comments on how the top-$k$ spatial skylines can be obtained are provided. In Section 6, the presented algorithms are experimentally analyzed and compared. In Section 7, the conclusions are presented. We also provide an Appendix that summarizes the properties and existent algorithms to solve the nearest, farthest and top-$k$ version of the problem under the non-weighted Euclidean distance.

## 2. Related work

Next, we give an overview of the works related to Spatial Skyline Queries. As we mentioned above, in the the Appendix we describe, in detail, the properties and algorithms provided in the literature to solve Spatial Skyline Queries under the Euclidean distance.

*Basic Skylines.* Given a set $P$ of n points in $\mathbb{R}^d$, a *skyline query* returns the points of $P$, called *skyline points*, that are not dominated by any other point. A point $p$ dominates another point $q$ if it is better than or equal to $q$ in all dimensions and better than $q$ in at least one dimension. Depending on the context, "better" means that has lower or higher coordinate values. Skyline queries have been studied in different disciplines. For example, in Computational Geometry it is known as the problem of the maximum vector [18] and in Operations Research as the Pareto-optimal set problem [22].

A commonly used example is assisting a tourist in choosing a set of interesting hotels in a city (see Figure 3). Each of the hotels has two attributes: room price and distance to the beach. For a tourist who prefers a low-cost hotel close to the beach, hotels $h_1$, $h_2$ and $h_3$ are interesting, these are exactly the skyline of the hotels in Figure 3.

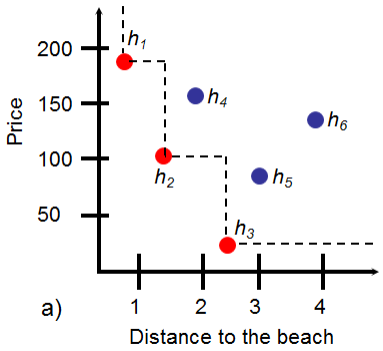


Fig. 3. Skyline of a hotel search scenario

Skyline queries were first studied as maximal vectors [22]. Börzsönyi et al. [5], that introduced skyline queries for database applications, presented divide-and-conquer techniques and index structures to solve the problem in $O(n \log^{d-2} n + n \log n)$ time. Since then, a number of different sequential [21,33,18] and parallel [6,31,28] algorithms for the basic skyline computation have been proposed. In [42] there is a good summary of them.

The top-$k$ dominating query returns the $k$ objects that dominate the maximum number of objects in a given dataset [33,46,17,26]. Since the top-$k$ dominating query identifies the most desirable objects, it is a decision making tool used to rank objects in many life applications.

For completeness, we want to mention that the basic skyline problem has some generalizations. For instances, there exist studies dealing with dynamic skylines queries, where there is a moving point of interest, this makes that the distance from this point to the candidates changes [33,38,41,24]. Some other papers discuss the reverse skyline queries which study the changes in the set of skylines when a new candidate is added. The solutions to this last problem are based on the dynamic problem [7,16]. There also exist skylines queries in temporary databases, where the period of validity or existence of the candidates changes throughout the year, for example simulating hotels opening only in summer [20].

*Spatial Skylines with Euclidean distance.* Given a set $P$ of $n$ data points and a set $Q$ of $m$ query points in the plane, the *nearest spatial skyline query* (NSSQ) retrieves the points of $P$ such that no other point of $P$ is closer to all of the query points of $Q$ simultaneously. Let us see an example. A visitor of a trade fair identifies a set of city locations, say for example the central train station $c$ and the exhibition center venue $v$, and wants to select interesting hotels in terms of Euclidean distance. Considering the scenery depicted in

Figure 4, we want to select the hotels that are no father simultaneously from $c$ and $v$ than any other hotel. Thus, the spatial skyline hotels $h_1$, $h_2$ and $h_4$ with respect to $c$ and $v$ are the most interesting hotels for the visitor. This problem is a two-dimensional problem, because for each hotel two properties are analyzed. The analyzed elements can be turned into real data and each candidate can be mapped into a point in $\mathbb{R}^2$ with coordinates equal to the distance to the beach and its price. Thus, it can be transformed to a traditional skyline problem of dimension 2.

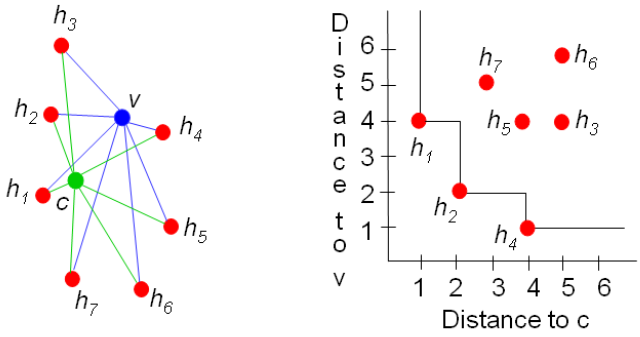


Fig. 4. Nearest spatial skyline hotels: $h_1$, $h_2$ and $h_4$

There can be applications of this problem in other areas, such as: events organization, disaster management decisions or facility location, among others. For instance, suppose that the members of a multidisciplinary task team that work in different offices, need to meet together regularly. The set of query points corresponds to the locations of the offices where the members usually work and the set of points of interest to the potential locations for their weekly meetings. Then, the best location for their weekly meeting is one of the skyline points. In disaster management, suppose that several residential buildings, defining the set of points of interest, must be evacuated because of several explosions or fires defining the query points. The first buildings to be evacuated are the spatial skylines buildings. Finally, consider a business plan for opening a number of shops, points of interest, near a set of residential areas, that define the set of query points. Again, the best locations for opening the new shops are the spatial skyline locations. Crisis management applications is another example. Assume that a number of waterborne infectious disease cases were confirmed at different locations. People who live at spatial skyline places with respect to those locations should be alerted and examined first, because there might be a higher probability that these people may have been exposed to contagious water.

Analogously, given a set $P$ of $n$ data points and a set $Q$ of $m$ query points in the plane, the *farthest spatial skyline query* (FSSQ) retrieves the points of $P$ such that no other point of $P$ is farther from all of the query points of $Q$ simultaneously. Such queries are helpful in identifying spatial locations faraway from undesirable locations, e.g., unpleasant facilities (nuclear power plants, garbage dumps or chemical plants) or business competitors (hotels, restaurants). Figure 5 illustrates an example where we aim at finding an optimal subset of locations among 7 potential locations for a new park. We want the park to be far, in terms of Euclidean distance, from two query points that represent a chemical plant and a garbage dump. The most desirable locations for the park are the farthest spatial skyline locations $p_3$, $p_6$ and $p_7$.

Any NSSQ or FSSQ problem can be treated as a basic $d$-dimensional skyline query, considering for each point of interest its distance to each query point. Applying the method of Börzsönyi et al. [5] nearest and farthest spatial skyline queries can be solved in $O(n \log^{m-2} n + n \log n)$ time if all the $O(nm)$ distances are already computed. Note that distances need to be recomputed for each different data set query $Q$. The goal of existent approaches is to efficiently solve spatial skyline queries, exploiting the geometric properties of the problem, without such transformation [38,39,3,23,35,25]. Sharifzadeh and Shahabi [38,39] first introduced spatial skyline queries and presented R-tree based and Voronoi-based algorithms for solving the problem.

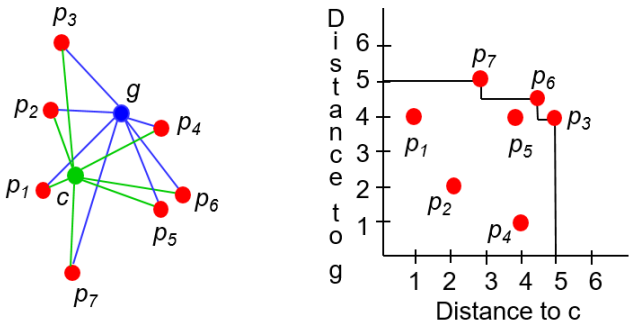


Fig. 5. Farthest spatial skyline parks: $p_3$, $p_6$ and $p_7$

By exploiting geometric properties, they avoid the exhaustive examination of all the point pairs in $P$ and $Q$. Bhattacharya et al. [3], reduce the spatial skyline problem to the problem of finding data points that have non-empty cells in an additively weighted Voronoi diagram under convex distance function. The weights of the Voronoi diagram are derived from the co-ordinates of the points of $P$ and the convex distance function is derived from $Q$. They present a randomized incremental algorithm to find the set of non-dominated points. No results of the algorithm implementation are reported. Lee et al. [23] take advantage of some geometric properties to sort the candidate points from less to greater distance to a query point to speedup the answers. This is, by now, the best existing sequential algorithm to solve the NSSQ problem. The same paper also provides an approximate algorithm to select a representative subset of skylines at a much lower cost. They also analyze the problem when considering a dynamic set of query points. Finally, Wenly et al. [42] provide the only existent parallel solution for the NSSQ problem that uses the MapReduce technique.

The furthest spatial skyline query FSSQ problem has been studied in [45]. First the problem is solved with a baseline algorithm and, then, an efficient progressive algorithm is proposed. The latter significantly outperforms the former by exploiting spatial locality. They also develop an efficient approximation algorithm to trade accuracy for efficiency.

*General spatial skylines.* General spatial skyline query (GSSQ) was introduced at the same time by Lin et al. under this name [25] and by Soudani and Baraani-Dastgerdi under the name of the spatial nearest neighbor skyline query [35]. Instead of a single set of query points, they considered several sets of query points $Q_1$, $Q_2$, ..., $Q_l$ of different types. In this case, it is wanted to select, for example, the set of desirable *hotels* among a set of candidates such that there is no other hotel that has a beach, a shopping center, a public transport stop and a museum closer. Therefore, the query or candidate points are analyzed in relation to the distance to the nearest point of each set of query points $Q_i$.

*Spatial skyline queries with non-Euclidean distance.* There exist algorithms to obtain the skylines using road-network distances [39] and Manhattan distance [36]. They both face the same problem but analyzing proximity with a different distance function. In fact, the only difference is the way in which proximity is measured because both use a metric function that has exactly the same properties as the Euclidean distance.

*Top-k spatial skyline queries.* In this case, the subset of the top-$k$ spatial skylines is selected among all the existent spatial skylines. The top-$k$ spatial skylines are the $k$ skylines optimizing a specific objective function. There are several papers dealing with top-$k$ spatial skylines [32,40]. Son et al. [40] deal with top-$k$ spatial skylines under Manhattan and Euclidean distances. Particularly, they improve an existent approach for the Manhattan distance and present a solution to obtain the top-$k$ skylines.

Recently, different aspects have been added in the spatial skyline problem such as uncertainty or sematics. Elmi and Min [10] develop efficient techniques to compute the spatial skyline over uncertain data. On the other hand, Sohail et al. [34] propose two new types of queries, which enrich the semantics of the conven-

tional spatial queries by introducing social relevance components.

## 3. Nearest and farthest weighted skyline queries

In this section, we formalize the problem tackled in this paper which deals with the multiplicative weighted Euclidean distance and the nearest and farthest spatial skyline points. We start by defining the multiplicative weighted Euclidean distance and then analyze the properties and particularities of the spatial skyline points under this distance (mainly in comparison with those of the Euclidean distance).

### 3.1. *Multiplicative weighted Euclidean distances*

Let $P$ be a set of points in the plane. Assume that each $p \in P$ is associated with a positive real weight $w_p > 0$. The *multiplicative weighted (Euclidean) distance* $d_p(q)$ from the location $p \in P$ to an arbitrary point $q$ is defined as $d_p(q) = (1/w_p)\ d(p, q)$, where $d(p, q)$ denotes the *Euclidean distance* between points $p$ and $q$. Note that, the multiplicative weighted distance is a non-metric, it is not symmetric and the triangle inequality does not hold because the weights are point-depending.

As an example, in Figure 6 we have a set $P$ of seven points and the squared query point $q$. In Figure 6 a), where the Euclidean distance is considered, $q$ has one point at distance $u$, two points at distance $2u$, three points at distance $3u$ and one point at distance $4u$. In Figure 6 b) multiplicative weighted distances are considered and the weights of the points of $P$ are represented in brackets. In this case, $P$ has two points at weighted distance $u$ from $q$, one point at weighted distance $2u$, two points at weighted distance $3u$ and two points at weighted distance $4u$. Note that, in Figure 6 b) the point placed on the smallest semi-circumference has a weighted distance of $4u$ with respect to $q$, meanwhile the point placed on the greatest semi-circumference is only at weighted distance $2u$ from $q$.

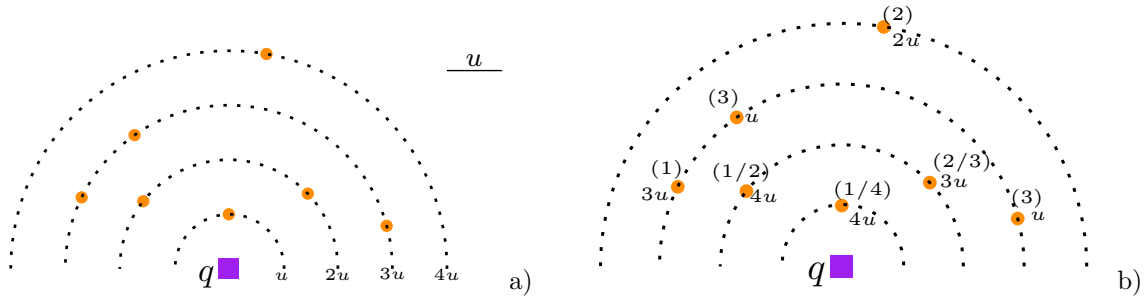


Fig. 6. a) Euclidean distances, b) Multiplicative weighted Euclidean distances

The multiplicative weighted distance transforms the *shape* of several geometric structures that have been used in the literature to solve the problem under the unweighted distance. Their definition and properties under the non weighted distance are summarized in the Appendix. Next, we analyze how they behave under the weighted distance.

– *Bisector between $p_i$ and $p_j$, $b(i, j)$*: the locus of points at identical distance to $p_i$ and $p_j$, i.e.

$$b(i,j) = \{s \in \mathbb{R}^2 \mid d_{p_i}(s) = d_{p_j}(s)\} = \{s \in \mathbb{R}^2 \mid d(p_i, s)/d(p_j, s) = w_{p_i}/w_{p_j}\}.$$

When $w_{p_i}/w_{p_j} \neq 1$ it defines the Apollonius circumference which has center $c_{ij} = (w_{p_i}^2 p_j - w_{p_j}^2 p_i)/(w_{p_i}^2 - w_{p_j}^2)$ and radius $r_{ij} = w_{p_i} w_{p_j} d(p_i, p_j)/|w_{p_i}^2 - w_{p_j}^2|$. The center $c_{ij}$ is aligned with points $p_i$ and $p_j$ and does not separate them. On the contrary, when the weights are the same it defines a straight line.

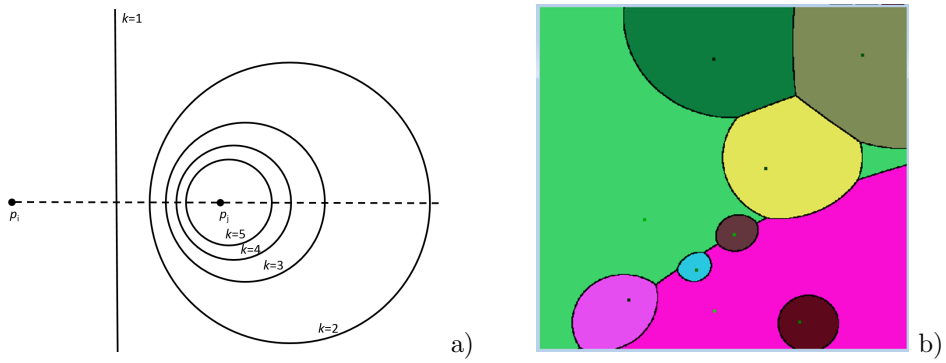


Fig. 7. a) Bisector circles for several ratios $k = w_{p_i}/w_{p_j}$, b) A Voronoi Diagram under weighted Euclidean distance

Figure 7 a) provides several examples of bisector circles $b(i, j)$ for fixed points $p_i$ and $p_j$ corresponding to different values of $k = w_{p_i}/w_{p_j}$, where $k \geq 1$ is assumed without loss of generality. The circles always surround point $p_j$. As $k$ grows, the radius of the circles decreases and the center becomes closer to $p_j$. On the contrary, as $k$ diminish the radius of the circles increases and the center of the circles goes away from $p_j$. In the special case of $k = 1$ ($w_{p_i} = w_{p_j}$), the bisector becomes a straight line (circle of infinite radius).

– *Closest region to $p_i$ with respect to $p_j$*, $r(p_i, p_j)$: the closed planar region bounded by $b(i, j)$ containing $p_i$. It may either by the interior or the exterior of the circle defined by $b(i, j)$. Regarding Figure 7 a) the closest region to $p_j$ is the interior of the circle and to $p_i$ the exterior one.

– *Voronoi diagram of $P$*: the partitioning of the plane that associates each point $p_i \in P$ to a region of the plane consisting of all the points closer to $p_i$ than to any other $p_j \in P - \{p_i\}$. When the weighted Euclidean distance is considered, the closest region to a point may be not convex nor connected and its boundary is defined by straight line, circular and parabolic arc segments [1]. Figure 7 b) is a weighted Voronoi diagram with one non connected region.

### 3.2. *Nearest and farthest weighted skyline queries*

Let $P$ be a set of $n$ weighted points and $Q$ be a set of $m$ query points in the plane. Next, we provide several definitions that are analogue to those of the traditional spatial skyline points presented in the Appendix. Assume that $p_i$ and $p_j$ are two weighted points of $P$ and take into account that the definitions are provided with respect to the set $Q$ of query points.

#### 3.2.1. *Nearest weighted spatial skyline queries*

We will say that:

– $p_i$ *spatially dominates* $p_j$ from *near* $\iff d_{p_i}(q_k) \leq d_{p_j}(q_k)\ \forall q_k \in Q$ and $\exists q_l \in Q\ d_{p_i}(q_l) < d_{p_j}(q_l)$.

– $p_i$ is *not spatially dominated* by $p_j$ from *near* $\iff \exists q_k \in Q$ with $d_{p_i}(q_k) < d_{p_j}(q_k)$ or $d_{p_i}(q_k) = d_{p_j}(q_k)\ \forall q_k \in Q \iff q_k \in r(p_i, p_j)$.

– $p_i$ *is a nearest spatial skyline point* $\iff$ $p_i$ is not spatially dominated from near by any other $p_j \in P$ $\iff \forall p_j \in P - \{p_i\},\ \exists q_k \in Q \mid q_k \in r(p_i, p_j)$.

The goal of a *nearest weighted spatial skyline query* is to retrieve the set $NWSSQ(P,Q)$ of all the nearest spatial skyline points of the set $P$ with respect to $Q$:

$$NWSSQ(P,Q) = \{p_i \in P \mid \forall p_j \in P - \{p_i\},\ \exists q_k \in Q \mid q_k \in r(p_i,p_j)\}\,.$$

Let us see with an example the different results obtained with a nearest spatial skyline query depending on whether weights are taken into account or not. Consider the scenery represented in Figure 8 a). In Figure 8 b) weights are not taken into account, and the nearest spatial skyline hotels are $h_1$, $h_2$ and $h_4$. In Figure 8 c) hotels have assigned weights $w_{h_1} = w_{h_2} = w_{h_3} = w_{h_4} = 1$, $w_{h_5} = 6$, $w_{h_6} = 2$, $w_{h_7} = 5$, and the nearest weighted spatial skyline hotels are $h_5$ and $h_7$.

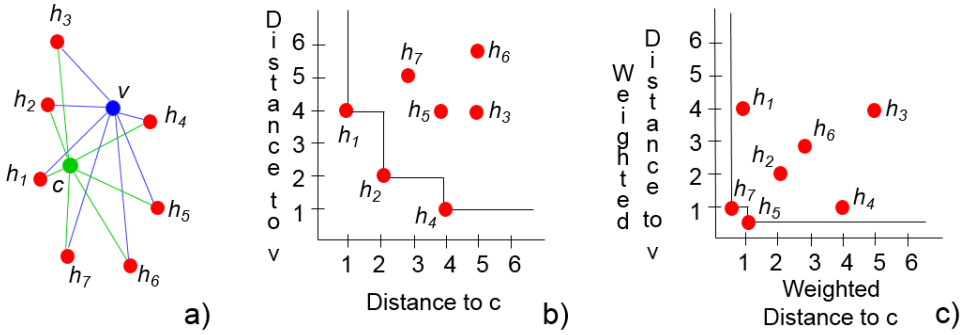


Fig. 8. b) Nearest spatial skyline hotels: $h_1$, $h_2$ and $h_4$; c) Nearest weighted spatial skyline hotels: $h_1$, $h_2$ and $h_4$.

3.2.2. *Farthest weighted spatial skyline queries*

We will say that:

- $p_i$ *spatially dominates* $p_j$ from *far* $\iff$ $d_{p_i}(q_k) \geq d_{p_j}(q_k)\ \forall q_k \in Q$ and $\exists q_l \in Q\ d_{p_i}(q_l) > d_{p_j}(q_l)$.
- $p_i$ is *not spatially dominated* by $p_j$ from *far* $\iff$ $\exists q_k \in Q$ with $d_{p_i}(q_k) > d_{p_j}(q_k)$ or $d_{p_i}(q_k) = d_{p_j}(q_k)\ \forall q_k \in Q \iff q_k \in r(p_j,p_i)$.
- $p_i$ *is a farthest spatial skyline point* $\iff$ $p_i$ is not spatially dominated by any other $p_j \in P$ $\iff \forall p_j \in P - \{p_i\},\ \exists q_k \in Q \mid q_k \in r(p_j,p_i)$.

The goal of a *farthest weighted spatial skyline query* is to retrieve the set $FWSSQ(P,Q)$ of all the farthest spatial skyline points of the set $P$ with respect to $Q$:

$$FWSSQ(P,Q) = \{p_i \in P \mid \forall p_j \in P - \{p_i\},\ \exists q_k \in Q \mid q_k \in r(p_j,p_i)\}\,.$$

Next, we present an example of farthest spatial skyline showing the difference between considering weighted or unweighted points. Consider the scenery represented in Figure 9 a) with $P = \{p_1, p_2, \ldots, p_7\}$ and $Q = \{c, g\}$. In Figure 9 b) $P$ is a set of unweighted points and the farthest spatial skyline parks are $p_3$, $p_6$ and $p_7$. In Figure 9 c) $P$ is a set of weighted points, the weights associated to its points are $w_{p_1} = w_{p_2} = w_{p_5} = w_{p_7} = 1$, $w_{p_5} = 6$, $w_{p_3} = w_{p_4} = w_{p_6} = 2$. In this case, the farthest weighted spatial skyline points are $p_5$ and $p_7$.

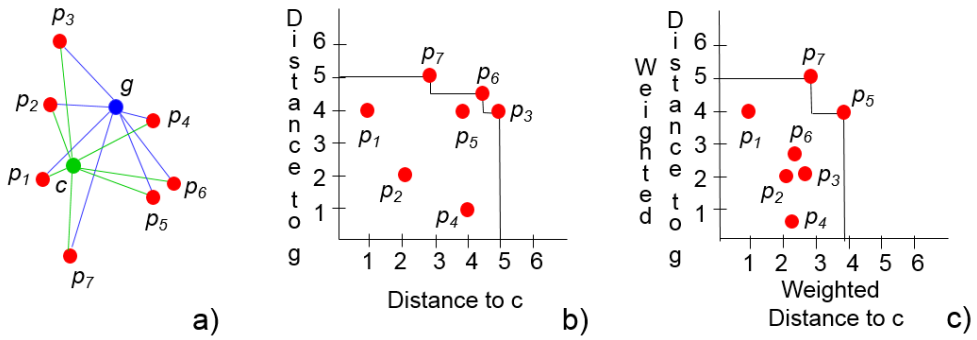


Fig. 9. b) Farthest unweighted spatial skyline: $p_3$, $p_6$ and $p_7$; c) Farthest weighted spatial skyline parks: $p_5$ and $p_7$;

### 3.3. *Top-k skylines*

Sometimes, as it happens with the traditional spatial skylines, after determining the skylines there are still too many desirable points to choose. In these cases, extracting the best $k$ ones is useful. With this aim, users specify some ranking or scoring function $f$ that reflects the relevance of each point in the skyline.

A scoring function $f$ is said to be monotone if and only if for any two points $p$, $p'$, such that $d_p(q) \leq d_{p'}(q)$ for every $q \in Q$, implies $f(p) \leq f(p')$. Typical monotone scoring functions ([32,40]) are:

$$\delta_{max}(p) = \max_{q \in Q} d_p(q),$$
$$\delta_{min}(p) = \min_{q \in Q} d_p(q), \text{ and}$$
$$\delta_{sum}(p) = \sum_{q \in Q} d_p(q).$$

The top-$k$ skylines are those minimizing the monotone scoring function in the from near case and those maximizing it in the from far case.

Along the paper, and from now on, we will omit the from near/far, or nearst/farthest specification whenever possible and we will add them when it is necessary for the understanding of the paper. Accordingly, we will talk about the weighted spatial skyline points and denote the set as $WSSQ(P, Q)$. We also use the terms *spatial skyline points* and *non spatially dominated points* interchangeably.

### 3.4. *Weighted spatial skyline properties*

When using the weighted Euclidean distance, some of the well known properties of the traditional spatial skyline points under the Euclidean distance presented in the Appendix still hold. But most of them turn to be not true. Next, we provide several lemmas and observations related to the geometric properties of the weighted spatial skyline queries. Most of them point out differences between the weighted and unweighted version of the problem.

**Observation 1** *Let $E(Q)$ denote the subset of extreme points of $Q$, those defining the convex hull of $Q$ (see the Appendix for further details). It may happen that $E(Q) \subset r(p_i, p_j)$ while $p_i$ does not spatially dominate from near $p_j$ and $p_j$ does not spatially dominate from far $p_i$.*

For example in Figure 10, $E(Q)$ is contained in $r(p_i, p_j)$, but $p_i$ does not spatially dominate from near $p_j$ and $p_j$ does not spatially dominate from far $p_i$.

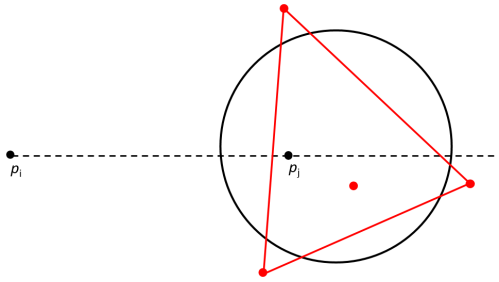


Fig. 10. Example for Observations 1 and 3

**Observation 2** *The bisector of two points in $P$ may intersect $CH(Q)$ while one of the weighted points spatially dominates the other from near or does not spatially dominate it from far.*

Next follows an example of each of the two cases. In Figure 11 a) the bisector of $p_i$ and $p_j$ intersects $CH(Q)$, but $p_i$ dominates from near $p_j$. In Figure 11 b) the bisector of $p_i$ and $p_j$ intersects $CH(Q)$, but $p_j$ dominates from far $p_i$.

**Observation 3** *The set of weighted skyline points of $P$ with respect to $Q$ does not depend only on $E(Q)$.*

For example, in Figure 10 the point $p_i$ dominates from near $p_j$ with respect to $E(Q)$, but $p_j$ is not dominated from near by $p_i$ with respect to $Q$. Point $p_j$ dominates from far $p_i$ with respect to $E(Q)$, but $p_i$ is not dominated from far by $p_j$ with respect to $Q$.

**Observation 4** *A point of $P$ inside $CH(Q)$ may not be a nearest nor a farthest skyline point.*

For example, in Figure 11 a) the point $p_j$ is inside $CH(Q)$ but it is not a nearest skyline point because $p_i$ dominates it from near. In Figure 11 b) the point $p_i$ is inside $CH(Q)$ but it is not a farthest skyline point because $p_j$ dominates it from far.

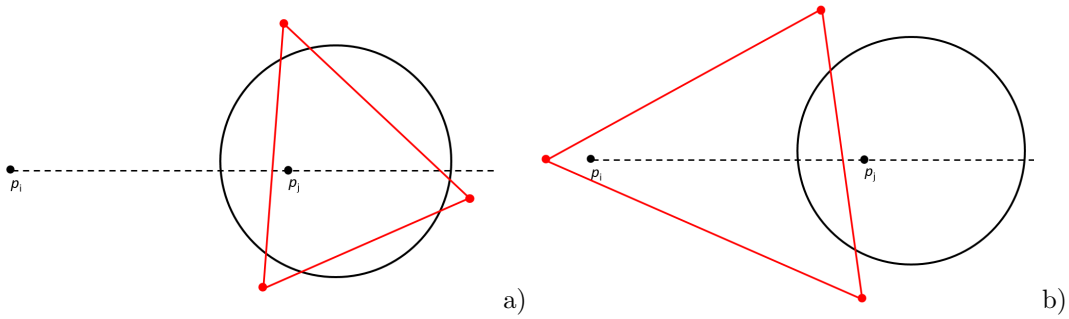


Fig. 11. Examples for Observations 2, 4 and 5: a) from near, b) from far

**Observation 5** *If the Voronoi region of a point of $P$ intersects with $CH(Q)$, the point may not be a nearest skyline point and the point may dominate from far an other point of $P$.*

In Figure 11 a) the Voronoi region of $p_j$ intersects with $CH(Q)$ but $p_j$ is not a nearest skyline point because $p_i$ dominates it from near. In b) the Voronoi region of $p_j$ intersects with $CH(Q)$ and $p_j$ dominates from far $p_i$.

**Lemma 1** *Let $p \in P$ be the closest point to a point $q \in Q$, assuming uniqueness, i.e.*

$$\forall p' \in P - \{p\} d_p(q) < d_{p'}(q),$$

*then $p$ is a weighted spatial skyline point from near.*

**Lemma 2** *Let $p \in P$ be the farthest point to a point $q \in Q$, assuming uniqueness, i.e.*

$$\forall p' \in P - \{p\} d_p(q) > d_{p'}(q),$$

*then $p$ is a weighted spatial skyline point from far.*

**Lemma 3** [*Transitivity*] *If $p_i$ spatially dominates $p_j$ and $p_j$ spatially dominates $p_k$; then $p_i$ spatially dominates $p_k$.*

We do not provide the profs of the lemmas because they are trivial and can be directly adapted from those for the unweighted case.

### 3.5. *Unweighted versus weighted spatial skyline queries comparison*

Many of the properties used to solve unweighed spatial skyline queries are not true when weights are associated to the points of $P$. In this section we analyze it in depth. Figure 12 provides a summary of the properties of the nearest and farthest spatial skyline problems for the unweighed and weighted problem. It also summarizes the properties used in each of the existent algorithms to determine the unweighted skyline points which are described in the Appendix.

As it can be seen in the table, most of the existent algorithms for the unweighted case use $E(Q)$, the Voronoi diagram and the shape of several geometric regions such as $D(q_k, p_i)$ or $r(p_i, p_j)$ to reduce the number of tests done. However, according to the observations related to the weighted spatial skyline points, neither the convex hull nor the Voronoi diagrams can be used to accelerate the weighted skylines determination. On the other hand, the geometric regions they use to bound the area where the skylines can be found, called search and dominator regions, can be analogously defined and would still contain the weighted skyline points. But, meanwhile under the unweighted Euclidean distance these regions are unions or intersections of circles and can are easily approximated or bounded, under the weighted distance they turned to be bounded by circular arcs and may have several disconnected components. Their irregular shapes make them difficult or impossible to bound or approximate. Moreover, in the unweighted case, several data structures are used to reduce the number of dominance tests, but the properties allowing their use mainly rely on the triangular inequality that does no longer hold when weights are considered. Finally, there also exist several properties used in the unweighted case that are proven by considering an arbitrary point in $\mathbb{R}^2$ and assuming that it is a point of $P$. These proves can not be generalized to the weighted case, an arbitrary point $p \in \mathbb{R}^2$ can not be considered as a point of $P$ because points of $P$ have a weight associated. Since the weight can not be arbitrarily chosen, because its value notoriously affects results, many of the proved results and strategies used for the unweighted case do not generalize to the weighted one.

As it can also be seen in Figure 12, there are two basic properties that are used in all the algorithms used for the non-weighted case: the reduction of $Q$ to $E(Q)$ and answering the dominance test by checking whether $b(p_i, p_j)$ intersects $CH(Q)$. They can not longer be used with the weighted distance. By taking into account the properties used in each of the existent algorithms for the unweighted case, we conclude that: i) $BF$ algorithm can be extended to the weighted case, but its accelerated version, $ABF$, can not. ii) $B^2S^2$ and $VS^2$ are not adaptable because they relay on the geometric shape of the search and dominator regions which can no longer be easily determined nor bounded. iii) $DS$, the fastest known sequential algorithm, simplifies the problem by using $E(Q)$ (wich can not be done) and uses the transitivity of the dominance relation (which still holds). Hence, it can be adapted. iv) $PA$ is based on the triangular inequality of the Euclidean distance which is not fulfilled and, hence, it can not be adapted to the weighted case. v) $BBFS$ uses the shape of the search regions to accelerate the process, which makes it not adaptable to the weighted case neither.

It is not surprising that most of the properties and algorithms can not be extended to the weighted case because the nature of the problems are completely different. The weight makes that each point of $P$ derives to $|Q|$ virtual points. Each point $p \in P$ is virtually placed at a different location for each query of $Q$. This is represented in Figure 13 where the set of points $P$ is virtually transformed to $\{p_{1,j}, p_{2,j}\}$ according to each $q_j \in Q$.

| | Weighted Euclidean distance | | Euclidean distance | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Properties | | Properties | | Existent algorithms | | | | | | |
| | nearest | farthest | nearest | farthest | BF | ABF | $B^2S^2$ | $VS^2$ | DS | PA | BBFS |
| $D(q_k,p_i) = \{ s \mid d_s(q_k) \leq d_{pi}(q_k) \}$ | not connected region | | disk | | | | * | * | | * | * |
| $b(p_i,p_j) = \{s \mid d_{pi}(s) = d_{pi}(s) \}$ | a circle, with center $c_{ij}$ and radius $r_{ij}$ $c_{ij} = (w^2_i p_j - w^2_j p_i)/(w^2_i - w^2_j)$ $r_{ij} = w_i w_j d(p_i,p_j)/\lvert w^2_i - w^2_j \rvert$ | | a line, the median of the segment $p_i p_j$ | | | | * | * | | | * |
| $r(p_i,p_j) = \{s \mid d_{pi}(s) \leq d_{pi}(s) \}$ | if $w_i \neq w_j$ the interior or exterior of a circle | | a half plane | | * | * | | | | | * |
| if $E(Q) \subset r(p_i p_j)$ | false **Observation 1** | false **Observation 1** | $p_i$ spatially dominates $p_j$ **Lemma 7** | $p_j$ spatially dominates $p_i$ **Lemma 14** | | * | * | * | * | * | * |
| If $b(p_i,p_j) \cap CH(Q) \neq \emptyset$ they do not spatially dominate each other | false **Observation 2** | false **Observation 2** | true **Lemma 8** | true **Lemma 8** | | * | * | * | * | | |
| SSQ(P,Q) only depends on E(Q) | false **Observation 3** | false **Observation 3** | true **Lemma 9** | true **Lemma 9** | | * | * | * | * | * | * |
| $p \in P$ is inside CH(Q) is a skyline | false **Observation 4** | false **Observation 4** | true **Lemma 10** | false **Observation 6** | | | | * | * | | |
| $p \in P$ is a skylines if, being $q \in Q$, if it is the | closest point of P to q **Lemma 1** | farthest point of P to q **Lemma 2** | closest point of P to q **Lemma 11** | farthest point of P to q **Lemma 15** | | | | * | * | | * |
| if the Voronoi region of p intersects CH(Q) | – **Observation 5** | – **Observation 5** | p is a skyline **Lemma 12** | p does not dominate **Lemma 16** | | | | * | | | * |
| Transitivity: if $p_i$ dominates $p_j$ and pj dominates $p_k$ then $p_k$ dominates $p_i$ | true **Lemma 3** | true **Lemma 3** | true **Lemma 13** | true **Lemma 13** | | | | | * | | * |
| Triangular inequality | false | false | true | true | | | | | | * | * |
| Other properties | | | **Search region** **Lemma 4** **Lemma 5** **Lemma 6** | | | | | * * * | | * * | * |

Fig. 12. Geometric properties and algorithms summary

## 4. Weighted spatial skyline obtention

In this section, we present two algorithms to solve the weighted spatial skyline problem. One of them solves the problem sequentially in the CPU and the other takes advantage of the parallel and compute capabilities of the GPU to accelerate the process. The GPU has been used in last years to accelerate the resolution of many problems and it is shown that it provides very good results [11–15,43,44]. Hence, we want to exploit its parallel and compute capabilities to try to solve the problem faster in parallel.

The sequential algorithm, which is called the *weighted distance sorting algorithm*, $WDS$, is an adaptation of $DS$, the fastest known algorithm for the unweighted case (see the Appendix), to the weighted Euclidean distance. The parallel proposal is a parallel version of the brute force algorithm under the weighted distance, $PWBF$, which is based on the brute force algorithm, $BF$ (see the Appendix). These two presented algorithms are theoretically analyzed and compared in terms of time and space complexity. As we already justified in

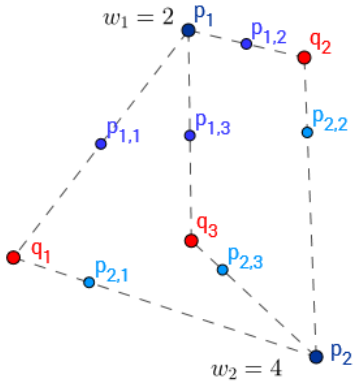


Fig. 13. $P = \{p_1, p_2\}$ is virtually transformed to $\{p_{1,j}, p_{2,j}\}$ according to $q_j \in Q = \{q_1, q_2, q_3\}$

Section 3 they do not use data structures such as Q-trees nor exploit geometric properties because the weighted Euclidean distance is a non metric function. On the other hand, transforming this problem to a traditional spatial skyline implies doing extra computations and requires using much more memory than solving it as a spatial skyline problem. Hence, this option is also discarded.

### 4.1. *Sequential algorithms*

The $WDS$ sequential algorithm solves the problem in four main steps:

1. Sort $P$ according to the distance to $\tilde{q}$, an arbitrary point of $Q$
2. Set the closest point of $P$ to $\tilde{q}$, $p_0$, as a $WSSQ$
3. Set each $p_i \in P$ not dominated by any of the points already stored in $WSSQ$ as a $WSSQ$
4. Report $WSSQ$

Concretely, the $WDS$ algorithm starts sorting the points of $P$ according to their weighted distance to an arbitrary point $\tilde{q} \in Q$. If the problem is solved *from-near*, the points of $P$ are sorted by increasing distance order to $\tilde{q}$, so that, after sorting, $d_{p_i}(\tilde{q}) \leq d_{p_j}(\tilde{q})$ whenever $i < j$ and $p_i, p_j \in P$. Meanwhile, when solving the problem *from-far* they are sorted by decreasing distance order, so that, once sorted, $d_{p_i}(\tilde{q}) \geq d_{p_j}(\tilde{q})$, whenever $i < j$.

After sorting $P$, we store the first point of $P$ in $WSSQ$ for being a spatial skyline (due to Lemmas 1 and 2). Then, we analyze the points $p_i \in P$ in increasing distance order and determine whether they are spatial skylines by checking wether point $p_i$ is dominated by any of the currently detected spatial skyline points (those stored in $WSSQ$). If it is not dominated, it is a spatial skyline and consequently is added to $WSSQ$. Lemma 3 (which states that any not-skyline point $p_i$ will be dominated by one of the already detected skyline points $p_j$ with $j < i$) guarantees that the obtained $WSSQ$ contains all the spatial skylines points. The pseudocode of the algorithm is provided in Figure 14.

```
algorithm WDS_algorithm( Input: WPointSet P , QuerySet Q, bool from_near;
                         Output: WPointSet WSSQ)
1.    sort_acording_to_distance(P,Q[0],from_near);
2.    P[0]->WSSQ
3.    for (unsigned i=1; i<|P|; i++)
4.      bool skyline = true;
5.      unsigned j = 0
6.      while (skyline and j<|WSSQ|)
7.         skyline = not is_dominated(P[i], WSSQ[j], Q, from_near); //dominance test
8.         j++;
9.      if (skylines) P[i]->WSSQ
```

Fig. 14. $WDS$ sequential algorithm pseudocode

To determine whether point $p_i \in P$ is dominated by point $\tilde{p}_j \in WSSQ$ a dominance test is performed. This test depends on whether we solve the problem *from-near* or *from-far*. When it is solved *from-near* $p_i$ is not dominated by $p_j$ if there exist a point $q \in Q$ closer to $p_i$ than to $p_j$. Hence we traverse $Q$ computing and comparing the distances $d_{p_i}(q)$ and $d_{p_j}(q)$. As soon as we find a point $q \in Q$ with $d_{p_i}(q) < d_{p_j}(q)$ we know that $p_j$ does not dominate $p_i$ and the dominance test is answered. If for all $q \in Q$ the inequality $d_{p_i}(q) \geq d_{p_j}(q)$ holds, the point $p_j$ dominates $p_i$ and hence $p_i$ is not a spatial skyline. When the problem is solved *from-far*, $p_i$ is not dominated by $p_j$ if there exists a point $q \in Q$ farther from $p_i$ than to $p_j$. Consequently, the test is answered as soon as we find a point $q$ with $d_{p_i}(q) > d_{p_j}(q)$, in this case $p_i$ is not dominated by $p_j$. See Figure 15 for the pseudocode of the dominance test.

```
bool is_dominated(WPoint pi, WPoint pj, QuerySet Q, bool from_near)
1.  bool is_dominated = true;
2.  unsinged k = 0;
3.  while (is_dominated and k < |Q|)
4.    double di = dist(pi,Q[k]), dj = dist(pj,Q[k]);
5.    if (from_near and di < dj) is_dominated = false;
6.    else if (!from_near and di > dj) is_dominated = false;
7.    k++
8.  return is_dominated;
```

Fig. 15. Dominance test pseudocode

**Complexity analysis.** We denote $n = |P|$, $m = |Q|$ and $r = |WSSQ(P, Q)| \in O(n)$ and analyze, first, the memory and, then, the running time requirements of the $WDS$. The algorithm only stores $P$, $Q$ and $WSSQ$, hence its memory requirements are $O(n + m + r) = O(n + m)$. Concerning the time complexity, it requires $O(n \log n)$ to sort $P$ according to the distance to $\tilde{q} \in Q$ and performs at most $O(nr)$ dominance tests that take $O(m)$ time each, in the worst case. It leads to a worst case time complexity of $O(n \log n + nmr)$. Since $r = O(n)$ and there exist cases in which the number of skylines is $O(n)$, in the worst case, the complexity of this algorithm is $O(n^2 m)$. On the other hand, there also exist sceneries in which the $n$ dominance tests can be solved in $O(1)$ time and where $r = O(1)$; in this best case, the algorithm could run in $O(n \log n)$ time.

### 4.2. *Parallel algorithm*

By using the parallel computing capability of the GPU, we design a parallel robust algorithm that improves the running time of the sequential algorithm. In fact, we present the parallel version, PWBF, of the brute force algorithm under the weighted distance. The PWBF algorithm can be subdivided in three main parts (see the pseudocode in Figure 16).

1. **Transfer information to the GPU** Before calling the kernel, $P$ is transferred to a GPU global memory array $hP$ and $Q$ to a GPU constant memory array $chQ$. Since $|Q|$ tends to be small it can be directly transferred to constant memory, a very fast access memory (similar to the shared memory) without problems of coalesced accesses.
2. **Perform the dominance tests in the GPU** The kernel output is stored in array $hS$. The kernel is launched considering $n$ threads, one per point of $P$. Each thread considers its corresponding point $p_i = hP[i]$ and determines whether $p_i$ is a skyline by analyzing as many points of $hP$ and $chQ$ as necessary. As before, it looks for a point $p_j \in P$, $j \neq i$, dominating $p_i$. If it is found, $p_i$ is not a skyline. To check whether $p_j$ dominates $p_i$, the thread computes and compares the distances from $p_i$ and $p_j$ to the points in $chQ$ (as it is done in the sequential algorithm). Once the thread has determined if $p_i$ is a skyline, it stores the information in $hS$ (an uninitialized output integer array of size $n$ stored in global memory). The thread sets $hS[i]$ to 1 if $p_i$ is a skyline or to 0 otherwise.

```
kernel PWBF_kernel( Input: Wpoint global hP[], Query constant chQ[], int n, int m, bool from_near,
                    Output: int global hS[])
1.    integer ip = global_id
2.    sharedMemory Point sP[BLOCK_SIZE]
3.    Point p;
4.    if(ip<n) p = hP[ip];
5.    bool is_dominated = false;
6.    for(unsigned i = 0; i < n; i += BLOCK_SIZE)
7.       if(i+local_id < n) sP[local_id] = hP[i+local_id];
8.       synchronize_threads
9.
10.      if(ip < n and !is_dominated)
11.         for(j=0; j<BLOCK_SIZE and i+j<n; j++)
12.           is_dominated = is_dominated(p, sP[j], chQ, n, m, from_near); //dominance test
13.           if(is_dominated) break;
14.      synchronize_threads
15.
16.   if(is_dominated) hS[ip] = 0
17.   else hS[ip] = 1;
```

Fig. 16. PWBF algorithm pseudocode

**3. Extract the skyline points in the CPU** Finally, $hS$ is transferred to a CPU array $S$ which is traversed to store in $WSSQ$ all the points $P[i]$ occupying positions with $S[i]$ equal to 1.

Note that in step 2, all the threads analyze the set $hP$ to determine whether their corresponding point $p_i$ is a skyline. We use this fact to make that the threads in a block cooperate to transfer $hP$ from global to shared memory in blocks of BLOCK_SIZE size. It requires two synchronization points: one after the thread transfers a point of $hP$ to shared memory, and the other after the thread has finished analyzing the points stored in shared memory. Moreover, the kernel has to be carefully programmed because those threads that had already determined that their associated point is dominated have to keep on transferring points to shared memory. However, the threads that have already finished will not analyze them.

Something similar could be done with $Q$ if it contains too much points to be stored in constant memory. The constant memory size depends on the device properties, but it can typically store up to 8000 points, which represents a huge cardinality for $Q$. Constant memory has a very fast access and generally the cardinality of $Q$ would allow storing it in constant memory. It makes this type of memory the best kind of memory to store $Q$. However, if $Q$ was too big, it can be stored in global memory and posteriorly transferred to shared memory per blocks with thread-cooperation, as we are doing with $P$.

**Complexity analysis.** We start analyzing the GPU and CPU memory requirements of the algorithm. Concerning the GPU, it stores arrays $hP$ and $hS$ in global memory, $hP$ is transferred to shared memory in blocks of BLOCK_SIZE, and $chQ$ is stored to constant memory. Accordingly, it uses $3n$ real and $n$ integer values in global memory, $2n$ real values in constant memory and $3BLOCK\_SIZE$ real values in shared memory per block. Among all, it needs $O(n+m)$ memory in the GPU. In the CPU we store $P$, $Q$, $S$ and $WSSQ$, i.e. it requires $O(2n+m+r) = O(n+m)$ memory.

Typically the time complexity analysis of the parallel algorithms analyze the worst case *total work* and the *execution time* of the algorithm when $p$ threads are used, which is what we analyse next. The worst case total work is the number of executed operations over all the processors or threads. In the parallel part of the algorithm $n$ threads are used and each thread does $O(nm)$ work in the worst case. The final sequential part that traverses $S$ to determine $WSSQ$ does $O(n)$ work. Hence, the total work done by the algorithm, in the worst case, is $O(n^2m)$. On the other hand, the execution time of the parallel part of algorithm when $p \in O(n)$ threads are considered is $O(n^2m/p)$ and the extraction of $WSSQ$ has an execution time of $O(n)$. Consequently, the execution time of $PWFB$ is $O(n^2m/p + n) = O(n^2m/p)$. If instead of the worst case we analyze the best case, each thread does $O(1)$ work leading to a total work of $O(n)$ done by $p = O(n)$ threads. Taking into account the sequential extraction of $WSSQ$ which requires $O(n)$ work and time, the total work is $O(n)$ in the best case, and the execution time $O(n/p + n) = O(n)$ as well.

To end with we want to mention that the small memory requirements of the kernel allows exploiting all the parallel capabilities of the GPU by setting $p$ as big as possible according to the hardware limitations. In practice $p$ is limited by the maximum number of threads a device can handle in parallel, but some times it may have to be diminished to accelerate the running times due to the amount of memory used by the threads of each block. The memory requirements per block of this algorithm are really small and hence this algorithm allows us to make use of all the parallelism of the GPU without problems. Going a bit further in this analysis, since the number of multiprocessors of the devices is limited, designing parallel algorithms that need a huge number of threads and that each thread did few work is not a good option. The number of threads needed and the work done per thread have to be balanced. This is the reason why we parallelize the brute fore algorithm in this way, each thread determines wether a point $p \in P$ is or is not a weighted spatial skyline point and makes up to $n$ dominance tests. Making that each thread did a single dominance test would lead to a strategy that would need $n^2$ threads and each thread would have done $O(m)$ work, a huge number of threads doing too few work each.

### 4.3. *Theoretical comparison between the parallel and sequential algorithm*

By comparing the complexities of the parallel and sequential presented strategies, we can see that the total work done in the worst case for the sequential WDS algorithm is output-dependent, meanwhile that of parallel PWBF algorithm is independent from the output. Hence, in general, the total work complexity of the PWBF algorithm is worse than that of the WDS. Moreover, in the case that the output is as big as possible, both complexities coincide. Since an important part of the work done by PWBF is performed in parallel, it would turn PWBF to a more efficient algorithm than WDS in terms of running time.

Apart from this initial observations, the standard measures to theoretically compare a parallel and a sequential algorithm are the parallel speedup and the work efficiency of the parallel algorithm. They refer, respectively, to the time complexity $t$ of the best known sequential algorithm to solve the problem, which is $WSD$, and the execution time $t_p$ and the total work $w$ of the parallel algorithm, WDS. Next, we use these measures to compare WDS and PWBF.

**Work efficiency:** it is defined as the ratio $\frac{w}{t}$, which in our case is

$$\frac{w}{t} = O(n^2 m / rnm) = O(n/r).$$

It reflects that PWBF does more work than the WDS, which is due to the fact that WDS is output dependent and PWBF is not.

**Theoretical parallel speedup:** it is given by the ratio $\frac{t}{t_p}$, which in our case is

$$\frac{t}{t_p} = O\left(\frac{rnm}{\frac{n^2 m}{p}}\right) = O\left(\frac{p\ r}{n}\right).$$

Again, the extra work done by PWBF algorithm penalises it. This ratio would generally be greater than 1 and PWBF would theoretically perform better than WDS. However, for very small outputs the theoretical parallel speedup would be smaller than 1, which would mean that WDS algorithm would perform better than PWBF. Hence, the worst scenery for PWBF are problems with a very small number of spatial skyline points.

**Space complexity comparison:** The space complexity of both algorithms is $O(n + m)$. Thus, the space complexity of the parallel algorithm is optimal.

Accordingly, even though the PWBF parallel algorithm does more work than the sequential WDS one, it proceeds in parallel and will perform better than WDS except for cases in which very small outputs, in percentage, were obtained.

## 5. Top-$k$ skylines

When the user is interested in the top-$k$ skylines, the spatial skyline points have to be evaluated according to a scoring function, and then the $k$ skylines being better scored are selected. The user must provide the integer value $k$ and select a scoring function among the typical scoring functions presented in Section 3.3. In the particular case in which $k$ is greater than the number of skylines, all skylines are returned.

In this section, we adapt or complet the presented algorithms so that they report the top-$k$ skylines. We differentiate two sceneries depending on whether the user directly asks for the top-$k$ spatial skylines or contrarily he/she asks for the top-$k$ after determining the spatial skylines. In the latter case, using the GPU makes no sense, transferring the skylines back to the GPU is too expensive in comparison with the work needed to extract the top-$k$ skylines in the CPU. However, in the former case, both, the sequential $WDS$ and the parallel $PWBF$ algorithms can be adapted. Next, we detail the particularities of each algorithm.

**Top-$k$ extraction once the skylines are known:** To determine the top-$k$ skylines once the skylines have already been detected, we traverse sequentially, in the CPU, array $WSSQ$. The scoring function is evaluated for each $p \in WSSQ$ and the top-k scoring values, together with the skylines were they are achieved, are stored. During the scoring function evaluation the already obtained value is compared with the $k$-th obtained one so that a skyline could be discarded of being a top-$k$ skyline as soon as possible. For instance, if we are interested in minimizing the maximal distance, as soon as the maximal distance of the current skyline to a point of $Q$ is bigger than the $k$-th one, that skyline can be discarded. When it happens, we stop evaluating the scoring function of that point and start analyzing another skyline.

**Top-$k$ extraction from the beginning:** In this case, we evaluate the scoring function of each spatial skyline point just after detecting it. Hence, the scoring function is only evaluated for the spatial skyline points (as before). Note that the scoring value can not be computed during the dominance tests because, to obtain it, all the points of $Q$ have to be considered. Meanwhile, the dominance tests for those points that are spatial skyline points usually partially analyze $Q$. We have guarantees that a dominance test analyzes the whole set $Q$ if and only if the point is dominated and, hence, if and only if it is discarded as a skyline point. Hence, we adapt the presented algorithms as follows:

**Top-$k$ WDS Sequential algorithm** Once we detect that a point is a skyline, its scoring function value is computed. While it is being computed it is compared with the current $k$th value, and it is again discarded as soon as possible. Finally, the spatial skyline point is stored, if and only if, it is one of the top-$k$ spatial skyline points. Hence, we only maintain the top-$k$ skylines which are always stored in a sorted way according to their scoring values.

**Top-$k$ PWBF Parallel algorithm** Each thread determines whether its associated point $p_i$, stored in $hP[i]$, is a skyline point. If it is a skyline, the thread evaluates the scoring function at $p_i$ by traversing $chQ$. Finally, each thread stores in the $i$-th position of an output uninitialized real array: the value of the scoring function of $p_i$, when it is a skyline, or a $-1$, otherwise (see the pseudocode in Figure 17). This real output array is transferred to the CPU where it is traversed while the top-$k$ values and their positions are determined (only the positive values are considered, the $-1$ values are ignored). The weighted points occupying these positions are the top-$k$ weighted skyline points. Obtaining the top-$k$ skyline points in the GPU is not appropriate, it requires sorting them performing many global memory accesses and atomic operations leading to a slower strategy.

**Complexity analysis.** Determining the top-$k$ weighted spatial skylines in the CPU, either after detecting the $r$ skylines or directly, requires $O(rm)$ time to evaluate the scoring function and $O(r \log k)$ time to

```
kernel Top_k_PWBF_kernel(Input: Wpoint global hP[], Query constant cQ[], int n, int m, bool from_near, int op,
                         Output: int global hS[])
1.    int i = thread glboal id
2.    sharedMemory Point sP[BLOCK_SIZE]
3.    Point p;
4.    if(ip<n) p = hP[ip];
5.    bool is_dominated = false;
6.    for(unsigned i = 0; i < n; i += BLOCK_SIZE)
7.       if(i+local_id < n) sP[local_id] = hP[i+local_id];
8.       synchronize_threads
9.
10.      if(ip < n and !is_dominated)
11.         for(j = 0; j < BLOCK_SIZE and i+j < n; j++)
12.           is_dominated = is_dominated(p, sP[j], cQ, n, m, from_near);
13.           if(is_dominated) break;
14.      synchronize_threads
15.
16.  if (is_dominated) hS[ip]=-1;
17.  else
18.     real val=0;
19.     for(unsigned i = 0; i < m; i++)
20.         val + = scoring_funcion_according_op(op, dis(p, cQ[i]), val);
21.     hS[ip]=val;
```

Fig. 17. Top-$k$ PWBF algorithm pseudocode

determine the top $k$, i.e. $O(rm + r \log k)$ extra time. It requires $O(k)$ extra space.

Concerning the parallel algorithm, the modifications in the $PBF$ algorithm makes that each of the $r$ threads detecting spatial skylines does $O(m)$ extra work. Since the work needed to determine the skyline points is $O(nm)$, obtaining the value of the scoring function does not increment the order of complexity of the work done by the threads. In fact, it does again $O(rm)$ extra work. Hence, the complexity analysis of the parallel part of the algorithm does not change. We have to add the time needed to obtain the top-$k$ values in the CPU which is $O(n+r \log k)$ extra time. Hence, the total work done by this algorithm is $O(mn^2+r \log k)$ and when $p \in O(n)$ threads are used the execution time becomes $O(mn^2/p + n + r \log k)$. Concerning the extra space, it can be done with $O(k)$ extra space.

## 6. Experimental results

In this section we provide the experimental settings, present a simple interface we used to deal with the problem, and analyze the running time performance of the presented algorithms.

### 6.1. *Experimental settings*

All the implementations have been done in C++, Cuda C has been used for the parallel part, and the visualization is done by using OpenGL. The experimental results have been obtained using an Intel Core i7-7700 CPU @3.6GHz with 32GB of RAM and a GPU Nvidia GTX 1060 6GB. Experiments have been mainly done with synthetic data. Real weighted data sets are not easy to obtain, even thought, we build a small real weighted data set.

As *synthetic data* we consider several sets $P$ of cardinality varying from 2K to 100K and we considered sets $Q$ having from 50 to 150 queries because according to its role in the problem its cardinality has to be much smaller than that of $P$. Sets $P$ and $Q$ have been randomly generated using the standard C++ random library within a squared domain $[0, 1] \times [0, 1]$ and the weights have also been randomly generated within $[0, 10]$.

As *real data* we work with a set $P$ defined by 379 points corresponding to the locations of the hotels of the seaside city of Barcelona (in the south-west of Europe) that are registered at OpenSteetsMap, a collaborative project to create a free editable map of the world, [29] in July 2018. We considered the hotels

with latitude between 2.100 and 2.25 and longitude between 41.38 and 41.44 mapped to points in the square $[0,1] \times [0,1]$ maintaining the original width-height ratio. As weights we consider the between 0 and 5 clients-given metadata-score, of October 2018, mapped to integer numbers between 0 and 10. This score is obtained from google, there, each hotel has a punctuation based on customer reviews. The better the hotel, the higher the score. The set of queries $Q$ is defined by 8 tourist places of Barcelona: *El Born, Sagrada Familia, La Pedrera (Gaudi house), El Liceu, Agbar Tower, The Cathedral, Tibidabo* and *Montjuïc Castel* which are displaced along the whole city. These sets are represented in Figure 19. We did not managed to find any bigger set of real data with a set of weighted points of interest. There exist several sets of points of interest but all of them are unweighted. Hence, they are not suitable to test our algorithms which are specially designed for weighted points of interest. Setting their weights arbitrarily turn that real data to synthetic data and is not fare.

### 6.2. *Results visualization*

We have developed a simple interface to deal with the weighted spatial skyline problem which allows to easily obtain, visualize and store the sets $P$, $Q$ and $WSSQ$. Sets $P$ and $Q$ can be placed manually, generated randomly or read from a *.json* or *binary* file. The user can zoom-in and out with the mouse and also select any of the points to edit it modifying its coordinates or weight. The set $WSSQ$ is computed once all the parameters are specified. If desired, the used data and the obtained $WSSQ$ set can be stored in *.json* or *binary* files.

In Figures 18 and 19 the obtained results for a synthetic and the real data set are respectively shown. In them queries are colored in blue, the weighted points are gradated in red according to their weight, the darker the smaller the weight, and the obtained skylines are remarked with a green background.

Figure 18 corresponds to the spatial skyline points obtained with a set $P$ with 1000 weighted points and a set $Q$ with 50 queries. The nearest and farthest skylines are remarked with the green square in the first row and the top-10 ones in the second one. Note that the images reflect that: i) not all the points in the convex hull of $Q$ are weighted skyline points, ii) the nearest skylines tend to be not too far from the points of $Q$, and iii) the farthest ones are either in the center of $Q$ or to the boundaries of $P$.

Figure 19 presents the real data sets used. The images correspond, from left to right, to all the nearest skyline points, the top-5 ones and all the farthest skyline points. We do not extract the 5-top farthest skylines because there only exist 8 points in c) and 3 in f). Concerning the top-5 nearest ones, somehow, one could expect that the top-5 ones where selected closer to the city center, where the density of hotels is greater. However, they are quite displaced to the left due to the query point placed al the top-left corner of the image, which corresponds to *Montjuïc Castel*. In the second row, we solved the same problems after eliminating *Montjuïc Castel* from $Q$. In this case, the nearest skylines and the top-5 ones are mainly in the center of the city and of $Q$. Meanwhile there disappear the furthest skylines that were located in the city center that are present in the image of the first row.

### 6.3. *Performance analysis*

In this section we analyze the running times of our algorithms testing them with several synthetic data sets. We present the running time of $WDS$ and $PWBF$ algorithms. According to our experimentation, extracting the top$-k$ skylines, independently of whether it is done a posteriorly or form the beginning, does not produce important changes on the running times. In fact, the most important delay does not exceed 60 milliseconds. Hence, there would be no visible differences between the graphics we would obtain with the running times obtained to determine the top-$k$ spatial skylines and those presented in Figures 20 and 21.

Figures 20 and 21 provide the information related to the performance of the sequential $WDS$ and the parallel $PWBF$ algorithms when several sets of points of interest are considered. Each of its charts contains information of 14 different settings corresponding to different $|P|$ and, even though we ran the algorithms in several cardinalities of $Q$, the results of the figures correspon to sets $Q$ with cardinality fixed to 100.

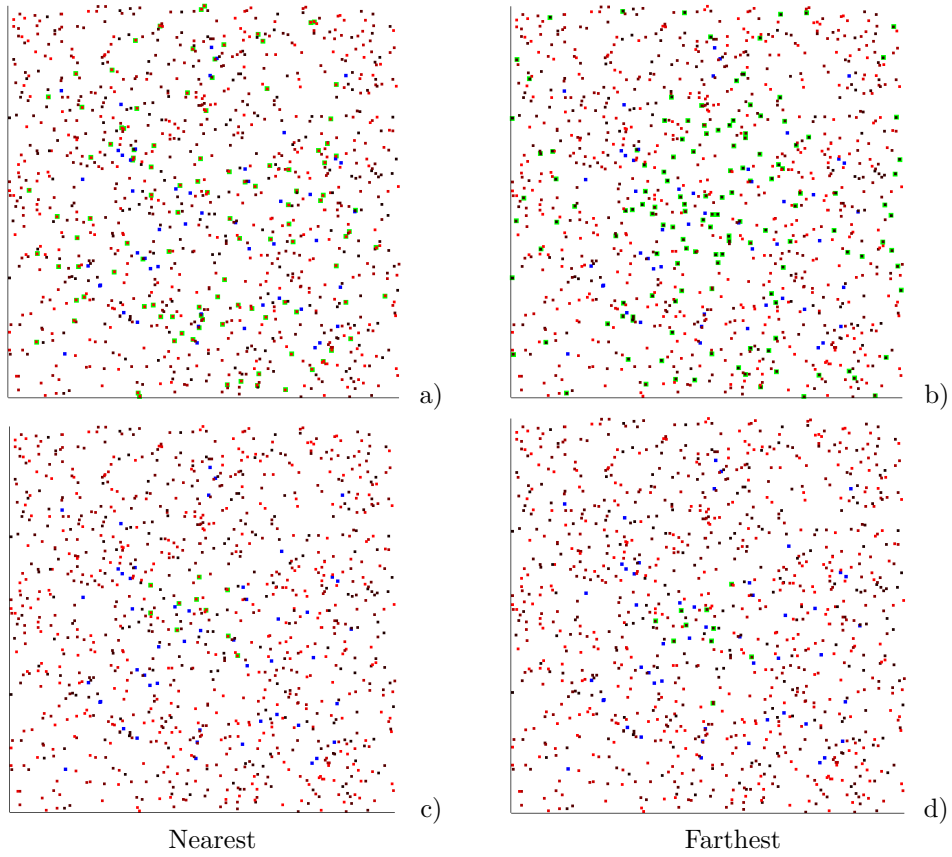


Fig. 18. In red 1000 weighted points, in blue 50 query points, in green all the skylines in a) and b), the top-10 in c) and d)

We run the algorithm with many paris of sets $P$ and $Q$. The obtained running times corroborate that, having $|P|$ and $|Q|$ fixed, the more skylines we obtain the longer the execution time is. To clearly show this fact, among the analyzed pairs of sets $P$ and $Q$, we selected sets $P$ and $Q$ providing a small and a big number of spatial skyline points and we report the obtained running times. Concretely, the results obtained with sets leading to a small output, at most the 20% of the points of $P$ are skylines, are provided in Figure 20. On the other hand, those results corresponding to sets with a big output, at least the 70% of the points of $P$ are skylines, are represented in Figure 21. After analyzing the configurations leading to small and big outputs, we noticed that when the differences between the weights associated to the points of $P$ decrease, the number of skylines increase. Meanwhile, when the differences between the weights increase, the number of skylines decrease.

Figures 20 and 21 contain several experimental results. In chart a) we can see the size of the outputs, the number of spatial skyline points, when we solve problem from near (salmon color) and from far (ocher color). We can see that generally there are more nearest than farthest spatial skyline points. In the remaining parts of the figure, left charts correspond to solving the problem from near and the right ones from far. They contain information of $WDS$ colored purple and of $PWBF$ colored blue. Charts $b$) and $c$) contain the running times of the presented algorithms. It can be clearly seen that $WDS$ is always slower than $PWBF$, for both the near and farthest case and independently that we obtain a small or big output. To easily visualize how many times $PWBF$ is faster than $WDS$, in charts d) and e) we present the speedup of $PWBF$ with respect to the $WDS$. These speedups vary from 2 to 10 when we obtain small outputs and from 5 to 45 for big outputs. Finally, in charts $f$) and $g$) we provide the number of computations done for each algorithm to solve the problem. As it is expected, $PWBF$ does much more computations than the $WSD$, but since they are done in parallel, its running times are better than those of the $WSD$. Note that,

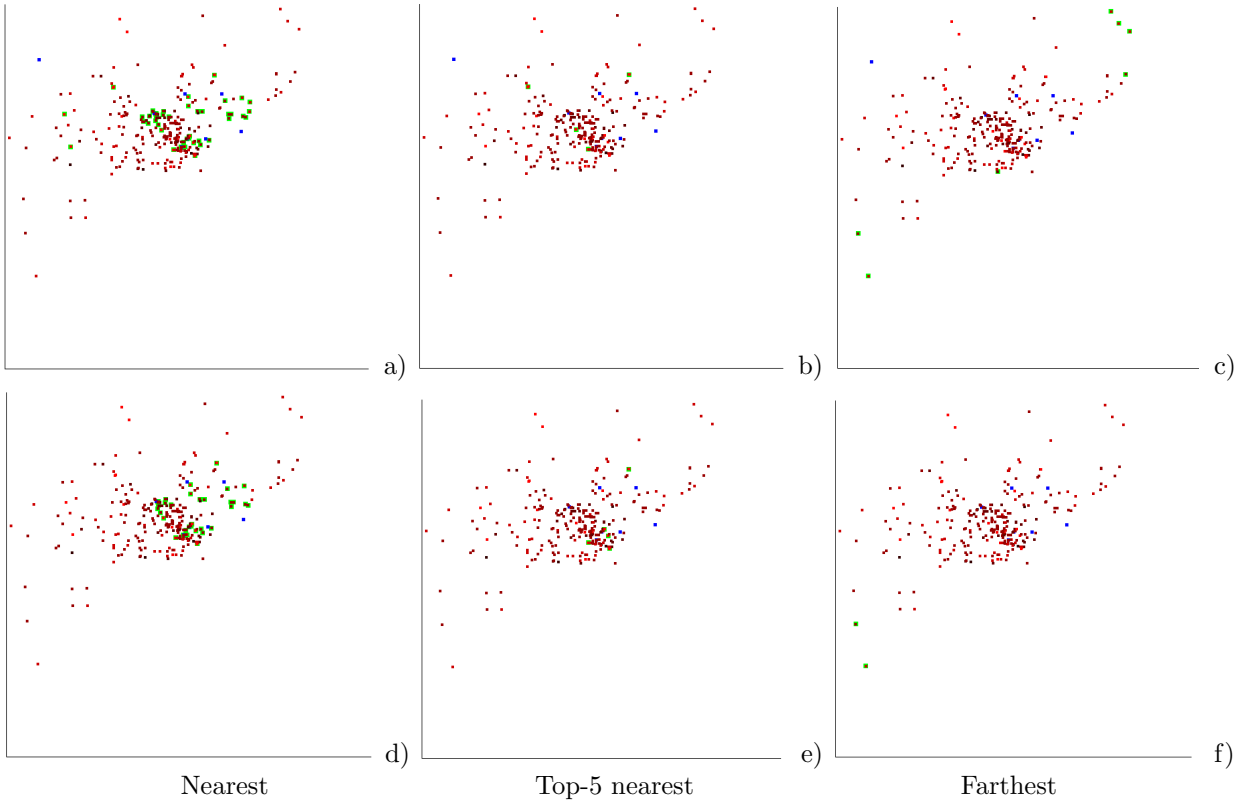


Fig. 19. In red the hotels of Barcelona, in blue 6 (top) and 5 (bottom) query points, in green the obtained skyline hotels

$WSD$ is really eficient when a small output is obtained because it does a very small number of dominance tests. In this case, $PWBF$ does more than 6 times more operations than $WSD$. Meanwhile, with big outputs both algorithms increment the number of operations done, $PWBF$ still does more operations than $WDS$, but it turns to double the operations done by $WSD$. This fact, that is expected from the algorithms desing, is experimentally corroborated in charts $f$) and $g$). Analyzing a bit more these figures, we see how the speedup of the $PWBF$ increases when the difference between the number of operations done by the algorithms diminishes. When it happens the parallelism of $PWBF$ is really exploited because it does not much extra work but it is done in parallel. However, independently from the extra work done, $PWBF$ is always faster than $WSD$. Hence, the parallel $PWBF$ algorithm outperforms the sequential $WSD$ in all the cases except for the smallest from near case in which case the running times are almost the same. Hence, exploiting the parallel capabilities of the GPU is worthwhile and drives to a robust and fast algorithm.

## 7. Conclusions and future work

We have presented, theoretically studied and sequentially and parallelly solved the spatial skyline problem under the weighted Euclidean distance for the first time. These queries find applications in decision-making support systems, facility location, crisis management and in trips or events planning.

We have proven that most of the properties of the traditional spatial skyline problems are no longer true under the weighted Euclidean distance, which does not fulfill the triangular inequality. The from near, from far and top-$k$ versions of the problem have been analyzed. Two fast and robust algorithms to solve the problem have been presented in detail, one runs sequentially in the CPU ($WDS$) and the other exploits the

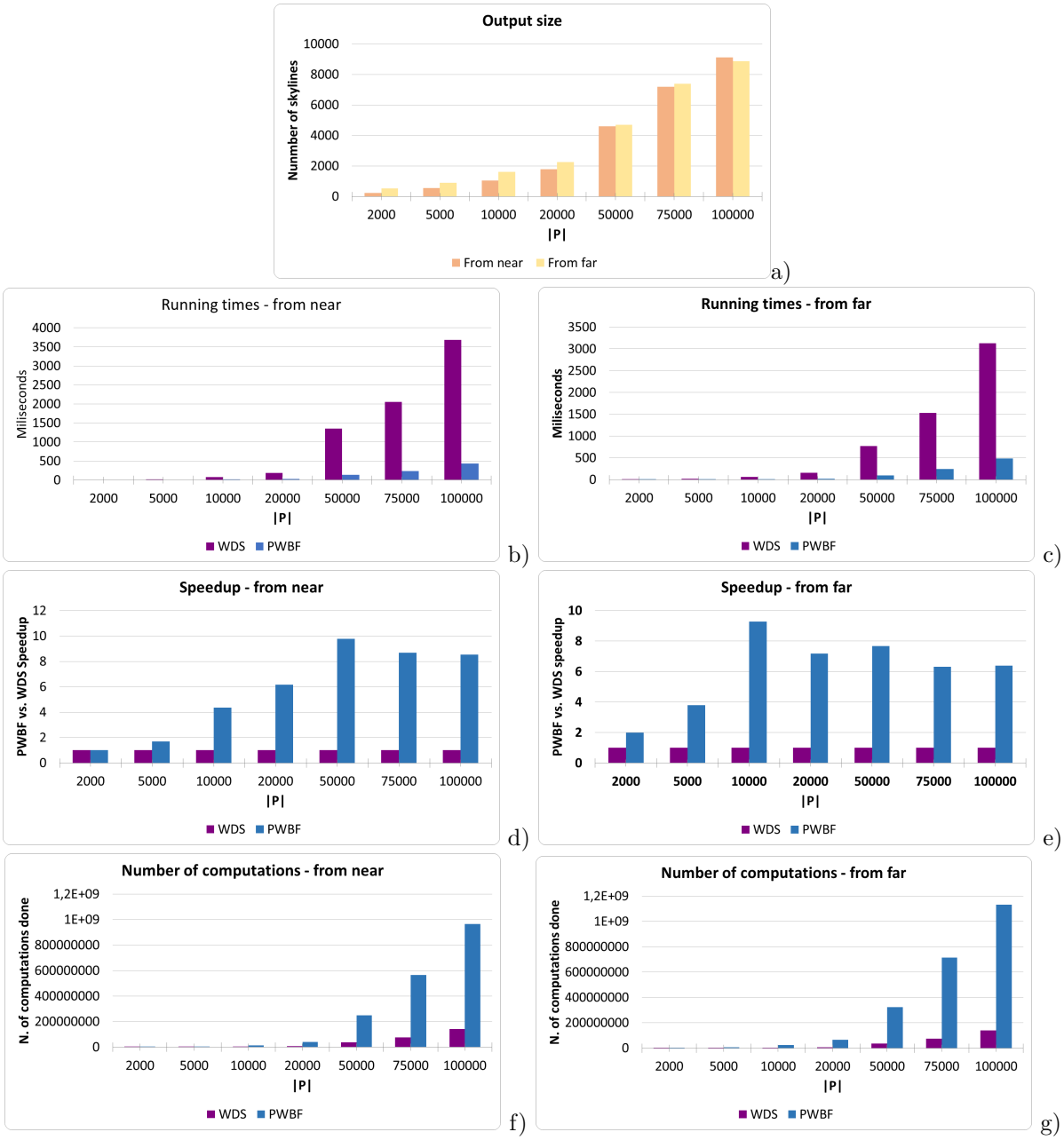


Fig. 20. Synthetic data dealing to a small output: a) Output size. Running times of WDS and PWBF from: b) near c) far. Speedup of PWBF vs. WDS from: d) near, e) far. Number of computations done from: f) near, g) far

parallel capabilities of the GPU (PWBF). These algorithms have been analyzed and compared theoretically and experimentally. The parallel algorithm outperforms the sequential one by a factor between 6 and 10 in most of the analyzed settings as it was predicted by the theoretical complexity analysis comparison. We have also compared these two algorithms with several other described, implemented and tested parallel algorithms to solve the problem and that have been discarded to be slower.

We also developed a simple interface to work with the problem that allows to easily visualize the obtained solution and to store it in a *.json* or *binary* file. Since the obtained solution sometimes still has too much candidates, we allow the user to obtain the top-$k$ spatial skylines according to a scoring function that can be selected among four predefined ones. However, the scoring function could easily be defined by the user

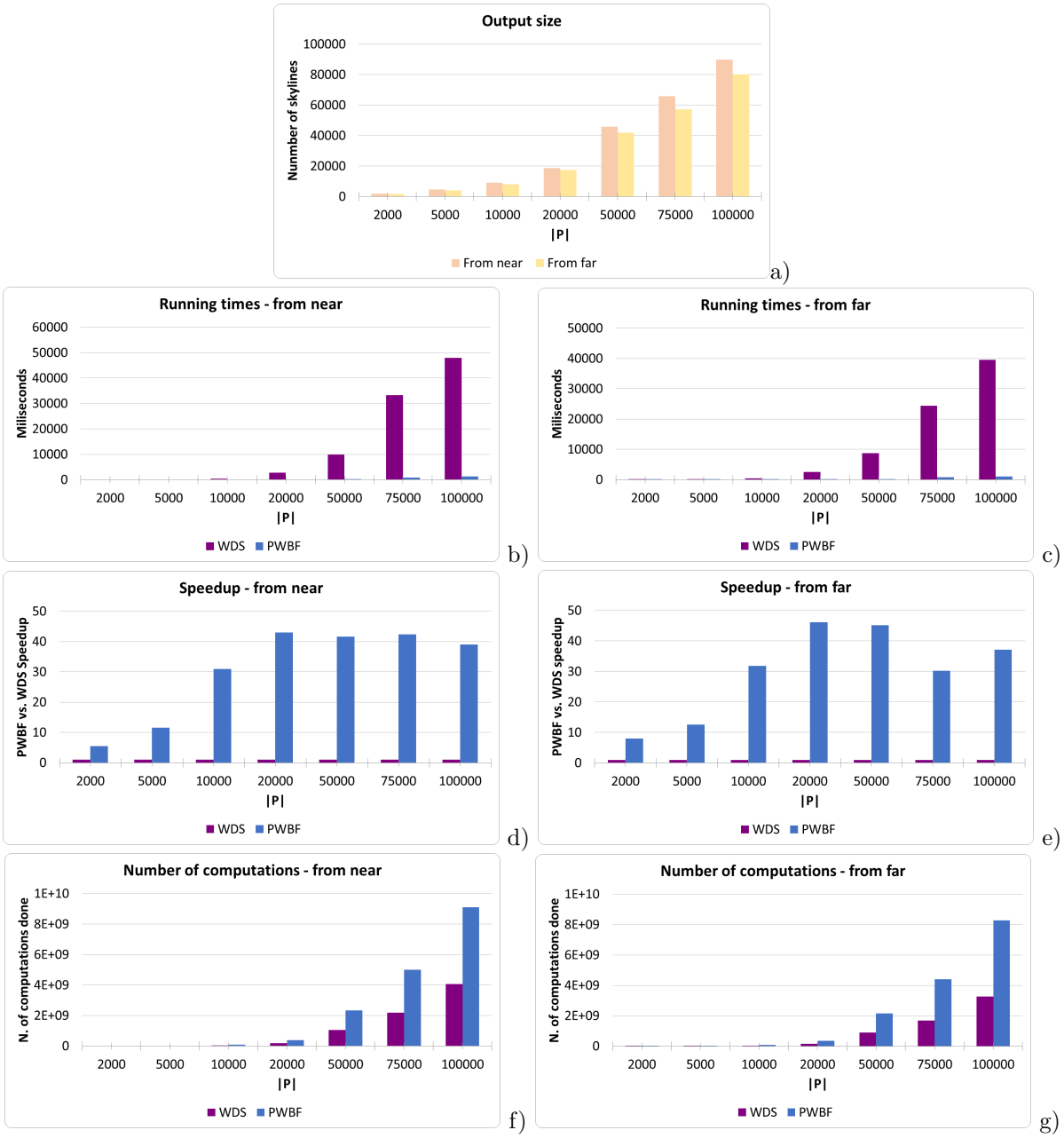


Fig. 21. Synthetic data dealing to a big output: a) Output size. Running times of WDS and PWBF from: b) near c) far. Speedup of PWBF vs. WDS from: d) near, e) far. Number of computations done from: f) near, g) far

meanwhile it only depends on the distances from the skyline to the query points. We allow the user to first obtain the skylines and then identify the top$-k$ among them, or to directly determine only the top-$k$ ones.

As future work, we are interested in combining nearest and farthest spatial skyline queries to retrieve a subset of points of interest such that no other point of interest is simultaneously closer to a set of desirable facilities and farther from a set of undesirable facilities. For example, to select a set of desirable hotels of a city located near tourist attractions and sights of interest and, by the other side, far away from noisy nightlife places. Moreover trying to put the current work into a broader context, we will study whether the nearest and farthest spatial skyline queries may concern the decision analysis of some place choice applying

this study in game theory [19,47].

## References


[1] Aurenhammer, F., Edelsbrunner, H. An optimal algorithm for constructing the weighted voronoi diagram in the plane. Pattern Recognition, 17(2) (1984) 251-257.

[2] W.T. Balke, U. Güntzer, J.X. Zheng, Efficient distributed skylining for web information systems, Proceedings of EBT, 2004.

[3] Bhattacharya, B., Bishnu, A., Cheong, W., Das, S., Karmakar, A., Snoeyink, J., Computation of non-dominated points using Voronoi diagrams compact, in: Walcom: Algorithms and Computation, Proceedings, LNCS, 5942, (2010) 82–93.

[4] B. Boots, R. South, Modeling retail trade areas using higher-order, multiplicatively weighted Voronoi diagrams, Journal of Retailing 73(3) (1997) 519–536.

[5] Borzsony, S., Kossmann, D., Stocker, K., The Skyline Operator, in: Proceedings of the 17th International Conference on Data Engineering, (2001) 421–430.

[6] Choi, W., Liu, L., Yu, B., 2012. Multi-criteria decision making With skyline computation, in: 13th International Conference on Information Reuse and Integration, IEEE, 316–323.

[7] Dellis, E., Seeger, B., 2007. Efficient computation of reverse skyline queries, in: Proceedings of the 33rd International Conference on Very large data bases (VLDB '07), 291–302.

[8] T. Drezner, Optimal continuous location of a retail facility, facility attractiveness, and market share: An interactive model. Journal of Retailing. 70(1) (1994) 49–64.

[9] T. Drezner, Z. Drezner, 2002. Validating the Gravity-Based Competitive Location Model Using Inferred Attractiveness. Annals OR 111(1-4), 227–237.

[10] S. Elmi, J.-K. Min, *Spatial skyline queries over incomplete data for smart cities* Journal of Systems Architecture, 90 (2018) 1–14.

[11] M. Fort and J.A. Sellarès, *Finding influential location regions based on reverse k-neighbor queries*, Knowledge-Based Systems, 47 (2013) pp. 35-52.

[12] M. Fort and J.A. Sellarès, *Solving the k-influence region problem with the GPU*, Information Sciences, 269 (2014) 255–269.

[13] M. Fort and J.A. Sellarès, *Common influence region problems*, Information Sciences, 321 (2015) 116–135.

[14] M. Fort and J.A. Sellarès, *Efficient multiple bichromatic mutual nearest neighbor query processing*, Information Systems, 62 (2016) 136–154.

[15] M. Fort and J.A. Sellarès *Solving multiple kth smallest dissimilarity queries for non-metric dissimilarities with the GPU*, Information Sciences, Volumes 361-362 (2016) pp. 66-83.

[16] Gao, Y., Liu, Q., Zheng, B., Chen, G., On reverse skyline efficient query processing, Expert Systems with Applications, 41(7) (2014) 3237–3249.

[17] Y. Gao, Q. Liu, L. Chen, G. Chen, Q. Li *Efficient algorithms for finding the most desirable skyline objects*, Knowledge-Based Systems, 89 (2015) 250–264.

[18] Godfrey, P., Shipley, R., Gryz, L., Algorithms and analyzes for maximal vector computation, The VLDB Journal, 16(1) (2007) 5–28.

[19] Hofbauer, J. and Sigmund, K., Evolutionary Games and Population Dynamics, Cambridge University Press,Cambridge, 1998.

[20] Kalyvas, C., Tzouramanis, T., Manolopoulos And Processing 2017. Temporal Databases Skyline Queries in, in: Proceedings of the Symposium on Applied Computing (SAC '17), ACM, New York, 893–899.

[21] Kossmann, D., Ramsak, F., Rost, S., 2002. Shooting stars in the sky: An online algorithm for skyline queries, in: Proceedings of the 28th International Conference on Very Large Data Bases (VLDB '02 ), 275–286.

[22] Kung, H.-T., Luccio, F. Preparata, FP, On finding the maximum of a set of vectors, Journal of the ACM, 22(4) (1975) 469–476.

[23] M. Lee, W. Son, H. Ahn, and S. Hwang, *Spatial skyline queries: exact and approximation algorithms*, GeoInformatica 15(4) (2011) 665–697.

[24] Li, Z. Peng, Z. Yan, J., Li, T., Continuous dynamic skyline queries over data stream, Journal of Computer Research and Development, 48(1) (2011) 77–85.

[25] Lin, Q., Zhang, Y., Zhang, W., Li, A., 2013. General Efficient computation space skyline, World Wide Web, 16(3), 247–270.

[26] X. Miao, Y. Gao, B. Zheng, G. Chen, H. Cui, Top-k Dominating Queries on Incomplete Data, IEEE Transactions on Knowledge and Data Engineering, 28(1) (2016) 252–266.

[27] B. Mendes, I.H. Themido *Multi-outlet retail site location assessment*, International Transactions in Operational Research 11 (1) (2004), 1–18.

[28] Mullesgaard, K., Pedersen, JL, Lu, H., Zhou, Y., 2014. Efficient Computation Skyline in MapReduce, in: Proceedings of 17th International Conference on Extending Databases Technology (EDBT), 37–48.

[29] *OpenStreetMap*, https://www.openstreetmap.org, last accesed at 20 july 2018.

[30] Papadias, D., Tao, Y., Fu, G., Seeger, B., 2005. Progressive skyline computation in database systems, ACM Transactions on Database Systems, 30 (1), 41–82.

[31] Park, Y., Min, J.-K., Shim, K., 2013. Parallel computation of skyline and reverse skyline queries using mapreduce, Proceedings of the VLDB Endowment, 6 (14), 2002–2013.
[32] Prema Steffi, R., Sundaramoorthy, S. *Top-K Spatial Preference Query with Range Based Skyline Query in Mobile Environment*, International Journal of Computer Science International Journal of Computer Science and Engine and Engineering, 2(3), 2014, pp. 46–50.
[33] D. Papadias, Y. Tao, G. Fu, and B. Seeger, Progressive skyline computation in database systems, ACM Trans. Database Syst., 30(1) (2005) 41–82.
[34] A. Sohail, M. A. Cheema, D. Taniar, *Social-Aware Spatial Top-k and Skyline Queries* The Computer Journal, 61(11) (2018), pp. 1620–1638.
[35] Soudani, NM, Baraani-Dastgerdi, A., 2011. The Spatial Nearest Neighbor Skyline Queries, International Journal of Database Management Systems, 3(4) (2011) 65–79.
[36] W. Son, S.W. Hwang, H.K. Ahn, *MSSQ: Manhattan Spatial Skyline Queries*, Inf. Syst. 40 (2014) pp. 67–83.
[37] Md.S Islam, C. Liu, J.W. Rahayu, T. Anwar, Q+Tree: An Efficient Quad Tree based Data Indexing for Parallelizing Dynamic and Reverse Skylines, CIKM '16 Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (2016), pp. 1291–1300.
[38] M. Sharifzadeh, and C. Shahabi, The Spatial Skyline Queries. Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB '06), 2006, pp. 751–762.
[39] M. Sharifzadeh, C. Shahabi, and L. Kazemi, *Processing spatial skyline queries in both vector spaces and spatial network databases*, ACM Trans. Database Syst.. 34(3) (2009), pp. 1–45.
[40] W. Son, F. Stehn, C. Knauer, H.K. Ahn, *Top-k Manhattan spatial skyline queries*, Information Processing Letters., 123 (2017) pp. 27–35.
[41] Tao, Y., Xiao, X., Pei, J., *Efficient skyline and top-k retrieval in subspaces*, IEEE Transactions on Knowledge and Data Engineering, 19(8) (2007) pp. 11072–1088.
[42] W. Wang, J. Zhang, M.T. Sun, and W.S. Ku, *Efficient Parallel Spatial skyline Evaluation using MapReduce*, 2oth International Coferecne on Extending Database Technology (EDBT'17) (2017), pp. 426–437.
[43] T. Xu, G. Wang *Finding strongly connected components of simple digraphs based on generalized rough sets theory*, Knowledge-Based Systems, Volume 149 (2018) pp. 88-98.
[44] K. Xu, X. Zheng, Y. Cai, H. Min, T. Wong *Improving user recommendation by extracting social topics and interest topics of users in uni-directional social networks*, Knowledge-Based Systems, 140 (2018) pp. 120-133.
[45] G-W. You, M-W. Lee, H. Im,S-W. Hwang, *The Farthest Spatial Skyline Queries*, Inf. Syst. 38 (2013) pp. 286–301.
[46] M.L. Yiu and N. Mamoulis, Multi-dimensional top-k dominating queries, VLDB J., 18(3) (2009) pp. 695–718.
[47] Y. Zhang, J. Wang, C. Ding, C. Xia, Impact of individual difference and investment heterogeneity on the collective cooperation in the spatial public goods game. Knowledge-Based Systems 136 (2017) pp. 150-158.

## Appendix A. Nearest and farthest spatial skylines under Euclidean distance

In this Appendix, we provide the background for the spatial skyline queries. We start with the formal definition of the problem, the most important and used properties of the unweighted spatial skyline queries and the existent algorithms designed to solve them. Several of these properties and most of the algorithms relay on geometric aspects that are not true when a not metric function is used to measure proximity, which is the case of the multiplicative weighted Euclidean distance. We do not extensively provide all the existent properties but only those that are meaningful for the new results we present in this paper. We start with the more studied problem, the nearest spatial skyline queries, continue with the farthest spatial skyline problem and end with the nearest and farthest top-$k$ spatial skyline queries.

### A.1. *Nearest spatial skyline queries*

We denote by $P$ a set of $n$ points of interest and by $Q$ a set of $m$ query points in the Euclidean plane. The Euclidean distance between points $p$ and $q$ is denoted $d(p, q)$.

When referring to the nearest spatial skyline queries (under the Euclidean distance), it is said that:

- $p_i$ *spatially dominates from near* $p_j \iff d(p_i, q_k) \leq d(p_j, q_k)\ \forall q_k \in Q$ and $\exists q_l \in Q\ d(p_i, q_l) < d(p_j, q_l)$

- $p_i$ *is not spatially dominated from near by* $p_j \iff \exists q_k \in Q$ with $d(p_i, q_k) < d(p_j, q_k)$ or $d(p_i, q_k) = d(p_j, q_k) \forall q_k \in Q$

– $p_i$ *is a nearest spatial skyline point* $\iff$ $p_i$ is not spatially dominated from near by any other point of $P$

According to the previous definitions, the terms *nearest spatial skyline points* and *non spatially dominated points from near* can be interchangeably used. In the example of Figure 4, point $h_2$ spatially dominates from near $h_5$ but it is not spatially dominated from near by any other point of $P$. Hence $h_2$ is a spatial nearest skyline point. The same happens with $h_1$ and $h_4$.

In fact, the *nearest spatial skyline query* is formally defined as the problem of retrieving the set $NSSQ(P,Q)$ of all the nearest spatial skyline points of the set $P$ with respect to $Q$, or equivalently all the points that are not spatially dominated from near by any point in $P$:

$$NSSQ(P,Q) = \{p_i \in P \mid \forall p_j \in P - \{p_i\},\ p_i \text{ is not spatially dominated from near by } p_j\}\,.$$

The nearest spatial skyline points can be easily obtained by checking, for each point in $P$, whether it is spatially dominated from near by any other point in $P$ with respect to $Q$. However, there exist several geometric properties of the set of nearest spatial skyline points that restrict the region where the nearest spatial skyline point can be located. These properties can be used to design more efficient algorithms avoiding some dominance tests. Next, we summarize the most important definitions and lemmas of the nearest spatial skyline points.

– *Closest region to $p_i$ with respect to $p_j$*: $r(p_i,p_j)$ is the set of points of the Euclidean plane closer to $p_i$ than to $p_j$, it is the half plane containing $p_i$ and delimited by the bisector $b(p_i,p_j)$ of the points $p_i$ and $p_j$. The bisector $b(p_i,p_j)$ is the locus of equidistant points between $p_i$ and $p_j$ that defines the straight line perpendicular to the line connecting $p_i$ and $p_j$ through their midpoint.

– *Convex hull of $Q$*: $CH(Q)$ denotes the *convex hull* or *convex envelope* of the set of points $Q$ which is the smallest convex set that contains $Q$. Its vertices are called *extreme points* and define the set $E(Q)$.

– *Voronoi diagram of $P$*: $VD(P)$ denotes the *Voronoi diagram* of $P$ defined by the planar partition of the Euclidean plain in to $|P|$ cells, one associated to each point $p_i \in P$, called the Voronoi cell of $p_i$ which contains the points of plane that are closer to $p_i$ than to any other point in $P \setminus \{p_i\}$.

– *Independent region of $p$ and $q$*: $D(q,p)$ is the disk centered at $q$ and radius $d(q,p)$ [42].

– *Search region* also called *Independent Region Group* of $p_i$ with respect to $Q$: $SR(p_i,Q)$ includes $p_i$ and all the points of $P$ that are not spatially dominated by $p_i$, and hence the spatial skyline is contained in $SR(p_i,Q)$ $\forall p_i \in P$. It is defined as

$$SR(p_i,Q) = \bigcup_{q_k \in Q} D(q_k,p_i).$$

– *Dominator region* of $p_i$ with respect to $Q$: denotes the region that is dominated by $p_i$, it is defined as

$$DR(p_i,Q) = \bigcap_{q_k \in Q} D(q_k,p_i)\,.$$

The search and dominator regions restrict the region where the skyline points can be located to a bounded region of the Euclidean plane and provide a characterization of the skyline points. This is stated in the following lemmas extracted from [38,39,23,42].

**Lemma 4** *The spatial skyline is contained in the intersection of all the search regions which is, meanwhile, contained in the intersection of the search regions of the points of any subset $P' \subset P$, i.e.*

$$NSSQ(P,Q) \subset \bigcap_{p_i \in P} SR(p_i,Q) \subset \bigcap_{p_i \in P'} SR(p_i,Q) \quad \textit{for any } P' \subset P.$$

**Lemma 5** *A point $p_i$ is a skyline point if and only if it is the only point of $P$ contained in its dominator region $DR(p_i, Q)$, i.e.*

$$p_i \text{ is a skyline point } \iff P \cap DR(p_i, Q) = \{p_i\}.$$

According to Lemma 5 the spatial skyline can be obtained as the set of points of $P$ whose dominator region does not contain any other point of $P$, i.e.

$$NSSQ(P, Q) = \{p_i \in P \mid P \cap DR(p_i, Q) = \{p_i\}\}.$$

There also exist several other important properties that help in the detection of the spatial skyline and that are used in the algorithms proposed to solve the problem.

**Lemma 6** *Being $q \in Q$ and $p_i \in P$, any point $p \in D(q, p_i) \cap P$ can only be dominated by a point of $P \cap D(q, p_i)$.*

**Lemma 7** *If $E(Q) \subset r(p_i, p_j)$ then $p_i$ spatially dominates $p_j$.*

**Lemma 8** *The bisector of two data points in $P$ intersects the interior of $CH(Q)$ if and only if they do not spatially dominate each other.*

**Lemma 9** *The set of skyline points of $P$ with respect to $Q$ depends only on $E(Q)$, which defines $CH(Q)$.*

**Lemma 10** *Each point $p \in P$ that is inside $CH(Q)$ is a skyline point.*

**Lemma 11** *The closest point $p_j \in P$ to point $q_k \in Q$, assuming uniqueness, is a spatial skyline point.*

**Lemma 12** *Each $p \in P$ whose Voronoi cell intersects the interior of $CH(Q)$ is a spatial skyline point.*

**Lemma 13** *If point $p_i \in P$ spatially dominates $p_j \in P$ and $p_j$ spatially dominates $p_k \in P$, then $p_i$ spatially dominates $p_k$.*

A.1.1. *Existent algorithms*

There exist several approximated and exact algorithms to solve nearest spatial skyline queries under the Euclidean distance. Among them, we briefly explain the most relevant and used ones. We do not pay attention to the approximated ones because are not directly related to our work. We explain the force brute algorithm to solve the problem, the fastest sequential one and the most known algorithm used until the fastest one was presented. Finally, we also present the fastest parallel algorithm.

The simplest algorithm, the *brute force algorithm* ($BF$), analyzes all the pairs $(p_i, p_j) \in P \times P$ with $i \neq j$ and performs its dominance test, i.e. it checks whether $p_j$ dominates $p_i$. If $p_i$ is not dominated by any point $p_j \in P - \{p_i\}$ the point $p_i$ is set as a spatial skyline. The dominance test associated to $p_i$ and $p_j$ can be done considering the half-space of $p_i$ with respect to $p_j$, $h(p_i, p_j)$. Since $p_j$ dominates $p_i$ if all the points of $Q$ are closer to $p_j$ than to $p_i$; it is checked whether there exists a point $q_k \in Q$ lying in $h(p_i, p_j)$. If such a point exists for every $q_j$, $p_i$ is stored as a skyline point. Checking wether a point $p_i \in P$ is a spatial skyline point with this algorithm takes $O(nm)$ time in the worst case. Hence, the total time complexity of this force brute algorithm $O(n^2 m)$. This algorithm can be easily improved by considering $E(Q)$ instead of $Q$ (Lemma 9), this reduces the worst case time complexity to $O(n^2 m')$, where $m' = |E(Q)|$. Moreover, the dominance test can be also accelerated by using Lemma 8 and an appropriate algorithm to check whether the bisector of the two points intersects the convex polygon $CH(Q)$, the time complexity of these *accelerated brute force* algorithm ($ABF$) is reduced to $O(n^2 \log m')$.

Sharifzadeh et al [38] proposed the Branch and Bond spatial skyline algorithm ($B^2S^2$) that searches the spatial skyline candidates by visiting an R-tree from top to bottom which avoids several dominance tests. They also presented the Voroni-based Spatial Skyline ($VS^2$) which starts with the closest data points to

query points and searches in the space by visiting the neighbors of the visited data points over the Voronoi diagram. They used Lemmas 4, 5, 9, 10, and 15. In their experiments $VS^2$ preformed better than $B^2S^2$. However, themselves posteriorly show that the version of the $VS^2$ that had proposed was not completely correct and fixed the errors in [39] by using Lemma 12. Fixing the errors lowered the performance of the algorithm, but it was still the best algorithm at that moment.

Posteriorly, Lee et al. presented the fastest known algorithm in [23], which we call *distance-sorting* ($DS$). They manage to notoriously reduce the number of tests done. After reducing $Q$ to $E(Q)$ (Lemma 9) they select an arbitrary point $q \in E(Q)$ and sort the points of $P$ according to increasing distance to $q$. Then, they start finding the spatial skyline points; the closest point to $q$ is a spatial skyline point (Lemma 15) and the rest of points of $P$ are analyzed in increasing distance to $q$ order. They use the transitivity of the dominance property and the fact that the point being analyzed can only be spatially dominated by the already detected skyline points (Lemma 13). It drastically reduces, in practice, the number of performed dominance test. Again to check whether a point is dominated by any of the already obtained skylines, they check whether the bisector of the two points intersects the $CH(Q)$ (Lemma 8). This algorithm takes $O(m \log m)$ to construct the convex hull, $O(n \log n)$ to sort $P$ in increasing distance order, and preforms $O(|S|)$ dominance tests in $O(\log m')$ time each, where $S$ denotes the set of non-dominated points. Since generally $m < n$ it leads to a total time complexity of $O(n(|S|m + \log n)$. Lee et al. [23] also provide a fastest approximate algorithm to solve the problem that consists in computing the $VD(P)$ and $CH(Q)$ and approximate the skyline by the set of spatial skyline points as the points of $P$ whose Voronoi cell intersects the interior of $CH(Q)$ which is a subset of the real skyline.

The first *parallel approach* ($PA$) to detect the spatial skyline was presented in [42]. The approach relays on Lemma 9, 6 and 4. The base idea is that, each search region is the union of several disks that define an independent region when looking for spatial skyline points (Lemma 6). Hence, they chose the search region defined by a point $p \in P$ which is called the *independent region pivot* and manage to turn the problem into $m'$ independent problems, one for each of the disks defining the search region. These $m'$ problems are solved in parallel. In fact, they propose a three-phase MapReduce-based solution. In the first phase they calculate $CH(Q)$, $Q$ is partitioned, several local convex hulls are first output and then merged to produce the global convex hull. In the second MapReduce-phase they determine the independent region pivot, they look for the point $p \in P$ whose search region has minimal area; the area of the search regions is first locally and then globally minimized. Finally, in the third phase each point of $P$ is associated to the independent regions defining the search region of the chosen pivot, where it is contained. Then $|E(Q)|$ independent spatial skyline problems are solved in parallel by using the independent regions and only the points contained in the independent region. They also avoid dominance tests by taking advantage of the pruning regions, which are defined by circular ring sectors, because the points inside these regions are dominated. The algorithm also uses two multi-level grids and ends by eliminating potentially repeated skyline points. This is necessary because the same skyline can be obtained in different independent regions and hence there will appear duplicated in the output.

### A.2. *Farthest spatial skyline queries*

The farthest spatial skyline query problem was first introduced in 2013 by You et al in [45]. In that paper, they reverse proximity for remoteness. A point of interest $p \in P$ is desirable if no other point $p' \in P$ is farther from all the query points of $Q$. Unlike the nearest spatial skylines points that are clustered within around the query convex hull, the farthest ones are scattered across the data space. They prove that there does not exist duality between the nearest and farthest spatial skylines and provide the following problem definitions:

- $p_i$ *spatially dominates from far* $p_j \iff d(p_i, q_k) \geq d(p_j, q_k)\ \forall q_k \in Q$ and $\exists q_l \in Q\ d(p_i, q_l) > d(p_j, q_l)$
- $p_i$ *is not spatially dominated from far by* $p_j \iff \exists q_k \in Q$ with $d(p_i, q_k) > d(p_j, q_k)$

or $d(p_i, q_k) = d(p_j, q_k) \forall q_k \in Q$

– $p_i$ *is a farthest spatial skyline point* $\iff$ $p_i$ is not spatially dominated from far by any other point of $P$

The *farthest spatial skyline query* is formally defined as the problem of retrieving the set $FSSQ(P, Q)$ of all the farthest spatial skyline points of the set $P$ with respect to $Q$, or equivalently of all the points that are not spatially dominated from far by any point in $P$:

$$FSSQ(P, Q) = \{p_i \in P \mid \forall p_j \in P - \{p_i\},\ p_i \text{ is not spatially dominated from far by } p_j\}.$$

They prove how Lemma 8, Lemma 9 and Lemma 13 also hold for the farthest spatial skyline points and provide the following properties.

**Lemma 14** *If $E(Q) \subset r(p_i, p_j)$ then $p_j$ spatially dominates from far $p_i$.*

**Observation 6** *A point of $P$ inside $CH(Q)$ may not be a farthest skyline point.*

**Lemma 15** *The farthest point $p_j \in P$ to point $q_k \in Q$, assuming uniqueness, is a farthest spatial skyline point.*

**Lemma 16** *If the Voronoi region of a point of $P$ intersects with $CH(Q)$ then the point does not dominate from far any other point of $P$.*

A.2.1. *Existent algorithms*

In [45], two solutions are proposed. One based on the transformation of the problem to a $n$-dimensional general skyline problem, in which the Improved Distributed Skyline (IDS) algorithm of [2] is adapted by using an $R$-tree. The second proposal, the Branch and Bound farthest spatial skyline ($BBFS$), significantly outperforms the initial one by using the geometric properties of the problem. It uses Lemmas 10, Lemma 16 and Lemma 13 indirectly, i.e., the fact that any point whose Voronoi diagram cell intersects $CH(Q)$ does not dominate from far any other point of $P$. In the paper, they also provide an approximate algorithm that reduces the processing cost without significantly affecting the quality of the results. The paper does not provide the complexity of any of the two algorithms.

A.3. *Top-k Nearest and Farthest Spatial Skyline Queries*

The top-$k$ spatial skyline query retrieves the subset of the best (lowest/highest scores) $k$ skyline objects with respect to $f$ for small $k$. Users specify some ranking function $f$ that reflects the relevance of each point in the skyline.

A scoring function $f$ is said to be monotone if and only if for any two points $p, p'$, such that $d(p, q) \leq d(p', q)$ for every $q \in Q$, implies $f(p) \leq f(p')$. Typical monotone scoring functions are

$$\delta_{max}(p) = \max_{q \in Q} dp, q);\ \ \delta_{min}(p) = \min_{q \in Q} d(p, q) \ \text{ and } \delta_{sum}(p) = \sum_{q \in Q} d(p, q).$$

The top-$k$ nearest spatial skyline points are the points minimizing the monotone scoring functions and the furthest ones those maximizing them.

The existent studies on the top-$k$ skyline points deal with the nearest case, mobile environments [32] and the Manhattan distance [40].