Journal of Global Optimization
https://doi.org/10.1007/s10898-023-01280-1

Faster distance-based representative skyline and k-center along pareto front in the plane

Sergio Cabello[1,2]

Received: 25 September 2021 / Accepted: 1 March 2023

Abstract

We consider the problem of computing the *distance-based representative skyline* in the plane, a problem introduced by Tao, Ding, Lin and Pei [Proc. 25th IEEE International Conference on Data Engineering (ICDE), 2009] and independently considered by Dupin, Nielsen and Talbi [Mathematics; Optimization and Learning - Third International Conference, OLA 2020] in the context of multi-objective optimization. Given a set P of n points in the plane and a parameter k, the task is to select k points of the skyline defined by P (also known as Pareto front for P) to minimize the maximum distance from the points of the skyline to the selected points. We show that the problem can be solved in $O(n \log h)$ time, where h is the number of points in the skyline of P. We also show that the decision problem can be solved in $O(n \log k)$ time and the optimization problem can be solved in $O(n \log k + n \operatorname{loglog} n)$ time. This improves previous algorithms and is optimal for a large range of values of k.

Keywords Geometric optimization · Skyline · Pareto front · Clustering · k-center

1 Introduction

For each point p in the plane, let $x(p)$ and $y(p)$ denote its x- and y-coordinate, respectively. Thus $p = (x(p), y(p))$. A point p ***dominates*** a point q if $x(p) \geq x(q)$ and $y(p) \geq y(q)$. Note that a point dominates itself. For a set of points P, its ***skyline*** is the subset of points of P that are not dominated by any other point of P. We denote by sky(P) the set of skyline points of P. Formally

$$\operatorname{sky}(P) = \{p \in P \mid \forall q \in P \setminus \{p\} : \; x(q) < x(p) \text{ or } y(q) < y(p)\}.$$

See the left of Fig. 1 for an example. The skyline of P is also called the ***Pareto front*** of P. A set P of points is a skyline or a Pareto front if $P = \operatorname{sky}(P)$. Note that some authors

Supported by the Slovenian Research Agency (P1-0297, J1-9109, J1-1693, J1-2452, N1-0218).

✉ Sergio Cabello
sergio.cabello@fmf.uni-lj.si

1 Univerza v Ljubljani Fakulteta za matematiko in fiziko, Ljubljana, Slovenia

2 Institute of Mathematics, Physics and Mechanics, Ljubljana, Slovenia

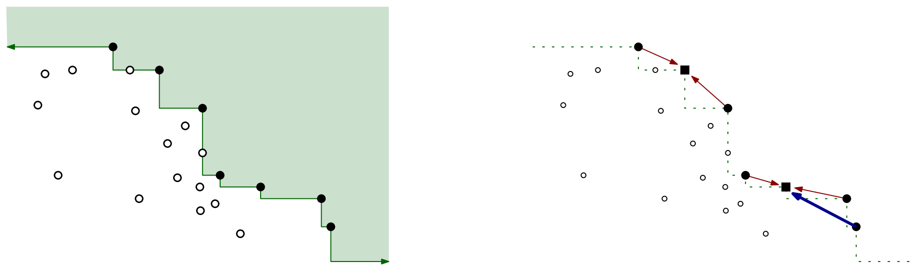


Fig. 1 Left: Example of point set P with the points of $\mathrm{sky}(P)$ marked as filled dots; the shaded region has to be empty of points of P. Right: if Q is the two-point set marked with squares, then the length of the longest arrow is $\psi(Q, P)$; in the figure this arrow is thicker and in a different color

exchange the direction of the inequalities, depending on the meaning of the coordinates in the application domain.

For each subset of points $Q \subseteq \mathrm{sky}(P)$ we define

$$\psi(Q, P) := \max_{p \in \mathrm{sky}(P)} \ \min_{q \in Q} \ d(p, q),$$

where $d(p, q)$ denotes the ***Euclidean distance*** between p and q. Note that $\psi(Q, P) = \psi(Q, \mathrm{sky}(P))$ and $\psi(\mathrm{sky}(P), \mathrm{sky}(P)) = 0$. An alternative point of view for $\psi(Q, P)$ is the following: it is the smallest value λ such that disks centered at the points of Q with radius λ cover the whole $\mathrm{sky}(P)$. See Fig. 1, right. One can consider $\psi(Q, P)$ as how big error we make when approximating $\mathrm{sky}(P)$ by Q.

The aim of this paper is to provide efficient algorithms for the following optimization problem: for a given point set P in the plane and a positive integer k, compute

$$\mathrm{opt}(P, k) := \min\{\psi(Q, P) \mid Q \subseteq \mathrm{sky}(P) \text{ and } |Q| \leq k\}.$$

A ***feasible solution*** is any subset $Q \subseteq \mathrm{sky}(P)$ with $|Q| \leq k$. An ***optimal solution*** for $\mathrm{opt}(P, k)$ is a feasible solution Q^* such that $\psi(Q^*, P) = \mathrm{opt}(P, k)$. Thus, we use $\mathrm{opt}(P, k)$ both as the optimal value and the description of the problem. Note that $\mathrm{opt}(P, k) = \mathrm{opt}(\mathrm{sky}(P), k)$.

We will also consider the ***decision problem***: given a set P of points in the plane, a positive integer k and a real value $\lambda \geq 0$, decide whether $\mathrm{opt}(P, k) \leq \lambda$ or $\mathrm{opt}(P, k) > \lambda$.

Motivation Skylines (or Pareto fronts) can be considered for arbitrary dimensions and play a basic role in several areas. Börzsönyi, Kossmann and Stocker [2] advocated their use in the context of databases, where they have become an ubiquitous tool; see for example [25, 26]. A traditional example in this context is querying a data base of hotels using, say, three criteria, such as the price, the distance to a conference venue, and the average review score. If we ignore other parameters, there is no reason to pay more for a hotel that is further from the conference venue and has worse reviews. In different words, there is no need to consider a hotel whenever there is another hotel that is better in all criteria. Using those numerical criteria as coordinates, each hotel is represented by a point in three-dimensional space. After reversing some of the coordinates to reflect that smaller distance or smaller price is better, we can phrase this property using dominance. In this context, we are telling that we can restrict our attention to hotels represented by points on the skyline.

Skylines also play a key role in multiobjective optimization and decision analysis; see for example the surveys [8, 19, 23]. In this setting, we have different criteria in the space of feasible solutions. Like in the previous setting, there is no benefit to consider a feasible solution whenever there exists another solution that outperforms it in each criteria. In such a case, it suffices to restrict attention to feasible solutions on the skyline. Such feasible solutions on the skyline can be presented to a person or a team to make decisions, possibly taking into account additional criteria, or they may be used as starting data for the next iteration of a more complex algorithm. This latter setting occurs for example in evolutionary algorithms.

We have argued that in several important cases it suffices to restrict our attention to points on the skyline. In some cases, the point set has already been processed, perhaps for some other tasks, and its skyline is already available. In some other cases, we are given the point set as raw data and has to be processed from scratch.

However, quite often the skyline contains too many points. The question is then how to select a representative subset of the points in the skyline. The representative may be a final answer or just an intermediary step, as it happens for example in evolutionary algorithms, where diversity plays a key role.

As one can imagine, several different meaningful definitions of *best representative* subset are reasonable, and different applications advocate for different measures on how good the representatives are. The use of randomization is also relevant in the context of evolutionary algorithms.

The problem we are considering, $\text{opt}(P, k)$, was introduced in the context of databases by Tao et al. [27] under the name of *distance-based representative* of $\text{sky}(P)$. Although we have presented the problem in the plane, it can be naturally extended to arbitrary dimensions. Tao et al. argue the benefit of using such representatives. One of the key properties is that it is not sensitive to the density of P, a desirable property when P is not uniformly distributed.

The very same problem, $\text{opt}(P, k)$, was considered in the context of optimization by Dupin, Nielsen and Talbi [7], who noted the connection to clustering, multiobjective optimization and decision analysis. They noticed that $\text{opt}(P, k)$ is the (discrete) k-center problem for $\text{sky}(P)$. Clustering problems in general, and k-center problem in particular [3, 15] are fundamental in several areas; thus it is relevant to find particular cases that can be solved efficiently.

Another very popular and influential measure to select representatives in the context of databases was introduced by Lin et al. [20]. In this case, we want to select k points of $\text{sky}(P)$ so as to maximize the number of points from P dominated by some of the selected points. See our work [5] for a recent algorithmic improvement in the planar case. In the context of evolutionary algorithms one of the strongest measures is the hypervolume of the space dominated by the selected points [1] with respect to a reference point, that can be taken to be the origin. As it can be seen in the survey by Li and Yao [19], an overwhelming amount of different criteria to select representatives from $\text{sky}(P)$ has been considered in the area of multiobjective optimization.

Previous algorithms We will be using three parameters to analyze the time complexity of algorithms. We will use n for the size of P, k for the bound on the number of points to be selected, and h for the number of points in $\text{sky}(P)$. Unless explicitly mentioned, the discussion is for the plane.

We start reviewing the computation of the skyline. Kung, Luccio and Preparata [18] showed that the skyline of P in the plane can be computed in $O(n \log n)$ time, and this is asymptotically optimal. Kirkpatrick and Seidel [17] noted that the lower bound of $\Omega(n \log n)$ does not hold when h is small, and they provided an algorithm to compute $\text{sky}(P)$ (in the plane) in $O(n \log h)$ time. Nielsen [24] provided an algorithm to compute the top k skylines, that is, $\text{sky}(P)$, $\text{sky}(P \setminus \text{sky}(P))$, ... up to k iterations. The running time is $O(n \log H)$, where H is the size of the top k skylines together.

Kirkpatrick and Seidel [17] also showed that computing the skyline of P takes $\Omega(n \log h)$ time in the algebraic decision tree model. In fact, they showed that, for a given h', deciding whether P has exactly h' points in the skyline takes $\Omega(n \log h')$ time. This implies that finding any k points in the skyline takes $\Omega(n \log k)$ time. Indeed, if we could find k points in the skyline in $o(n \log k)$, then we could run the test for $k = h'$ and $k = h' + 1$ and decide in $o(n \log(k + 1)) = o(n \log h')$ time whether $\mathrm{sky}(P)$ has exactly h' points, contradicting the lower bound.

Now we move onto computing $\mathrm{opt}(P, k)$. Tao et al. [27] showed that the problem can be solved in $O(kh^2)$ time assuming that the skyline is available and sorted by x-coordinate. Thus, it takes $O(n \log h + kh^2)$ time. In the full version of the work [28] they improved the time bound of the second phase to $O(kh)$, which implies a time bound of $O(n \log h + kh)$. They showed that already the three-dimensional version of the problem is NP-hard by noticing that the classical two-dimensional center problem without any constraints regarding the skyline, known to be NP-hard [14], can be encoded as the three-dimensional problem along the skyline.

Dupin, Nielsen and Talbi [7] solve the $\mathrm{opt}(P, k)$ problem in $O(kh \log^2 h)$ time, again assuming that the skyline is available and sorted. Thus, starting from P, the time bound is $O(n \log h + kh \log^2 h)$. They also provide an algorithm for $\mathrm{opt}(P, 1)$ taking $O(\log h)$ time and an algorithm for $\mathrm{opt}(P, 2)$ taking $O(\log^2 h)$ time, again assuming that the skyline is available and sorted by x-coordinate.

Mao et al. [21] provided an exact algorithm for $\mathrm{opt}(P, k)$ that takes $O(k^2 \log^3 h)$ time, assuming that the skyline is stored for binary searches, and an approximation with error of ε in $O(k \log^2 h \log(T/\varepsilon))$ time, where T is the distance between the extreme points of the skyline. Again, if we start from P, we should add the time $O(n \log h)$ to both algorithms to compute the skyline. The decision problem is solved in linear time in the work by Yin and Wei and Liu [31], assuming that the skyline is available.

If the points on the skyline are collinear and sorted, we can compute $\mathrm{opt}(P, k)$ in linear time using the algorithm of Frederickson [9, 10]. See [13] for a recent account that may be easier to understand. The author believes that the same approach can be used to solve $\mathrm{opt}(P, k)$ in linear time, once the skyline is available sorted. However, as the details in those algorithms are very complicated, the author is reluctant to make a firm claim.

Our contribution We show that $\mathrm{opt}(P, k)$ can be solved in $O(n \log h)$ time. We provide a quite simple algorithm for this. Compared to a possible adaptation of [9, 10], we get the added value of simplicity within the same time bound that we need to compute the skyline itself. If the skyline is already available, the running time becomes $O(h \log h)$. In all cases we improve all previous works considering algorithms for the problem $\mathrm{opt}(P, k)$ explicitly [7, 21, 27, 28].

At first, this may seem optimal because computing the skyline of P takes $\Omega(n \log h)$ time. However, do we really need to compute the skyline? After all, we only need to select a particular subset of k points from the skyline, and this has a lower bound of $\Omega(n \log k)$ time, as mentioned above. We show that the decision problem for $\mathrm{opt}(P, k)$ can be solved in $O(n \log k)$ time. This is asymptotically optimal if we want to find a solution because such a solution has to find k points in $\mathrm{sky}(P)$. In fact, we show that with some additional preprocessing we can solve several decision problems of the form: for a given λ, is it $\mathrm{opt}(P, k) \leq \lambda$ or $\mathrm{opt}(P, k) > \lambda$? More precisely, for a given point set P and an integer $\kappa = \Theta(k^2)$, we can preprocess P using $O(n \log \kappa)$ time such that each subsequent decision problem for $\mathrm{opt}(P, k)$ can be solved in $O(k(n/\kappa) \log \kappa)$ time. For example, taking $\kappa = O(k^2)$, this implies that we can solve a *sequence* of $O(k^2)$ decision problems in $O(n \log k)$ time altogether, where each value λ for a decision problem is available only after the previous decision problems have been answered. (If all the values λ of all decision problems would be given at once, this task is easier because we could make a binary search among those values to find the boundary between positive and negative answers.) We combine the decision problem with parametric search and selection in sorted arrays to show that $\text{opt}(P, k)$ can be solved in $O(n \log k + n \operatorname{loglog} n)$ time. This is asymptotically optimal whenever $\log k = \Omega(\operatorname{loglog} n)$, if we want to find also an optimal solution.

Finally, we provide additional results for the case when k is very small. The driving question here is, for example, how fast can we solve $\text{opt}(P, 15)$. We show that a $(1 + \varepsilon)$-approximation can be computed in $O(kn + n \operatorname{loglog}(1/\varepsilon))$ time. We obtain this by finding a 2-approximation in $O(kn)$ time and then making use of repetitive binary searches using the decision problem.

Besides improving previous algorithms, we believe that the conceptual shift of solving $\text{opt}(P, k)$ without computing $\text{sky}(P)$ explicitly is a main contribution of our work. This is also the only way to get around the existing lower bound for computing $\text{sky}(P)$. Nevertheless, even when $\text{sky}(P)$ is available, we improve previous works, with the possible exception of [9]. We provide very detailed description of our algorithms to indicate that they are implementable.

We start the paper discussing in detail an optimal algorithm for computing the skyline in $O(n \log h)$ time. The algorithm uses the techniques by Chan [4] to compute the convex hull and the techniques by Nielsen [24] to compute the top-k skylines. We decided to provide a self-contained presentation of this part. The purpose is two-fold. First, we believe that the presentation of this part is didactic, slightly simpler than the presentation by Nielsen because we try to solve a simpler problem, and may find interest within the community interested in computing with skylines in the plane. Second, it provides the basic idea for our later approach, and thus it should be discussed to some level of detail in any case. Most precisely, in our approach we split P into t sets arbitrarily, compute the skyline of each of those subsets separately, and use those skylines to compute certain points along $\text{sky}(P)$, as we need them, by merging information from the t skylines.

Organization of the paper In Sect. 2 we provide some basic observations. In Sect. 3 we explain a simple, optimal algorithm to compute the skyline. Section 4 is devoted to the problem of finding the skyline representatives through the computation of the skyline, while Sect. 5 looks at the problem without computing the skyline. Results for very small k are considered in Sect. 6. We conclude in Sect. 7, where we also provide further research questions.

2 Basic tools

We are usually interested in the order of the points along the skyline $\text{sky}(P)$. For this we store the points of $\text{sky}(P)$ sorted by increasing x-coordinate in an array. We could also use any other data structure that allows us to make binary searches among the elements of $\text{sky}(P)$, such as a balanced binary search tree. Note that sorting $\text{sky}(P)$ by x- or by (decreasing) y-coordinate is equivalent.

Quite often we use an expression like "select the highest point among $q_1, \dots, q_t$, breaking ties in favor of larger $x(\cdot)$". This means that we take the point among $q_1, \dots, q_t$ with largest y-coordinate; if more points have the largest y-coordinate $y_{\max}$, then we select among those with $y(q_i) = y_{\max}$ the one with largest x-coordinate. This can be done iterating over $i = 1, \dots, t$, storing the best point q_* so far, and updating

$$q_* \leftarrow q_i \text{ if and only if } y(q_*) < y(q_i) \text{ or } \ (y(q_*) = y(q_i) \text{ and } x(q_*) < x(q_i)).$$

This wording appears to take care of the cases where more points have the same x- or y-coordinate. Let us explain an intuition that may help understanding how ties are broken.

Conceptually, think that each point $p = (x, y) \in P$ is replaced by the point $p_\varepsilon = (x + y\varepsilon, y + x\varepsilon)$ for an infinitesimally small $\varepsilon > 0$, and let P_ε be the resulting point set. A point $p \in P$ belongs to $\text{sky}(P)$ if and only if p_ε belongs to $\text{sky}(P_\varepsilon)$. Moreover, no two coordinates are the same in P_ε, if $\varepsilon > 0$ is sufficiently small. Thus, whenever we have to break ties, we think what we would do for P_ε.

For correctness and making binary searches we will often use the following monotonicity property. The proof is folklore; see Dupin, Nielsen and Talbi [7] for a generalization to other metrics.

Lemma 1 *If p, q, r are points of* $\text{sky}(P)$ *and $x(p) < x(q) < x(r)$, then $d(p, q) < d(p, r)$.*

Proof From the hypothesis we have $y(p) > y(q) > y(r)$ and therefore

$$\begin{aligned}\big(d(p,q)\big)^2 &= \big(x(q) - x(p)\big)^2 + \big(y(q) - y(p)\big)^2 \\ &< \big(x(r) - x(p)\big)^2 + \big(y(r) - y(p)\big)^2 = \big(d(p,r)\big)^2.\end{aligned}$$

□

A first consequence of this lemma is that any disk centered at a point of $\text{sky}(P)$ contains a contiguous subsequence of $\text{sky}(P)$. In particular, if $\text{sky}(P)$ is stored for binary searches along x-coordinates, then we can perform binary searches to find the boundary elements of $\text{sky}(P)$ contained in any given disk *centered at a point of* $\text{sky}(P)$. (The claim is not true for disks centered at arbitrary points of P.)

For an x-coordinate x_0 and a set of points Q, let $\text{succ}(Q, x_0)$ be the leftmost point of Q in the open halfplane to the right of the vertical line $x = x_0$. Similarly, we denote by $\text{pred}(Q, x_0)$ the rightmost point of Q in the open halfplane to the left of the vertical line $x = x_0$. Thus

$$\begin{aligned}\text{succ}(Q, x_0) &= \arg\min\{x(q) \mid q \in Q,\ x(q) > x_0\},\\ \text{pred}(Q, x_0) &= \arg\max\{x(q) \mid q \in Q,\ x(q) < x_0\}.\end{aligned}$$

We will use $\text{succ}(Q, x_0)$ and $\text{pred}(Q, x_0)$ when Q is a skyline; see Fig. 2. In our use we will also take care that $\text{succ}(Q, x_0)$ and $\text{pred}(Q, x_0)$ are well-defined, meaning that there is some point of Q on each side of the vertical line $x = x_0$.

When Q is a skyline stored for binary searches along x-coordinates, the points $\text{succ}(Q, x_0)$ and $\text{pred}(Q, x_0)$ can be obtained in $O(\log |Q|)$ time.

Looking into the case of skylines we can observe that

$$\begin{aligned}\text{succ}(\text{sky}(P), x_0) = \arg\max\{y(p) \mid p \in P,\ x(p) > x_0\},\\ \text{breaking ties in favor of larger } x(\cdot),\end{aligned}$$

while using $y_0 = \max\{y(p) \mid p \in P,\ x(p) \geq x_0\}$ we have

$$\begin{aligned}\text{pred}(\text{sky}(P), x_0) = \arg\max\{x(p) \mid p \in P,\ y(p) > y_0\},\\ \text{breaking ties in favor of larger } y(\cdot).\end{aligned}$$

In several of our algorithms we will use a divide and conquer approach. The following observation is one of the simple but key observations. See Fig. 2, right.

Lemma 2 *Assume that P is the disjoint union of $P_1, \ldots, P_t$. For each real value x_0, the point* $\text{succ}(\text{sky}(P), x_0)$ *is the highest point among* $\text{succ}(\text{sky}(P_1), x_0), \ldots, \text{succ}(\text{sky}(P_t), x_0)$, *breaking ties in favor of larger $x(\cdot)$. Moreover, if for each $i \in \{1, \ldots, t\}$ the points of* $\text{sky}(P_i)$

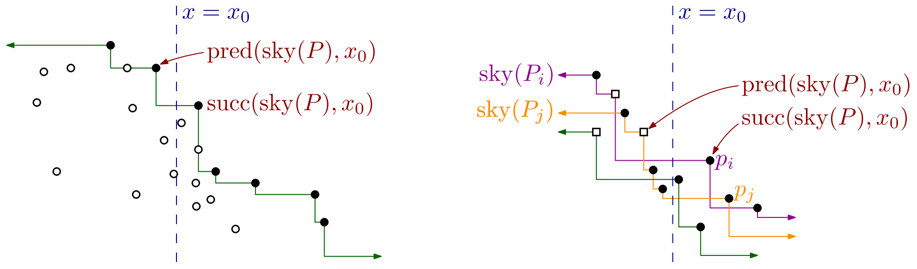


Fig. 2 Left: $\mathrm{succ}(\mathrm{sky}(P), x_0)$ and $\mathrm{pred}(\mathrm{sky}(P), x_0)$ for a set of points P. Right: Computing $\mathrm{succ}(\mathrm{sky}(P), x_0)$ and $\mathrm{pred}(\mathrm{sky}(P), x_0)$ when we have $\mathrm{sky}(P_1), \dots, \mathrm{sky}(P_t)$ for $P = P_1 \cup \dots \cup P_t$. The points $q_1, \dots, q_t$ are marked with empty squares

are sorted by x-coordinate for binary searches, then we can compute $\mathrm{succ}(\mathrm{sky}(P), x_0)$ *in* $O(t + \sum_i \log |P_i|)$ *time.*

Proof The point $\mathrm{succ}(\mathrm{sky}(P), x_0)$ is the highest point of P in the halfplane $x > x_0$, breaking ties in favor of larger $x(\cdot)$. Since

$$\begin{aligned} y\big(\mathrm{succ}(\mathrm{sky}(P), x_0)\big) &= \max\{y(p) \mid p \in P,\ x(p) > x_0\} \\ &= \max_{i=1,\dots,t} \ \max\{y(p_i) \mid p_i \in P_i,\ x(p_i) > x_0\} \\ &= \max_{i=1,\dots,t} \ y\big(\mathrm{succ}(\mathrm{sky}(P_i), x_0)\big)\}, \end{aligned}$$

the claim about the characterization of $\mathrm{succ}(\mathrm{sky}(P), x_0)$ follows. The algorithmic claim is obtained by making a binary search in each $\mathrm{sky}(P_i)$ to obtain $\mathrm{succ}(\mathrm{sky}(P_i), x_0)$ and choosing the highest point; in case of ties we select the one with largest x-coordinate. $\square$

We do not provide pseudocode for the simple algorithm of Lemma 2, but the idea will be used inside pseudocode provided later. We also have the following tool, which is slightly more complicated. It tests whether a point belongs to the skyline and computes the predecessor. Pseudocode under the name of TestSkylineAndPredecessor is described in Fig. 3.

Lemma 3 *Assume that P is the disjoint union of $P_1, \dots, P_t$ and for each $i \in \{1, \dots, t\}$ the points of* $\mathrm{sky}(P_i)$ *are sorted by x-coordinate for binary searches. For any point $p \in P$, the algorithm TestSkylineAndPredecessor solves the following two problems in* $O(t + \sum_i \log |P_i|)$ *time:*

- *decide whether $p \in$* $\mathrm{sky}(P)$*;*
- *compute* $\mathrm{pred}(\mathrm{sky}(P), x(p))$.

Proof For each $i \in \{1, \dots, t\}$ we use a binary search along $\mathrm{sky}(P_i)$ to find the point p_i of $\mathrm{sky}(P_i)$ with smallest x-coordinate among those with $x(p) \geq x(p_i)$. Let p_0 be the highest point among $p_1, \dots, p_t$, breaking ties in favor of those with larger x-coordinate. Note that p_0 is the highest point of $\mathrm{sky}(P)$ in $x \geq x(p)$; the proof of Lemma 2 can be trivially adapted for this. The point p is in $\mathrm{sky}(P)$ if and only if $p = p_0$.

Computing $\mathrm{pred}(P, x(p))$ is similar to computing $\mathrm{succ}(P, x(p))$, if we exchange the roles of x and y-coordinates. See Fig. 2, right, for an example. For each $i \in \{1, \dots, t\}$, we use a binary search along $\mathrm{sky}(P_i)$ *with respect to the y-coordinate* and key $y(p_0)$, to find the point $q_i \in \mathrm{sky}(P_i)$ with smallest y-coordinate among those with $y(q_i) > y(p_0)$. The point among

```
**Algorithm** *TestSkylineAndPredecessor*
**Input:** $\text{sky}(P_1), \ldots, \text{sky}(P_t)$ stored for binary searches and a point $p$.
**Output:** A pair $(X, q)$ where $X$ is True if $p \in \text{sky}(P_1 \cup \cdots \cup P_t)$ and False otherwise,
    and $q = \text{pred}(\text{sky}(P_1 \cup \cdots \cup P_t), x(p))$.
1.  **for** $i = 1, \ldots, t$ **do**
2.      $p_i \leftarrow$ point of $\text{sky}(P_i)$ in $x \geq x(p)$ with smallest $x$-coordinate, using binary
        search
3.  $p_0 \leftarrow$ highest point among $p_1, \ldots, p_t$, breaking ties in favor of larger $x(\cdot)$
4.  **for** $i = 1, \ldots, t$ **do**
5.      $q_i \leftarrow$ point of $\text{sky}(P_i)$ in $y > y(p_0)$ with smallest $y$-coordinate, using
        binary search
6.  $q_0 \leftarrow$ rightmost point among $q_1, \ldots, q_t$, breaking ties in favor of larger $y(\cdot)$
7.  **return** $(p = p_0?,\ q_0)$
```

Fig. 3 Algorithm described in Lemma 3 for testing whether $p \in \text{sky}(P)$ and computing $\text{pred}(\text{sky}(P), x(p))$

$q_1, \ldots, q_t$ with largest x-coordinate (and breaking ties in favor of the largest y-coordinate) is $\text{pred}(\text{sky}(P), x(p))$. (As seen in the example of Fig. 2, right, $\text{pred}(\text{sky}(P), x(p))$ is not necessarily any of the points among $\text{pred}(\text{sky}(P_1), x(p)), \ldots, \text{pred}(\text{sky}(P_t), x(p))$.)

Algorithm TestSkylineAndPredecessor implements the idea of this proof. □

3 Computing the skyline optimally

In this section we show that the skyline of a set P of n points can be computed in $O(n \log h)$, where $h = |\text{sky}(P)|$. The algorithm is simple and reuses the ideas exploited by Chan [4] and Nielsen [24]. We include it because of its simplicity and because we will modify it later on. Compared to Nielsen [24], our algorithm is slightly simpler because we are computing a slightly simpler object.

It is well-known and very easy to see that $\text{sky}(P)$ can be computed in $O(n \log n)$ time. Indeed, we just need to sort the points lexicographically: a point p precedes a point q if $x(p) < x(q)$ or if $x(p) = x(q)$ and $y(p) < y(q)$. Then we make a pass over the reversed list of sorted points, and maintain the point with largest $y(\cdot)$ scanned so far. The skyline of P is the list of local maxima we had during the procedure. See Fig. 4 for the idea. Details are given as Algorithm SlowComputeSkyline in Fig. 5. Note that the lexicographic order is important for correctness: if two points p_i and p_{i-1} have the same x-coordinate, then the point p_i has larger y-coordinate and thus p_{i-1} does not make it to the skyline. We will be using that $\text{sky}(P)$ is returned sorted by x-coordinate.

We now describe the algorithm computing $\text{sky}(P)$ in $O(n \log h)$ time. The core of the procedure is a subroutine, called ComputeSkylineBounded and described in Fig. 6, that takes as input P and an integer value parameter $s \geq 1$. The subroutine returns $\text{sky}(P)$ when $h \leq s$, and a warning "*incomplete*", when $h > s$. The role of s is that of a rough guess for the size of $\text{sky}(P)$.

The idea of the subroutine ComputeSkylineBounded is as follows. First we compute a value M that is an upper bound on the absolute values of all coordinates. We will use the points $(-M, M)$ and $(M, -M)$ as dummy points to avoid having boundary cases. Let

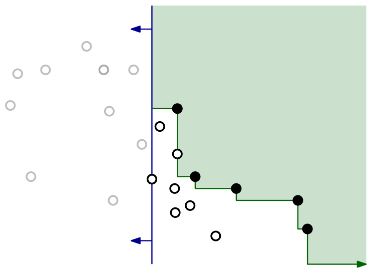


Fig. 4 Idea in SlowComputeSkyline

```
Algorithm SlowComputeSkyline
Input: A set $P$ of $n$ points.
Output: sky($P$) sorted by $x$-coordinate.
1.  $p_1, \dots, p_n \leftarrow$ sort $P$ lexicographically increasing
2.  output $\leftarrow$ new list with $p_n$
3.  $p \leftarrow p_n$      (* last point added to output *)
4.  for $i = n-1, \dots, 1$ do
5.      if $y(p_i) > y(p)$ then
6.          $p \leftarrow p_i$
7.          append $p_i$ to output
8.  return output
9.  (* we may reverse output if wished by increasing $x$-coordinate *)
```

Fig. 5 Algorithm to compute sky(P) in $O(n \log n)$ time

$\tilde{P} = P \cup \{(-M, M), (M, -M)\}$. We split P arbitrarily into $t \approx n/s$ groups of points, $P_1, \dots, P_t$, each with roughly s points, and add the dummy points $(-M, M)$ and $(M, -M)$ to each group P_i. We then compute the skyline of each group P_i using an algorithm with time complexity $O(n \log n)$, for example SlowComputeSkyline. Each outcome sky(P_i) is stored in an array sorted by increasing x-coordinate, so that we can perform binary searches along sky(P_i).

Now we can compute sky$(\tilde{P})$ = sky$(P) \cup \{(-M, M), (M, -M)\}$. Assuming that we are at a point p of sky$(\tilde{P})$, we can compute the next point $p' = \text{succ}(\text{sky}(\tilde{P}), x(p))$ along sky$(\tilde{P})$ by taking the highest among succ(sky$(P_1), x(p)), \dots,$ succ(sky$(P_t), x(p)$); ties are broken by taking the point with largest x-coordinate. The correctness of this is formalized in Lemma 2. The procedure is started from the dummy point $(-M, M)$, and iteratively compute the next point s times. If at some point we reach the dummy point $(M, -M)$, we know that we computed the whole sky(P) (we do not report $(-M, M)$ and $(M, -M)$ as part of the output). If after $s+1$ iterations we did not arrive to the dummy point $(M, -M)$, then we have computed at least $s+1$ points of sky(P) and correctly conclude that $|\text{sky}(P)| > s$.

Lemma 4 *Algorithm ComputeSkylineBounded has the following properties:*

- *ComputeSkylineBounded*(P, s) *returns* sky(P), *if* $|\text{sky}(P)| \le s$,
- *ComputeSkylineBounded*(P, s) *returns* "incomplete", *if* $|\text{sky}(P)| > s$,
- *ComputeSkylineBounded*(P, s) *takes* $O(n \log s)$ *time.*

Proof Because of Lemma 2, in lines 11-13 we correctly replace p by succ(sky$(\tilde{P}), x(p)$) in each iteration of the repeat loop. If we reach $p = (M, -M)$ at some point, then we know

```
Algorithm ComputeSkylineBounded
Input: A set $P$ of points, a positive integer $s$.
Output: skyline for $P$, if it contains at most $s$ points, "incomplete" otherwise.
1.  $M \leftarrow 1 + \max \bigcup_{p \in P} \{|x(p)|, |y(p)|\}$
2.  $t \leftarrow \lceil n/s \rceil$
3.  split $P$ arbitrarily into $t$ groups $P_1, \ldots, P_t$, each of at most $s$ points
4.  for $i = 1, \ldots, t$ do
5.      append the points $(-M, M)$ and $(M, -M)$ to $P_i$        (* dummy end points *)
6.      $S_i \leftarrow$ SlowComputeSkyline($P_i$)
7.      store $S_i$ for binary searches along $x$-coordinate
8.  output $\leftarrow$ new empty list
9.  $p \leftarrow (-M, M)$        (* dummy starting point *)
10. repeat $s+1$ times
11.     for $i = 1, \ldots, t$ do
12.         $p_i \leftarrow \mathrm{succ}(S_i, x(p))$, using binary search
13.     $p \leftarrow$ highest point among $p_1, \ldots, p_t$, breaking ties in favor of larger $x(\cdot)$
14.     if $x(p) = M$ then (* we arrived to $p = (M, -M)$ *)
15.         return output
16.     else
17.         append $p$ to output
18. return "incomplete" (* because skyline has more than $s$ points *)
```

Fig. 6 Algorithm to compute sky(P) in $O(n \log s)$ time, if $h \le s$

that we computed the whole sky(P). If after $s+1$ iterations we did not reach $p = (M, -M)$, then we computed $s+1$ points of sky(P) and the algorithm returns "*incomplete*".

It remains to bound the running time. For each P_i we spend $O(s \log s)$ time to compute sky(P_i). This means that we spend $O((n/s) \cdot s \log s) = O(n \log s)$ time to compute all sky$(P_1), \ldots,$ sky(P_t). In each iteration of the repeat loop we make $O(t) = O(n/s)$ binary searches among $O(s)$ elements, because each S_i has $O(|P_i|) = O(s)$ elements. Thus, in one iteration we compute the points $p_1, \ldots, p_t$ in $O(t \log s) = O((n/s) \log s)$ time, and selecting p then takes additional $O(n/s)$ time. We conclude that each iteration takes $O((n/s) \log s)$ time, and since there are $s+1$ iterations, the time bound follows. □

Note that Algorithm ComputeSkylineBounded can also be used to decide whether $h \le s$ for a given s in $O(n \log s)$ time.

To compute the skyline we make repetitive use of ComputeSkylineBounded(P, s) increasing the value of s doubly exponentially: if s is too small, we reset s to the value s^2, and try again. The intuition is that we want to make an exponential search for the value $\log h$ using terms of the form $\log s$. Thus, when we are not successful in computing the whole skyline because $s < h$, we want to set s to double the value of $\log s$. See the algorithm OptimalComputeSkyline in Fig. 7.

Theorem 5 *The skyline of a set of n points can be computed in $O(n \log h)$ time, where h is the number of points in the skyline.*

Proof Consider the algorithm OptimalComputeSkyline in Fig. 7. In the calls to ComputeSkylineBounded(P, s) we have s of the form 2^{2^r} for $r = 1, 2, 3 \ldots,$ until $h \le 2^{2^r}$.

```
**Algorithm** *OptimalComputeSkyline*
**Input:** A set $P$ of points.
**Output:** skyline for $P$.
1.  $output \leftarrow$ "incomplete"
2.  $s \leftarrow 4$
3.  **while** $output =$ "incomplete" **do**
4.      $output \leftarrow$ ComputeSkylineBounded$(P, s)$
5.      $s \leftarrow s^2$
6.  **return** $output$
```

Fig. 7 Algorithm to compute sky(P) in $O(n \log h)$ time

Thus, we finish when $r = \lceil \log_2(\log_2 h) \rceil$ and $s^{1/2} < h \leq s$. Thus the algorithm uses

$$\sum_{r=1}^{\lceil \log_2(\log_2 h)\rceil} O(n \log 2^{2^r}) = \sum_{r=1}^{\lceil \log_2(\log_2 h)\rceil} O(n2^r) = O\big(n2^{\lceil \log_2(\log_2 h)\rceil}\big) = O(n \log h)$$

time. □

A bit of thought shows that the exponent 2 in Line 5 of OptimalComputeSkyline can be replaced by any constant larger than 1. Thus, we could replace it by $s \leftarrow s^3$, for example.

4 Optimization via computation of the skyline

In this section we show how to solve the problem opt(P, k) in $O(n \log h)$ time, independently of the value of k. First we show how to solve the decision problem in linear time, assuming that sky(P) is available, and then consider the optimization problem.

4.1 Decision problem

Assume that sky(P) is already available and sorted by increasing x-coordinate.

For any point $p \in$ sky(P) and any $\lambda \geq 0$, the ***next relevant point*** for p with respect to λ, denoted by nrp(p, λ), is the point q of sky(P) that is furthest from p subject to the constraints that $d(p, q) \leq \lambda$ and $x(q) \geq x(p)$. See Fig. 8. Note that the next relevant point may be p itself, as it may be the only point of sky(P) that satisfies the conditions. Because of the monotonicity property of Lemma 1, we can find nrp(p, λ) scanning sky(P) from p onwards; the running time is proportional to the number of points of sky(P) between p and nrp(p, λ). (One could also use an exponential search, which is significantly better when $k \ll h$.)

For a given $\lambda \geq 0$ we can decide in $O(h)$ time whether opt$(P, k) \leq \lambda$. The idea is a simple greedy approach that is used for several other "1-dimensional" problems: we iteratively compute next relevant points to compute the centers. Detailed pseudocode is in Fig. 9.

Lemma 6 *Given a skyline S sorted by x-coordinate, an integer k and a value λ, we can decide in $O(h)$ time whether* opt$(S, k) \leq \lambda$.

Proof Consider the algorithm DecisionSkyline1 in Fig. 9. The input is an array $S[1 \ldots h]$ describing the skyline sorted by x-coordinate. We use an index i that scans the skyline $S[\ldots]$.

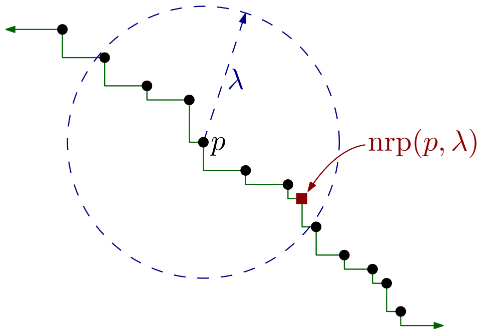


Fig. 8 The next relevant point for p with respect to λ is indicated with a squared mark

At the ath iteration of the for-loop, we have an index ℓ_a and we find the largest indices c_a, r_a such that $S[c_a] = \mathrm{nrp}(S[\ell_a], \lambda)$ and $S[r_a] = \mathrm{nrp}(S[c_a], \lambda)$. (We use ℓ_a, c_a and r_a because of *left*, *center* and *right* for the ath cluster.)

With a simple inductive argument we have the following property: after a iterations of the for-loop, the index i is the maximum index with the property that $S[1 \ldots (i-1)]$ can be covered with a disks of radius λ centered at points of S.

The procedure finishes when we use k centers but the last cluster does not cover $S[h]$ because $r_k < h$, or when we use at most k centers and we are covering $S[r_a] = S[h]$ because $i = h + 1$. In the former case we have $\mathrm{opt}(S, k) > \lambda$, while in the latter case we have $\mathrm{opt}(S, k) \leq \lambda$ and *output* has the list of centers.

The running time is $O(h)$ because the index i only increases and between consecutive increases we spend $O(1)$ time. □

4.2 Optimization problem

Let us turn our attention now to the optimization problem of computing $\mathrm{opt}(P, k)$ and an optimal solution.

We start computing $\mathrm{sky}(P)$ explicitly and storing it in an array $S[1 \ldots h]$ sorted by increasing x-coordinate. This takes $O(n \log h)$ time using the optimal algorithm presented in Theorem 5 or the algorithms in [17, 24].

Consider the $h \times h$ matrix A defined by

$$[A]_{i,j} = \begin{cases} d(S[i], S[j]) & \text{if } i < j, \\ -d(S[i], S[j]) & \text{if } i \geq j. \end{cases}$$

Lemma 1 implies that A is a ***sorted matrix***: each row has increasing values and each column has decreasing values. The matrix A is not constructed explicitly, but we work with it implicitly.

Note that $\mathrm{opt}(P, k)$ is one of the (non-negative) entries of A because $\mathrm{opt}(P, k)$ is an interpoint distance between points in $\mathrm{sky}(P)$. We perform a binary search among the entries of A to find the smallest entry λ^* in A such that $\mathrm{opt}(P, k) \leq \lambda^*$. We then have $\mathrm{opt}(P, k) = \lambda^*$.

To perform the binary search, we use an algorithm for the *selection problem* in sets of real numbers: given a set A and a natural number a with $1 \leq a \leq |A|$, we have to return the element with rank a when A is sorted non-decreasingly. Frederickson and Johnson [12, Theorem 1] show that the selection problem for the entries of a sorted matrix A of size $h \times h$

```
Algorithm DecisionSkyline1
Input: A skyline given as an array $S[1 \dots h]$ sorted by increasing $x$-coordinate, a positive
       integer $k \le h$ and a real value $\lambda \ge 0$.
Output: A solution $Q \subseteq S$ with $|Q| \le k$ and $\psi(Q,S) \le \lambda$, if $\mathrm{opt}(S,k) \le \lambda$, and
       "incomplete" if $\lambda < \mathrm{opt}(S,k)$.
1.  output ← new empty list
2.  $i \leftarrow 1$ (* counter to scan $S[\cdot]$ *)
3.  for $a = 1, \dots, k$ do
4.      $\ell_a \leftarrow i$ (* $S[\ell_a]$ is the first point to be covered by the $a$th center *)
5.      (* find $\mathrm{nrp}(S[\ell_a], \lambda)$ *)
6.      while $d(S[\ell_a], S[i]) \le \lambda$ and $i \le h$ do
7.          $i \leftarrow i + 1$
8.      $c_a \leftarrow i - 1$      (* $S[c_a]$ is the center for the $a$th cluster *)
9.      (* find $\mathrm{nrp}(S[c_a], \lambda)$ *)
10.     while $d(S[c_a], S[i]) \le \lambda$ and $i \le h$ do
11.         $i \leftarrow i + 1$
12.     $r_a \leftarrow i - 1$      (* dummy operation for easier analysis *)
13.     append $S[c_a]$ to the list output   (* $S[c_a]$ is a center that covers $S[\ell_a], \dots, S[r_a]$*)
14.     if $i > h$ then (* we finished scanning $S$ because $S[i-1] = S[h]$ *)
15.         return output
16. return "incomplete"
```

Fig. 9 Decision algorithm for testing whether $\mathrm{opt}(S, k) \le \lambda$. The index a is superfluous but helps analyzing the algorithm

can be solved in $O(h)$ time looking at $O(h)$ entries of A. This means that, for any given a with $1 \le a \le h^2$, we can select in $O(h)$ time the ath element of the entries of A.

This implies that we can make a binary search among the elements of A without constructing A explicitly. More precisely, we make $O(\log(h^2)) = O(\log h)$ iterations of the binary search, where at each iteration we select the ath element λ_a in A, for a chosen a, and solve the decision problem in $O(h)$ time using Lemma 6. In total we spend $O(h \log h)$ time. It is easy to obtain an optimal solution once we have $\lambda^* = \mathrm{opt}(P, k) = \mathrm{opt}(S, k)$ by calling DecisionSkyline1(S, λ^*). We summarize.

Theorem 7 *Given a set P of n points in the plane and an integer parameter $k \le h$, we can compute in $O(n \log h)$ time* $\mathrm{opt}(P, k)$ *and an optimal solution, where h is the number of points in the skyline* $\mathrm{sky}(P)$. *If* $\mathrm{sky}(P)$ *is already available, then* $\mathrm{opt}(P, k)$ *and an optimal solution can be found in $O(h \log h)$ time.*

An alternative approach is explained and justified in detail in Frederickson and Zhou [13, Lemma 2.1]. The bottom line is that we can perform a binary search in A using $O(h)$ time and looking at $O(h)$ entries in A, plus the time needed to solve $O(\log h)$ decision problems. We provide a high-level description and refer to [13] for the details, including pseudocode. The algorithm is iterative and maintains a family $\mathbb{A}$ of submatrices of A. We start with $\mathbb{A} = \{A\}$. At the start of each iteration we have a family $\mathbb{A}$ of submatrices of A that may contain the value λ^*, are pairwise disjoint, and have all the same size. In an iteration, we divide each submatrix in $\mathbb{A}$ into four submatrices of the same size, and perform two pruning steps, each using a median computation and solving a decision problem. At each iteration the size of the submatrices in $\mathbb{A}$ halves in each dimension and the number of submatrices in

$\mathbb{A}$ at most doubles. After $O(\log h)$ iterations, the family $\mathbb{A}$ contains $O(h)$ matrices of size 1×1, and we can perform a traditional binary search among those values. Actually, the algorithm of Frederickson and Johnson [12] for the selection problem follows very much the same paradigm, where a counter is used instead of a decision problem.

Both papers [12, 13] include pseudocode. For a practical implementation it is probably better to replace the deterministic linear-time computation of medians by a simple randomized linear-time computation [6, Section 2.4].

5 Optimization without computation of the skyline

We now describe another algorithm for the decision problem and for the optimization problem where we do not have $\mathrm{sky}(P)$ available, and we do not compute it.

5.1 Decision problem

Assume that we are given P, a positive integer k and a real value $\lambda \geq 0$. We want to decide whether $\mathrm{opt}(P, k) \leq \lambda$.

We use the same idea that was exploited in the algorithm ComputeSkylineBounded. We split the point set arbitrarily into groups and compute the skyline of each group. Then we note that we can use a binary search along the skyline of each group to find the next relevant point on the skyline of each group. First we provide a simple technical lemma and then show how to compute the next relevant point along $\mathrm{sky}(P)$. Details are provided as algorithm NextRelevantPoint in Fig. 12.

In our discussion we will be using the infinite curve $\alpha(p, \lambda)$ obtained by concatenating the infinite upward vertical ray from $p + (\lambda, 0)$, the circular arc centered at p and radius λ from $p + (\lambda, 0)$ to $p + (0, -\lambda)$ clockwise, and the vertical downward infinite ray from $p + (0, -\lambda)$. See Fig. 10.

Lemma 8 *Given a skyline Q sorted by x-coordinate such that Q has points on both sides of $\alpha(p, \lambda)$, we can compute in $O(\log |Q|)$ time the point with maximum x-coordinate among the points of Q lying to the left of $\alpha(p, \lambda)$ and and the point with minimum x-coordinate among the points lying to the right of $\alpha(p, \lambda)$.*

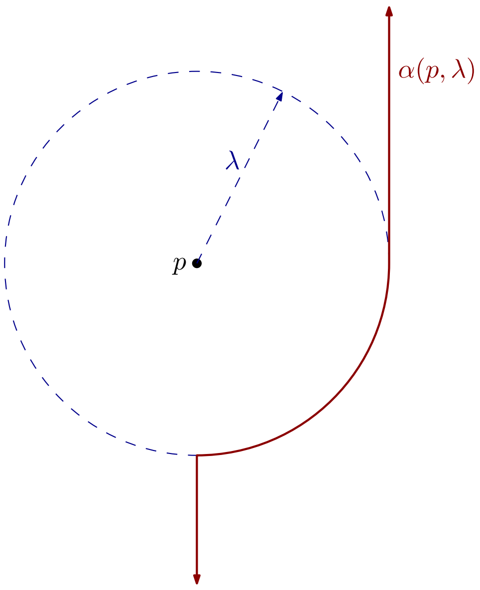
Proof Note that the set of points from Q to the left of $\alpha(p, \lambda)$ forms a contiguous subsequence of Q (sorted by x-coordinate). Moreover, for any point q, we can decide in constant time whether it is to the left or the right of $\alpha(p, \lambda)$. Thus, we can perform a binary search along Q to find where the change occurs. □

Lemma 9 *Assume that P is split into $P_1, \ldots P_t$, and for each $i \in \{1, \ldots, t\}$ the points of* $\mathrm{sky}(P_i)$ *are sorted by x-coordinate for binary searches. Let p be a point of* $\mathrm{sky}(P)$. *Then we can find the next relevant point* $\mathrm{nrp}(p, \lambda)$ *in $O(t + \sum_i \log |P_i|)$ time.*

Proof For each $i \in \{1, \ldots, t\}$, let q_i and q'_i be the points along $\mathrm{sky}(P_i)$ immediately before and after $\alpha(p, \lambda)$, respectively. The points q_i, q'_i can be computed with a binary search in $O(\log |P_i|)$ time; see Lemma 8.

Let q and q' be the points of $\mathrm{sky}(P)$ immediately before and after $\alpha(p, \lambda)$, respectively. The task is to find $q = \mathrm{nrp}(p, \lambda)$. Let γ be the L-shaped curve connecting q to $(x(q), y(q'))$ and then to q'. See Fig. 11. Let P_i be the group that contains q and let P_j be the group that

Fig. 10 The curve $\alpha(p, \lambda)$



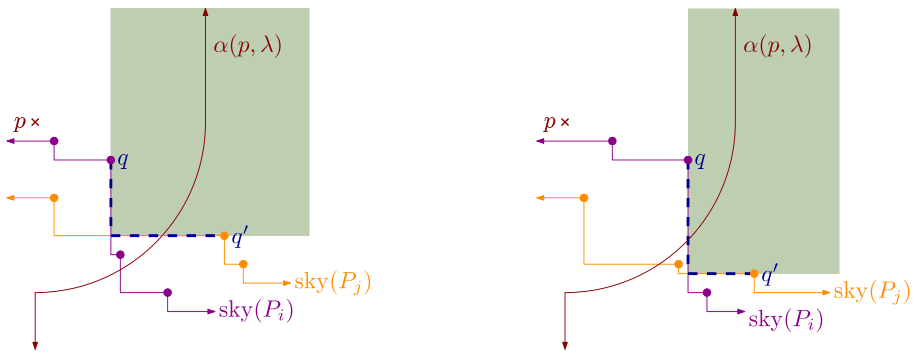


Fig. 11 Cases in the proof of Lemma 9; the shaded region is empty of points. Left: the case when the segment connecting $(x(q), y(q'))$ to q' crosses $\alpha(p, \lambda)$. Right: the case when the segment connecting q to $(x(q), y(q'))$ crosses $\alpha(p, \lambda)$

contains q'. If the horizontal segment connecting $(x(q), y(q'))$ to q' crosses $\alpha(p, \lambda)$, then q' is the highest point to the right of $\alpha(p, \lambda)$ (breaking ties in favor of larger x-coordinates). It follows that $q' = q'_j$ and moreover q' is the highest point among $q'_1, \ldots, q'_t$ (breaking ties in favor of larger x-coordinates). If the vertical segment connecting q to $(x(q), y(q'))$ crosses $\alpha(p, \lambda)$, then there is no point of P to the right of $x = x(q)$ and to the left of $\alpha(p, \lambda)$. If follows that $q = q_i$ and moreover q is the rightmost point among $q_1, \ldots, q_t$ (breaking ties in favor of larger y-coordinates).

We have seen that q is the rightmost point among $q_1, \ldots, q_t$ (breaking ties in favor of larger $y(\cdot)$) or q' is the highest point among $q'_1, \ldots, q'_t$, (breaking ties in favor of larger $x(\cdot)$), or both. As it can be seen in the examples of Fig. 11, it may be that only one of the claims is true.

```
Algorithm NextRelevantPoint
Input: sky($P_1$),...,sky($P_t$) stored for binary searches, a point $p \in$ sky($P_1 \cup \cdots \cup P_t$)
       and a real value $\lambda \geq 0$.
Output: nrp($p, \lambda$) for $P = P_1 \cup \cdots \cup P_t$.
1.  for $i = 1, \ldots, t$ do
2.      $q_i \leftarrow$ last point in sky($P_i$) to the left of or on $\alpha(p,\lambda)$, using binary search
3.      $q_i' \leftarrow$ succ(sky($P_i$), $x(q_i)$)      (* $q_i'$ to the right of $\alpha(p,\lambda)$ *)
4.  $q_0 \leftarrow$ rightmost point among $q_1, \ldots, q_t$, breaking ties in favor of larger $y(\cdot)$
5.  $q_0' \leftarrow$ highest point among $q_1', \ldots, q_t'$, breaking ties in favor of larger $x(\cdot)$
6.  $(X, r) \leftarrow$ TestSkylineAndPredecessor((sky($P_1$),...,sky($P_t$)), $q_0'$)
7.  if $X$ = True
8.       then return $r$
9.  else
10.     return $q_0$
```

Fig. 12 Algorithm to find the next relevant point using sky$(P_1), \ldots,$ sky(P_t)

Algorithmically we proceed as follows, using Lemmas 2 and 3 for some of the computations. We set q_0 to be the rightmost point among $q_1, \ldots, q_t$ (breaking ties in favor of larger $y(\cdot)$) and q_0' to be highest point among $q_1', \ldots, q_t'$ (breaking ties in favor of larger $x(\cdot)$). Using Lemma 3 we check whether q_0' belongs to sky(P). If q_0' belongs to sky(P), then we have $q' = q_0'$ and return pred(sky$(P), x(q_0')$), where we use Lemma 3 to compute pred( ). If q_0' does not belong to sky(P), then we have $q = q_0$ and return q_0. Details are provided in the algorithm NextRelevantPoint of Fig. 12. □

Now we can solve the decision problem using at most $2k$ calls to the function that finds the next relevant point. As we did in previous cases, we add dummy points to avoid boundary cases where the binary searches would have to return null pointers. Details are provided as algorithm DecisionSkyline2 in Fig. 13

For later use, we distinguish a preprocessing part, which is independent of λ and k, and a decision part that depends on λ and k. Indeed, later on we will be using the preprocessing once and the decision part multiple times.

Lemma 10 *Given a set P of n points and an integer parameter $\kappa \leq n$, we can preprocess P in $O(n \log \kappa)$ time such that, for any given real value $\lambda \geq 0$ and any positive integer k, we can decide in $O(k(n/\kappa) \log \kappa)$ time whether* opt$(P, k) \leq \lambda$.

Proof See DecisionSkyline2 in Fig. 13. Lines 1–11 correspond to the preprocessing, independent of λ and k, while lines 12–27 correspond to the decision problem. The running time for the preprocessing is $O(n + \sum_i |P_i| \log |P_i|) = O(n + (n/\kappa)\kappa \log \kappa) = O(n \log \kappa)$. In the decision part we perform at most k iterations, where each iteration takes $O(t) = O(n/\kappa)$ time plus the time for $O(1)$ calls to NextRelevantPoint (Lemma 9) with $t = O(n/\kappa)$. Thus, each query takes $O(k) \cdot \big(O(n/\kappa) + O((n/\kappa) \log \kappa)\big)$ time. The claim about the running times follow.

Regarding correctness, the same argument that was used in the proof of Lemma 6 applies. At the end of the ath iteration of the for-loop of lines 17–26 we have the following properties:

- the points $c_1, \ldots, c_a, \ell_1, \ldots, \ell_a, r_1, \ldots r_a$ belong to sky(P);
- the point r_a is the rightmost point of sky(P) with the property that the portion of sky(P) in $x \leq x(r_a)$ can be covered with a disks of radius λ centered at points of sky(P);

```
Algorithm DecisionSkyline2
Input: A set $P$ of points, positive integers $k$ and $\kappa \le n$ and a real value $\lambda \ge 0$.
Output: A solution $Q \subseteq \mathrm{sky}(P)$ with $|Q| \le k$ and $\psi(Q,P)$, if $\mathrm{opt}(P,k) \le \lambda$, and
    "incomplete" if $\lambda < \mathrm{opt}(P,k)$.
1.  (* Preprocessing *)
2.  $p_0 \leftarrow$ highest point of $P$, breaking ties in favor of larger $x(\cdot)$
3.  $q_0 \leftarrow$ rightmost point of $P$, breaking ties in favor of larger $y(\cdot)$
4.  $\lambda_{\max} \leftarrow 1 + d(p_0, q_0)$      (* upper bound for $\mathrm{opt}(P,k)$ *)
5.  $M \leftarrow 2\lambda_{\max} + \max \bigcup_{p \in P} \{|x(p)|, |y(p)|\}$
6.  $t \leftarrow \lceil n/\kappa \rceil$
7.  split $P$ arbitrarily into $t$ groups $P_1, \dots, P_t$, each of at most $\kappa$ points
8.  for $i = 1, \dots, t$ do
9.      append the points $(-M, M)$ and $(M, -M)$ to $P_i$      (* dummy end points *)
10.     $S_i \leftarrow \mathrm{SlowComputeSkyline}(P_i)$
11.     store $S_i$ in an array by increasing $x$-coordinate
12. (* Decision *)
13. if $\lambda \ge \lambda_{\max}$ then
14.     return $p_0$ (* or any other point of $\mathrm{sky}(P)$ *)
15. output $\leftarrow$ new empty list
16. $\ell_1 \leftarrow p_0$ (* first non-dummy point of $\mathrm{sky}(P)$ *)
17. for $a = 1, \dots, k$ do
18.     $c_a \leftarrow \mathrm{NextRelevantPoint}((S_1, \dots, S_t), \ell_a)$
19.     $r_a \leftarrow \mathrm{NextRelevantPoint}((S_1, \dots, S_t), c_a)$
20.     append $c_a$ to the list output      (* $c_a$ is a center that covers the portion of
        $\mathrm{sky}(P)$ from $\ell_a$ to $r_a$ *)
21.     (* compute $\ell_{a+1} = \mathrm{succ}(\mathrm{sky}(P), x(r_a))$ *)
22.     for $i = 1, \dots, t$ do
23.         $p_i \leftarrow \mathrm{succ}(S_i, x(r_a))$, using binary search
24.     $\ell_{a+1} \leftarrow$ highest point among $p_1, \dots, p_t$, breaking ties in favor of larger $x(\cdot)$
25.     if $x(\ell_{a+1}) = M$ then (* $\ell_{a+1}$ is the dummy point and we have finished *)
26.         return output
27. return "incomplete" (* because $\lambda < \mathrm{opt}(P,k)$ *)
```

Fig. 13 Decision algorithm for testing whether $\mathrm{opt}(P,k) \le \lambda$. The index a is superfluous but helps analyzing the algorithm

- the disks centered at $c_1, \dots, c_a$ cover the portion of $\mathrm{sky}(P)$ in $x \le x(r_a)$.

This claim holds by induction because $\mathrm{NextRelevantPoint}(p, \lambda)$ computes $\mathrm{nrp}(p, \lambda) \in \mathrm{sky}(P)$ whenever $p \in \mathrm{sky}(P)$; see Lemma 9. □

Setting $\kappa = k$ in Lemma 10 we obtain one of our main results, where we see that computing the skyline, which takes $O(n \log h)$ time, is not needed to solve the decision problem. The result is relevant when $\log k = o(\log h)$; for example, when $k = \log h$.

Theorem 11 *Given a set P of n points in the plane, a positive integer parameter k and a real value $\lambda \ge 0$, we can decide in $O(n \log k)$ time whether* $\mathrm{opt}(P,k) \le \lambda$.

Proof We use the Algorithm of Lemma 10, which is actually DecisionSkyline2 in Fig. 13, with $\kappa = k$. The preprocessing takes $O(n \log k)$ and deciding (the unique) λ takes $O(k(n/k) \log k) = O(n \log k)$ time. □

5.2 Optimization problem

Now we turn to the optimization problem. We use the paradigm of *parametric search*; see for example [13, 22, 29]. Our presentation can be understood without a previous knowledge of the technique.

For convenience we set $\lambda^* = \text{opt}(P, k)$.

We simulate running the decision algorithm DecisionSkyline2, analyzed in Lemma 10, for the *unknown* value λ^*. Whenever we run into a computation that depends on the value of λ^*, we apply a binary search to find an interval $(\lambda_1, \lambda_2]$ with the following properties: (i) $(\lambda_1, \lambda_2]$ contains λ^*, and (ii) the next step of the algorithm would be the same for any $\lambda \in (\lambda_1, \lambda_2]$. We can then perform the next step of the algorithm for λ^* as we would perform it for λ_2.

For the binary search we will use the following simple algorithm to select the element with a given rank in a collection of sorted arrays. As discussed in the proof, the lemma is not optimal but it suffices for our purpose and its proof is simple enough to implement it.

Lemma 12 *Assume that we have t arrays $S_1, \ldots, S_t$, each with numbers sorted increasingly. Set S to be the union of the elements in $S_1, \ldots, S_t$ and let n be the number of elements in the union. For any given value λ^*, we can find $\lambda' = \min\{x \in S \mid x \geq \lambda^*\}$ in $O(t \log^2 n)$ time plus the time needed to perform $O(\log n)$ comparisons between an element of S and λ^*.*

Proof We use a recursive algorithm. For simplicity we describe the algorithm assuming that the elements of $S_1, \ldots, S_t$ are pairwise different. Breaking ties systematically we can assume this; for example replacing each element $S_i[j]$ with the triple $(S_i[j], i, j)$ and making comparisons lexicographically. We keep an *active* contiguous portion of each S_i; we store it using indices to indicate which subarray is active. Thus, the recursive calls only takes $2t$ indices as input.

Let m_i be the median of the active subarray of S_i; assign weight w_i to m_i, where w_i is the number of active elements in S_i. We can find m_i and w_i in $O(1)$ time using arithmetic of indices. We select the weighted median M of $m_1, \ldots, m_t$ in $O(t)$ time. We compare M against λ^* to decide whether $\lambda^* \leq M$ or $M < \lambda^*$. If $\lambda^* \leq M$, then we clip the active part of each subarray S_i to the elements at most M. If $M < \lambda^*$, then we clip the active part of each subarray S_i to the elements at least M. Then we call recursively to search for λ^* in the active subarrays.

Note that at some point the active subarray of an array may be empty. This does not affect the approach; we just skip that subarray (or use weight 0 for that). Clipping one single subarray can be done in $O(\log n)$ time using a binary search. Each call to the function takes $O(t \log n)$ time, plus one comparison between an element of S and λ^*, plus the time for recursive calls. Since at each call we reduce the size of the active arrays by at least $1/4$, we make $O(\log n)$ recursive calls. The result follows.

It should be noted that there is also a very simple randomized solution that probably works better in practice: at each iteration choose one entry uniformly at random among all active subarrays of the arrays, compare the chosen element to λ^*, and clip the active subarrays accordingly. One can do faster deterministically [11, 12, 16], but it does not matter in our application. □

We next provide the parametric version of Lemma 9; the algorithm ParamNextRelevant-Point in Fig. 14 is the parametric counterpart of algorithm NextRelevantPoint.

Lemma 13 *Assume that P is split into $P_1, \ldots P_t$, and for each $i \in \{1, \ldots, t\}$ the points of* $\text{sky}(P_i)$ *are sorted by x-coordinate for binary searches. For any given point $p \in$* $\text{sky}(P)$ *we*

```
Algorithm ParamNextRelevantPoint
Input: $\mathrm{sky}(P_1), \dots, \mathrm{sky}(P_t)$ given as arrays $S_1[1 \dots h_1], \dots, S_t[1 \dots h_t]$ sorted by in-
    creasing $x$-coordinate, and a point $p \in \mathrm{sky}(P_1 \cup \dots \cup P_t)$.
Output: $\mathrm{nrp}(p, \lambda^*)$ for $P = P_1 \cup \dots \cup P_t$.
1.  for $i = 1, \dots, t$ do
2.      $q_i \leftarrow S_i[h_i]$        (* last point of $\mathrm{sky}(P_i)$ *)
3.      $a_i \leftarrow \min\{j \mid x(S_i[j]) \geq x(p)\}$
4.      (* $S_i[a_i, \dots, h_i]$ stores $\{q \in \mathrm{sky}(P_i) \mid x(p) \leq x(q)\}$ *).
5.  $q_0 \leftarrow$ rightmost point among $q_1, \dots, q_t$, breaking ties in favor of larger $y(\cdot)$
6.  (* $q_0$ is rightmost point of $\mathrm{sky}(P)$ *)
7.  (* boundary cases for the binary search *)
8.  if DecisionSkyline2$(P, k, \kappa, 0) \neq$ "incomplete" then
9.      return $p$
10. if DecisionSkyline2$(P, k, \kappa, d(p, q_0)) =$ "incomplete" then
11.     return "impossible" (* this never happens in our application *)
12. (* binary search in sorted lists $\bigcup_i \Lambda_i = \bigcup_i \{d(p,q) \mid q \in S_i[a_i, \dots, h_i]\}$ *)
13. use Lemma 12 to find $\lambda' = \min\{\lambda \in \bigcup_i \Lambda_i \mid \lambda^* \leq \lambda\}$
14. returnNextRelevantPoint$((\mathrm{sky}(P_1), \dots, \mathrm{sky}(P_t)), p, \lambda')$
```

Fig. 14 Algorithm to find the next relevant point for the unknown λ^* using $\mathrm{sky}(P_1), \dots, \mathrm{sky}(P_t)$

can compute the next relevant point $\mathrm{nrp}(p, \lambda^*)$ *in* $O(t \log^2 n)$ *time plus the time needed to solve* $O(\log n)$ *decision problems, without knowing* λ^*.

Proof Consider the algorithm ParamNextRelevantPoint given in Fig. 14. The input $\mathrm{sky}(P_i)$ is stored in an array $S_i[1 \dots h_i]$. In line 3 we perform a binary search along $S_i[\dots]$ to find the index a_i such that $S_i[a_i, \dots, h_i]$ contains the points in $x \geq x(p)$.

Let $\Lambda = \{d(p,q) \mid q \in \mathrm{sky}(P),\ x(p) \leq x(q)\}$ and for each $i \in \{1, \dots, t\}$ let $\Lambda_i = \{d(p,q) \mid q \in \mathrm{sky}(P_i),\ x(p) \leq x(q)\}$. Obviously we have $\Lambda \subseteq \bigcup_i \Lambda_i$ and $0 \in \Lambda$. The sets Λ and Λ_i are not constructed explicitly, but are manipulated implicitly.

The boundary cases, when $\lambda^* = 0 = \min \Lambda$ and when $\lambda^* > \max \Lambda$, are treated in lines 5–11. In the remaining case we have $0 < \lambda^* \leq \max \Lambda$.

Consider any index $i \in \{1, \dots, t\}$. Since $p \in \mathrm{sky}(P)$, the point p also belongs to $\mathrm{sky}(P_i \cup \{p\})$. From Lemma 1 we then obtain that the distances from p increase along $\mathrm{sky}(P_i)$, if we restrict to the right of the vertical line $x = x(p)$. It follows that

$$\forall j, j' \text{ with } a_i \leq j < j' \leq h_i : \ d(p, S_i[j]) < d(p, S_i[j']).$$

In short, the values of Λ_i, which are of the form $d(p,q)$, are sorted increasingly when q iterates over $S_i[a_i, \dots, h_i]$.

Since $\Lambda \subseteq \bigcup_i \Lambda_i$, we can perform a binary search in the union of the t sorted lists $\Lambda_1, \dots, \Lambda_t$ to find the smallest element λ' of $\bigcup_i \Lambda_i$ with $\lambda^* \leq \lambda'$. The next relevant point from p for λ^* and for λ' is the same. Thus, we can return the next relevant point for λ'.

To analyze the running time, we note that lines 1–5 perform $O(t)$ binary searches, each with a running time of $O(\log n)$, and some other operations using $O(t)$ time. Thus, this part takes $O(t \log n)$ time. In lines 8–11 we are making two calls to the decision problem plus $O(1)$ time. In lines 12-13 we use Lemma 12, which takes $O(t \log^2 n)$ time plus the time needed to solve $O(\log n)$ decision problems. □

```
Algorithm ParametricSearchAlgorithm
Input: A set $P$ of points and a positive integer $k \le n^{1/4}$.
Output: A solution $Q \subseteq \mathrm{sky}(P)$ with $|Q| \le k$ and $\psi(Q,P) = \mathrm{opt}(P,k)$.
1.  (* Preprocessing – same as in DecisionSkyline2 *)
2.  $p_0 \leftarrow$ highest point of $P$, breaking ties in favor of larger $x(\cdot)$
3.  $q_0 \leftarrow$ rightmost point of $P$, breaking ties in favor of larger $y(\cdot)$
4.  $\lambda_{\max} \leftarrow 1 + d(p_0, q_0)$      (* upper bound for $\mathrm{opt}(P,k)$ *)
5.  $M \leftarrow 2\lambda_{\max} + \max \bigcup_{p\in P}\{|x(p)|, |y(p)|\}$
6.  $\kappa \leftarrow k^3 \log^2 n$
7.  $t \leftarrow \lceil n/\kappa \rceil$
8.  split $P$ arbitrarily into $t$ groups $P_1, \dots, P_t$, each of at most $\kappa$ points
9.  for $i = 1, \dots, t$ do
10.     append the points $(-M, M)$ and $(M, -M)$ to $P_i$      (* dummy end points *)
11.     $S_i \leftarrow \mathrm{SlowComputeSkyline}(P_i)$
12.     store $S_i$ in an array by increasing $x$-coordinate
13. (* Parametric part *)
14. $output \leftarrow$ new empty list
15. $\ell_1 \leftarrow p_0$ (* first non-dummy point of $\mathrm{sky}(P)$ *)
16. for $a = 1, \dots, k$ do
17.     $c_a \leftarrow \mathrm{ParamNextRelevantPoint}((S_1, \dots, S_t), \ell_a)$
18.     $r_a \leftarrow \mathrm{ParamNextRelevantPoint}((S_1, \dots, S_t), c_a)$
19.     append $c_a$ to the list $output$      (* $c_a$ is a center that covers portion of $\mathrm{sky}(P)$
        from $\ell_a$ to $r_a$ *)
20.     (* compute $\ell_{a+1} = \mathrm{succ}(\mathrm{sky}(P), x(r_a))$ *)
21.     for $i = 1, \dots, t$ do
22.         $p_i \leftarrow \mathrm{succ}(S_i, x(r_a))$, using binary search
23.     $\ell_{a+1} \leftarrow$ highest point among $p_1, \dots, p_t$, breaking ties in favor of larger $x(\cdot)$
24.     if $x(\ell_{a+1}) = M$ then (* $\ell_{a+1}$ is the dummy point and we have finished *)
25.         return $output$
```

Fig. 15 Algorithm to find an optimal solution to $\mathrm{opt}(P,k)$. The index a is superfluous but helps analyzing the algorithm

Theorem 14 *Given a set P of n points in the plane and an integer parameter k, we can compute* $\mathrm{opt}(P,k)$ *and an optimal solution in $O(n \log k + n \log\log n)$ time.*

Proof If $k \ge n^{1/4}$, then $\log k = \Theta(\log n)$ and we can use Theorem 7 to obtain an algorithm with running time $O(n \log h) = O(n \log n) = O(n \log k)$.

For the case $k < n^{1/4}$, we consider the algorithm ParametricSearchAlgorithm given in Fig. 15. In lines 1–12 we make exactly the same preprocessing as in the algorithm DecisionSkyline2 with $\kappa = k^3 \log^2 n \le n$. As stated in Lemma 10, the preprocessing takes $O(n \log \kappa) = O(n(\log k + \log\log n))$ time and each subsequent decision problem can be solved in time

$$O(k(n/\kappa)\log\kappa) = O(k(n/k^3 \log^2 n)(\log k + \log\log n)) = O(n/k \log n).$$

The rest of the algorithm, lines 13–25, follows the paradigm of the decision part of DecisionSkyline2. We make k iterations, where at the ath iteration we compute points $\ell_a, c_a, r_a \in \mathrm{sky}(P)$ such that $c_a = \mathrm{nrp}(\ell_a, \lambda^*)$ and $r_a = \mathrm{nrp}(c_a, \lambda^*)$. The point ℓ_{a+1} is then set to be the point after r_a along $\mathrm{sky}(P)$.

We are making $O(k)$ calls to the function ParamNextRelevantPoint and, as shown in Lemma 13, each of them takes $O(t \log^2 n)$ time plus the time needed to solve $O(\log n)$ decision problems. In total we spend in all calls to ParamNextRelevantPoint

$$O(k) \cdot \big(O((n/k^3 \log^2 n) \log^2 n) + O(\log n) \cdot O(n/k \log n)\big) = O(n)$$

time. In lines 21-23 we spend additional $O(k) \cdot O(t \log \kappa) = O(n)$ time. The claim about the running time follows.

Correctness follows the same line of thought as for algorithm DecisionSkyline2 (see Lemma 10), using that ParamNextRelevantPoint correctly finds $\mathrm{nrp}(p, \lambda^*)$ for the unknown value $\lambda^* = \mathrm{opt}(P, k)$.

Note that the algorithm finds an optimal solution, but, as written, does not return the value $\mathrm{opt}(P, k)$. Assuming, for simplicity, that the returned solution has k points (it could have fewer), we then have

$$\mathrm{opt}(P, k) = \psi(P, \{c_1, \ldots, c_k\}) = \max\Big(\bigcup_{a=1,\ldots,k} \{d(c_a, \ell_a), d(c_a, r_a)\}\Big).$$

This quantity can be computed through the iterations of the for-loop. □

6 Algorithms for very small *k*

In this section we show that the problem $\mathrm{opt}(P, k = 1)$ can be solved in linear time; note that for this running time we cannot afford to construct the skyline explicitly. We also provide a $(1 + \varepsilon)$-approximation for $\mathrm{opt}(P, k)$ that is fast when k is very small. The main tool is the following result.

Lemma 15 *Let P be a set of n points in the plane and let p_0 and q_0 be two distinct points of $\mathrm{sky}(P)$ with $x(p_0) < x(q_0)$. Consider the portion of the skyline $S' = \{p \in \mathrm{sky}(P) \mid x(p_0) \le x(p) \le x(q_0)\}$; this portion or the whole skyline is not available. In $O(n)$ time we can compute points r_* and r'_* of S' such that*

$$r_* = \arg\min_{p \in S'} \max\{d(p, p_0), d(p, q_0)\} \quad \textit{and} \quad r'_* = \arg\max_{p \in S'} \min\{d(p, p_0), d(p, q_0)\}.$$

Proof We can assume that all the points p of P have $x(p_0) \le x(p) \le x(q_0)$ because other points can be ignored. Let β be the bisector of $p_0 q_0$; note that β has positive slope and is not vertical. Let p' be the point of $\mathrm{sky}(P)$ to the left of or on β with largest x-coordinate. Let q' be the point of $\mathrm{sky}(P)$ to the right of β with smallest x-coordinate. We show that $\{r_*, r'_*\} \subseteq \{p', q'\}$ and that p', q' can be computed in linear time. The result follows because we can just evaluate $\max\{d(p, p_0), d(p, q_0)\}$ and $\min\{d(p, p_0), d(p, q_0)\}$ for $p = p'$ and $p = q'$ and select the best.

First we show that $r_* = p'$ or $r_* = q'$. Consider any point $p \in S'$ with $x(p) < x(p')$; we then have $x(p_0) \le x(p) < x(p')$. Because of Lemma 1 and because p, p' are to the left of the bisector β we have

$$\max\{d(p, p_0), d(p, q_0)\} = d(p, q_0) > d(p', q_0) = \max\{d(p', p_0), d(p', q_0)\},$$

which means that p cannot be the optimal point r_*. A symmetric argument shows that any point $p \in S'$ with $x(q') < x(p) \le x(q_0)$ cannot be optimal. We conclude that r_* is p' or q'. A similar argument shows that r'_* is also p' or q'. Indeed, for any point $p \in S'$ with

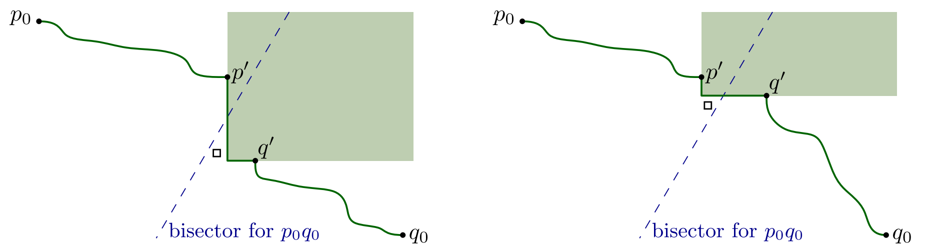


Fig. 16 Cases in the proof of Lemma 15; the shaded region is empty of points. Left: the case when the segment connecting p' to $(x(p'), y(q'))$ crosses β; the square represents a point showing that sometimes $q_1 \notin \text{sky}(P)$. Right: the case when the segment connecting $(x(p'), y(q'))$ to q' crosses β; the square represents a point showing that sometimes $p_1 \notin \text{sky}(P)$

$x(p_0) \le x(p) < x(p')$ we have

$$\min\{d(p, p_0), d(p, q_0)\} = d(p, p_0) < d(p', p_0) = \min\{d(p', p_0), d(p', q_0)\},$$

and the argument for $p \in S'$ with $x(q') < x(p) \le< x(q_0)$ is similar.

It remains to explain how to compute p' and q' in linear time. Let p_1 be the point to the left of or on β with largest x-coordinate (breaking ties in favor of larger y-coordinate), and let q_1 be the point to the right of β with largest y-coordinate (breaking ties in favor of larger x-coordinate).

The same argument that was used in the proof of Lemma 9 shows that $p_1 \in \text{sky}(P)$ or $q_1 \in \text{sky}(P)$ (or both). We repeat the argument to make the proof self-contained. See Fig. 16 Consider the last point $p' \in \text{sky}(P)$ to the left of or on β and the first point $q' \in \text{sky}(P)$ to the right of β. Define γ as the L-shape curve connecting p' to $(x(p'), y(q'))$ and to q'. If γ crosses β at the vertical segment connecting p' to $(x(p'), y(q'))$, then the point p' has to be the rightmost point to the left of or on β and thus $p' = p_1$. If γ crosses β at the horizontal segment connecting $(x(p'), y(q'))$ to q', then q' has to be the topmost point to the right of β and thus $q' = q_1$.

We check whether q_1 belongs to $\text{sky}(P)$ in linear time. Checking this amounts to checking whether $y(q_1)$ is the unique maximum among $y(q)$ for all the points $q \in P$ with $x(q_1) \le x(q) \le x(q_0)$. If $q_1 \in \text{sky}(P)$, then it must be $q_1 = q'$ and we can compute p' using that

$$x(p') = \max\{x(p) \mid x(p) < x(q_1),\ y(p) > y(q_1)\}$$

If $q_1 \notin \text{sky}(P)$, we infer that $p_1 \in \text{sky}(P)$ and thus $p' = p_1$. We can compute the point q' using that

$$y(q') = \max\{y(p) \mid x(p_1) < x(p)\}.$$

The whole procedure to find p' and q', as described, can be implemented to take linear time because we only need $O(1)$ scans of the point set P. $\square$

The 1-center problem can be solved in linear time using Lemma 15.

Theorem 16 *Given a set P of n points in the plane, we can compute in $O(n)$ time* $\text{opt}(P, 1)$ *and an optimal solution.*

Proof Let p_0 be the point with largest y-coordinate (breaking ties in favor of larger x-coordinate) and let q_0 be the point with the largest x-coordinate (breaking ties in favor of larger y-coordinate). The points p_0 and q_0 are the extreme points of $\text{sky}(P)$. Note that for each point $p \in \text{sky}(P)$ we have $\psi(\{p\}, P) = \max\{d(p, p_0), d(p, q_0)\}$ because of Lemma 1.

If $p_0 = q_0$, we return p_0 as the solution, which has cost 0. If $p_0 \neq q_0$, we apply Lemma 15 to compute

$$r_* = \arg \min_{p \in \text{sky}(P)} \max\{d(p, p_0), d(p, q_0)\} = \arg \min_{p \in \text{sky}(P)} \psi(\{p\}, P)$$

in linear time, and return r_*. □

We next provide a 2-approximation for $\text{opt}(P, k)$ that is relevant when k is very small. As soon as $k = \Omega(\log\log n)$, Theorem 14 is better, as it can compute an optimal solution.

Lemma 17 *Given a set P of n points in the plane and a positive integer k, we can compute in $O(kn)$ time a feasible solution $Q \subseteq \text{sky}(P)$ with at most k points such that $\psi(Q, P) \leq 2 \cdot \text{opt}(P, k)$. In the same time bound we also get $\psi(Q, P)$.*

Proof We assume $k \geq 2$ because the case $k = 1$ is covered by Theorem 16.

Let p_0 be the point with largest y-coordinate (breaking ties in favor of larger x-coordinate) and let q_0 be the point with the largest x-coordinate (breaking ties in favor of larger y-coordinate). The points p_0 and q_0 are the extreme points of $\text{sky}(P)$.

Set $c_1 = p_0$, $c_2 = q_0$, and for $i = 3, \ldots k$, let c_i be the point of $\text{sky}(P)$ that is furthest from $c_1, \ldots, c_{i-1}$. A classical result by Gonzalez [14] included in textbooks (for example [30, Section 2.2]) shows that $C = \{c_1, \ldots, c_k\}$ is a 2-approximation: we have $\psi(C, P) \leq 2 \cdot \text{opt}(P, k)$.

The points in C can be computed iteratively in $O(kn)$ time as follows. Assume that we computed $c_1, \ldots, c_i$ and we want to compute c_{i+1}. The vertical lines though $c_1, \ldots c_i$ split the region $x(p_0) \leq x \leq x(q_0)$ into slabs. We maintain for each slab σ the points of P that are inside σ. For each slab σ, let $c(\sigma) \in \{c_1, \ldots, c_i\}$ be the point that defines its left boundary and let $c'(\sigma) \in \{c_1, \ldots, c_i\}$ be the point defining its right boundary. Obviously $x(c) < x(c')$. We can use Lemma 15 to compute the point

$$r'_*(\sigma) = \arg \max_{p \in \text{sky}(P) \cap \sigma} \min\{d(p, c(\sigma)), d(p, c'(\sigma))\}.$$

Because of Lemma 1, for each slab σ and each $p \in \text{sky}(P) \cap \sigma$ we have

$$\max\{d(p, c_1), d(p, c_2), \ldots, d(p, c_i)\} = \max\{d(p, c(\sigma)), d(p, c'(\sigma)).$$

If follows that the point c_{i+1} has to be one of the points $r'_*(\sigma)$, where σ iterates over all slabs defined by $c_1, \ldots, c_i$. Thus we take c_{i+1} to be the point achieving the maximum

$$\max_{r'_*(\sigma)} \min\{d(r'_*(\sigma), c(\sigma)), d(r'_*(\sigma), c'(\sigma))\}.$$

Let σ^* be the slab defining c_{i+1}; thus $c_{i+1} = r'_*(\sigma^*)$. Then the slab σ^* has to be split into two subslabs with the vertical line $x = x(c_{i+1})$, and the points of $P \cap \sigma^*$ have to be rearranged into the two new subslabs. Since the slabs are interior disjoint, the use of Lemma 15 over all slabs takes linear time. The rest of the work to compute c_{i+1} and update the split of P into slabs also takes linear time. We conclude that the construction of C takes $O(kn)$ time.

Note that with an extra round, we can compute within each slab the point that is furthest from any point of C. This would be the steps to compute c_{k+1}. Computing the distance from c_{k+1} to C we obtain $\psi(C, P)$. □

From a 2-approximation we can compute a $(1+\varepsilon)$-approximation using binary search. This is relevant when $k = o(\log\log n)$, as otherwise Theorem 14 computes an optimal solution.

Theorem 18 *Given a set P of n points in the plane, a positive integer k and a real value ε with $0 < \varepsilon < 1$, we can compute in $O(kn + n \operatorname{loglog}(1/\varepsilon))$ time a feasible solution Q with at most k points such that $\psi(Q, P) \leq (1 + \varepsilon)\operatorname{opt}(P, k)$.*

Proof We use Lemma 17 to compute a value λ such that $\lambda \leq 2\operatorname{opt}(P, k) \leq 2\lambda$. This takes $O(kn)$ time. Then we perform a binary search among the $O(1/\varepsilon)$ values

$$\lambda,\ \lambda + \tfrac{\varepsilon\lambda}{2},\ \lambda + 2\tfrac{\varepsilon\lambda}{2},\ \lambda + 3\tfrac{\varepsilon\lambda}{2},\ \lambda + 4\tfrac{\varepsilon\lambda}{2}, \ldots, \approx 2\lambda$$

to find the index j such that

$$\lambda + (j-1)\tfrac{\varepsilon\lambda}{2} \ <\ \operatorname{opt}(P, k)\ \leq\ \lambda + j\tfrac{\varepsilon\lambda}{2}.$$

For this we have to solve $O(\log(1/\varepsilon))$ decision problems. Once we have the index j, we solve the decision problem for $\lambda + j\frac{\varepsilon\lambda}{2}$ and return the feasible solution Q that we obtain. This is a $(1 + \varepsilon)$ approximation because

$$\psi(Q, P)\ \leq\ \lambda + j\tfrac{\varepsilon\lambda}{2}\ =\ \left(\lambda + (j-1)\tfrac{\varepsilon\lambda}{2}\right) + \tfrac{\varepsilon\lambda}{2} < \operatorname{opt}(P, k) + \varepsilon\operatorname{opt}(P, k).$$

To analyze the running time of this step, we note that we have to solve $O(\log(1/\varepsilon))$ decision problems, all for the same number of points k. Using Theorem 10 with $\kappa = k^2\log^2(1/\varepsilon)$, we spend

$$O\Big(n\log(k^2\log^2\tfrac{1}{\varepsilon})\Big) + O(\log\tfrac{1}{\varepsilon}) \cdot O\Bigg(k\frac{n}{k^2\log^2\frac{1}{\varepsilon}}\log(k^2\log^2\tfrac{1}{\varepsilon})\Bigg) = O(n(\log k + \operatorname{loglog}\tfrac{1}{\varepsilon}))$$

time to solve the $O(\log(1/\varepsilon))$ decision problems. The term $O(n\log k)$ is absorbed by $O(kn)$. □

The theorem implies that for any constant k we can compute a $(1 + \varepsilon)$-approximation in $O(n \operatorname{loglog}(1/\varepsilon))$ time.

7 Discussion

We have improved previous results for computing the distance-based representatives of the skyline in the plane, a problem that is relevant in the context of databases, evolutionary algorithms and optimization. We have shown that such representatives can be computed without constructing the skyline for the whole point set, which is a conceptual shift with respect to previous works and approaches. For example, with a relatively simple algorithm we can solve the decision problem in $O(n\log k)$ time, which is asymptotically optimal. We also provided algorithms for the optimization problem that are asymptotically optimal for a large range of values of k: Theorem 7 is optimal when $\log k = \Omega(\log h)$ and Theorem 14 is optimal when $k = \Omega(\log n)$.

Most of our algorithms are easy enough to be implemented. Specially simple is the new decision algorithm that does not require computing the skyline. In several cases, the decision combined with a trivial binary search may suffice to get reasonable results. A practical implementation would use randomized algorithms for selection, instead of deterministic linear-time median finding.

One may wonder whether using the Euclidean distance is a reasonable choice. That depends on the application. The approach can be modified easily to work with other distances, such as the L_∞ or the L_1-metric. The work of Dupin, Nielsen and Talbi [7] also works for quite general metrics. The main property that we need is that any disk (in the chosen metric) centered at a point of the skyline intersects the skyline in a connected subpiece. From this we can infer monotonicity of the distances (Lemma 1) and we can perform binary searches along the skyline.

We next describe two research directions where progress is awaiting. The first question is whether we can compute $\mathrm{opt}(P, k)$ in $O(n \log k)$ for all values k. It may be good to start considering the case when k is constant. For example, can we compute $\mathrm{opt}(P, 15)$ in $O(n)$ time when $h = \Theta(\sqrt{n})$?

One can imagine scenarios where we want to solve $\mathrm{opt}(P, k)$ for several different values k. Can we exploit correlation between the problems in a non-trivial way? More precisely, what is the computational complexity of the following problem: given a set P of points in the plane and a set $K \subseteq \{1, \dots, n\}$, compute $\mathrm{opt}(P, k)$ for all $k \in K$.

References

1. Beume, N., Naujoks, B., Emmerich, M.T.M.: SMS-EMOA: multiobjective selection based on dominated hypervolume. Eur. J. Oper. Res. **181**(3), 1653–1669 (2007). https://doi.org/10.1016/j.ejor.2006.08.008
2. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proc. 17th International Conference on Data Engineering, ICDE 2001, pages 421–430. IEEE Computer Society, (2001). https://doi.org/10.1109/ICDE.2001.914855
3. Çalı, H., Labbé, M., Yaman, H.: p-center problems. In:Gilbert Laporte, Stefan Nickel, and Francisco Saldanha da Gama, editors, Location Science, chapter 3, pages 51–65. Springer, 2nd edition, (2019). https://doi.org/10.1007/978-3-030-32177-2_3
4. Chan, T.M.: Optimal output-sensitive convex hull algorithms in two and three dimensions. Discret. Comput. Geom. **16**(4), 361–368 (1996). https://doi.org/10.1007/BF02712873
5. Choi, J., Cabello, S., Ahn, H.-K.: Maximizing dominance in the plane and its applications Algorithmica, to appear. Preliminary Vers. in WADS (2019). https://doi.org/10.1007/s00453-021-00863-2
6. Dasgupta, S., Papadimitriou, C. H., Vazirani, U. V.: Algorithms. McGraw-Hill, (2008)
7. Dupin, N., Nielsen, F., Talbi, El-G., Unified polynomial dynamic programming algorithms for p-center variants in a 2d Pareto front. Mathematics, 9(4),: Preliminary version in Optimization and Learning - Third International Conference. OLA **2020**,(2021). https://doi.org/10.3390/math9040453
8. Emmerich, M.T.M., Deutz, A.H.: A tutorial on multiobjective optimization: fundamentals and evolutionary methods. Nat. Comput. **17**(3), 585–609 (2018). https://doi.org/10.1007/s11047-018-9685-y
9. Frederickson, G.N.: Optimal algorithms for tree partitioning. In: Proc. 2nd Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, SODA 1991, pages 168–177. ACM/SIAM, (1991). URL: http://dl.acm.org/citation.cfm?id=127787.127822
10. Frederickson, G.N.: Parametric search and locating supply centers in trees. In: Algorithms and Data Structures, 2nd Workshop WADS 1991, volume 519 of Lecture Notes in Computer Science, pages 299–319. Springer, (1991). https://doi.org/10.1007/BFb0028271
11. Frederickson, G.N., Johnson, D.B.: The complexity of selection and ranking in X+Y and matrices with sorted columns. J. Comput. Syst. Sci. **24**(2), 197–208 (1982). https://doi.org/10.1016/0022-0000(82)90048-4
12. Frederickson, G.N., Johnson, D.B.: Generalized selection and ranking: Sorted matrices. SIAM J. Comput. **13**(1), 14–30 (1984). https://doi.org/10.1137/0213002

13. Frederickson, G.N., Zhou, S.: Optimal parametric search for path and tree partitioning. CoRR, abs/1711.00599, (2017). arXiv:1711.00599
14. Gonzalez, T.F.: Clustering to minimize the maximum intercluster distance. Theor. Comput. Sci. **38**, 293–306 (1985). https://doi.org/10.1016/0304-3975(85)90224-5
15. Hochbaum, D.S., Shmoys, D.B.: A best possible heuristic for the k-center problem. Math. Oper. Res. **10**(2), 180–184 (1985). https://doi.org/10.1287/moor.10.2.180
16. Kaplan, H., Kozma, L., Zamir, O., Zwick, U.: Selection from heaps, row-sorted matrices, and X+Y using soft heaps. In: 2nd Symposium on Simplicity in Algorithms, SOSA@SODA,: volume 69 of OASICS, pages 5:1–5:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik **2019**,(2019). https://doi.org/10.4230/OASIcs.SOSA.2019.5
17. Kirkpatrick, D.G., Seidel, R.: Output-size sensitive algorithms for finding maximal vectors. In: Joseph O'Rourke, editor, Proc. 1st Annual Symposium on Computational Geometry, SoCG 1985, pages 89–96. ACM, (1985). https://doi.org/10.1145/323233.323246
18. Kung, H.T., Luccio, F., Preparata, F.P.: On finding the maxima of a set of vectors. J. ACM **22**(4), 469–476 (1975). https://doi.org/10.1145/321906.321910
19. Li, M., Yao, X.: Quality evaluation of solution sets in multiobjective optimisation: A survey. ACM Comput. Surv. **52**(2), 26:1-26:38 (2019). https://doi.org/10.1145/3300148
20. Lin, X., Yuan, Y., Zhang, Q., Zhang, Y.: Selecting stars: The k most representative skyline operator. In: Proc. 23rd International Conference on Data Engineering, ICDE 2007, pages 86–95. IEEE Computer Society, (2007). https://doi.org/10.1109/ICDE.2007.367854
21. Mao, R., Cai, T., Li, R.-H., Yu, J.X., Li, J.: Efficient distance-based representative skyline computation in 2D space. World Wide Web **20**(4), 621–638 (2017). https://doi.org/10.1007/s11280-016-0406-0
22. Megiddo, N.: Combinatorial optimization with rational objective functions. Math. Oper. Res. **4**(4), 414–424 (1979). https://doi.org/10.1287/moor.4.4.414
23. Mukhopadhyay, A., Maulik, U., Bandyopadhyay, S., Coello, C.A.C.: A survey of multiobjective evolutionary algorithms for data mining: Part I. IEEE Trans. Evol. Comput. **18**(1), 4–19 (2014). https://doi.org/10.1109/TEVC.2013.2290086
24. Nielsen, F.: Output-sensitive peeling of convex and maximal layers. Inf. Process. Lett. **59**(5), 255–259 (1996). https://doi.org/10.1016/0020-0190(96)00116-0
25. Peng, Raymond C-W., Wong, P.: Skyline queries and pareto optimality. In: M. Tamer Liu, Lingand Özsu, editor, Encyclopedia of Database Systems, pages 1–4. Springer New York, New York, NY, (2016). https://doi.org/10.1007/978-1-4899-7993-3_80684-1
26. Stefanidis, K., Koutrika, G., Pitoura, E.: A survey on representation, composition and application of preferences in database systems. ACM Trans. Database Syst. **36**(3), 19:1-19:45 (2011). https://doi.org/10.1145/2000824.2000829
27. Tao, Y., Ding, L., Lin, X, Pei, J.: Distance-based representative skyline. In: Proc. 25th International Conference on Data Engineering, ICDE 2009, pages 892–903. IEEE Computer Society, (2009). https://doi.org/10.1109/ICDE.2009.84
28. Tao, Y., Li, J., Ding, L., Lin, X., Pei, J.: On representing skylines by distance, (2013). Long version of [27] available at https://www.cse.cuhk.edu.hk/~taoyf/paper/icde09-long.pdf via https://www.cse.cuhk.edu.hk/~taoyf/pub.html
29. van Oostrum, R., Veltkamp, R.C.: Parametric search made practical. Comput. Geom. **28**(2–3), 75–88 (2004). https://doi.org/10.1016/j.comgeo.2004.03.006
30. Williamson, D.P., Shmoys, D.B.: The Design of Approximation Algorithms. Cambridge University Press, (2011). http://www.cambridge.org/de/knowledge/isbn/item5759340/?site_locale=de_DE
31. Yin, B., Wei, X., Liu, Y.: Finding the informative and concise set through approximate skyline queries. Expert Syst. Appl. **119**, 289–310 (2019). https://doi.org/10.1016/j.eswa.2018.11.004