Approximation Algorithms for Projective Clustering [*] [†]

Pankaj K. Agarwal[‡] Cecilia M. Procopiuc [‡]

Abstract

We consider the following two instances of the projective clustering problem: Given a set S of n points in $\mathbb{R}^d$ and an integer $k > 0$, cover S by k slabs (resp. d-cylinders) so that the maximum width of a slab (resp., the maximum diameter of a cylinder) is minimized. Let w^* be the smallest value so that S can be covered by k slabs (resp. d-cylinders), each of width (resp. diameter) at most w^*. This paper contains three main results: (i) For $d = 2$, we present a randomized algorithm that computes $O(k \log k)$ strips of width at most w^* that cover S. Its expected running time is $O(nk^2 \log^4 n)$ if $k^2 \log k \leq n$; for larger values of k, the expected running time is $O(n^{2/3}k^{8/3} \log^4 n)$. (ii) For $d = 3$, a cover of S by $O(k \log k)$ slabs of width at most w^* can be computed in expected time $O(n^{3/2}k^{9/4}\,\mathrm{polylog}(n))$. (iii) We compute a cover of $S \subset \mathbb{R}^d$ by $O(dk \log k)$ d-cylinders of diameter at most $8w^*$ in expected time $O(dnk^3 \log^4 n)$. We also present a few extensions of this result.

1 Introduction

Problem statement and motivation. A typical *projective clustering* problem can be defined as follows. Given a set S of n points in $\mathbb{R}^d$ and two integers $k < n$ and $q \leq d$, find k q-dimensional flats $h_1, \ldots, h_k$ and partition S into k subsets $S_1, \ldots, S_k$ so that

$$\max_{1 \leq i \leq k} \max_{p \in S_i} d(p, h_i)$$

is minimized. That is, we partition S into k clusters and each cluster S_i is projected onto a q-dimensional linear subspace so that the maximum distance between a point p and its projection p^* is minimized. Other objective functions have also been proposed [25, 30] for projective clustering. If $q = d - 1$, the above problem is equivalent to finding k hyper-strips that contain S so that the maximum width of a hyper-strip is minimized. If $q = 1$, then we want to cover S by k congruent hyper-cylinders of smallest radius. In typical applications, k is a small constant.

[*] Work on this paper is supported in part by National Science Foundation research grants CCR–9732287 and EIA–9870724, by Army Research Office MURI grant DAAH04–96–1–0013, by an NYI award, by a Sloan fellowship, and by a grant from the U.S.-Israeli Binational Science Foundation.

[†] A preliminary version of this paper appeared in the Proceedings of the 11th ACM-SIAM Symp. on Discrete Algorithms, 2000, pp. 538–547.

[‡]Center for Geometric Computing, Department of Computer Science, Box 90129, Duke University, Durham, NC 27708-0129, USA. E-mail: pankaj@cs.duke.edu, magda@cs.duke.edu

Projective clustering has recently received attention as a tool for creating more efficient nearest neighbor structures, as searching amid high dimensional point sets is becoming increasingly important. Since higher dimensional datasets are typically sparse (in the sense that the difference in some of the coordinates is large for every pair of points), clustering them using lower dimensional linear subspaces (or surfaces) is more meaningful than clustering them using full dimensional regions (e.g., covering points by balls or cubes) [8, 11, 19, 29]. For example, this approach is commonly used to classify image databases, using color and texture values [10]. Not only has clustering been used in data mining and classification applications [28, 37], it is also useful for constructing hierarchical index structures (e.g., R-trees, kd-trees) [11, 27, 40]. Since the query time and/or size of most of the geometric searching data structures increase exponentially with the dimension, projective clustering is used to reduce the dimensionality of the dataset. Various ad-hoc techniques have been proposed to reduce the dimensionality of the data, such as feature selection [31], subspace clustering [8, 7], and transformation of the data space into a lower dimensional space [19]. In this paper we develop approximation algorithms for some of the projective clustering problems.

Previous results. Clustering has been widely studied in several areas of computer science, including database systems [21, 28, 39], information retrieval, image processing, data compression, combinatorial optimization, and computational geometry; see the survey [5] and references therein for a sample of results. Although many theoretical results are known on facility location and related clustering problems [6, 13, 20], very few theoretical results are known on projective clustering.

Meggido and Tamir [36] showed that it is NP-complete to decide whether a set S of n points in $\mathbb{R}^2$ can be covered by k lines. This immediately implies that projective clustering is NP-Complete even in the planar case. In fact, it also implies that approximating the minimum width of strip covers (i.e., covering S by k congruent strips) within a constant factor is NP-Complete. Approximation algorithms for hitting compact sets by minimum number of lines are presented in [22]. Fitting a $(d-1)$-hyperplane through S, i.e., projective clustering with $q = d-1$ and $k = 1$, is the classical *width problem*. The width of a point set can be computed in $\Theta(n \log n)$ time for $d = 2$ [24, 32], and in $O(n^{3/2+\varepsilon})$ expected time for $d = 3$ [4]. Duncan *et al.* gave an algorithm for computing the width approximately in higher dimensions [17]. Several algorithms with near-quadratic running time are known for covering a set of n points in the plane by two strips of minimum width; see [26] and references therein. It is an open problem whether a subquadratic algorithm exists for this problem.

Besides these results, very little is known about the projective clustering problem, even in the plane. A few Monte Carlo algorithms have been developed for projecting S onto a single subspace [25]. We can also use the greedy algorithm [15] to cover points by congruent q-dimensional hypercylinders. More precisely, if S can be covered by k hyper-cylinders of radius r, then the greedy algorithm covers S by $O(k \log n)$ hyper-cylinders of radius r in time $n^{O(d)}$. The approximation factor can be improved to $O(k \log k)$ using the technique by Brönimann and Goodrich [12]. For example, this approach computes a cover of $S \subseteq \mathbb{R}^2$ by $O(k \log k)$ strips of a given width r in time $O(n^3 k \log k)$, assuming that S can be covered by k strips of width r each. No algorithm with roughly nk running time is known for this problem.

Our results. In this paper we first consider the simplest case of covering a set of points in the plane by strips. That is, given a set S of n points and a positive integer k, we want to cover S by k strips so that the maximum width of a strip is minimized. This is known as the *k-line-center* problem. Let w^* denote the minimum value so that S can be covered by k strips of width at most w^*. In typical applications, k is a small constant and n is very large, so an algorithm with near-linear running time as a function of n is desirable. For simplicity, we will assume that $k^2 \log k \leq n$. Since we cannot hope for an efficient algorithm that covers S by k strips of width cw^*, we relax the constraint on the number of strips (e.g., as in many of the k-median algorithms [33]). We propose a randomized algorithm that computes a cover of S by $O(k \log k)$ strips of width at most w^* in expected time $O(nk^2 \log^3 n \log(k \log n))$. Note that for constant k the running time becomes $O(n \log^c n)$. Our algorithm also works for larger values of k, but then the expected running time is $O(n^{2/3}k^{8/3} \log^4 n)$. The main feature of our algorithm is that its running time is near-linear as a function of n. Our method has the flavor of EM (expected maximization) algorithms, a widely used approach in discriminant analysis [34].

We then extend our approach to $\mathbb{R}^d$ and present an $O(dnk^3 \log^4 n)$-time algorithm to cover a set of n points in $\mathbb{R}^d$ by $O(dk \log k)$ hyper-cylinders of diameter $8w^*$, where w^* is the minimum diameter to cover S by k congruent hyper-cylinders. We also obtain efficient algorithms for covering $S \subseteq \mathbb{R}^3$ by hyper-strips, and for covering $S \subseteq \mathbb{R}^d$ by hyper-strips whose normals belong to a fixed set of vectors.

The paper is organized as follows: In Section 2, we introduce some definitions and prove a few simple results that are later used in our algorithm. Our approximation algorithm for the k-line-center problem is described in Section 3; we start by presenting a decision algorithm in Section 3.1, and then use it to develop the approximation algorithm in Section 3.2. We extend our algorithms to $\mathbb{R}^3$ in Section 4 and to higher dimensions in Section 5.

2 Preliminaries

A *strip* σ in the plane is the region lying between two parallel lines ℓ_1 and ℓ_2. The *width* of σ is the distance between ℓ_1 and ℓ_2, and the *direction* of σ is the direction of ℓ_1 and ℓ_2. A set Σ of strips is called a *strip cover* of S if each point of S lies in one of the strips of Σ. The *size* of Σ is the number of strips in Σ, and the *width* of Σ is the maximum width of a strip in Σ. For a fixed k, a strip cover Σ of size k is an *optimal strip cover* if its width is minimum among all strip covers of size k.

Let w^* denote the width of an optimal strip cover of S. Our algorithm first computes a strip cover Σ of S of size $O(k \log k)$ and width at most $6w^*$. We then divide each strip in Σ into six smaller strips by equidistant lines parallel to the direction of the strip. The resulting set of strips is a cover of S of size $O(k \log k)$ and width at most w^*. For simplicity, we refer to this as the "cutting technique." The computation of Σ is described in the next section. In the remainder of this section, we introduce a few useful notations and prove a result that will be crucial to our algorithm.

For any pair of points p, q, let ℓ_{pq} denote the line passing through p and q. If $p = q$, ℓ_{pq} is the horizontal line through p. For any three, not necessarily distinct, points p, q, r in the plane, we denote by $\sigma(p, q, r)$ the strip of width $2 \cdot d(r, \ell_{pq})$ having ℓ_{pq} as the median line. If $r \in \ell_{pq}$, $\sigma(p, q, r)$

is the same as ℓ_{pq}. Let $\Pi = \{\sigma(p,q,r) \mid p,q,r \in S\}$. We also use the notation $\sigma(p,q;w)$ to denote the strip of width $2w$ whose median line is ℓ_{pq}.

Lemma 2.1 *Let σ be a strip of width w, and let p be any point in $S \cap \sigma$. Then there exist points $q, r \in S$ so that $\sigma(p,q,r)$ covers all points of $S \cap \sigma$ and $d(r, \ell_{pq}) \leq 3w$.*

Proof: Let $S_\sigma = S \cap \sigma$. If $|S_\sigma| = 1$, then choose $q = r = p$ and the lemma follows. Suppose $|S_\sigma| > 1$ and let D be the diameter of S_σ. We choose q to be the point that is farthest from p; obviously, $d(p,q) \geq D/2$.

Let ρ be the smallest rectangle containing S_σ whose two edges lie on the boundary of σ (see Figure 1). The width of ρ is w. Let L be the length of ρ. Since the two sides of ρ that are perpendicular to the direction of σ must each pass through a point of S_σ, $L \leq D$. Let σ' be the thinnest strip in direction parallel to the line ℓ_{pq} that contains ρ. We denote by w' the width of σ'. Using the notations of Figure 1, we deduce:

$$w' = w_1 + w_2 \leq w + L \sin\alpha \leq w + D \cdot \frac{2w}{D} = 3w.$$

We choose $r \in S_\sigma$ to be the point that is farthest away from the line through p and q. Since $r \in \rho$, $d(r, \ell_{pq}) \leq 3w$. Moreover, $\sigma(p,q,r) \supset S_\sigma$, and the lemma follows. □

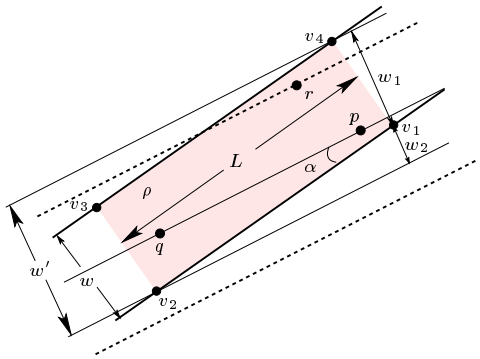


Figure 1. Finding a strip $\sigma(p,q,r)$ (dashed boundaries) that covers $S \cap \sigma$.

Let σ_i be a strip in an optimal cover of S, and $S_i = S \cap \sigma_i$. Lemma 2.1 implies that there exist at least $|S_i|$ pairs of input points $p, q \in S_i$ so that $\sigma(p,q;3w^*) \supseteq S_i$. This gives us some freedom in choosing one such pair of points for each σ_i. We use this fact in order to design a fast algorithm for computing a strip cover of S, of width at most $6w^*$.

Let $\tilde{w} \leq 3w^*$ denote the maximum distance between a point $r \in S$ and a line ℓ_{pq}, for $p, q \in S$, i.e.,

$$\tilde{w} = \max\{d(r, \ell_{pq}) \mid p,q,r \in S,\ d(r,\ell_{pq}) \leq 3w^*\}.$$

Lemma 2.1 and the definitions of Π and $\tilde{w}$ imply the following.

Corollary 2.2 *There exists a strip cover Σ of S of size k so that $\Sigma \subset \Pi$ and the width of Σ is at most $2\tilde{w} \leq 6w^*$.*

3 Approximating k-line-center

We first describe a decision algorithm that, given a real number w, returns "yes" if $w \geq \tilde{w}$. As a by-product, whenever it returns "yes," it also returns a strip cover of S of size $O(k \log k)$ and of width $2w$. We then use this decision algorithm and the parametric search technique by Meggido [35] to compute a strip cover of S of size $O(k \log k)$ and of width at most $2\tilde{w}$. As we show in Section 3.2, the parametric search is considerably simplified using the geometry of the problem, so our optimization algorithm is of practical value as well.

3.1 Decision algorithm

Since w remains unchanged throughout this subsection, we use $\sigma(p, q)$ to denote the strip $\sigma(p, q; w)$. We define the set system $\mathcal{X} = (E, \mathcal{R})$ as follows: $E = \{(p, q) \mid p, q \in S, p \neq q\}$ and $\mathcal{R} = \{\mathcal{R}_p \mid p \in S\}$, where $\mathcal{R}_p = \{(q, r) \in E \mid p \in \sigma(q, r)\}$. The elements of E are called *objects* and the elements of $\mathcal{R}$ are called *ranges*.

The problem of computing a strip cover $\Sigma \subseteq \Pi$ of S of width w is equivalent to finding a *hitting set* for the set system $\mathcal{X}$. We compute a hitting set by a method based on Clarkson's algorithm for polytope approximation [16], which was later extended by Brönimann and Goodrich [12]. Since we will be using the framework of Clarkson's algorithm, we sketch its main ideas. The algorithm works in phases. It assigns weights to each element in E. Initially, $wt(e) = 1$ for all $e \in E$. In each phase, it chooses a random subset $M \subseteq E$ of size $O(k \log k)$ so that each element of E is chosen with probability proportional to its weight. If M is a hitting set, it returns M. Otherwise, let $\mathcal{R}_p$ be a range so that $M \cap \mathcal{R}_p = \emptyset$. If $\sum_{e \in \mathcal{R}_p} wt(e) \leq wt(E)/(2k)$, it doubles the weight of each element in $\mathcal{R}_p$. Brönimann and Goodrich [12] show that, if $\mathcal{X}$ has a hitting set of size k and if the VC-dimension of $\mathcal{X}$ is finite, the algorithm returns a hitting set of size $O(k \log k)$ in $O(k \log n)$ expected phases; see [23] for definitions and results on VC-dimension and related notions.

Overall algorithm. If we apply the above method directly to our problem, the time taken by one phase is $\Omega(n^2)$ because $|E| = \Theta(n^2)$. We modify the basic algorithm, by using Lemma 2.1 and by taking advantage of the geometric properties of our problem, so that each phase is executed in $O(n \log(k \log n))$ time. A simpler variant of our algorithm will take $O(nk \log n)$ time per phase. There are three main ideas:

(i) It suffices to choose the random set M only from a small subset $N \subset E$ of size $O(nk \log n)$.

(ii) If we find a range $\mathcal{R}_p \in \mathcal{R}$ not hit by M, we double the weights of only a suitable subset of $\mathcal{R}_p$ of size $O(nk \log n)$. This significantly reduces the execution time of a phase, while still ensuring that the algorithm stops in $O(k \log n)$ expected number of phases.

(iii) We maintain the weights of the elements in N implicitly, so that they can be updated efficiently, and so that the weight of an element in N can be computed quickly.

DECISION PROCEDURE FastStrips (w)
$wt(e) = 1, \forall e \in E$; $successful = 0$;
$A \subset S$: random set of size $k \log n$; $N = S \times A$;
`repeat`
1. Select random set $R \subset S$ of size $ck \log k$;
   /* $\Pr[q \in S$ *is chosen*$] = wt(q)/wt(N)$. */
2. $\forall q \in R$, choose an object $(q, r),\ r \in A$;
   /* $\Pr[r \in A$ *is chosen*$] = wt((q, r))/wt(q)$. */
   $M : \{(q, r) \mid (q, r)$ chosen in Step 2$\}$;
   $\Sigma : \{\sigma(q, r) \mid (q, r) \in M\}$;

3. `if` $S \subset \bigcup_{\sigma \in \Sigma} \sigma$
   `return` "yes"; /* Σ is a strip cover.*/

4. Compute $p \in S \setminus \bigcup_{\sigma \in \Sigma} \sigma$;

5. `if` $wt(\mathcal{R}_p \cap N) \leq wt(E)/(2k)$
   5.1 $successful{+}{+}$;
   5.2 $wt(e) := 2wt(e), \quad \forall e \in \mathcal{R}_p \cap N$;
   5.3 $A := A \cup \{p\}$; $N = S \times A$;
`until` $successful = 12k \log n$;
`return` "no";

Figure 2. A fast decision procedure.

The algorithm is described in Figure 2. We need the following notation. We define the *weight function* $wt : E \to \mathbb{Z}^+$. Initially, $wt(e) = 1$ for all $e \in E$. The algorithm maintains a set $A \subseteq S$ so that the random set of objects M is chosen from the set $N = S \times A$. During the execution, new points may be added to A, but they are never deleted from A. For any point $p \in S$, we define $wt(p) = \sum_{q \in A} wt((p, q))$. We deduce that $\sum_{p \in S} wt(p) = wt(N)$. The weights $wt(p)$ are updated every time the set A changes. The set M is computed in two steps (Steps 1 and 2). To simplify notation, we denote by Σ the set of strips $\sigma(q, r)$, where $(q, r) \in M$. We say that a phase is *successful* if Steps 5.1–5.3 are executed, and *unsuccessful* otherwise.

Correctness. We prove that the decision procedure always returns "yes" if $w \geq \tilde{w}$, although it may return "yes" even if $w < \tilde{w}$.

Let S be a set of n points in the plane, and let $\Sigma^* = \{\sigma_1^*, \ldots, \sigma_k^*\}$ be an optimal strip cover of S. We define the *strip subsets* of S with respect to Σ^* to be the sets

$$S_i^* = \{p \in S \mid p \in \sigma_i^* \text{ and } p \notin \sigma_j^*, \forall j < i\}.$$

Thus, $S_1^*, \ldots, S_k^*$ partition S.

We prove that, for any $1 \leq m \leq k$, the set A intersects at least $m+1$ distinct strip subsets after $12m \log n$ successful phases, provided $w \geq \tilde{w}$.[1] As there are only k strip subsets, it follows that the decision algorithm must terminate (and return "yes") before executing $12k \log n$ successful phases.

Lemma 3.1 *Let $w \geq \tilde{w}$ be a real number. For any positive integer $m \leq k$, the set A intersects at least $m+1$ distinct strip subsets after the decision procedure* FastStrips*(w) has executed $12m \log n$ successful phases.*

Proof: Let $x = 12m \log n$. For any $0 \leq s \leq x$, let A^s denote the set A after s successful phases. Clearly, $A^{s_1} \subseteq A^{s_2}$ for any $s_1 \leq s_2$. Suppose on the contrary, A^x intersects only j strip subsets, $j \leq m$. Since $\bigcup S_i^* = S$ and $A^0 \subseteq A^x$, the set A^x intersects at least one strip subset, hence $j \geq 1$. Without loss of generality, suppose A^x intersects $S_1^*, \ldots, S_j^*$ and is disjoint from $S_{j+1}^*, \ldots, S_k^*$.

Let $r_i \in A^x \cap S_i^*, 1 \leq i \leq j$, be j points so that r_i is the first point of S_i^* that is added to A in Step 5.3 of the algorithm. By this we mean that either $r_i \in A^0 \cap S_i^*$, or there exists $1 \leq s \leq x$ so that $r_i \in S_i^* \cap (A^s \setminus A^{s-1})$ and $S_i^* \cap A^{s-1} = \emptyset$. Using Lemma 2.1, the definition of $\tilde{w}$, and the fact that $w \geq \tilde{w}$, we can prove that there exist j points $q_1, \ldots, q_j \in S$ so that $\sigma(q_i, r_i) \supset S_i^*, 1 \leq i \leq j$. For $1 \leq s \leq x$, let p_s be the point computed in Step 4 of the sth successful phase. Then $p_s \in \bigcup_{i=1}^{j} S_i^*$ (since $p_s \in A^x$ and A^x does not intersect $S_{j+1}^*, \ldots, S_k^*$). Let S_i^* be the strip subset containing p_s, $i \leq j$. If $r_i \in A^{s-1}$, then $wt((q_i, r_i))$ is doubled, because $(q_i, r_i) \in N \cap \mathcal{R}_{p_s}$. Otherwise, $p_s = r_i$. For $1 \leq i \leq j$, we denote by x_i the number of times $wt((q_i, r_i))$ is doubled during the first x successful phases. The observation above implies that $\sum_{i=1}^{j} x_i \geq x - j$. Since $wt(E)$ increases by a factor of at most $(1 + 1/(2k))$ after each successful phase, after $12m \log n$ successful phases we have

$$\prod_{i=1}^{j} \frac{wt((q_i, r_i))}{wt(E)} \geq \prod_{i=1}^{j} \frac{2^{x_i}}{n^2(1+1/2k)^x} \geq \frac{2^{x-j}}{n^{2j} e^{jx/2k}}.$$

Then

$$\begin{aligned}
\log\left(\frac{2^{x-j}}{n^{2j} e^{\frac{jx}{2k}}}\right) &= x - j - 2j \log n - \frac{j \log e}{2k} x \\
&= \left(1 - \frac{j \log e}{2k}\right) \cdot x - j - 2j \log n \\
&> \left(1 - \frac{\log e}{2}\right) \cdot \frac{3}{1 - \frac{\log e}{2}} m \log n - m - 2m \log n \\
&> 0
\end{aligned}$$

(recall that $x = 12m \log n$ and that $j \leq m$). Hence

$$\prod_{i=1}^{j} \frac{wt((q_i, r_i))}{wt(E)} > 1.$$

[1] The base of all logarithms is 2 unless stated explicitly.

This implies that for at least one i we have $wt((q_i, r_i)) > wt(E)$, a contradiction. □

Lemma 3.1 immediately implies the following:

Lemma 3.2 *For any value $w \geq \tilde{w}$, the decision procedure* FastStrips *(w) returns "yes" within $12k \log n$ successful phases. Whenever it returns "yes," the procedure also computes a cover of S of size $O(k \log k)$ and width $2w$.*

Remark 3.3 The proof of Lemma 3.1 implies that there exists a set A of size $O(k \log n)$ so that a strip cover of S of size k and width at most $2\tilde{w}$ can be chosen from the set $\{\sigma(q, r, p) \mid p, q \in S, r \in A\}$. However, we do not know how to compute $\tilde{w}$ and A directly.

Running time. For any $r \in A$, the object (q, r) is selected with probability $wt((q, r))/wt(N)$. Applying the result from the ε-net theory [23] to the set system induced by $\mathcal{R}$ on N (which has a finite VC-dimension), we have that

$$\Pr[wt(\mathcal{R}_p \cap N) \leq wt(N)/(2k)] \geq 1/2.$$

Since $wt(N) \leq wt(E)$, we deduce

$$\Pr[wt(\mathcal{R}_p \cap N) \leq wt(E)/(2k)] \geq 1/2.$$

Thus, a phase is successful with probability at least $1/2$. The expected number of phases is $O(k \log n)$.

We now analyze the time required by each phase. To simplify the analysis, we make the assumption that $k^2 \log k \leq n$. Let $a = |A| = O(k \log n)$. A naïve algorithm for executing a phase takes $O(na)$ time. Such an algorithm simply checks for each $p \in S$ and $(q, r) \in M$ whether $p \in \sigma(q, r)$. If p is not covered and $wt(N \cap \mathcal{R}_p) \leq wt(E)/2k$ (which we check by looking at all elements in $N \cap \mathcal{R}_p$), we double the weights for all objects in $N \cap \mathcal{R}_p$. We now show how each phase can be executed in $O(n \log a)$ time.

First, let us consider Step 3. Let M be the set of objects chosen in Step 2. Compute the arrangement $\mathcal{A}$ of the lines that form the boundaries of the strips $\sigma(q, r)$, for all $(q, r) \in M$; mark every cell of $\mathcal{A}$ that is contained in at least one strip $\sigma(q, r), (q, r) \in M$. Using an optimal planar point-location algorithm [18], for each point $p \in S$, find the cell of $\mathcal{A}$ that contains p. If the cell is marked, p is covered by at least one strip; otherwise, p is not covered by any strip. Obviously, this also solves Step 4. The time for computing the arrangement $\mathcal{A}$ is $O(|M|^2) = O(k^2 \log^2 k) = O(n \log k)$ since $n \geq k^2 \log k$, and we spend $O(n \log k)$ time to locate all points. Thus, Steps 3 and 4 are executed in $O(n \log k)$ time.

Data structures. To implement Steps 1, 2, and 5 efficiently, we use data structures that support the following three operations:

(i) Return a point q in S with probability $wt(q)/wt(N)$.

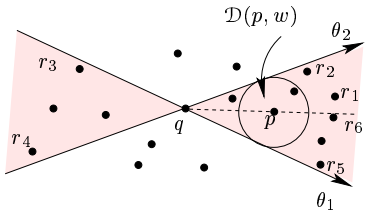


Figure 3. A_{pq} consists of (at most) three intervals in S_q.

(ii) Given a point $q \in S$, return an object $(q, r),\ r \in A$, with probability $wt((q, r))/wt(q)$.

(iii) For a given pair $p, q \in S$, double the weights of all objects in the set $\mathcal{R}_{pq} = \{(q, r) \in N \mid p \in \sigma(q, r)\}$.

For each point $q \in S$, we maintain a dynamic segment tree T_q as explained below. The underlying structure of a dynamic segment tree is a weighted $BB[\alpha]$-tree. Since we do not need the full functionality of a dynamic segment tree, we use a less sophisticated tree structure, such as a *red-black* tree.

Fix a point $q \in S$. Let A_q be the sequence of points in A sorted by the orientation of the rays $\overrightarrow{qr}, r \in A$. For $p \in S$, let $A_{pq} = \{r \in A \mid p \in \sigma(q, r)\} \subseteq \mathcal{R}_p$. Let $\mathcal{D}(p, w)$ denote the disk of radius w centered at p. If $q \in \mathcal{D}(p, w),\ A_{pq} = A$. Otherwise, $A_{pq} \subseteq A$ is the set of points that lie in the double wedge formed by the two tangent lines of $\mathcal{D}(p, w)$ passing through q (see Figure 3). Thus, A_{pq} can be represented by at most three linear intervals $[r_1, r_2], [r_3, r_4]$, and $[r_5, r_6]$, where $r_i \in A_q$. If all three intervals are nonempty, then r_1 (resp. r_6) is the first (resp. last) point of S_q.

Let J_q be the set of intervals representing the subsets A_{pq}, where p is a point reported in Step 4 of a successful phase. After each successful phase, at most one point is added to A_q and at most three new intervals are added to J_q. We store J_q in a dynamic segment tree T_q to maintain the weights of objects (q, r), $r \in A_q$. The leaves of T_q are labeled by the points in A_q in increasing order from left to right. For simplicity, we identify a leaf with its label. Each interior node v of T_q is associated with a canonical set S_v, which is the set of all points in A_q associated with the leaves of the subtree rooted at v. Each node v is also associated with a subset of intervals $J_q^v \subseteq J_q$. Instead of storing J_q^v explicitly at v, we will simply store the number of intervals in J_q^v, i.e., the value $k_v = |J_q^v|$. In a standard segment tree an interval J is stored at a node v if $S_v \subseteq J$ but $S_{p(v)} \not\subseteq J$. When we insert an interval into T_q we will use the same rule, but later, as other points and intervals are being inserted, we will not enforce the above condition, in the sense that J might be stored at a node v even if $S_{p(v)} \subseteq J$. The reason is that, as the structure of T_q changes due to the insertion of new points into A_q, we have to perform rotations on T_q to keep it balanced (see below). It is expensive to update the set J_q^v during rotations. However, we will always maintain the following invariant:

($\star$) *If an interval J is stored at nodes $v_1, \ldots, v_k$, then $J = \bigcup_{i=1}^{k} S_{v_i}$ and $S_{v_i} \cap S_{v_j} = \emptyset$.*

If T_q were static, the invariant above would imply that the weight $wt((q, r))$ of any pair $(q, r) \in S \times A$ can be computed as $2^{\sum_y k_y}$, where the summation is taken over all nodes of T_q on the path

from the root to the leaf labeled r. However, this does not hold for a dynamic tree: when adding a new leaf r' to T_q, the weight $wt((q, r'))$ is always 1, irrespective of how many intervals are already stored in the nodes from the root to r'. To solve this, we associate a value w_v to each node v of T_q such that

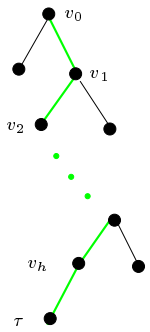
$$w_v = \frac{\sum_r wt((q, r))}{2^{\sum_y k_y}}, \tag{1}$$

where r ranges over all leaves in the subtree of v, and y ranges over all nodes on the path from the root to the parent of v. In the following, we use the notation Π_v to denote the path in T_q from the root to the parent of v. Initially, w_v is the number of leaves in the subtree of v. When inserting a new leaf r', we set $w_{r'} = \frac{1}{2^{\sum_y k_y}}$, where y ranges over all nodes on $\Pi_{r'}$. We also update the values w_v for the ancestors of r', as detailed below. Note that (1) implies $w_{root} = wt(q)$.

An interval $[r_i, r_j]$ is inserted into T_q using the standard segment tree procedure. If an interval is stored at an ancestor of v, then all leaves in the subtree rooted at v lie in this interval, so their weights double. Since both the numerator and the denominator in the expression of w_v increase by a factor of 2, w_v does not change. Thus, w_v changes only when an interval is stored at v or at one of its descendents. Using this observation, the values w_v can be updated while inserting an interval, without affecting the asymptotic running time of the segment-tree insertion procedure.

A point is inserted into A_q using the standard red-black tree insertion procedure. That is, we first traverse a path of T_q from the root to a leaf and insert the new point there; this is called the top-down phase. In the second phase, called the bottom-up phase, we traverse the same path backwards and perform $O(1)$ rotations to rebalance the tree. In each of the two phases, the values k_v and w_v are updated as follows.

Let τ be the new leaf that was created to insert a new point r in A_q, and let $\Pi_\tau = v_0, v_1, \ldots, v_h$ be the path from the root to the parent of τ. The procedure described in Figure 4 updates the values w_v in $O(\log a)$ time during the top-down phase. It can be checked that the values stored at the nodes after the top-down phase satisfy (1).

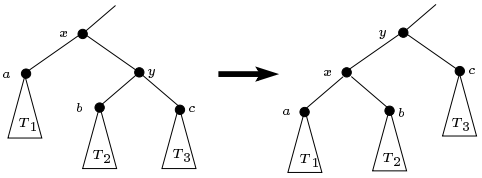


$w_{v_0} := w_{v_0} + 1;$
$P = 1;\ k = k_{v_0};$
for $i = 1$ **to** h **do**
begin
  $P := P \cdot 2^{k_{v_{i-1}}};$
  $w_{v_i} := w_{v_i} + 1/P;$
  $k := k + k_{v_i};$
end
$k_\tau := 0;\ w_\tau := \frac{1}{2^k};$

Figure 4. Updating values w_v in the top-down phase.

In the bottom-up phase, suppose we perform a left rotation at a node v. Figure 5 describes the structural change in T_q and the update of the values stored at each node. Intuitively, J_q^y, the set of intervals stored at y, is pushed to its children b and c, i.e., $J_q^b := J_q^b \cup J_q^y$ and $J_q^c := J_q^c \cup J_q^y$. Since

the set S_y after the rotation is the same as S_x before the rotation, we move J_q^x to J_q^y and set $J_q^x = \emptyset$. It can be checked that the new values stored at the nodes x, y, a, b, and c after the rotation satisfy (1).



$$
\begin{aligned}
&k_c := k_c + k_y;\\
&w_c := w_c \cdot 2^{k_y}\\
&k_b := k_b + k_y;\\
&w_b := w_b \cdot 2^{k_y}\\
&k_y := k_x;\ k_x := 0;\\
&w_x := w_a + w_b;\\
&w_y := 2^{k_y}(w_x + w_c);
\end{aligned}
$$

Figure 5. Maintaining values k_v and w_v under left rotation.

Details of the algorithm. With this definition for the trees T_q, Steps 1 and 2, as well as the test in Step 5 are executed as follows.

STEP 1: Check the roots of all segment trees and pick the points $q \in S$ with probability $w_{root(T_q)}/wt(N)$, where $wt(N) = \sum_{q \in S} w_{root(T_q)}$. Choosing one point $q \in S$ takes $O(n)$ time, so the total time for this step is $O(nk \log k)$.

STEP 2: For each point $q \in R$, we want to choose an object $(q, r), r \in A$, with probability $wt((q, r))/wt(q)$. Let $\mathrm{rk}(r)$ denote the rank of r in the order A_q. We choose a random number m between 1 and $wt(q)$ with uniform probability and then traverse T_q to determine the leaf $r \in A_q$ such that

$$\sum_{\substack{r' \in A_q \\ \mathrm{rk}(r') < \mathrm{rk}(r)}} wt((q, r')) < m \quad \text{and} \quad \sum_{\substack{r' \in A_q \\ \mathrm{rk}(r') \le \mathrm{rk}(r)}} wt((q, r')) \ge m. \tag{2}$$

The object (q, r) is returned. The traversal procedure maintains the invariant that when we are at a node v, we are given the value $\mu_v = 2^{\sum_{y \in \Pi_v} k_y}$ and an integer $m_v \in [1, \mu_v w_v]$, and we want to return a point of A_q that satisfies (2) for $m = m_v$ and for r' ranging only over the points in the subtree rooted at v. If v is a leaf, then r is the point associated with v. Otherwise, let u and z be the left and right children of v. Let $mu_u = \mu_z = \mu_v 2^{k_v}$. If $m_v \le \mu_u w_u$, then we visit u with $m_u = m_v$. Otherwise, we visit z with $m_z = m_v - \mu_u w_u$. We spend $O(1)$ time per visited node. Hence, this step takes $O(k \log(k) \log a)$ time, over all $O(k \log k)$ points $q \in R$.

STEP 5: First we describe the execution of Step 5.2, and then we explain how to determine the test in Step 5. Let p be the point computed in Step 4. In view of the preceding discussion, for each point $q \in S$ we want to return A_{pq} as at most three linear intervals in the sorted sequence A_q. Let θ_1, θ_2 be the orientations of the two tangent lines of $\mathcal{D}(p, w)$ passing through q. Since the leaves of T_q are sorted by the orientation of points in A with respect to q, we can compute the required intervals in $O(\log a)$ time by searching T_q with θ_1 and θ_2. We then insert these intervals in T_q and add p as a leaf in T_q. The total time spent in Step 5.2 is $O(n \log a)$.

It remains to describe how to execute the test in Step 5. For a fixed point $q \in S$, let I_1, I_2, I_3 be the three intervals representing A_{pq} as discussed above. We compute the value $\sum_{r \in A_{pq}} wt((q, r))$

as follows. Let I be one of the three intervals above. We traverse T_q to find the nodes associated with I, and we use the values k_v and w_v and (1) to compute $w(I) = \sum_{r \in I} wt((q,r))$. Then,

$$\sum_{r \in A_{pq}} wt((q,r)) = w(I_1) + w(I_2) + w(I_3).$$

Finally,

$$wt(\mathcal{R}_p) = \sum_{q \in S} \sum_{r \in A_{pq}} wt((q,r)).$$

The time required for computing $wt(\mathcal{R}_p)$ is thus $O(n \log a)$.

Taking into account the fact that the decision procedure terminates after $O(k \log n)$ successful phases, we conclude with the following.

Theorem 3.4 *For any real number $w \geq 0$, the expected running time of the decision algorithm* FastStrips *(w) is $O(nk \log(n) \log(k \log n))$ (assuming $k^2 \log k \leq n$). If $w \geq \tilde{w}$,* FastStrips *(w) returns "yes," together with a cover of S by $O(k \log k)$ strips of width at most $2w$.*

Remark 3.5 If $k^2 \log k > n$, we modify the implementation of Step 3, as it is inefficient to compute the entire arrangement $\mathcal{A}$. Instead we use the algorithm described in [1, 14] to determine whether S lies in the union of the strips in Σ. This step takes

$$O(|\Sigma|^{2/3} n^{2/3} \log n) = O(k^{2/3} n^{2/3} \log^{5/3} n)$$

time. A phase can thus be implemented in $O(n^{2/3} k^{2/3} \log^{5/3} n)$ time, and the expected running time of the decision algorithm is $O(n^{2/3} k^{5/3} \log^{8/3} n)$.

3.2 Optimization algorithm

We are now ready to describe our algorithm that, given a set of n points in the plane, computes a strip cover $\tilde{\Sigma}$ of S of size $O(k \log k)$ and width at most $2\tilde{w} \leq 6w^*$. Using the "cutting technique" as outlined in Section 2, we can then transform $\tilde{\Sigma}$ into a new cover $\tilde{\Sigma}'$ of size $O(k \log k)$ and width at most w^*. In view of Remark 3.1, there exists a subset $A \subseteq S$ of size $O(k \log n)$ so that $\tilde{\Sigma}$ can be chosen as a subset of the set $\{\sigma(q,r,p) \mid p,q \in S,\ r \in A\}$. If we could compute A efficiently, we could do a binary search on the set

$$W = \{d(p, \ell_{qr}) \mid p,q \in S,\ r \in A\},$$

using the decision procedure FastStrips(w) at each step. However, we do not know how to compute A efficiently, as it depends on the unknown value $\tilde{w}$. By Corollary 2.2, another possibility is to choose $\tilde{\Sigma}$ as a subset of $\Pi = \{\sigma(q,r,p) \mid p,q,r \in S\}$. But in this case we have to do a binary search on the set $\{d(p, \ell_{qr}) \mid p,q,r \in S\}$, and we do not know how to choose an element of given rank from this set in time $o(n^2)$. Our solution is to do binary searches on appropriate subsets of W that

we can compute efficiently. We borrow ideas from Meggido's parametric searching technique [35] and exploit the geometry of our problem to compute these subsets efficiently.

Our goal is to simulate the decision procedure FastStrips(w) generically at $\tilde{w}$, without knowing the value of $\tilde{w}$. To simplify notation, let $\mathcal{F}(w)$ denote the result returned by FastStrips(w). The algorithm maintains an interval $I = [\alpha, \beta]$ that satisfies the following invariant.

$$[\alpha, \beta] \neq \emptyset,\ \alpha < \tilde{w},\ \text{and } \mathcal{F}(\beta) = \text{“yes.”} \qquad (\mathcal{I}')$$

Initially, $I = [-\infty, +\infty]$. Since FastStrips(w) could return yes even if $w < \tilde{w}$, unlike the standard parametric searching, $\tilde{w}$ is not guaranteed to lie in the interval $[\alpha, \beta]$. The generic algorithm will not be able to detect immediately when it tries to set β to a value less than $\tilde{w}$. If this happens, then the generic algorithm will no longer be simulating FastStrips at $\tilde{w}$. The generic algorithm will either detect the situation at a later stage and return a strip cover of width at most $2\tilde{w}$, or it will finish the simulation. In the latter case, we will argue that $\beta \leq \tilde{w}$. We simulate the simpler version of the decision procedure that takes $O(nk \log n)$ time per phase and does not maintain segment trees. Steps 1 and 2 do not depend on the value $\tilde{w}$. We describe the simulation of Steps 3, 4, and 5 below.

In order to simulate Steps 3 and 4, we compute the set

$$W_1 = \{d(p, \ell_{qr}) \mid p \in S,\ (q, r) \in M\}$$

and sort it in increasing order. We then perform a binary search over W_1 to compute a value $w_i \in W_1$ so that $\mathcal{F}(w_i) =$ “yes” and $\mathcal{F}(w_{i-1}) =$ “no.” If $w_i \leq \alpha$, we return the strip cover computed by FastStrips(w_i). If $\beta \leq w_{i-1}$, we return the strip cover computed by FastStrips(β) because $\beta \leq w_{i-1} \leq \tilde{w}$. If $S \subseteq \bigcup_{(q,r)\in M} \sigma(q, r; w_{i-1})$, we return the set $\{\sigma(q, r; w_{i-1}) \mid (q, r) \in M\}$ as the strip cover of S. Otherwise, we set I to be $I \cap [w_{i-1}, w_i]$. One can easily check that I satisfies invariant $(\mathcal{I}')$. Since FastStrips(w) could return “yes” even if $w < \tilde{w}$, it is possible that $w_i < \tilde{w}$, but we cannot test it efficiently and therefore cannot terminate the algorithm. Let $w_i = d(p, \ell_{qr})$ for $(q, r) \in M$ and $p \in S$. Then $p \notin \bigcup_{(q,r)\in M} \sigma(q, r; w_{i-1})$. We set p to be the point computed in Step 4. Since $\tilde{w}$ could be larger than w_i, it is possible that $p \in \sigma(q, r; \tilde{w})$.**Not clear.** Nevertheless, we assume that p is not covered and proceed. As we prove below, this does not affect the correctness of our algorithm.

To simulate Step 5, we compute and sort

$$W_2 = \{d(p, \ell_{qr}) \mid q \in S,\ r \in A\}$$

where p is the point computed in Step 4. We then perform a binary search over W_2 to compute a value $w_j \in W_1$ so that $\mathcal{F}(w_j) =$ “yes” and $\mathcal{F}(w_{j-1}) =$ “no.” If $w_j \leq \alpha$, we return the strip cover computed by FastStrips(w_j). If $\beta \leq w_{j-1}$, we return the strip cover computed by FastStrips(β). Otherwise, we set I to be $I \cap [w_{j-1}, w_j]$, and we let $\mathcal{R}_p \cap N$ be the set of objects $(q, r) \in N$ for which $d(p, \ell_{qr}) \leq w_{j-1}$. One can easily check that invariant $(\mathcal{I}')$ is maintained. If $wt(\mathcal{R}_p \cap N) \leq wt(E)/(2k)$, we double the weights of all the objects in $\mathcal{R}_p \cap N$. That is, for each $q \in S$, we compute at most three intervals that represent the set A_{pq} and insert p and these intervals in the segment tree T_q. This step is independent of $\tilde{w}$, so it can be performed as in the decision algorithm.

If the algorithm does not terminate during the simulation, let $[\alpha_f, \beta_f]$ be the final interval.

Lemma 3.6 *If the generic procedure has not returned a strip cover of size $O(k \log k)$ and width at most $2\tilde{w}$ within $12k \log n$ successful phases, then $\beta_f \leq \tilde{w}$ and FastStrips(β_f) returns "yes" together with a cover of size $O(k \log k)$ and width $2\beta_f$.*

Proof: A simple inductive argument proves that the interval $I = [\alpha, \beta]$ maintained by the algorithm always satisfies invariant $(\mathcal{I}')$.

Suppose the generic procedure returns within $12k \log n$ successful phases. This can happen either during the execution of Step 3, or during the execution of Step 5. An easy argument shows that in each case the returned cover has width smaller than $2\tilde{w}$.

Assume now that the generic procedure executes $12k \log n$ successful phases. Since $[\alpha_f, \beta_f]$ satisfies invariant $(\mathcal{I}')$, it follows that $\alpha_f < \tilde{w}$ and $\mathcal{F}(\beta_f) =$"yes." For the sake of contradiction, suppose $\beta_f > \tilde{w}$. We prove below that all tests executed during the simulated algorithm would return the same result if they were executed for $\tilde{w}$. Then, by the standard argument from parametric search, the simulated generic algorithm works exactly as FastStrips($\tilde{w}$). By Lemma 3.2, the algorithm must return within $12k \log n$ successful phases, a contradiction.

Let $w_{i-1}, w_i \in W_1$ be the two values determined during the simulation of Step 3 of some iteration. Let p be the point computed during the simulation of the subsequent Step 4. Since $[\alpha_f, \beta_f] \subseteq [w_{i-1}, w_i]$, it follows that $w_{i-1} < \tilde{w} < w_i$. Recall that w_{i-1} and w_i are consecutive values in the set W_1 and that $p \in \left(\bigcup_{(q,r) \in M} \sigma(q, r; w_i) \setminus \bigcup_{(q,r) \in M} \sigma(q, r; w_{i-1}) \right)$. It then follows that $p \notin \bigcup_{(q,r) \in M} \sigma(q, r; \tilde{w})$ and thus Steps 3 and 4 return the same result as if they were executed for $\tilde{w}$. Let $w_{j-1}, w_j \in W_2$ be the two values determined during the simulation of the subsequent Step 5. As above, we deduce that $w_{j-1} < \tilde{w} < w_j$. We claim that $\mathcal{R}_p \cap N$ is equal to the set of objects

$$X = \{(q, r) \in N \mid d(p, \ell_{qr}) \leq w_{j-1}\}.$$

Indeed, $w_{j-1} < \tilde{w}$ immediately implies $X \subseteq \mathcal{R}_p \cap N$. Conversely, let $(q, r) \in \mathcal{R}_p \cap N$, hence $d(p, \ell_{qr}) \leq \tilde{w}$. Suppose for a contradiction that $d(p, \ell_{qr}) > w_{j-1}$. Since w_{j-1} and w_j are consecutive values in W_2, it follows that $d(p, \ell_{qr}) \geq w_j > \tilde{w}$, a contradiction. Hence, $\mathcal{R}_p \cap N = X$ and the simulation of Step 5 proceeds as if it was executed for $\tilde{w}$. □

The simulation of each phase requires $O(nk \log^2 n)$ time to compute and sort W_1 and W_2, and calls the decision procedure FastStrips $O(\log n)$ times. We summarize our result in the following theorem.

Theorem 3.7 *Given a set S of n points in the plane and a positive integer k so that $k^2 \log k \leq n$, we can compute in $O(nk^2 \log^3 n \log(k \log n))$ expected time a cover of S of size $O(k \log k)$ and of width at most w^*. If $k^2 \log k > n$, the expected running time is $O(n^{2/3} k^{8/3} \log^{14/3} n)$.*

4 Covering by Slabs in $\mathbb{R}^3$

Let S be a set of n points in $\mathbb{R}^3$, and let $k > 0$ be an integer. A *slab* in $\mathbb{R}^3$ is the region bounded by two parallel planes. The *direction* of a slab is the line passing through the origin that is orthogonal

to the slab boundaries and that is oriented in the $(+z)$-direction. In this section we describe an algorithm for covering S by $O(k \log k)$ congruent slabs of width at most w^*, where w^* is the minimum width of k congruent slabs that cover S, by extending the algorithm described in Section 3. For simplicity, we assume the points of S to be in general position, i.e. no three points of S are colinear, and no four points of S are coplanar.

For any three (not necessarily distinct) points p, q, and r, let π_{pqr} be the plane passing through them. If all of them are the same point, then π_{pqr} is the horizontal plane passing through p, and if only two of them, say p and q, are the same, then π_{pqr} is the plane obtained by translating the line π_{pr} in the horizontal direction normal to it. For any four points $p, q, r, s \in \mathbb{R}^3$, let $\sigma(p, q, r, s)$ denote the slab of width $2d(\pi_{pqr}, s)$ with π_{pqr} as its median plane. Let $\Pi = \{\sigma(p, q, r, s) \mid p, q, r, s \in \mathcal{S}\}$. The following lemma is a generalization of Lemma 2.1.

Lemma 4.1 *Let P be a set of points in $\mathbb{R}^3$ lying in a slab σ of width w , and let ϕ be a direction so that the angle between ϕ and the direction of σ is α. Then the width of P in direction ϕ is at most $w + \sqrt{2}\,\mathrm{diam}(P)\sin\alpha$.*

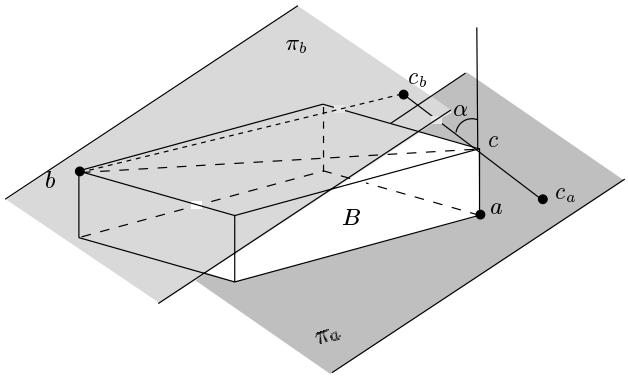


Figure 6. Bounding the width of a slab, in a given direction, that contains P.

Proof: Let B be the smallest box that covers P and whose two opposite faces are parallel to the boundary planes of σ. Let τ be the thinnest slab in direction ϕ that contains B; the boundary planes of τ pass through two diagonally opposite vertices, say a and b, of B. See Figure 6. Let π_a (resp. π_b) be the boundary plane of σ passing through a (resp. b). Let ac be the edge of B in the direction of σ, and let c_a (resp. c_b) be the projection of c onto π_a (resp. π_b). Then

$$\begin{aligned} \mathrm{width}(\tau) &= \|c_a c_b\| = \|c_a c\| + \|c c_b\| \\ &\leq w + \|bc\| \sin(\angle c_b b c) \\ &\leq w + \sqrt{2}\,\mathrm{diam}(P)\cos(\angle c_b c b). \end{aligned}$$

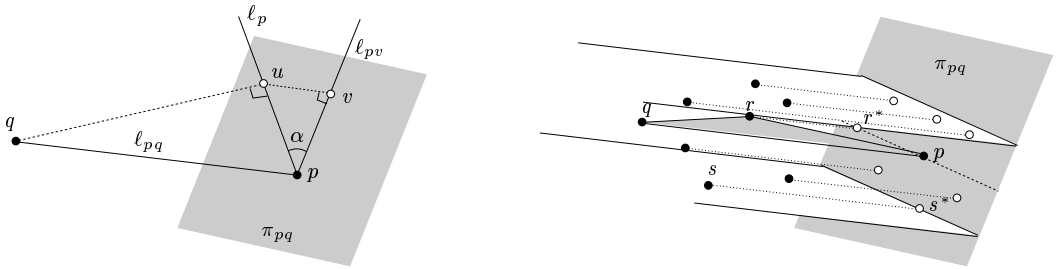
The angle $\angle c_b c b$ is minimized when c_b lies on the plane spanned by a, b, and c, in which case $\angle c_b c b = \pi/2 - \alpha$. Therefore $\mathrm{width}(\tau) \leq w + \sqrt{2}\,\mathrm{diam}(P)\sin\alpha$. □

Lemma 4.2 *Let S be a set of n points in $\mathbb{R}^3$, and let σ be a slab of width w. For any point $p \in S \cap \sigma$, there exist $q, r, s \in S$ so that $\sigma(p, q, r, s)$ covers $S \cap \sigma$ and $d(s, \pi_{pqr}) < 12w$.*

Proof: Let $\Delta = \mathrm{diam}(S \cap \sigma)$, let ϕ be the direction of σ, and let $q \in S \cap \sigma$ be the point farthest from p. Then $\|pq\| \geq \Delta/2$. Let π_{pq} be the plane passing through p and normal to ℓ_{pq}, and let ℓ_p be the line passing through p in direction ϕ. See Figure 7(i). Let u be the projection of q onto ℓ_p, and let v be the projection of u onto π_{pq}. For any point $r \in \mathbb{R}^3$, let r^* be the projection of r onto π_{pq}. Set $S^*_\sigma = \{r^* \mid r \in S \cap \sigma\}$. Since p, q, u, and v are coplanar, we can deduce that

$$\sin(\angle \ell_p, \ell_{pv}) = \sin(\angle \ell_{qu}, \ell_{pq}) = \frac{\|pu\|}{\|pq\|} \leq \frac{2w}{\Delta}.$$

By Lemma 4.1, the width of $S \cap \sigma$ in the direction of ℓ_{pv} is at most $(1 + 2\sqrt{2})w$, i.e., S^*_σ lies in a strip of that width in π_{pq}. By Lemma 2.1, there exist two points $r^*, s^* \in S^*_\sigma$ so that the strip $\sigma(p, r^*, s^*)$ covers S^*_σ and has width at most $6(1 + 2\sqrt{2})w$; see Figure 7(ii). Since the boundary planes of $\sigma(p, q, r, s)$ are parallel to ℓ_{pq} and $\sigma(p, q, r, s) \cap \pi_{pq}$ is the strip $\sigma(p, r^*, s^*)$, we deduce that $S \cap \sigma \subseteq \sigma(p, q, r, s)$. By construction, $d(s, \pi_{pqr}) \leq 3(1 + 2\sqrt{2})w < 12w$. □



(i) (ii)

Figure 7. Finding a slab $\sigma(p, q, r, s)$ that approximates σ.

Set $\tilde{w} = \max\{d(s, \pi_{pqr}) \mid p, q, r, s \in S, d(s, \pi_{pqr}) \leq 12w^*\}$. The above lemma implies that S can be covered by a set $\Sigma \subseteq \Pi$ of k slabs, each of width at most $2\tilde{w}$. As in Section 3, we first describe a decision algorithm that, given a real number w, returns ' 'yes" if $w \geq \tilde{w}$. As a by-product, whenever it returns "yes," it also returns a slab cover of S of size $O(k \log k)$ and of width $2w$. We then use the parametric-search technique to compute a slab cover of size $O(k \log k)$ of width at most $2\tilde{w}$.

4.1 Decision algorithm

Fix a real value $w > 0$. Let $\sigma(p, q, r)$ denote the slab with median plane π_{pqr} and width $2w$, where w is a fixed value.[2] We define the set system $\mathcal{X} = (E, \mathcal{R})$ as follows: $E = \{(p, q, r) \mid p, q, r \in S\}$

[2]The notation $\sigma(p, q, r)$ was also used to denote a planar strip in the previous sections. For the remainder of this section, it will only denote a 3-dimensional slab in the sense defined here.

and $\mathcal{R} = \{\mathcal{R}_p \mid p \in S\}$, where $\mathcal{R}_p = \{(q, s, r) \in E \mid p \in \sigma(q, s, r)\}$. Since a slab is the intersection of two halfspaces, it can be shown that the VC-dimension of $\mathcal{X}$ is a constant [38]. We modify Steps 1 and 2 of the algorithm described in Figure 2, but for the sake of completeness we describe the whole algorithm in Figure 8. As before, we maintain a small set A of size $O(k \log n)$. For any $(q, s) \in S \times A$, we define $wt((q, s)) = \sum_{r \in S} wt((q, s, r))$.

DECISION PROCEDURE $Fastslab(w)$
$wt(e) = 1, \forall e \in E$; $successful = 0$;
$A \subset S$: random set of size $k \log n$; $N = S \times A \times S$;
`repeat`
1. Select random set $R \subset S \times A$ of size $ck \log k$;
   /* $\Pr[(q, s) \in S \times A$ *is chosen*$] = wt((q, s))/wt(N)$. */
2. $\forall (q, s) \in R$, choose an object $(q, s, r),\ r \in S$;
   /* $\Pr[r \in S$ *is chosen*$] = wt((q, s, r))/wt((q, s))$. */
   $M : \{(q, s, r) \mid (q, s, r)$ chosen in Step 2$\}$;
   $\Sigma : \{\sigma(q, s, r) \mid (q, s, r) \in M\}$;

3. `if` $S \subset \bigcup_{\sigma \in \Sigma} \sigma$
   `return` "yes"; /* Σ is a strip cover.*/

4. Compute $p \in S \setminus \bigcup_{\sigma \in \Sigma} \sigma$;

5. `if` $wt(\mathcal{R}_p \cap N) \leq wt(E)/(2k)$
   5.1 $successful{+}{+}$;
   5.2 $wt(e) := 2wt(e), \quad \forall e \in \mathcal{R}_p \cap N$;
   5.3 $A := A \cup \{p\}$; $N = S \times A \times S$;
`until` $successful = 16k \log n$;
`return` "no";

Figure 8. The decision procedure for slab cover.

The proof of correctness follows a similar argument to the one in Lemma 3.1. Since the initial value of $wt(E)$ is n^3 (instead of n^2), we require more successful phases before terminating the algorithm.

Data structures. To implement a phase efficiently, we again observe that, for any $p \in S$, the set $\mathcal{R}_p \cap N$ can be represented by linear intervals. Indeed, let $\mathcal{D}(p, w)$ denote the sphere of radius w centered at p. For any fixed pair $(q, s) \in S \times A$ so that $q, s \notin \mathcal{D}(p, w)$, all points $r \in S$ for which $(q, s, r) \in \mathcal{R}_p \cap N$ lie in the double wedge $W_{qs}(p)$ formed by the two tangent planes $\pi^+_{qs}(p), \pi^-_{qs}(p)$ of $\mathcal{D}(p, w)$ passing through ℓ_{qs}. See Figure 4.1 (i). If $q \in \mathcal{D}(p, w)$ or $s \in \mathcal{D}(p, w)$, then $(q, s, r) \in \mathcal{R}_p \cap N$ for all $r \in S$.

Let us assume that ℓ_{qs} passes through origin, and let h_{qs} be the plane normal to ℓ_{qs} and passing through origin. Let $\rho \subseteq h_{qs}$ be the horizontal ray emanating from origin in $(+x)$-direction. For any point $r \in S$, let r^* be the orthogonal projection of a point r onto the plane h_{qs}. See Figure 4.1 (ii).

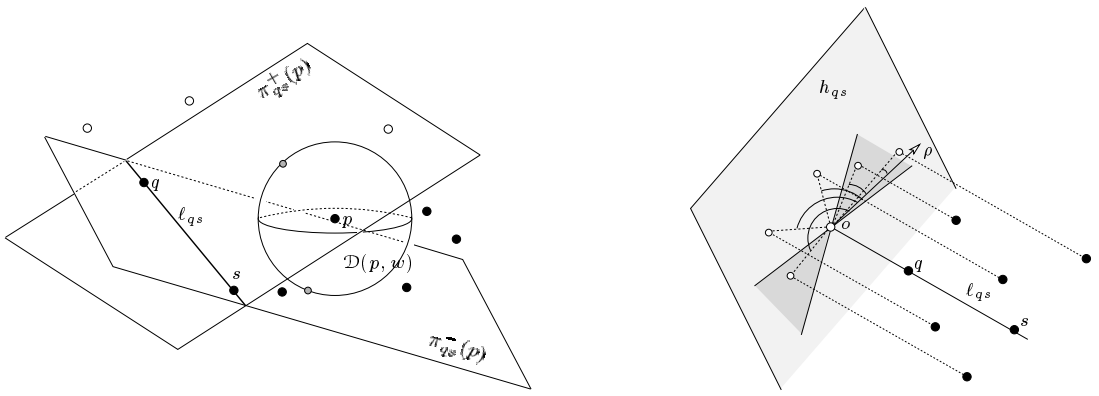


Figure 9. (i) Double-wedge $W_{qs}(p)$ bounded by two planes $\pi^+_{qs}(p), \pi^-_{qs}(p)$, each passing through ℓ_{qs} and tangent to $\mathcal{D}(p,w)$ (at points colored gray); black points lie inside the double wedge and white points lie outside the double wedge. (ii) Projection of S onto h_{qs}; the double wedge is the intersection of $W_{qs}(p)$ with h_{qs}.

Let S_{qs} be the set S sorted in nondecreasing order of the orientation of the rays $\overrightarrow{or^*}$ with respect to ρ. Then $S \cap W_{qs}(p)$ can be represented as three linear intervals (as in Section 3) $[r_1, r_2]$, $[r_3, r_4]$, and $[r_5, r_6]$, where $1 \le r_i \le r_{i+1} \le n$, in the sense that all points of S_{sq} whose ranks are in the range $[r_1, r_2] \cup [r_3, r_4] \cup [r_5, r_6]$ lie in $W_{qs}(p)$. Note that unlike Section 3 in which we used the points of A themselves to represent the intervals, we now use the ranks of points in S_{qs} to represent the intervals. We need a data structure that supports the following three operations:

(Q1) Given a pair $(q, s) \in S \times A$ and a point $r \in S$, return the rank of r, $\mathrm{rk}(r)$, in S_{qs}.

(Q2) Given a pair $(q, s) \in S \times A$ and a halfplane π bounded by ℓ_{qs}, determine the rank of the point that π meets first as we rotate it around ℓ_{qs} in clockwise (or counter-clockwise) direction.

(Q3) Given a pair $(q, s) \in S \times A$ and an integer $1 \le i \le n$, return the point in S_{sq} of rank i.

We fix a parameter $m \le n^3$ and preprocess S in $O(m \log m)$ time into a partition-tree based data structure Ψ of size $O(m)$ that answers a simplex-range query in $O((n/m^{1/3})\,\mathrm{polylog}(n))$ time [2]. Let γ_{qs} be the halfplane spanned by ℓ_{qs} and ρ, and for a point $r \in S$, let $\pi_{qs}(r)$ be the plane spanned by ℓ_{qs} and r. If $r \in \ell_{qs}$, let $\pi_{qs}(r) = \gamma_{qs}$. Then $\mathrm{rk}(r)$ in S_{qs} is determined by the number $n_{qs}(r)$ of points lying in the wedge bounded by γ_{qs} and $\pi_{qs}(r)$ and that contains r. More precisely, if the angle between ρ and $\overrightarrow{or^*}$ in counterclockwise direction is at most π, then $\mathrm{rk}(r) = n_{qs}(r)$. Otherwise, $\mathrm{rk}(r) = n - n_{qs}(r) + |\gamma_{qs} \cap S| + 1$. A (Q2) query can also be reduced to counting the number of points lying in a wedge bounded by π and γ_{qs}. As shown in [2], the simplex range-counting query can also be used to answer a (Q3) query within the same asymptotic time bound. Each r_i can be computed by answering (Q2) queries for the four halfplanes contained in $\pi^+_{qs}(p), \pi^-_{qs}(p)$ and bounded by ℓ_{qs}.

As in Section 3, we would like to maintain a segment tree T_{qs} for each pair $(q, s) \in S \times A$ to store the intervals representing $W_{qs}(p)$. Since T_{qs} stores intervals $[r_i, r_{i+1}] \subseteq [1, n]$, each T_{qs} would have n leaves — labeled 1 through n from left to right — and thus the total space and time needed to maintain these trees explicitly would be $\Omega(n^2 k)$, which is too expensive. We observe that only $O(\log n)$ nodes of each T_{qs} are visited in each phase, so the decision algorithm visits a total of $O(kn \log n)$ nodes in each phase. Keeping this in mind, we maintain each T_{qs} implicitly, by explicitly representing only those nodes of T_{qs} that have been accessed by the algorithm.

In more detail, for each internal node v of T_{qs}, we either explicitly represent both children of v or neither of them. If v is represented explicitly, then all of its ancestors are also represented explicitly. We thus explicitly represent a top subtree of T_{qs}. The ith leftmost leaf of T_{qs} is labeled i and is associated with the point of S_{qs} whose rank is i. For each explicitly represented node v, we store an integer k_v, the number of intervals stored at v, and an interval $[\alpha_v, \beta_v]$ where α_v (resp. β_v) is the label of the leftmost (resp. rightmost) leaf of the subtree rooted at v. If v is not explicitly represented, then $k_v = 0$. We also store an integer ω_v at v, defined as follows. If v is a leaf, then $\omega_v = 2^{k_v}$. If v is an internal node with children u and z then $\omega_v = 2^{k_v}(\omega_u + \omega_z)$. Hence, for any node v, the following holds:

$$\sum_r wt((q, s, r)) = \omega_v \cdot 2^{\sum_{y \in \Pi_v} k_y}, \tag{3}$$

where r ranges over all points associated with the leaves in the subtree of v, and Π_v is the path from the root to the parent of v.

Since the set S does not change over time, unlike Section 3, the set of nodes in T_{qs} remains fixed. But whenever we add a point to A, n new trees are created.

Implementing a phase. We now describe how we implement various steps of the decision algorithm.

STEP 1: If the root u of T_{qs} is not explicitly represented, then no interval is stored in T_{qs} yet, so $wt((q, s)) = n$. Otherwise, $\omega_u = wt((q, s))$. By examining all $O(nk \log n)$ trees, we can determine $wt((q, s))$ for all $(q, s) \in S \times A$. Compute $wt(N) = \sum_{(q,s) \in S \times A} wt((q, s))$ and choose $(q, s) \in S \times A$ with probability $wt((q, s))/wt(N)$. Choosing a pair (q, s) takes $O(nk \log n)$ time, so the total time spent in selecting R is $O(nk^2 \log(k) \log n)$.

STEP 2: For each point $(q, s) \in R$, we want to choose an object (q, s, r), for $r \in S$, with probability $wt((q, s, r))/wt((q, s))$. We have already computed $wt((q, s))$ in Step 1. We choose a random integer $m \in [1, wt((q, s))]$ with uniform probability and find the point $r \in S_{qs}$ such that

$$\sum_{\substack{r' \in S_{qs} \\ \mathrm{rk}(r') < \mathrm{rk}(r)}} wt((q, s, r')) < m \quad \text{and} \quad \sum_{\substack{r' \in S_{qs} \\ \mathrm{rk}(r') \leq \mathrm{rk}(r)}} wt((q, s, r')) \geq m. \tag{4}$$

If the root of T_{qs} is not explicitly represented, then we simply return the point in S_{qs} of rank m. Otherwise, we visit T_{qs} in a top-down manner as in Section 3. The algorithm maintains the invariant that when we are at a node v, we are given the value $\mu_v = 2^{\sum_{y \in \Pi_v} k_y}$ and an integer $m_v \in [1, \mu_v \omega_v]$,

and we want to return a point of S_{qs} that satisfies (4) for $m = m_v$ and for r' ranging only over the points in the subtree rooted at v. If v is a leaf, then we return the point whose rank is stored at v. If v is an internal node but its children are not explicitly represented, then we return the point of S_{qs} of rank $\alpha_v + \lceil m_v/(\mu_v 2^{k_v}) \rceil - 1$, using the data structure Ψ mentioned above. Finally, let u and z be the left and right children of v. Let $\mu_u = \mu_z = \mu_v 2^{k_v}$. If $m_v \leq \mu_u \omega_u$, then we visit u with $m_u = m_v$. Otherwise, we visit z with $m_z = m_v - \mu_u \omega_u$.

We spend $O((n/m^{1/3}) \operatorname{polylog}(n))$ time for each pair $(q, s) \in R$, therefore the total time spent in Step 2 is $O((nk/m^{1/3}) \operatorname{polylog}(n))$.

STEP 5: The test in Step 5 is executed in $O(nk \log n)$ time in the same way as in the procedure described in Section 3.1. In Step 5.2 we compute, for each $(q, s) \in S \times A$, the (at most) three intervals $[r_1, r_2]$, $[r_3, r_4]$, and $[r_5, r_6]$, where $1 \leq r_i \leq r_{i+1} \leq n$, that represent the points lying in the double wedge $W_{qs}(p)$. Each r_i is computed using the simplex range searching data structure Ψ as described above. We then insert each of these three intervals in T_{qs} using the standard segment-tree insertion procedure. The only difference is that when we try to access a node v that is not explicitly represented, we create it and store k_v, ω_v, and $[\alpha_v, \beta_v]$ at v. If v is the root of T_{qs}, then $\alpha_v = 1$ and $\beta_v = n$, otherwise if v is the left (resp. right) child of its parent u, then $\alpha_v = \alpha_u, \beta_v = \lfloor (\alpha_u + \beta_u)/2 \rfloor$ (resp. $\alpha_v = \lfloor (\alpha_u + \beta_u)/2 \rfloor + 1, \beta_v = \beta_u$). If v is a newly created node at which we store an interval, we set $k_v = 1$ and $\omega_v = 2(\beta_v - \alpha_v + 1)$. For all other new nodes, we set $k_v = 0$ and $\omega_v = \beta_v - \alpha_v + 1$. Note that we add p to A in Step 5.3 and create n new pairs $(q, p) \in S \times A$. Since no interval is stored in T_{qp}, we do not create it. The overall running time of Step 5 is $O((n^2 k/m^{1/3}) \operatorname{polylog}(n))$.

Hence, the total expected running time of the decision algorithm over all $O(k \log n)$ successful phases, excluding the time spent in preprocessing Ψ, is $O((n^2 k^2/m^{1/3}) \operatorname{polylog}(n))$.

4.2 Optimization algorithm

We extend the parametric-search approach from the previous section to run the decision algorithm generically at $\tilde{w} \leq 12 w^*$. Steps 1 and 2 do not depend on w, so we can execute them as in the decision algorithm. Steps 3 and 4 are executed generically as in Section 3.2, by doing a binary search. Finally, in Step 5, we need the value of w to compute, for each pair $(q, s) \in S \times A$, the boundaries r_i of the intervals that represent the points lying in $W_{qs}(p)$. Recall that each r_i is determined by counting the number of points lying in a wedge formed by γ_{qs} and $\pi^+_{qs}(p)$ or $\pi^-_{qs}(p)$; γ_{qs} does not depend on w, but $\pi^+_{qs}(p)$ and $\pi^-_{qs}(p)$ do. The data structure Ψ is a tree of height $O(\log n)$, and the number of points lying in a query wedge is computed by traversing $O((n/m^{1/3} \operatorname{polylog}(n))$ root-to-leaf paths of Ψ. Each node v of Ψ stores $O(1)$ points $\xi^v_1, \ldots \xi^v_l$ and the query procedure performs $O(1)$ point-plane tests of the following form for each plane h bounding the query wedge: Does h lie above, below, or on ξ^p_i, for $1 \leq i \leq l$? In our case, γ_{qs} does not depend on w, so we can perform these tests for γ_{qs} in $O(1)$ time, but we need to perform these tests *generically* for $\pi^+_{qs}(p), \pi^-_{qs}(p)$ as follows. Let $\pi_{qs}(\xi^v_i)$ be the plane spanned by ℓ_{qs} and ξ^v_i, and let δ be the distance between p and $\pi_{qs}(\xi^v_i)$. By determining whether $\delta < \tilde{w}$, which we can (almost) do using the decision algorithm, we can decide whether $\pi^+_{qs}(p), \pi^-_{qs}(p)$ lie above, below, or on ξ^v_i. It will be too expensive to invoke

the decision algorithm for each such test, so we proceed as follows.

Let $W_1, \ldots, W_\nu$ be the $O(nk \log n)$ query wedges generated in Step 5, for which we want to count the number of points of S lying inside. We answer these queries in $O(\log n)$ stages. In the ith stage we visit all nodes at level i in Ψ reached by the simplex-range-counting query procedure on all W_j's and perform all the necessary point-plane tests to determine which nodes of level $i+1$ should be visited by the query procedure. Let $\Delta = \{\delta_1, \ldots, \delta_\nu\}$ be the set of real values sorted in increasing order that we need to compare with $\tilde{w}$ to resolve the point-plane tests at the nodes of level i. We perform a binary search as in Section 3.2. The algorithm either returns a cover of S by $O(k \log k)$ congruent slabs of width at most $2\tilde{w}$, or it returns an index $j < \nu$ so that the decision algorithm returns "no" on δ_j and "yes" on δ_{j+1}. In the latter case, we can argue that $\delta_j < \tilde{w}$. Even though it is possible that $\tilde{w} > \delta_{j+1}$, we conclude that $\tilde{w} \leq \delta_l$ for all $l > j$. As argued in Section 3.2, this does not affect the correctness of our generic algorithm. Finally, when the generic algorithm stops, it returns a cover of S with $O(k \log k)$ congruent slabs of width at most $2\tilde{w} \leq 24w^*$. Extending the cutting technique from the previous section, we partition each slab into $O(1)$ thinner slabs to obtain a cover by $O(k \log k)$ slabs of width at most w^*.

Since the decision algorithm has $O(k \log n)$ successful phases and in each phase of the generic algorithm we spend $O((n^2k^2/m^{1/3}) \operatorname{polylog}(n))$ time, the overall running time of the algorithm, including the time spent in preprocessing Ψ is $O(m \log m + (n^2k^3/m^{1/3}) \operatorname{polylog}(n))$. By choosing $m = n^{3/2}k^{9/4}$, we obtain the following.

Theorem 4.3 *Given a set S of n points in $\mathbb{R}^3$ and a positive integer k we can compute in expected time $O(n^{3/2}k^{9/4} \operatorname{polylog}(n))$ a cover of S by $O(k \log k)$ slabs of width at most w^*, where w^* is the minimum width of a cover of S by k slabs.*

5 Extensions to Higher Dimensions

We extend our algorithm to cover a set S of n points in $\mathbb{R}^d$ by cylinders and by strips with fixed orientations.

5.1 Covering with cylinders

Given a line ℓ and a real number $w \geq 0$, the d-cylinder of diameter w with axis ℓ is the set of points in $\mathbb{R}^d$ that are wihin distance w from ℓ. For $q = 1$ the projective clustering problem is to cover $S \subseteq \mathbb{R}^d$ by k congruent d-cylinders of minimum diameter. Given a cylinder σ, we define its *length* L with respect to S to be the minimum distance between two parallel hyperplanes orthogonal to the axis of σ so that $S \cap \sigma$ lies between them. Then the maximum area of the planar projection of σ is $L \cdot w$. Hence, given any three points $p, q, r \in (S \cap \sigma)$, the area of the triangle Δpqr is at most $L \cdot w$. Using this observation, we can extend Lemma 2.1 as follows.

Lemma 5.1 *Let S be a set of points in $\mathbb{R}^d$, let σ be a d-cylinder of diameter w, and let p be any point in $S \cap \sigma$. Then there exist points $q, r \in S$ so that $d(r, \ell_{pq}) \leq 4w$ and the d-cylinder of axis ℓ_{pq} and radius $d(r, \ell_{pq})$ covers $S \cap \sigma$.*

We extend the two-dimensional decision procedure as follows. The set system is defined as before (see Section 3.1), except that we now denote by $\sigma(p, q; w)$ the d-cylinder of axis ℓ_{pq} and radius w. It can be shown that the VC-dimension of this set system is $O(d)$ [38]. In Step 1 of FastStrips, we choose a random sample R of size $c_1 dk \log k$, where c_1 is a constant. The expected number of phases remains $O(k \log n)$. Steps 1–5 of the decision procedure are executed by a naïve algorithm in $O(dnk \log n)$ time, i.e., we maintain the weights of all elements in N explicitly and do not maintain any data structure. The parametric search extends naturally from the two-dimensional case. We thus obtain the following result.

Theorem 5.2 *Given a set S of n points in $\mathbb{R}^d$ and a positive integer k, we can compute in expected time $O(dnk^3 \log^4 n)$ a cover of S by $O(dk \log k)$ d-cylinders of diameter at most $8w^*$, where w^* is the minimum value so that S can be covered by k d-cylinders of diameter w^*.*

Remark 5.3 Note that we cannot extend the "cutting technique" outlined in Section 2 to transform a cover of S by $O(dk \log k)$ cylinders of radius $\tilde{w} \leq 8w^*$ into a cover by $O(dk \log k)$ d-cylinders of diameter at most w^*. By a packing argument, we need C^d (for some constant C) d-cylinders of diameter w^* to ensure that we cover all points covered by a d-cylinder of diameter $8w^*$.

5.2 Covering with slabs of fixed orientations

In many practical applications, the orientations of flats on which S is projected belong to a fixed set of vectors. For example, one may want to cover the points by k slabs (i.e., region lying between two parallel hyperplanes) of minimum width, so that the bounding hyperplanes of each slab are orthogonal to a coordinate axis. In general, let $\mathcal{V}$ be a set of m vectors in $\mathbb{R}^d$. We want to compute a cover of S by k slabs so that each slab is orthogonal to one of the vectors in $\mathcal{V}$. We use the following straightforward result.

Lemma 5.4 *Let σ be a slab of width w and let v be the vector normal to the hyperplanes bounding σ. Then for any point $p \in S \cap \sigma$, the slab of width $2w$ whose median hyperplane passes through p and is normal to v covers $S \cap \sigma$.*

For a fixed value w, let $\sigma(v, p)$ denote the slab of width $2w$ whose median plane passes through p and is orthogonal to v. We define the set system $\mathcal{X} = (E, \mathcal{R})$ as follows: $E = \{(v, p) \mid v \in \mathcal{V}, p \in S\}$ and $\mathcal{R} = \{\mathcal{R}_p \mid p \in S\}$, where $\mathcal{R}_p = \{(v, q) \in E \mid p \in \sigma(v, q)\}$. Then the VC-dimension of $\mathcal{X}$ is proportional to $\min\{|\mathcal{V}|, d\}$. We modify the algorithm described in Figure 2 as follows. Let $N = \mathcal{V} \times A$, where $A \subseteq S$ is a set maintained as before. For any $v \in \mathcal{V}$, let $wt(v) = \sum_{p \in A} wt((v, p))$. In Step 1 we choose $R \subseteq \mathcal{V}$ of size $c_2 k \log k$ so that each v is chosen with probability $wt(v)/wt(N)$ (c_2 is a constant that depends on the VC-dimension). In Step 2 we choose, for each $v \in R$, an object (v, p), $p \in A$, with probability $wt((v, p))/wt(v)$. This guarantees that the expected number of iterations is $O(k \log n)$. To implement each iteration efficiently we maintain, for each $v \in \mathcal{V}$, the sorted order of A along v, denoted A_v. Then for any $p \in S$, the set $\mathcal{R}_p \cap N$ can be represented as the union of m intervals, one in each set A_v. For each $v \in \mathcal{V}$, we maintain a dynamic range tree T_v whose leaves are labeled by the points in A_v in increasing

order from left to right. The trees are maintained and queried in the same manner as the trees T_q discussed in Section 3. Thus, the decision procedure requires $O((nk^2 + mk)\log^2 n)$ time. The parametric search is a straightforward variant of the algorithm described in Section 3.2. We obtain the following result.

Theorem 5.5 *Let S be a set of n points in $\mathbb{R}^d$, let k be a positive integer, and let $\mathcal{V}$ be a set of m vectors in $\mathbb{R}^d$. We can compute in $O((nk^3 + mk^2)\log m\,\mathrm{polylog}(n))$ expected time a cover of S by $O(\min\{m, d\}k\log k)$ slabs of width w^* so that the normals to their bounding hyperplanes are in $\mathcal{V}$.*

6 Conclusions

We have proposed efficient approximation algorithms for a number of projective clustering problems. Recently we have developed an $(1+\varepsilon)$-approximation algorithm for covering a set of points in $\mathbb{R}^d$ by k cylinders [3]. Currently we do not know how to extend our techniques to higher dimensions without increasing the running time (as a function of n) for $q > 1$. An interesting special case, which arises frequently in database applications, is when we want to fit q-flats, each of which is normal to a subset of the coordinate axes. In this case, our method computes an approximate solution in expected time $(d^{\min\{q,d-q\}} + nk)k^2\log^{O(1)} n$. Hence, the method is efficient only for $q = 1, d-1$.

References

[1] P. K. Agarwal, Partitioning arrangements of lines: II. Applications, *Discrete Comput. Geom.*, 5 (1990), pp. 533–573.

[2] P. K. Agarwal and J. Erickson, Geometric range searching and its relatives, in: *Advances in Discrete and Computational Geometry* (B. Chazelle, J. E. Goodman, and R. Pollack, eds.), *Contemporary Mathematics*, Vol. 223, American Mathematical Society, Providence, RI, 1999, pp. 1–56.

[3] P. K. Agarwal, C. M. Procopiuc and K. R. Varadarajan, Approximation Algorithms for k-Line Center, *Proc. 10th Annu. European Sympos. Algorithms*, 2002, pp. 54–63.

[4] P. K. Agarwal and M. Sharir, Efficient randomized algorithms for some geometric optimization problems, *Discrete Comput. Geom.*, 16 (1996), pp. 317–337.

[5] P. K. Agarwal and M. Sharir, Efficient algorithms for geometric optimization, *ACM Comput. Surv.*, 30 (1998), pp. 412–458.

[6] P. K. Agarwal, M. Sharir, and E. Welzl, The discrete 2-center problem, *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, 1997, pp. 147–155.

[7] C. C. Aggarwal, C. M. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park, Fast algorithms for projected clustering, *Proc. of ACM SIGMOD Intl. Conf. Management of Data*, 1999, pp. 61–72.

[8] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, Automatic subspace clustering of high dimensional data for data mining applications, *Proc. ACM SIGMOD Conf. on Management of Data*, 1998, pp. 94–105.

[9] R. Agrawal, A. Ghosh, T. Imielinski, B. Iyer, and A. Swami, An interval classifier for database mining applications, *Proc. 18th Intl. Conf. Very Large Databases*, 1992, pp. 560–573.

[10] S. Belongie, C.Carson, H. Greenspan, and J. Malik, Color- and texture-based image segmentation using EM and its application to content-based image retrieval, *Proc. ICCV*, 1998, pp. 675–682.

[11] S. Berchtold, C. Böhm, and H. P. Kriegel, The pyramid-technique: Towards breaking the curse of dimensionality, *Proc. of ACM SIGMOD Intl. Conf. Management of Data*, 1998, pp. 142–153.

[12] H. Brönnimann and M. T. Goodrich, Almost optimal set covers in finite VC-dimension, *Discrete Comput. Geom.*, 14 (1995), 263–279.

[13] M. Charikar, S. Guha, E. Tardos, and D. B. Shmoys, A constant-factor approximation algorithm for the k-median problem, *Proc. 31st Annu. ACM Sympos. Theory Comput.*, 1999, pp. 1–10.

[14] B. Chazelle, Cutting hyperplanes for divide-and-conquer, *Discrete Comput. Geom.*, 9 (1993), pp. 145–158.

[15] V. Chvátal, A greedy heuristic for the set-covering problem, *Math. Oper. Res.*, 4 (1979), 233–235.

[16] K. L. Clarkson, Algorithms for polytope covering and approximation, *Proc. 3rd Workshop Algorithms Data Struct.*, *Lecture Notes Comput. Sci.*, Vol. 709, Springer-Verlag, 1993, pp. 246–252.

[17] C. A. Duncan, M. T. Goodrich, and E. A. Ramos, Efficient approximation and optimization algorithms for computational metrology, *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, 1997, pp. 121–130.

[18] H. Edelsbrunner, L. J. Guibas, and J. Stolfi, Optimal point location in a monotone subdivision, *SIAM J. Comput.*, 15 (1986), 317–340.

[19] C. Faloutsos and K.-I. Lin, FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia databases, *Proc. ACM SIGMOD Conf. on Management of Data*, 1995, pp. 163–173.

[20] T. Feder and D. H. Greene, Optimal algorithms for approximate clustering, *Proc. 20th Annu. ACM Sympos. Theory Comput.*, 1988, pp. 434–444.

[21] S. Guha, R. Rastogi, and K. Shim, CURE: An efficient clustering algorithm for large databases, *Proc. ACM SIGMOD Intl. Conf. Management of Data*, 1998, pp. 73–84.

[22] R. Hassin and N. Megiddo, Approximation algorithms for hitting objects by straight lines, *Discrete Appl. Math.*, 30 (1991), 29–42.

[23] D. Haussler and E. Welzl, Epsilon-nets and simplex range queries, *Discrete Comput. Geom.*, 2 (1987), 127–151.

[24] M. E. Houle and G. T. Toussaint, Computing the width of a set, *IEEE Trans. Pattern Anal. Mach. Intell.*, PAMI-10 (1988), 761–765.

[25] P. Indyk, R. Motwani, P. Raghavan, and S. Vempala, Locality-preserving hashing in multidimensional spaces, *Proc. 29th Annu. ACM Sympos. Theory Comput.*, 1997, pp. 618–625.

[26] J. W. Jaromczyk and M. Kowaluk, The two-line center problem from a polar view: A new algorithm and data structure, *Proc. 4th Workshop Algorithms Data Struct.*, *Lecture Notes Comput. Sci.*, Vol. 955, Springer-Verlag, 1995, pp. 13–25.

[27] N. Katayama and S. Satoh, The sr-tree: An index structure for high-dimensional nearest neighbor queries, *Proc. of ACM SIGMOD Intl. Conf. Management of Data*, 1997, pp. 369–380.

[28] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley & Sons, 1990.

[29] D. Keim, S. Berchtold, C. Böhm, and H. P. Kriegel, A cost model for nearest neighbor search in high-dimensional data space, *Proc. 16th Sympos. Principles of Database Systems*, 1997, pp. 78–86.

[30] J. Kleinberg, Two algorithms for nearest-neighbor search in high dimension, *Proc. 29th Annu. ACM Sympos. Theory Comput.*, 1997, pp. 599–608.

[31] R. Kohavi and D. Sommerfield, Feature subset selection using the wrapper method: Overfitting and dynamic search space topology, *Proc. 1st Intl. Conf. on Knowledge Discovery and Data Mining*, 1995.

[32] D. T. Lee and Y. F. Wu, Geometric complexity of some location problems, *Algorithmica*, 1 (1986), 193–211.

[33] J.-H. Lin and J. S. Vitter, Approximation algorithms for geometric median problems, *Inform. Process. Lett.*, 44 (1992), 245–249.

[34] G. J. McLachlan and K. E. Basford, *Mixture Models*, Marcel Dekker, Inc., 1988.

[35] N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, *J. ACM*, 30 (1983), 852–865.

[36] N. Megiddo and A. Tamir, On the complexity of locating linear facilities in the plane, *Oper. Res. Lett.*, 1 (1982), 194–197.

[37] R. T. Ng and J. Han, Efficient and effective clustering methods for spatial data min ing, *Proc. 20th Intl. Conf. Very Large Databases*, 1994, pp. 144–155.

[38] J. Pach and P. K. Agarwal, *Combinatorial Geometry*, John Wiley & Sons, New York, 1995.

[39] J. Shafer, R. Agrawal, and M. Mehta, Sprint: A scalable parallel classifier for data mining., *Proc. 22nd Intl. Conf. Very Large Databases*, Morgan Kauffman, 1996.

[40] D. A. White and R. Jain, Similarity indexing with the ss-tree, *Proc. 12th Intl. Conf. Data Engineering*, 1996, pp. 516–523.