Remarks on k-Level Algorithms in the Plane

Timothy M. Chan*

July 7, 1999

Abstract

In light of recent developments, this paper re-examines the fundamental geometric problem of how to construct the k-level in an arrangement of n lines in the plane.

- The author's recent dynamic data structure for planar convex hulls improves a decade-old sweep-line algorithm by Edelsbrunner and Welzl, which now runs in $O(n \log m + m \log^{1+\varepsilon} n)$ deterministic time and $O(n)$ space, where m is the output size and ε is any positive constant. We discuss simplification of the data structure in this particular application, by viewing the problem kinetically.
- Har-Peled recently announced a randomized algorithm with an expected running time of $O((n+m)\alpha(n)\log n)$. We observe that a version of an earlier randomized incremental algorithm by Agarwal, de Berg, Matoušek, and Schwarzkopf yields almost the same result.
- The current combinatorial bound by Dey shows that $m = O(nk^{1/3})$ in the worst case. We give an algorithm that guarantees $O(n \log n + nk^{1/3})$ expected time.

1 Introduction

The notion of k-levels [2, 20, 26, 34] has proved to be an important one in computational geometry, exploited directly or indirectly in various algorithms for computing higher-order Voronoi diagrams [1, 11, 14] (used, for instance, in finding clusters of points [3, 23]), designing data structures for halfspace range searching [16, 17], and solving hyperplane partitioning problems (such as ham-sandwich cuts [32] and weak line-separators [25]). The concept is also fundamental in the combinatorics of arrangements (e.g., in bounding the complexity of lower envelopes [27]). In this paper, we study one simplest version of the algorithmic problem: k-levels of lines in the plane.

There are several ways to define this two-dimensional problem. A "functional" (or "parametric") approach seems the most intuitive: given n univariate linear functions $f_1, \ldots, f_n : \mathbb{R} \to \mathbb{R}$ and a number $k \in \{1, \ldots, n\}$, construct $G : \mathbb{R} \to \mathbb{R}$, where

$$G(x) \;=\; \text{the } k\text{-th smallest of the numbers } f_1(x), \ldots, f_n(x).$$

This function G is continuous, piecewise-linear, and is called the *k-level of* $f_1, \ldots, f_n$. It generalizes the notion of the *lower envelope* ($k = 1$) and the *upper envelope* ($k = n$).

*Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada, e-mail: tmchan@math.uwaterloo.ca

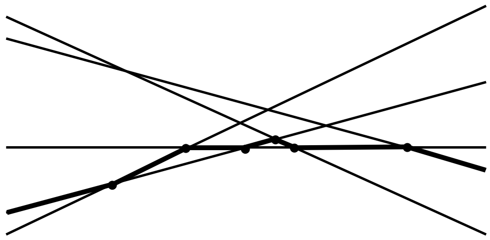

Figure 1: The 2-level of 5 lines in the plane

Viewed geometrically in the plane, the graph of each given linear function is a line, and the graph of the k-level resides in the arrangement of these n lines. In fact, the k-level forms an x-monotone polygonal chain that turns at every vertex of the arrangement it encounters (assuming no degeneracy); see Figure 1. "Constructing" the k-level means reporting the sequence of vertices/edges of this polygonal chain.

Combinatorially, obtaining tight worst-case bounds on the number m of vertices of the k-level turns out to be difficult and is basically equivalent (by duality) to the famous *k-set problem* [2, 4, 20], first studied by Simmons, Erdős, Lovász, and others in the early 1970s: given a planar n-point set, how many ways can it be partitioned by a line so that one side of the line has precisely k points? The long-held upper bound was $m = O(n\sqrt{k})$ (ignoring a minute improvement [36]). In 1997, Dey [19] obtained a breakthrough result, showing that $m = O(nk^{1/3})$. Configurations of lines are known [24] for which the k-level can have superlinear complexity $m = \Omega(n \log k)$. Resolving the gap between Dey's upper bound and the known lower bound remains a perplexing question.

Computationally, the problem of constructing k-levels was first introduced by Edelsbrunner and Welzl in a 1986 paper [22], describing a simple *sweep-line* algorithm in the plane that utilizes a (not-so-simple) dynamic convex hull data structure by Overmars and van Leeuwen [35] from 1981. Due to the unknown status on the combinatorial bound for m and the fact that m tends to be smaller than the worst-case bound in practical instances [5, 21], it makes sense to measure time bounds as a function of both the input size n and the output size m. Then the running time of Edelsbrunner and Welzl's algorithm is $O(n \log n + m \log^2 n)$. Space is $O(n)$.

Until very recently, no substantial improvement to this output-sensitive running time was reported, except for special cases. Cole, Sharir, and Yap [18] described a modification when k is much smaller than n; their algorithm takes $O(n \log n + (n + m) \log^2 k)$ time (see also [25]), using a convex layers subroutine by Chazelle [13]. The author [10] noted a modification for another case where the output size m is much smaller than n; the time bound becomes $O(n \log m + m \log^2 n)$. We were still no closer to the known lower bound $\Omega(n \log m + m)$ in terms of the number of logarithmic factors.

Important progress have been made very recently, however, in terms of both deterministic and randomized algorithms. The first comes from a breakthrough in the dynamic planar convex hull problem: the author [12] recently obtained a data structure that can be updated in $O(\log^{1+\varepsilon} n)$ amortized time and can support certain basic convex hull queries in $O(\log n)$ time. Here, $\varepsilon > 0$ is an arbitrarily small constant. The previous data structure by Overmars and van Leeuwen has $O(\log^2 n)$ update time [35]. The new data structure is a fairly nontrivial combination of several techniques: dynamization of a known optimal semi-dynamic hull structure, an interesting application of interval trees, along with bootstrapping. As a consequence, the running time of Edelsbrunner and Welzl's sweep-line algorithm reduces to $O(n \log n + m \log^{1+\varepsilon} n)$, and space remains $O(n)$.

In Section 2, we simplify the resulting algorithm somewhat, by eliminating the interval-tree component for this particular application of the dynamic hull structure. The simplification comes from the observation that queries generated from the sweep-line paradigm are x-monotone in a certain sense. Treating the x-coordinate as time, this enables a *kinetic* view of the data structuring problem. The viewpoint has other applications and should be of independent interest.

The second progress is on a different algorithmic paradigm: *randomized incremental construction*. Har-Peled [28] has just announced that a certain variant of this popular paradigm yields an $O((n+m)\alpha(n)\log n)$ expected time bound for planar k-level construction. Here, $\alpha(\cdot)$ is the inverse Ackermann function (which grows extremely slowly). This is marginally faster than the above deterministic bound $O(n \log n + m \log^{1+\varepsilon} n)$ for $m = \Omega(n)$, but it is unclear at the moment whether space can be made $O(n)$. Har-Peled actually considered the general problem of performing "on-line walks" in a planar arrangement and examined an unusual re-scheduling of a standard randomized incremental algorithm. Details seem intricate.

In Section 3, we observe that a more traditional randomized incremental algorithm produces essentially the same expected time bound, except for a mere $\alpha(n)$ factor (possibly an artifact of our analysis): $O((n+m)\alpha(n)^2 \log n)$. Actually, this algorithm is an already known one by Agarwal, de Berg, Matoušek, and Schwarzkopf [1] from 1994. They looked at the problem in both the planar and the three-dimensional case, and examined not only the k-level but also a related structure known as the "$(\leq k)$-level." Although they did not give an output-sensitive analysis for the planar k-level case, they provided basically all the ingredients that lead to the result. Note that the space usage of this algorithm is at least $\Omega(n+m)$.

In Section 4, we also point out an algorithm with a running time insensitive to the output size m. The above results imply that for a constant c close to 1, the k-level of any collection of n lines in the plane can be constructed within $O(nk^{1/3} \log^c n)$ time, according to Dey's combinatorial bound. This complexity is not the best in the worst case. Indeed, Agarwal, de Berg, Matoušek, and Schwarzkopf's randomized incremental algorithm [1] guaranteed $O(n \log^2 n + nk^{1/3} \log^{2/3} n)$ expected time. The author [11] recently brought this down to $O(n \log n + nk^{1/3} \log^{2/3} k)$ by random sampling and data structuring. Recently, Ramos [38] described further ideas that appear to yield $O(n \log n + nk^{1/3} 2^{O(\log^* k)})$ expected time with a fairly involved algorithm, though the planar case was not mentioned explicitly in his paper. Here, it is shown that $O(n \log n + nk^{1/3})$ expected time is possible. Of course, the interest of the result depends on whether Dey's bound is optimal, but our idea is simple: we observe that a random level near the k-level tends to have complexity smaller than $O(nk^{1/3})$; knowing nearby levels allows us to speed up the computation of the k-level.

We conclude with a sample list of applications in Section 5.

2 Sweep-Line Algorithms: A Kinetic Approach

The *sweep* paradigm is one of the most useful in geometry [9, 37]. We obtain sweep-line algorithms by the viewing our two-dimensional construction problem as a problem of designing one-dimensional kinetic data structures called *kinetic priority queues*, formally defined in Section 2.1.

The subject of kinetic data structures was first introduced by Basch, Guibas, and Hershberger [6] as a problem in itself, and has received a great deal of attention in recent years. Although our motivation is somewhat different (using these data structures to develop efficient static algorithms),[1] some of their ideas are applicable here. In fact, in a paper by Basch, Guibas, and Ramkumar [7], it was pointed out that their *kinetic tournament tree* implied a k-level algorithm running in $O((n + m)\alpha(n)\log^2 n)$ time. As we note, an r-ary version of this tournament tree (for an appropriate choice of r) actually yields a time bound of $O((n+m)\alpha(n)\log^2 n/\log\log n)$, which is already an improvement, albeit a very small one, over Edelsbrunner and Welzl's result (for m larger than n). All this is pointed out in Section 2.2 in fully self-contained terminology, with pseudocodes. The reason we choose to re-describe this suboptimal solution is threefold. First, the algorithm appears the simplest to implement among all published methods, and hence, we believe, should be of practical importance. (For example, unlike algorithms based on Overmars and van Leeuwen's hull structure, we do not need to perform repeated binary searches for bridge-finding [35].) Second, the approach generalizes to yield currently the best deterministic result for k-levels of curves. Finally, it prepares for the description of our next solution.

The best solution we get for kinetic priority queues is based on the author's dynamic hull structure [12], which involves dynamization of known semi-dynamic methods by Chazelle [13] and Hershberger and Suri [30] (which are modifications of Overmars and van Leeuwen's structure). This gives our fastest deterministic algorithm for planar k-level construction. Except for the semi-dynamic subroutine, we provide a self-contained description in Section 2.3, with pseudocodes for the most part.

2.1 A Kinetic Priority Queue Problem

Our version of a *kinetic priority queue* (KPQ) is basically a data structure that maintains, over time, the smallest element of a set of elements in $\mathbb{R}$ that are continuously moving along known "flight plans." Three operations are supported: "insert" a new element, "delete" an element, and "advance." The third operation must be called whenever an "event" is encountered. An example of an event is when the smallest element actually changes (such an event is called an "external" event), although for bookkeeping purposes, the data structure may also create other kinds of ("internal") events.

A more precise definition of the problem is helpful, especially in proving correctness. Although our particular formulation may not appear to be the most natural (nor exactly follow object-oriented principles), it is especially convenient for designing recursive solutions.

Problem definition. We assume that there is a single global time variable t maintained by the user, unmodified by our data structure. We also assume that t never decreases. Our data structure S maintains an element $S.min$ and a time value $S.next$. A formal specification of the three operations is as follows:

- $S.insert(s)$. Action: insert a new element s into S. Pre-condition: $t < S.next$.
- $S.delete(s)$. Action: delete element s from S. Pre-condition: $t < S.next$.
- $S.advance()$. No action. Pre-condition: $t = S.next < \infty$.

[1] For instance, since we are interested only in the overall running time, their notion of "responsiveness" is not a concern (except in an amortized sense), but a stronger measure of "efficiency" is required to bound the number of so-called "internal events" in terms the number of "external events."

After each operation, the data structure will modify $S.min$ and $S.next$ to ensure the following:

- Post-condition 1: $S.next > t$.
- Post-condition 2: $S.min$ is the smallest element in S at any time in the interval $[t, S.next)$.

What is described above is a min-KPQ. A max-KPQ is similarly defined for maintaining the largest element $S.max$. For simplicity, we assume no degenerate cases, i.e., that at most one pair of elements can meet at a given time, and the order of two elements must change after they meet.

Cost function. To measure the efficiency of a KPQ, it is not enough to bound the cost of each individual operation, since the number of advances (events) can be increased to lower the cost. Instead, we simply measure the worst-case total running time, $T(n, m)$, over a sequence of operations on an initially empty data structure S, in terms of two parameters: an upper bound n on the number of elements in S at any time, and an upper bound m on the number of insertions to S. Clearly, m is also an upper bound on the number of deletions. The number of times $S.min$ changes (external events) can be bounded in terms of these two parameters via geometric arguments.

Observation 2.1 *For elements moving linearly, the number of times $S.min$ changes is $O(m\alpha(n))$.*

Proof: The trajectory of an element s maps to a line in the plane, with the x-axis being time. Restricted to the time interval at which s is in the data structure S, we obtain a line segment. The vertices of the lower envelope of these m line segments correspond to changes in $S.min$, so the claim follows from known combinatorial bounds on lower envelopes [29, 40]. Note: the fact that every vertical line intersects at most n segments is needed to reduce the usual $O(m\alpha(m))$ bound to $O(m\alpha(n))$ (because we can divide the plane into $\lceil m/n \rceil$ vertical slabs each cutting $O(n)$ segments). $\square$

We note that there is a trivial KPQ with cost $T(n, m) = O(mn\alpha(n))$: we always set $S.next$ to be the next time after t when $S.min$ meets some other element in S. Each operation takes $O(n)$ time by brute force, and the number of advances in this case is $O(m\alpha(n))$ by Observation 2.1, since $S.min$ changes after each advance. We will give better KPQs in the next subsections.

Relation to k-levels. How can we use KPQs to construct k-levels of n lines in the plane? From the functional definition of k-levels, the problem is equivalent to tracking the k-th smallest of a given set of n linearly moving elements. We maintain two sets: a max-KPQ S_1 containing all elements of rank less than k, and a min-KPQ S_2 containing all elements of rank at least k.

Leaving the initialization part to the reader, we can write the main loop of our sweep-line algorithm as follows:

1. repeat:
2. $\quad t_0 \leftarrow$ the next time after t when $S_1.max$ and $S_2.min$ meet
3. $\quad t \leftarrow \min\{t_0,\ S_1.next,\ S_2.next\}$
4. $\quad$ if $t = \infty$ then exit
5. $\quad$ else if $t = t_0$ then
6. $\qquad s_1 \leftarrow S_1.max,\ s_2 \leftarrow S_2.min$
7. $\qquad S_1.delete(s_1),\ S_1.insert(s_2)$
8. $\qquad S_2.delete(s_2),\ S_2.insert(s_1)$
9. $\quad$ else if $t = S_1.next$ then $S_1.advance()$ else $S_2.advance()$

The correctness is easily verified from the pre- and post-conditions. We only have to print out $S_2.min$ whenever it changes. The number of insertions is bounded by n plus the output size. (In fact, because lines 7–8 are done only when $S_1.max$ meets $S_2.min$, the number of insertions is at most n plus the number of vertices that lie simultaneously on the $(k-1)$-level and the k-level.)

Lemma 2.2 *Given n lines in $\mathbb{R}^2$ in general position, we can compute all m vertices of the k-level in $O(T(n, n+m))$ time, where $T(\cdot,\cdot)$ denotes the cost function of the KPQ problem.*

2.2 First Solution: Divide-and-Conquer

It would seem difficult to apply the divide-and-conquer paradigm directly to construct the k-level in the plane (except for the special cases of lower/upper envelopes), but the kinetic viewpoint reduces the geometric problem to a one-dimensional problem, making this familiar paradigm possible.

Decomposability. Computing the smallest of n real numbers is certainly a "decomposable" problem: given the minimum of each subset, the minimum of the union is the minimum of these minima. The kinetic version is also decomposable in a similar way.

First some notation: given a (min-)KPQ structure S, we let $S.set$ denote the set of elements currently in S. Suppose that S is *decomposed* into a collection $S.\Pi$ of substructures, i.e., $S.set$ is partitioned into a collection of subsets $\{P.set \mid P \in S.\Pi\}$. Suppose further that we have an additional KPQ structure $S.Q$ to maintain the minimum of the minima of these substructures, i.e., $S.Q.set = \{P.min \mid P \in S.\Pi\}$. The following code fragment indicates how the advance operation can be implemented for S:

$S.maintain()$:

M1. $S.min \leftarrow S.Q.min$
M2. $S.next \leftarrow \min\{\ \min_{P \in S.\Pi} P.next,\ S.Q.next\ \}$

$S.advance()$:

A1. find $P \in S.\Pi$ with smallest $P.next$
A2. if $t = P.next$ then
A3. $\quad p \leftarrow P.min,\ P.advance(),\ p' \leftarrow P.min$
A4. $\quad$ if $p \neq p'$ then
A5. $\qquad S.Q.delete(p),\ S.Q.insert(p')$
A6. else if $t = S.Q.next$ then $S.Q.advance()$
A7. S.$maintain()$

Simple recursion. Fix a small number r. Our first method is straightforward divide-and-conquer: we simply decompose S into a collection $S.\Pi$ of r substructures, each containing at most n/r elements, and recursively generate a KPQ structure for each $P \in S.\Pi$. In addition, we need to implement the KPQ $S.Q$, but since this structure contains only r elements, the trivial method can be used.

Updates can be accomplished easily:

$S.insert(s)$:	$S.delete(s)$:
I1. $S.set \leftarrow S.set \cup \{s\}$	D1. $S.set \leftarrow S.set \setminus \{s\}$
I2. find $P \in S.\Pi$ with smallest $\lvert P.set\rvert$	D2. find $P \in S.\Pi$ with $s \in P.set$
I3. $S.Q.delete(P.min)$	D3. $S.Q.delete(P.min)$
I4. $P.insert(s)$	D4. $P.delete(s)$
I5. $S.Q.insert(P.min)$	D5. $S.Q.insert(P.min)$
I6. $S.maintain()$	D6. $S.maintain()$

Again, correctness of the method follows from checking the pre- and post-conditions.

Analyzing the recurrences. Let $m_1, \ldots, m_r$ be the number of insertions to the r substructures in $S.\Pi$ (note $\sum_i m_i = m$). We first bound the number of advances $N(n,m)$ on S. In each call to $S.advance()$, either a substructure $P \in S.\Pi$ is advanced or $S.Q$ is advanced. In the latter case, since $S.Q$ is implemented by the trivial method, $S.Q.min$ changes, implying that $S.min$ changes as well. This occurs at most $O(m\alpha(n))$ times by Observation 2.1. Therefore, we have the recurrence

$$N(n,m) \;=\; \sum_{i=1}^{r} N(n/r, m_i) \;+\; O(m\alpha(n)),$$

implying $N(n,m) = O(m\alpha(n)\log_r n)$.

To bound the overall running time, we first bound the total cost of all operations on $P \in S.\Pi$ by $\sum_i T(n/r, m_i)$. Each operation on the trivial KPQ $S.Q$ takes $O(r)$ time. We have argued that there are $O(m\alpha(n))$ advances for $S.Q$. We claim that there are $O(m\alpha(n))$ insertions for $S.Q$ as well. Why? The number of insertions is bounded by the number of times lines A5, I5, and D5 are executed. Since line A5 is done only when some $P.min$ changes, this number is $O(\sum_i m_i\alpha(n/r)) = O(m\alpha(n))$ by Observation 2.1. We conclude that the total cost of all operations on $S.Q$ is $O(mr\alpha(n))$.

Lines A1 and M2 can be performed in constant time if we use a heap of size r. The heap needs to be updated $O(N(n,m))$ times, so its total cost is $O(N(n,m)\log r) = O(m\alpha(n)\log n)$. Thus, the recurrence for the overall time is

$$T(n,m) \;=\; \sum_{i=1}^{r} T(n/r, m_i) \;+\; O(m\alpha(n)\log n + mr\alpha(n)),$$

which solves to

$$T(n,m) \;=\; O((m\alpha(n)\log n + mr\alpha(n))\log_r n).$$

Setting $r = 2$ gives $T(n,m) = O(m\alpha(n)\log^2 n)$. This is particularly suitable for implementation, as the underlying structure is a binary tree (a tournament tree) and separate heaps are avoided. For the best theoretical result, we can set $r = \lceil \log n \rceil$, yielding:

Theorem 2.3 *For elements moving linearly, a KPQ can be implemented with cost function* $T(n,m) = O(m\alpha(n)\log^2 n/\log\log n)$.

Remark. The above method easily extends to non-linearly moving elements and thus implies a planar k-level algorithm for continuous curves that runs in $O(\lambda_{a+2}(n+m)\log^2 n/\log\log n)$ time, where the maximum number of times two curves can meet is assumed to be a constant a. The function $\lambda_{a+2}(\cdot)$ here grows almost linearly [29, 40]. In contrast, Edelsbrunner and Welzl's algorithm [22] does not generalize to curves.

2.3 Second Solution: Exploiting Semi-Dynamic Data Structures

To speed up the method further, we need to borrow known data structures specific to the case of lines.

The deletion-only problem. We define a *semi-dynamic kinetic priority queue* (semi-KPQ) to be a KPQ S without the insert operation. We now need an operation "create" to initialize the structure. Specifically, given a set A, *create*(A) returns a new structure containing the elements of A. One parameter n—the size of A—is sufficient to measure the cost $T(n)$. It turns out that optimal semi-KPQs are known.

Lemma 2.4 *For elements moving linearly, a semi-KPQ can be implemented with cost function* $T(n) = O(n \log n)$*, where the total number of advances is* $O(n)$*.*

Proof: The trajectory of each element in the structure maps to a line in the plane. Take the lower envelope $\mathcal{L}$ of these lines. Set $S.next$ to be the x-coordinate of the next vertex of $\mathcal{L}$ to the right of $x = t$. Since $\mathcal{L}$ is dual to a convex hull, we can maintain $\mathcal{L}$ (and consequently, $S.next$) under any sequence of n deletions in $O(n \log n)$ time by the semi-dyamic hull structures of Chazelle [13] or Hershberger and Suri [30]. Over a sequence of deletions, the total structural change in $\mathcal{L}$ is linear (or if one thinks backwards, adding a line creates at most two new vertices). Since each advance can be charged to a distinct vertex of $\mathcal{L}$, there are $O(n)$ number of advances. □

Dynamizing. Bentley and Saxe [8] described how a semi-dynamic data structure for a decomposable problem implies a data structure supporting both insertions and deletions. This general dynamization technique uses a simple binary-counting trick and slows down operations by a logarithmic factor. We observe that the technique is applicable even in our kinetic setting. To avoid the slow-down, we actually apply a b-ary variant of the counting trick for a suitably chosen parameter b. One description is as follows.

Let $\ell = \lceil \log_b n \rceil$. The KPQ structure S is decomposed into a collection $S.\Pi$ of $O(b\ell)$ semi-KPQ structures. Each structure $P \in S.\Pi$ is assigned a *depth* value from $\{0, \ldots, \ell\}$, excluding one special structure at depth ∞. We will ensure that at most $b - 1$ structures reside at each depth.

The pseudocode for advance and delete is the same as before, exploiting the intermediate structure $S.Q$, which now contains $O(b\ell)$ elements. The pseudocode for insert proceeds as below, using a "union" subroutine that combines several structures of $S.\Pi$ into one (while destroying the old ones).

$S.union(\mathcal{C})$, where $\mathcal{C} \subseteq S.\Pi$:

U1. $P' \leftarrow create(\bigcup_{P \in \mathcal{C}} P.set)$
U2. for each $P \in \mathcal{C}$ do
U3. $\quad S.Q.delete(P.min)$
U4. $S.Q.insert(P'.min)$
U5. $S.\Pi \leftarrow S.\Pi \setminus \mathcal{C} \cup \{P'\}$
U6. return P'

$S.insert(s)$:

I1′. $S.set \leftarrow S.set \cup \{s\}$
I2′. $P' \leftarrow create(\{s\})$
I3′. $S.Q.insert(P'.min)$
I4′. $S.\Pi \leftarrow S.\Pi \cup \{P'\}$ and put P' at depth 0
I5′. $i \leftarrow 0$
I6′. while $|\{P \in S.\Pi \text{ at depth } i\}| = b$ do
I7′. $\quad$ put $S.union(\{P \in S.\Pi \text{ at depth } i\})$ at depth $i+1$
I8′. $\quad i \leftarrow i+1$
I9′. if $i = \ell$ then put $S.union(S.\Pi)$ at depth ∞
I10′. $S.maintain()$

As each structure $P \in S.\Pi$ is a semi-KPQ (with no insertion), it can be implemented by Lemma 2.4. As for $S.Q$, we assume for the moment that it is implemented by a KPQ with cost function $T_Q(\cdot,\cdot)$. As a result, we obtain a complete (non-recursive) method for S.

Accounting. To analyze this method, observe that a structure at depth $i+1$ is created from the union of b structures at depth i. Applying this observation inductively, we can deduce two facts for any index $i \in \{0, \ldots, \ell\}$:

(i) A structure at depth i holds at most b^i elements.

(ii) The number of depth-i structures created over time is at most m/b^i.

The number of depth-∞ structures created over time is the same as the number of depth-ℓ structures. Therefore, the total sizes of all sets created is at most

$$\left(\sum_{i=0}^{\ell}(m/b^i)b^i\right) \;+\; (m/b^\ell)n \;=\; O(m\ell).$$

By Lemma 2.4, the total number of advances in all $P \in S.\Pi$ is thus $O(m\ell)$, and the total cost of operations on all $P \in S.\Pi$ is $O(m\ell \log n)$.

What is the cost associated with the structure $S.Q$? We first bound the number of insertions to $S.Q$, i.e., the number of times lines A5, D5, I3′, and U4 are executed. Line A5 is executed only after an advance is performed on some $P \in S.\Pi$; thus, it happens $O(m\ell)$ times. Lines D5 and I3′ are executed $O(m)$ times. Line U4 is executed after some $P \in S.\Pi$ is created; as we have seen, this happens at most $O(m\ell)$ times. We conclude that the total cost of all operations on $S.Q$ is $T_Q(O(b\ell), O(m\ell))$.

An implicit heap is required to perform lines M2 and A1 and is updated after each operation on P. There are at most $O(m\ell)$ such operations, so the cost of the heap is $O(m\ell \log n)$. Each of the remaining instructions can be charged to an operation on either some $P \in S.\Pi$ or $S.Q$. Therefore, the running time of the entire method is $O(m\ell \log n \;+\; T_Q(O(b\ell), O(m\ell)))$, i.e.,

$$T(n,m) \;=\; O(m \log_b n \log n \;+\; T_Q(O(b \log_b n), O(m \log_b n))).$$

Bootstrapping. We can start with the bound $T_Q(n,m) = O(mn\alpha(n))$ by using the trivial method for $S.Q$ and set $b = 2$. This gives $T(n,m) = O(m\alpha(\log n)\log^2 n)$, which is almost same as in the recursive method from Section 2.2.

Suppose we instead have $T_Q(n,m) = O(m\log^\beta n)$ for some constant β. Then the running time becomes

$$T(n,m) \;=\; O(m\log_b n\cdot(\log n \;+\; \log^\beta(b\log_b n))).$$

Setting b so that $\log b = \lceil\log^{1/\beta} n\rceil$ implies $T(n,m) = O(m\log^{2-1/\beta} n)$.

In other words, the above method allows us to reduce the exponent in the logarithmic factor from β to $2 - 1/\beta$. Starting with $\beta \approx 2$ and repeating a finite number of times prove the theorem below.

Theorem 2.5 *For elements moving linearly, a KPQ can be implemented with cost function* $T(n,m) = O(m\log^{1+\varepsilon} n)$ *for any fixed constant* $\varepsilon > 0$.

Lemma 2.2 now implies a planar k-level algorithm running in $O((n+m)\log^{1+\varepsilon} n)$ time. This bound can be slightly improved to $O(n\log n + m\log^{1+\varepsilon} n)$ by observing that n insertions in the algorithm of Section 2.1 come from the initialization phase; these initial elements can be stored in a separate semi-KPQ. The $O(n\log n)$ overhead term can be reduced to $O(n\log m)$ for small values of m by a grouping trick of the author [10]. The space complexity is easily seen to be linear.

Corollary 2.6 *Given n lines in* $\mathbb{R}^2$ *in general position, we can compute all m vertices of the k-level in* $O(n\log m + m\log^{1+\varepsilon} n)$ *time and* $O(n)$ *space deterministically for any fixed constant* $\varepsilon > 0$.

Remark. Instead of bootstrapping, we can apply recursion on $S.Q$. The $\log^\varepsilon n$ factor can be replaced with a somewhat complicated function in n. The approach can be modified for degenerate arrangements, if the output size m is re-interpreted as the number of pairs of lines that define k-level vertices.

This second solution does not generalize to curves at the moment, because of Lemma 2.4; however, Theorem 2.5 can be used to construct the k-level of line segments in $O((n+m)\log^{1+\varepsilon} n)$ time.

3 A Randomized Incremental Algorithm

We now discuss a different algorithmic approach for planar k-levels based on another powerful paradigm in computational geometry—*randomized incremental construction* [9, 34]. We show that a certain randomized incremental algorithm by Agarwal, de Berg, Matoušek, and Schwarzkopf [1] is slightly faster (in expectation) than our best sweep-line algorithm, replacing the $\log^\varepsilon n$ factor by an even slower-growing function, at the expense of increasing space. The original analysis by Agarwal *et al.* is not output-sensitive.

Our description and analysis of the algorithm are necessarily sketchy, as most of the main points can be found in Agarwal *et al.*'s paper; we refer the interested reader to their paper for the missing details. There are actually two versions of the algorithm. The first version is simpler, but our analysis seems to yield an extra logarithmic factor. The second version removes this extra logarithmic factor for the planar case by a refined activity test. We describe the second. We admit that conceptually the sweep-line algorithms (particularly, the one in Section 2.2) are easier to grasp, since they are one-dimensional in nature.

Algorithm outline. The idea is to examine a random permutation $\ell_1, \ldots, \ell_n$ of the given set of non-degenerate lines and to maintain incrementally (for $r = 1, \ldots, n$) the "canonical triangulation" of a certain subcomplex $\mathcal{K}_r$ of the arrangement $\mathcal{A}_r$ of $\ell_1, \ldots, \ell_r$. For each triangle Δ of the canonical triangulation, we also maintain the "conflict list" $K(\Delta)$ (the set of all lines intersecting Δ), as well as a "level number" $k(q)$ (the number of lines below q) for some point $q \in \Delta$.

At the r-th iteration, $\mathcal{K}_r$ is formed by splitting each cell of $\mathcal{K}_{r-1}$ intersected by ℓ_r into two new sub-cells. We then re-triangulate, and update conflict lists (and level numbers) in a way typical in randomized incremental construction. In addition, we need to perform "activity tests" to throw away cells from $\mathcal{K}_r$ that are found irrelevant.

How does the activity test work on a triangle Δ of a given cell? We estimate the *distance* of Δ to the k-level, i.e., the minimum of $|k - k(q)|$ over all points $q \in \Delta$. We cannot afford to compute the exact distance, but we can approximate the distance (from above) to within an additive error of n/r by using known construction of an $\left(\frac{n}{r|K(\Delta)|}\right)$*-cutting* of $K(\Delta)$ in time $O(|K(\Delta)|^2 r/n)$; see Chazelle and Friedman [15] for a simple randomized algorithm. If the approximate distance exceeds n/r, then Δ is marked. If all triangles of the cell are marked, then the entire cell is deemed inactive and is discarded from $\mathcal{K}_r$.

It is clear that after all triangles of a cell are tested, (i) if the cell intersects the k-level, then it is kept, and (ii) if the cell is kept, then it intersects the region between the $(k - \lceil n/r \rceil)$-level and the $(k + \lceil n/r \rceil)$-level. However, we cannot afford to perform an activity test on all current triangles at every iteration. A lazy approach that is almost as good is as follows: perform an activity test on a triangle when it is created and whenever r reaches a power of 2. This implies the following invariants about the subcomplex $\mathcal{K}_r$:

(i) all cells that intersect the k-level are in $\mathcal{K}_r$, and

(ii) every cell in $\mathcal{K}_r$ intersects the region between the $(k - \lceil 2n/r \rceil)$-level and the $(k + \lceil 2n/r \rceil)$-level.

At the end with $r = n$, we have the k-level.

Analysis outline. Define $\widetilde{\mathcal{K}}_r$ to be the canonical triangulation of the subcomplex consisting of all cells in $\mathcal{A}_r$ that intersect the region between the $(k - \lceil 2n/r \rceil)$-level and the $(k + \lceil 2n/r \rceil)$-level, together with cells that are adjacent to these cells.

We first analyze the cost of the activity tests associated with creations of triangles at the r-th iteration. From the analysis of Agarwal *et al.*, this is asymptotically bounded by $\sum_\Delta |K(\Delta)|^2 r/n = O((n/r^2) f(r))$, where the sum is over all $\Delta \in \widetilde{\mathcal{K}}_r \setminus \widetilde{\mathcal{K}}_{r-1}$, and $f(r)$ is a well-behaved upper bound on the expected size of $\widetilde{\mathcal{K}}_r$.

To account for the cost of activity tests performed when r is a power of 2, we need to add $\sum_\Delta |K(\Delta)|^2 r/n = O((n/r) f(r))$, according to the analysis of Agarwal *et al.*, where this time the sum is over all $\Delta \in \widetilde{\mathcal{K}}_r$.

These yield the dominate costs, with the total expected running time being

$$O\left(\sum_{r=1}^{n} \frac{n}{r^2} f(r) \;+\; \sum_{i=1}^{\log_2 n} \frac{n}{2^i} f(2^i)\right) \;=\; O\left(\sum_{r=1}^{n} \frac{n}{r^2} f(r)\right).$$

Bounding the size of $\widetilde{\mathcal{K}}_r$. It remains to specify the function $f(r)$ and here the analysis deviates somewhat from Agarwal *et al.*'s in that we take into account the size m of the k-level. We begin with a combinatorial fact:

Observation 3.1 *If the k-level has m vertices, then the $(k-j)$-level, $(k-j+1)$-level, ..., and the $(k+j)$-level have a total of $O((n+m)\alpha(n)j)$ vertices.*

Proof: Break each line into segments by intersecting it with the region strictly above the k-level. As a result, we get a collection S' of $O(n+m)$ line segments. The $(k+1)$-level, $(k+2)$-level, ..., and $(k+j)$-level in the original arrangement coincides with the 1-level, 2-level, ..., and j-level in the arrangement of the segments S'. The latter is known to have combined complexity $O(|S'|\alpha(|S'|)j)$ [39], which implies one half of the lemma. The other half is of course similar. □

By the above observation, there exists an index $\ell \in (2n/r, 3n/r]$ such that the size of the $(k-\ell)$-level and the $(k+\ell)$-level is $O((n+m)\alpha(n))$. As we want a bound $f(r)$ on the number of vertices in $\widetilde{\mathcal{K}}_r$, we classify vertices into three types: 1. those that lie between the $(k-\ell)$-level and $(k+\ell)$-level, 2. those that lie strictly above the $(k+\ell)$-level, and 3. those that lie strictly below the $(k-\ell)$-level.

For type-1 vertices, note that by Observation 3.1, the region between the $(k-\ell)$-level and the $(k+\ell)$-level has $O((n+m)\alpha(n)n/r)$ vertices in total. Since the probability that a fixed vertex is a vertex of $\mathcal{A}_r$ is $O((r/n)^2)$, the expected number of type-1 vertices in $\widetilde{\mathcal{K}}_r$ is $O((n+m)\alpha(n)n/r \cdot (r/n)^2) = O((n+m)\alpha(n)r/n)$.

To count the number of type-2 vertices, we break each line of $\mathcal{A}_r$ into segments by intersecting it with the region strictly above the $(k+\ell)$-level. As a result, we get a collection R' of line segments. Now, an endpoint in R' is a vertex of the $(k+\ell)$-level, which has $O((n+m)\alpha(n))$ vertices in total. Since the probablity that a fixed vertex lies on a line of $\mathcal{A}_r$ is $O(r/n)$, the expected number of endpoints in R', and hence the expected number of segments in R', is bounded by $E[|R'|] = O((n+m)\alpha(n)r/n)$.

Type-2 vertices in $\widetilde{\mathcal{K}}_r$ can be subclassified into (i) vertices of cells in $\mathcal{A}_r$ that intersect the $(k+\ell)$-level, and (ii) vertices of adjacent cells. Type-2(i) vertices belong to a common face in the arrangement of the segments in R'. Known combinatorial bounds imply that there are $O(|R'|\alpha(|R'|))$ such vertices [29, 40]. The number of type-2(ii) vertices is of the same order in expectation, by a standard argument of Clarkson and Shor [17]. Therefore, the expected number of type-2 vertices in $\widetilde{\mathcal{K}}_r$ is $O((n+m)\alpha(n)^2 r/n)$.

Type-3 vertices in $\widetilde{\mathcal{K}}_r$ can be counted in an analogous way. Thus, we can set $f(r) = (n+m)\alpha(n)^2 r/n$ and obtain the overall expected time bound:

$$O\left(\sum_{r=1}^{n} \frac{n}{r^2} \cdot (n+m)\alpha(n)^2 r/n\right) = O\left((n+m)\alpha(n)^2 \sum_{r=1}^{n} \frac{1}{r}\right) = O((n+m)\alpha(n)^2 \log n).$$

Theorem 3.2 *Given n lines in $\mathbb{R}^2$ in general position, we can compute all m vertices of the k-level in $O((n+m)\alpha(n)^2 \log n)$ expected time.*

Remark. Perhaps a better analysis would resolve the $\alpha(n)$-factor difference with Har-Peled's result [28], but for all practical purposes, $\alpha(n)$ can be viewed as just a constant. It remains to be seen whether the algorithm by Har-Peled or the above algorithm by Agarwal *et al.* is better fitted for implementation. One important simplification of the latter lies in the activity test: by observing that the distance of a triangle Δ to the k-level is identical to the distance of the boundary $\partial\Delta$ to the k-level, two-dimensional cuttings can be replaced just by one-dimensional cuttings.

Both algorithms generalize to curves: Har-Peled's expected time bound is $O(\lambda_{a+2}(n+m)\log n)$ [28] and ours is almost the same (if α-like factors are ignored).

4 An Output-Insensitive Algorithm

It is of theoretical interest to obtain provable worst-case time bounds for constructing planar k-levels. The preceding output-sensitive algorithms do not necessarily yield the best result in this respect, and we describe, in this section, a simple way to achieve a time bound that matches with the current best combinatorial bound, using randomization in a mild way.

Our output-insensitive result is an immediate consequence of two facts, the first on the combinatorics of consecutive levels, the second on constructing a level sandwiched between two pre-computed levels.

Observation 4.1 *Given n lines and a random integer $k' \in (k-j, k]$, the expected number of vertices in the k'-level is $O(n(k/j)^{1/3})$.*

Proof: Dey [19] proved that the total number of vertices in the k-level, $(k-1)$-level, ..., and the $(k-j+1)$-level is $O(nk^{1/3}j^{2/3})$. The quantity of interest is this divided by j. □

Lemma 4.2 *Let $k' \in (k-j, k)$ be an integer. Given the $(k-j)$-level and the k-level, we can construct the k'-level in $O((n+M)\log^c j)$ time, where c is a constant, and M is the total number of vertices in the $(k-j)$-level, k-level, and k'-level.*

Proof: The result can be obtained by a straightforward modification of a sweep-line algorithm, for example, the one in Section 2.1. Basically, the only changes lie in the two KPQs S_1 and S_2, which now holds $O(j)$ elements instead of $O(n)$: S_1 contains all elements of rank from $k-j$ to $k'-1$, and S_2 contains all elements of rank from k' to k. Whenever the sweep line $x=t$ passes through a vertex of the $(k-j)$-level, we check how the $(k-j)$-th smallest element changes and update S_1 if necessary. Similarly, when a vertex of the k-level is encountered, S_2 is updated accordingly. The data structures in Sections 2.2 and 2.3 yield $c \approx 2$ and $c \approx 1$ respectively. □

Let $r_0 = 0$ and r_i be a random integer in $(2^{i-1}, 2^i]$ for each $i = 1, \ldots, \ell$, where $\ell = \lfloor \log_2 k \rfloor$. The following constructs a sequence of levels converging to the k-level:

```
1.  construct the (k − r_ℓ)-level and the (k + r_ℓ)-level
2.  for i = ℓ − 1 down to 0 do
3.      construct the (k − r_i)-level and the (k + r_i)-level
            from the (k − r_{i+1})-level and the (k + r_{i+1})-level
```

To analyze this simple strategy, let m_i denote the number of vertices in the $(k-r_i)$-level and $(k+r_i)$-level. Line 1 takes $O((n+m_\ell)\log^c n)$ time. The expected value of m_ℓ is $O(n)$ by Observation 4.1. Lemma 4.2 implies that the cost of line 3 is $O((n+m_i+m_{i+1})\log^c r_{i+1}) = O((n+m_i+m_{i+1})i^c)$.

The expected value of m_i is $O(n(k/2^i)^{1/3})$ by Observation 4.1. Therefore, the expected running time sums to

$$O\left(n\log^c n \ + \ \sum_{i=0}^{\ell-1} nk^{1/3}\frac{i^c}{2^{i/3}}\right) \ = \ O(n\log^c n + nk^{1/3}).$$

Theorem 4.3 *Given n lines in $\mathbb{R}^2$, we can construct the k-level in $O(n\log^c n + nk^{1/3})$ expected time for some constant c.*

The time bound is the same even if we set $\ell = \lfloor \min\{\log_2 k,\ 3c\log_2\log_2 n\}\rfloor$. The advantage of this variant is that the number of random bits is reduced. In fact, since independence of the r_i's is not necessary, only $O(\log\log n)$ random bits suffice. Also, the space can be made $O(n)$ by a non-trivial variant of the algorithm which we leave to the reader.

The overhead term in the time bound can be brought down to $O(n\log n)$, if we sacrifice simplicity and use the author's random sampling technique [11] on top of this algorithm (which increases the space and the number of random bits).

Corollary 4.4 *Given n lines in $\mathbb{R}^2$, we can construct the k-level in $O(n\log n + nk^{1/3})$ expected time.*

Remark. The approach applies to other types of arrangements of curves in the plane; an analogue to Observation 4.1 is needed, however. Observation 4.1 and Lemma 4.2 also give us a simple way to reduce an $O((n+m)\log^c n)$ output-sensitive time bound to $O(n\log^c n + m\log^c k)$; alternatively (and deterministically), one could apply the convex layers idea of Cole, Sharir, and Yap [18].

5 Applications

We briefly mention a few consequences for some specific problems studied in computational geometry.

k-sets. First, by duality, all the algorithms can be used to enumerate all *k-sets* [2, 4, 20] of a planar n-point set. Care must be taken in defining this particular problem, as often we do not want to report an $O(k)$-size description for each k-set. Also, the number of k-sets is not exactly the same as the number of k-level vertices, but rather is related to the number of vertices in the intersection of two consecutive levels; the algorithms in Sections 2 and 3 are actually sensitive to this latter parameter.

Higher-order Voronoi diagrams. For an application of our deterministic output-sensitive algorithm, we improve an early algorithm of Chazelle and Edelsbrunner [14] for *higher-order Voronoi diagrams* [2, 20, 34], which originally ran in $O(n^2\log^2 n)$ worst-case time. These diagrams are used to solve a number of problems on a planar point set, such as finding a k-point subset of minimal variance [3, 23], and finding the smallest circle enclosing all but k points [33].

Corollary 5.1 *The order-k Voronoi diagram of n points in $\mathbb{R}^2$ can be constructed in $O(n^2\log^{1+\varepsilon} n)$ time deterministically.*

Proof: First, by a standard "lifting map" [20], the problem reduces to constructing the k-level of a set H of n planes in $\mathbb{R}^3$, where each plane touches the lower envelope. Given $h \in H$, the k-level restricted to the plane h is equivalent to a two-dimensional k-level of lines after another transformation that maps a point in the lower envelope of H into $(0, -\infty)$. Corollary 2.6 implies an $O(n \log n + m_h \log^{1+\varepsilon} n)$ time bound, where m_h denotes the number of k-level vertices lying on h. Repeating this process for every $h \in H$ and gluing the structure together yield an overall running time

$$O\left(\sum_{h\in H} (n \log n + m_h \log^{1+\varepsilon} n)\right) \;=\; O(n^2 \log n + m \log^{1+\varepsilon} n),$$

where m, the size of the output diagram, is known to be $O(nk)$ [31]. □

Using more complicated machinery (namely, shallow cuttings [11]), we can make the above result sensitive to k, leading to the current best deterministic time bound for planar order-k Voronoi diagrams: $O(nk \log^{1+\varepsilon} k \cdot (\log n/ \log k)^{O(1)})$. However, simpler and slightly faster randomized algorithms are known [1, 11, 38].

Ham-sandwich cuts. For one application of the output-insensitive algorithm (Theorem 4.3), we mention Lo, Matoušek, and Steiger's method [32] for the well-known *ham-sandwich cut* problem [20], which now implies:

Corollary 5.2 *Given three n-point sets in $\mathbb{R}^3$, we can find a plane that simultaneously bisects the three sets in $O(n^{4/3})$ expected time.*

Proof: Lo *et al.* [32] bounded the time to compute a d-dimensional ham-sandwich cut by the time to construct a level in a $(d-1)$-dimensional hyperplane arrangement. □

Bichromatic segment intersection: connected case. For an application of kinetic priority queues (Theorem 2.5), we speed up an $O((n+m)\alpha(n) \log^3 n)$-time algorithm by Basch, Guibas, and Ramkumar [7] for a variant of the standard *red/blue segment intersection* problem in the connected case:

Corollary 5.3 *Given a connected family R of n red line segments and a connected family B of n blue line segments, we can report all m intersecting pairs from $R \times B$ in $O((n+m) \log^{2+\varepsilon} n)$ time and $O(n)$ space deterministically.*

Proof: Basch *et al.*'s sweep-line algorithm [7] solves the problem by using multiple KPQs, with a total of $O((n+m) \log n)$ insertions. □

References

[1] P. K. Agarwal, M. de Berg, J. Matoušek, and O. Schwarzkopf. Constructing levels in arrangements and higher order Voronoi diagrams. *SIAM J. Comput.*, 27:654–667, 1998.

[2] P. K. Agarwal and M. Sharir. Arrangements and their applications. To appear in *Handbook of Computational Geometry* (J. Urrutia and J. Sack, ed.), North-Holland.

[3] A. Aggarwal, H. Imai, N. Katoh, and S. Suri. Finding k points with minimum diameter and related problems. *J. Algorithms*, 12:38–56, 1991.

[4] A. Andrzejak and E. Welzl. k-sets and j-facets: a tour of discrete geometry. Manuscript, 1997.

[5] I. Bárány and W. Steiger. On the expected number of k-sets. *Discrete Comput. Geom.*, 11:243–263, 1994.

[6] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 747–756, 1997. *J. Algorithms*, to appear.

[7] J. Basch, L. J. Guibas, and G. Ramkumar. Reporting red-blue intersections between two sets of connected line segments. In *Proc. 4th European Sympos. Algorithms*, Lect. Notes in Comput. Sci., vol. 1136, Springer-Verlag, pages 302–319, 1996.

[8] J. Bentley and J. Saxe. Decomposable searching problems I: static-to-dynamic transformation. *J. Algorithms*, 1:301–358, 1980.

[9] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, 1997.

[10] T. M. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discrete Comput. Geom.*, 16:369–387, 1996.

[11] T. M. Chan. Random sampling, halfspace range reporting, and construction of ($\leq k$)-levels in three dimensions. In *Proc. 39th IEEE Sympos. Found. Comput. Sci.*, pages 586–595, 1998.

[12] T. M. Chan. Dynamic planar convex hull operations in near-logarithmic amortized time. Accepted to *40th IEEE Sympos. Found. Comput. Sci.*, 1999.

[13] B. Chazelle. On the convex layers of a planar set. *IEEE Trans. Inform. Theory*, IT-31:509–517, 1985.

[14] B. Chazelle and H. Edelsbrunner. An improved algorithm for constructing kth-order Voronoi diagrams. *IEEE Trans. Comput.*, C-36:1349–1354, 1987.

[15] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10:229–249, 1990.

[16] B. Chazelle and F. P. Preparata. Halfspace range search: an algorithmic application of k-sets. *Discrete Comput. Geom.*, 1:3–93, 1986.

[17] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[18] R. Cole, M. Sharir, and C. K. Yap. On k-hulls and related problems. *SIAM J. Comput.*, 16:61-77, 1987.

[19] T. K. Dey. Improved bounds on planar k-sets and k-levels. *Discrete Comput. Geom.*, 19:373–382, 1998.

[20] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Berlin, 1987.

[21] H. Edelsbrunner, P. Valtr, and E. Welzl. Cutting dense point sets in half. *Discrete Comput. Geom.*, 17:243–255, 1997.

[22] H. Edelsbrunner and E. Welzl. Constructing belts in two-dimensional arrangements with applications. *SIAM J. Comput.*, 15:271–284, 1986.

[23] D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete Comput. Geom.*, 11:321–350, 1994.

[24] P. Erdős, L. Lovász, A. Simmons, and E. Straus. Dissection graphs of planar point sets. In *A Survey of Combinatorial Theory* (J. N. Srivastava, ed.), North-Holland, Amsterdam, Netherlands, pages 139–154, 1973.

[25] H. Everett, J.-M. Robert, and M. van Kreveld. An optimal algorithm for the ($\leq k$)-levels, with applications to separation and transversal problems. *Int. J. Comput. Geom. Appl.*, 6:247–261, 1996.

[26] D. Halperin. Arrangements. In *Handbook of Discrete and Computational Geometry* (J. E. Goodman and J. O'Rourke, ed.), pages 389–412, CRC Press, 1997.

[27] D. Halperin and M. Sharir. New bounds for lower envelopes in three dimensions with applications to visibility of terrains. *Discrete Comput. Geom.*, 12:313–326, 1994.

[28] S. Har-Peled. Taking a walk in a planar arrangement. `http://www.math.tau.ac.il/~sariel/papers/98/walk.html`, December, 1998 (revised February, 1999).

[29] S. Hart and M. Sharir. Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes. *Combinatorica*, 6:151–177, 1986.

[30] J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. *BIT*, 32:249–267, 1992.

[31] D. T. Lee. On k-nearest neighbor Voronoi diagrams in the plane. *IEEE Trans. Comput.*, C-31:478–287, 1982.

[32] C.-Y. Lo, J. Matoušek, and W. L. Steiger. Algorithms for ham-sandwich cuts. *Discrete Comput. Geom.*, 11:433–452, 1994.

[33] J. Matoušek. On geometric optimization with few violated constraints. *Discrete Comput. Geom.*, 14:365–384, 1995.

[34] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice-Hall, Englewood Cliffs, N.J., 1994.

[35] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Sys. Sci.*, 23:166–204, 1981.

[36] J. Pach, W. Steiger, and E. Szemerédi. An upper bound on the number of planar k-sets. *Discrete Comput. Geom.*, 7:109–123, 1992.

[37] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

[38] E. Ramos. On range reporting, ray shooting, and k-level construction. To appear in *Proc. 15th ACM Sympos. Comput. Geom.*, 1999.

[39] M. Sharir. On k-sets in arrangements of curves and surfaces. *Discrete Comput. Geom.*, 6:593–613, 1991.

[40] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications.* Cambridge University Press, New York, 1995.