Constructing Planar Cuttings in Theory and Practice*

Sariel Har-Peled†

May 1, 2000‡

Abstract

We present several variants of a new randomized incremental algorithm for computing a cutting in an arrangement of n lines in the plane. The algorithms produce cuttings whose expected size is $O(r^2)$, and the expected running time of the algorithms is $O(nr)$. Both bounds are asymptotically optimal for nondegenerate arrangements. The algorithms are also simple to implement, and we present empirical results showing that they perform well in practice. We also present another efficient algorithm (with slightly worse time bound) that generates small cuttings whose size is guaranteed to be close to the best known upper bound of [Mat98].

1 Introduction

A natural approach for solving various problems in computational geometry is the divide-and-conquer paradigm. A typical application of this paradigm to problems involving a set $\hat{S}$ of n lines in the plane, is to fix a parameter $r > 0$, and to partition the plane into regions $R_1, \ldots, R_m$ (those regions are usually vertical trapezoids, or triangles, but we will consider here also convex polygons with more edges), such that the number of lines of $\hat{S}$ that intersect the interior of R_i is at most n/r, for any $i = 1, \ldots, m$, see Figure 1.1. This allows us to split the problem at hand into subproblems, each involving the subset of lines intersecting a region R_i. Such a partition is called a $(1/r)$-*cutting* of the plane. See [Aga91] for a survey of algorithms that use cuttings. For further work related to cuttings, see [AM95].

The first (though not optimal) construction of cuttings is due to Clarkson [Cla87]. Chazelle and Friedman [CF90] showed the existence of $(1/r)$-cuttings with $m = O(r^2)$ (a bound that is worst-case tight). They also showed that such cuttings, consisting of vertical trapezoids, can be computed in $O(nr)$ time. An optimal deterministic algorithm for generating cuttings was given by [Cha93]. Although those constructions are asymptotically optimal, they do not seem to produce a practically small number of regions. Coming up with a really small number of regions (i.e., reducing the constant of proportionality) is important for the efficiency of (recursive) data structures and algorithms that use cuttings. Currently, the best lower bound on the number of vertical trapezoids in a $(1/r)$-cutting in an arrangement of lines is $2.54(1 - o(1))r^2$, and the optimal cutting has at most $8r^2 + 6r + 4$ trapezoids; see [Mat98]. Improving the upper and lower bounds on the size of cuttings is still open, indicating that our understanding of

*A preliminary version of the paper appeared in the *14th ACM Symposium of Computational Geometry*, 1998. This work has been supported by a grant from the U.S.–Israeli Binational Science Foundation. This work is part of the author's Ph.D. thesis, prepared at Tel-Aviv University under the supervision of Prof. Micha Sharir.

†School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978, Israel; urlhttps://sarielhp.org.

‡ReLaTeXed April 24, 2019.

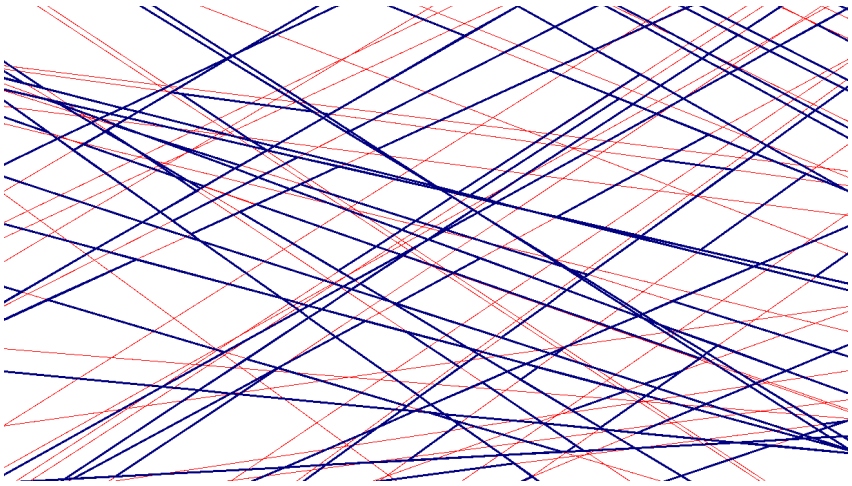

Figure 1.1: (1/20)-cutting of 100 lines as computed by our demo program [Har98], using the `PolyTree` algorithm (see. Section 4.2). The boundaries of the cutting regions are marked by thick lines, and each such region intersects at most 5 lines in its interior.

cuttings is still far from being fully satisfactory. In Section 3, we outline Matoušek's construction for achieving the upper bound and show a slightly improved construction (see below for details).

In this paper we propose several variants of a new and simple randomized incremental algorithm `CutRandomInc` for constructing cuttings, and prove the expected worst-case tight performance bounds, as stated in the abstract, for `CutRandomInc` and for some of its variants. We also present empirical results on several algorithms/heuristics for computing cuttings that we have implemented. They are mostly variants of our new algorithm, and they all perform well in practice.[1] As already stated, $O(r^2)$ bounds on the expected size of the cuttings for some of those variants can be proved. For the other improved algorithms, no formal proof of performance is currently available, and we leave this as an open question for further research.

Matoušek [Mat98] gave an alternative construction for cuttings, showing that there exists a $(1/r)$-cutting with at most (roughly) $8r^2$ vertical trapezoids. Unfortunately, this construction relies on computing the whole arrangement, and its computation thus takes $O(n^2)$ time. We present a new randomized algorithm that is based on Matoušek's construction; it generates a $(1/r)$-cutting of size $\leq (1+\varepsilon)8r^2$, in $O\left(\frac{nr}{\varepsilon}\log^2 n\right)$ expected time, where $0 < \varepsilon \leq 1$ is any prescribed constant.

In Section 2, we present the main two variants of the new algorithm `CutRandomInc`, and analyze their expected running time and the expected number of trapezoids that they produce. Specifically, the expected running time is $O(nr)$ and the expected size of the output cutting is $O(r^2)$. We also analyze, in Section 2.1, another variant `CRIVPolygon` of the algorithm that also has similar performance bounds. In Section 3 we present and analyze our variant of Matoušek's construction. In Section 4 we present our empirical results, comparing the new algorithms with several other algorithms/heuristics for constructing cuttings. These algorithms are mostly also variants of `CutRandomInc`, (except that we still do not have a formal analysis of their performance bound), but they also include a variant of the older algorithm of Chazelle and Friedman. The first batch of the implemented algorithms generate cuttings that consist of vertical trapezoids. Our empirical results show that the cuttings generated by the new algorithm `CutRandomInc` and its variants have between $10r^2$ and $14r^2$ vertical trapezoids. (The algorithms generate smaller cuttings when r is small. For example, for $r = 2$ the constant is about 9.)

[1] In spite of the theoretical importance of cuttings (in the plane and in higher dimensions), we are not aware to any (other) implementation of efficient algorithms for constructing cuttings.

In contrast, the Chazelle-Friedman algorithm generates cuttings of size roughly $70r^2$. Some variants of our algorithm are based on cuttings by convex polygons with a small number of edges rather than by vertical trapezoids. These perform even better in practice, and we have a proof of optimality for one of the methods `CRIVPolygon`, which can be interpreted as an extension of `CutRandomInc` (see Section 2.1). We conclude in Section 5 by mentioning a few open problems. A program with a GUI demonstrating the algorithms and heuristics presented in the paper, is available on the web in source form [Har98].

2 Incremental Randomized Construction of Cuttings

Given a set $\hat{S}$ of n lines in the plane, let $\mathcal{A}(\hat{S})$ denote the arrangement of $\hat{S}$; namely, the partition of the plane into faces, edges, and vertices as induced by the lines of $\hat{S}$ [Ede87]. Let $\mathcal{A}_{\mathcal{VD}}(\hat{S})$ denote the partition of the plane into vertical trapezoids (i.e., the vertical decomposition of $\mathcal{A}(\hat{S})$), obtained by erecting two vertical segments up and down from each vertex of $\mathcal{A}(\hat{S})$, and extending each of them until it either reaches a line of $\hat{S}$, or otherwise all the way to infinity.

Computing the decomposed arrangement $\mathcal{A}_{\mathcal{VD}}(\hat{S})$ can be done as follows. Pick a random permutation $S = \langle s_1, \ldots, s_n \rangle$ of $\hat{S}$. Put $S_i = \langle s_1, \ldots, s_i \rangle$, for $i = 1, \ldots, n$. We compute incrementally the decomposed arrangements $\mathcal{A}_{\mathcal{VD}}(S_i)$, for $i = 1, \ldots, n$, by inserting the ith line s_i of S into $\mathcal{A}_{\mathcal{VD}}(S_{i-1})$. To do so, we compute the *zone* Z_i of s_i in $\mathcal{A}_{\mathcal{VD}}(S_{i-1})$, which is the set of all trapezoids in $\mathcal{A}_{\mathcal{VD}}(S_{i-1})$ that intersect s_i. We split each trapezoid of Z_i into at most 4 trapezoids, such that no trapezoid intersects s_i in its interior, as in [SA95]. Finally, we perform a pass over all the newly created trapezoids, merging vertical trapezoids that are adjacent, and have identical top and bottom lines. The merging step guarantees that the resulting decomposition is $\mathcal{A}_{\mathcal{VD}}(S_i)$, independently of the insertion order of elements in S_i; see [dBCKO08].

However, if we decide to skip the merging step, the resulting structure, denoted as $\mathcal{A}^{|}(S_i)$, depends on the order in which the lines are inserted into the arrangement. In fact, $\mathcal{A}^{|}(S_i)$ is $\mathcal{A}_{\mathcal{VD}}(S_i)$ with additional superfluous vertical walls. Each such vertical wall is a fragment of a vertical wall that was created at an earlier stage and got split during a later insertion step.

Definition 2.1. Let $\hat{S}$ be a set of n lines in the plane, and let $0 < c < 1$ be a constant. A *c-cutting* of $\hat{S}$ is a partition of the plane into regions $R_1, \ldots, R_m$, such that, for each $i = 1, \ldots, m$, the number of lines of $\hat{S}$ that intersect the interior of R_i is at most cn.

A region C in the plane is *c-active* if the number of lines of $\hat{S}$ that intersect the interior of C is larger than cn.

A $(1/r)$-*cutting* is thus a partition of the plane into m regions such that none of them is $(1/r)$-active. Chazelle and Friedman [CF90] showed that one can compute, in $O(nr)$ time, a $(1/r)$-cutting that consists of $O(r^2)$ vertical trapezoids. Both bounds are asymptotically tight in the worst case.

We propose a new algorithm for computing a cutting that works by incrementally computing the arrangements $\mathcal{A}^{|}(S_i)$, using a random insertion order S of the lines. The new idea in the algorithm is that any "light" trapezoid (i.e., a trapezoid that is not $(1/r)$-active) constructed by the algorithm is immediately added to the final cutting, and the algorithm does not maintain the arrangement inside such a trapezoid from this point on. In this sense, one can think of the algorithm as being greedy; that is, it adds a trapezoid to the cutting as soon as one is constructed, and proceeds in this manner until the whole plane is covered. The algorithm, called `CutRandomInc`, is depicted in Figure 2.1.

The algorithm has two variants. One does not merge adjacent trapezoids (as in the construction of $\mathcal{A}^{|}(S)$), while the other performs such mergings (as in the construction of $\mathcal{A}_{\mathcal{VD}}(S)$).

ALGORITHM `CutRandomInc`($\hat{S}$, r, $merge\text{–}flag$)
`Input:` A set $\hat{S}$ of n lines, a positive integer r, and
a flag $merge\text{–}flag$ that indicates whether merging is used or not
`Output:` A $(1/r)$-cutting of $\hat{S}$ by vertical trapezoids
begin
Choose a random permutation $S = \langle s_1, s_2, \ldots, s_n \rangle$ of $\hat{S}$.
$\mathcal{C}_0 \leftarrow \{\mathbb{R}^2\}$.
$i \leftarrow 0$.
while there are $(1/r)$-active trapezoids in $\mathcal{C}_i$ **do**
$i \leftarrow i + 1$
$Zone_i \leftarrow$ The set of $(1/r)$-active trapezoids in $\mathcal{C}_{i-1}$ that intersect s_i.
$Zone'_i \leftarrow \cup_{\Delta \in Zone_i} split(\Delta, s_i)$,
where $split(\Delta, s)$ is the operation of splitting a vertical trapezoid Δ
crossed by a line s into at most four vertical trapezoids, as in [dBCKO08],
such that the new trapezoids cover Δ, and they do not intersect
s in their interior.
if $merge\text{–}flag$ **then**
Merge adjacent trapezoids in $Zone'_i$ that have the same top
and bottom lines (one of which is s_i).
end if
$\mathcal{C}_i \leftarrow (\mathcal{C}_{i-1} \setminus Zone_i) \cup Zone'_i$.
end while

return $\mathcal{C}_i$
end `CutRandomInc`

Figure 2.1: Algorithm for constructing a $(1/r)$-cutting of an arrangement of lines

If `CutRandomInc` outputs $\mathcal{C}_k$, for some $k < n$, then $\mathcal{C}_k$ has no $(1/r)$-active trapezoids, and it is thus a $(1/r)$-cutting. After the ith line is being inserted, it is guaranteed that no active trapezoid of $\mathcal{C}_j$ intersects s_i in its interior, for $j \geq i$. This remains true even if merging is done by `CutRandomInc`. In particular, $\mathcal{C}_n$ has no active region, and it is a cutting.

To appreciate the following proof of correctness and optimality of `CutRandomInc`, one has to observe that the covering $\mathcal{C}_i$ of the plane maintained by `CutRandomInc` depends heavily on the order in which the lines are inserted into the arrangement. Indeed, the set of active trapezoids maintained by `CutRandomInc` falls outside the classical frameworks of Clarkson and Shor [CS89], lazy randomized incremental construction [dBDS95], and epsilon nets [HW87]. See Figure 2.4 and Figure 2.5, for situations that illustrate the difference between these frameworks and ours. In order to analyze our algorithms, new techniques need to be developed.

In the following, we denote by R a *selection* of $\hat{S}$ of length $r \leq n$, i.e., an ordered sequence of r distinct elements of $\hat{S}$. By a slight abuse of notation, we also denote by R the unordered set of its elements. We define the *weight* of a trapezoid to be the number of lines that cross its interior.

Definition 2.2. Let $\mathcal{T} = \mathcal{T}(\hat{S})$ denote the set of all vertical trapezoids whose top and bottom edges are contained in lines of $\hat{S}$, and whose vertical sides are contained in lines that pass through vertices of $\mathcal{A}(\hat{S})$.

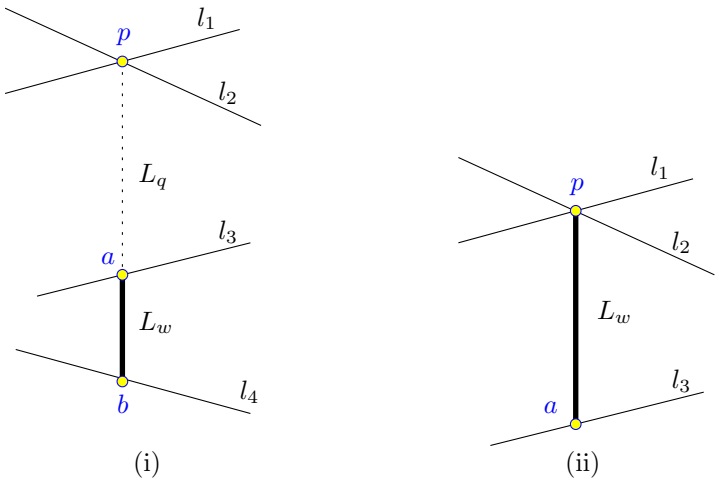

Figure 2.2: Splitters created by `CutRandomInc`

For a selection R of $\hat{S}$, let $\mathcal{CT}_{\mathcal{VD}}(R)$ denote the set of trapezoids of $\mathcal{A}_{\mathcal{VD}}(R)$. A trapezoid in $\mathcal{CT}_{\mathcal{VD}}(R)$ is defined by at most 4 lines. For an integer $0 \leq k \leq n$, let $\mathcal{CT}_{\mathcal{VD}}(R, k)$ denote the trapezoids of $\mathcal{CT}_{\mathcal{VD}}(R)$ having weight at least k.

Definition 2.3. A vertical segment that serves as a left or right side of a trapezoid in $\mathcal{T}(\hat{S})$ is called a *splitter*. The *weight* of a splitter is the number of lines of $\hat{S}$ that cross the relative interior of the splitter. For a selection R of $\hat{S}$, let $\mathcal{CT}_{\mathcal{SP}}(R)$ denote the set of splitters of the trapezoids of $\mathcal{A}^{|}(R)$, and let $\mathcal{CT}_{\mathcal{SP}}(R, k)$ denote the set of splitters in $\mathcal{CT}_{\mathcal{SP}}(R)$ of weight at least k, where $0 \leq k \leq n$. In general, a splitter in $\mathcal{CT}_{\mathcal{SP}}(R)$ is uniquely defined by 4 lines: two define the vertex of the arrangement through which the vertical line containing the splitter passes, and two define (pass through) its top and bottom endpoints; see Figure 2.2 (i). There are also splitters that are adjacent to the vertex that induces them (see Figure 2.2 (ii)); these are defined uniquely by 3 lines.

In the following, S denotes the random permutation of $\hat{S}$ used by `CutRandomInc`.

Lemma 2.4. *Let s be a splitter induced by $\{l_1, l_2, l_3, l_4\} \subseteq \hat{S}$, so that l_1 and l_2 intersect at a vertex p, s is contained in the vertical line l passing through p, and the endpoints of s are $a = l_3 \cap l$, $b = l_4 \cap l$, with a nearer to p than b. Let L_w be the set of lines of $\hat{S}$ that intersect the relative interior of s, and let L_q be the set of lines of $\hat{S}$ that intersect the relative interior of ap; see Figure 2.2 (i). Then the probability of s to be created by* `CutRandomInc` *is bounded by*

$$\frac{8}{(w+1)^2(q+w+3)^2},$$

where $w = |L_w|, q = |L_q|$.

If one of the endpoints of the splitter is p, then the probability of s to be created is $\leq 6/(w+1)^3$; where $w = |L_w|$, and L_w is the set of lines crossing the relative interior of s; see Figure 2.2 (ii).

Proof: Let A denote the event that l_1, l_2 appear in S before all the lines of $L_q \cup L_w \cup \{l_3\}$, and let B denote the event that l_3, l_4 appear in S before all the lines of L_w.

The event A is a necessary and sufficient condition for pb (or a longer segment) to be created, when merging is not used. The event B, conditioned on A, is a necessary and sufficient condition for the vertices delimiting s to be created before s is being "killed". Hence s is created by `CutRandomInc`

(without merging) if and only if $A \cap B$ occurs for S. (If merging is used, only one implication holds: If s is created then $A \cap B$ occurs for S.)

To compute $P(A \cap B)$ it suffices to consider permutations of only the $q + w + 4$ lines in $L_q \cup L_w \cup \{l_1, l_2, l_3, l_4\}$. We distinguish between two cases:

(i) The first three lines in such a permutation are the lines l_1, l_2, l_4, in any order in which l_4 is not the third line. This ensures the occurrence of A. We now choose the $w+1$ locations of the lines in $L_w \cup \{l_3\}$ in the permutation, and place l_3 at the first of the these locations, thus ensuring B. The number of such permutations is $(6-2)\binom{q+w+1}{w+1} w!q! = 4(q+w+1)!/(w+1)$.

(ii) The first two lines in the permutation are l_1, l_2 (in any order). Again, A is ensured. To ensure B, we choose the $w+2$ locations of the lines in $L_w \cup \{l_3, l_4\}$, and place l_3, l_4 as the first two of them (in any order). The number of such permutations is

$$2!\binom{q+w+2}{w+2} 2!w!q! = \frac{4(q+w+2)!}{(w+1)(w+2)}.$$

It is easily verified that these two cases exhaust all possibilities of $A \cap B$ to arise. Hence,

$$\begin{aligned} P(A \cap B) &= \frac{4(q+w+1)!}{(w+1)(q+w+4)!} + \frac{4(q+w+2)!}{(w+1)(w+2)(q+w+4)!} \\ &\leq \frac{4}{(w+1)(q+w+2)(q+w+3)^2} + \frac{4}{(w+1)^2(q+w+3)^2} \\ &\leq \frac{8}{(w+1)^2(q+w+3)^2} \end{aligned}$$

The proof of the second part of the lemma follows by observing that ap is created iff l_1, l_2, l_3 appear in S before all the lines of L_w. The probability for this to happen is $3!w!/(w+3)! \leq 6/(w+1)^3$. ∎

Definition 2.5. Let $\mathcal{T}_{\mathcal{SP}}(S)$ denote the set of splitters in $\bigcup_{i=1}^{n} \mathcal{CT}_{\mathcal{SP}}(S_i)$, let $\mathcal{T}_{\mathcal{VD}}(S)$ denote the set of trapezoids in $\bigcup_{i=1}^{n} \mathcal{CT}_{\mathcal{VD}}(S_i)$. Let $\mathcal{T}^{\mathcal{A}}_{\mathcal{SP}}(S) = \bigcup_{i=1}^{n} \mathcal{CT}_{\mathcal{SP}}(S_i, n/(2r))$, and let $\mathcal{T}^{\mathcal{A}}_{\mathcal{VD}}(S) = \bigcup_{i=1}^{n} \mathcal{CT}_{\mathcal{VD}}(S_i, n/r)$.

Lemma 2.6. *Let S be a random permutation of $\hat{S}$. Then*

$$E\Big[\sum_{s \in \mathcal{T}^{\mathcal{A}}_{\mathcal{SP}}(S)} (w(s))^c\Big] = O\left(n^c r^{2-c}\right),$$

for $c = 0$ or $c = 1$.

Proof: Let p be a vertex of $\mathcal{A}(\hat{S})$. The expected contribution, of all the splitters that lie on the vertical line passing through p, to the above sum is at most

$$\begin{aligned} O & \left(\sum_{w=n/2r}^{\infty} \sum_{q=0}^{\infty} \frac{w^c}{(w+1)^2(q+w+3)^2} + \sum_{w=n/2r}^{\infty} \frac{w^c}{(w+1)^3} \right) \\ &= O\left(\sum_{w=n/2r}^{\infty} \sum_{q=0}^{\infty} \frac{w^{c-2}}{(q+w+3)^2} + \left(\frac{n}{r}\right)^{c-2} \right) = O\left(\sum_{w=n/2r}^{\infty} w^{c-3} + \left(\frac{n}{r}\right)^{c-2} \right) \\ &= O\left(\left(\frac{n}{r}\right)^{c-2} \right), \end{aligned}$$

for $c = 0, 1$.

Since there are $O(n^2)$ vertices in $\mathcal{A}(\hat{S})$, it follows that

$$E\left[\sum_{s \in \mathcal{T}^{\mathcal{A}}_{\mathcal{SP}}(S)} (w(s))^c\right] = O\left(n^c r^{2-c}\right).$$

■

Lemma 2.6 implies that the number of "heavy" splitters generated by `CutRandomInc`, with or without merging, is $O(r^2)$, and their total weight is $O(nr)$.

Lemma 2.7. *Let S be a random permutation of $\hat{S}$. Then*

$$W = E\left[\sum_{\Delta \in \mathcal{T}^{\mathcal{A}}_{\mathcal{VD}}(S)} (w(\Delta))^c\right] = O\left(n^c r^{2-c}\right),$$

for $c = 0, 1$.

Proof: The probability of a trapezoid Δ of weight w to be created during the computation of $\mathcal{CT}_{\mathcal{VD}}(S)$, if it is defined by $d \leq 4$ lines, is proportional to $1/w^d$. Let f^d_w denote the number of trapezoids of $\mathcal{T}_{\mathcal{VD}}(S)$ that are defined by d lines of S and have weight w. Let $F^d_{\leq w} = \sum_{q=0}^{w} f^d_q$. By the Clarkson-Shor probabilistic technique [CS89], we have $F^d_{\leq w} = O\left((n/w)^2 w^d\right) = O(n^2 w^{d-2})$. Let W_d denote the contribution to W made by trapezoids defined by d lines. Then

$$\begin{aligned}
W_d &= \sum_{w=n/r}^{n-d} \frac{f^d_w w^c}{w^d} = \sum_{w=n/r}^{n-d} f^d_w w^{c-d} \leq \sum_{w=n/r}^{n-d} (F^d_{\leq w} - F^d_{\leq w-1}) w^{c-d} \\
&\leq F^d_{\leq n} n^{c-d} + \sum_{w=n/r}^{n-d-1} F^d_{\leq w}(w^{c-d} - (w+1)^{c-d}) = O\left(n^c + \sum_{w=n/r}^{n-d-1} F^d_{\leq w} w^{c-d-1}\right) \\
&= O\left(n^c + \sum_{w=n/r}^{n-d-1} n^2 w^{d-2} w^{c-d-1}\right) = O\left(n^c + n^2 \sum_{w=n/r}^{\infty} w^{c-3}\right) \\
&= O\left(n^c + n^2 (n/r)^{c-2}\right) = O\left(n^c r^{2-c}\right).
\end{aligned}$$

Overall, $W = \sum_{d=1}^{4} W_i = O(n^c r^{2-c})$. ■

By Lemma 2.7, the expected number of trapezoids in $\mathcal{T}^{\mathcal{A}}_{\mathcal{VD}}(S)$ is $O(r^2)$, and their expected total weight is $O(nr)$.

Remark 2.8. Lemma 2.6 and Lemma 2.7 hold for any $0 \leq c < 2$, but we only need the results for $c = 0, 1$.

Let $\nabla\mathcal{VD}_i = \mathcal{CT}_{\mathcal{VD}}(S_i, n/r) \setminus \mathcal{CT}_{\mathcal{VD}}(S_{i-1}, n/r)$, $\nabla\mathcal{SP}_i = \mathcal{CT}_{\mathcal{SP}}(S_i, n/2r) \setminus \mathcal{CT}_{\mathcal{SP}}(S_{i-1}, n/2r)$, and let $\nabla\mathcal{C}_i$ be the set of *active* trapezoids in $\mathcal{C}_i \setminus \mathcal{C}_{i-1}$; namely, the new active trapezoids created in the ith iteration of the algorithm.

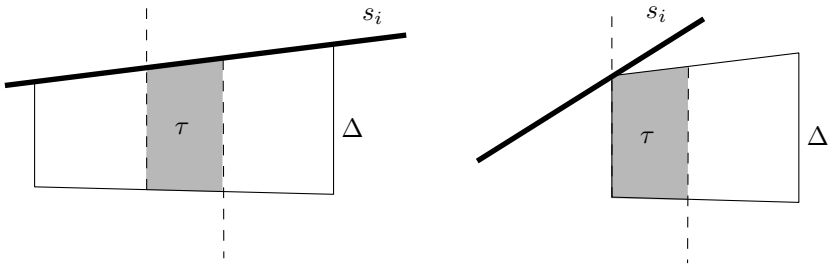

Figure 2.3: A new trapezoid $\tau \in \nabla\mathcal{C}_i$ must lie inside a new trapezoid $\Delta \in \nabla\mathcal{VD}_i$.

Lemma 2.9. *(a) Each new trapezoid $\tau \in \nabla\mathcal{C}_i$ must be contained in a new trapezoid $\Delta \in \nabla\mathcal{VD}_i$.*

(b) Let Δ be a $(1/r)$-active trapezoid in $\nabla\mathcal{VD}_i$, and let $\nabla\mathcal{C}_i(\Delta)$ (resp. $\nabla\mathcal{SP}_i(\Delta)$) denote the set of trapezoids in $\nabla\mathcal{C}_i$ (resp. splitters in $\nabla\mathcal{SP}_i$) that are contained in Δ. Then

$$\sum_{\tau\in\nabla\mathcal{C}_i(\Delta)} w(\tau) = O\left(w(\Delta) + \sum_{s\in\nabla\mathcal{SP}_i(\Delta)} w(s)\right),$$

and

$$\left|\nabla\mathcal{C}_i(\Delta)\right| = O\left(\frac{r}{n}w(\Delta) + \left|\nabla\mathcal{SP}_i(\Delta)\right|\right).$$

The lemma holds regardless of whether or not `CutRandomInc` *performs merging.*

Proof: (a) Let $\tau \in \nabla\mathcal{C}_i$ be an active trapezoid created in the ith iteration of `CutRandomInc`, and let Δ be the trapezoid of $\mathcal{CT}_{\mathcal{VD}}(S_i)$ that contains Δ. If the line s_i is the top or bottom line of τ then clearly Δ is also newly created. Otherwise s_i must delimit one of the vertical sides of τ. This side is also a side of Δ, so Δ is newly created. Hence $\Delta \in \nabla\mathcal{VD}_i$. See Figure 2.3.

(b) Let τ be a trapezoid in $\nabla\mathcal{C}_i(\Delta)$. We charge the weight of τ either to (a portion of) the weight of the (active) trapezoid of Δ, or to a new "heavy" splitter that bounds τ.

As noted in (a), at least one of the splitters of τ must be new (i.e., created in the ith iteration), and this remains true even if τ was created by merging a few active trapezoids of $\mathcal{C}_{i-1}$.

Consider the lines $l \in \hat{S}$ that cross τ. If at least half of these lines intersect the ceiling and/or floor of Δ, we charge $w(\tau)$ to those intersection points, whose number is at least $w(\tau)/2$. Since there are at most $2w(\Delta)$ such intersections on the boundary of Δ, it follows that the sum of the weights $w(\tau)$ of such trapezoids is at most $4w(\Delta)$.

So one can assume that at least half the lines that cross τ do not intersect either the floor or the ceiling of τ. This implies that the new splitter must intersects at least $w(\tau)/2 > n/(2r)$ of these lines, which implies that $s \in \nabla\mathcal{SP}_i$. Thus, we charge $w(\tau)$ to $w(s)$. Since $w(\tau) \le 2w(s)$, and each such splitter can be charged at most twice, the first inequality of (b) follows.

As for the second inequality, if a $(1/r)$-active trapezoid $\tau \in \nabla\mathcal{C}_i(\Delta)$ does not have a splitter of $\nabla\mathcal{SP}_i$ as one of its sides, then there at least n/r intersections between the lines crossing τ and the bottom and top edges of τ. The number of such trapezoids within Δ is at most $2w(\Delta)/(n/r) = O((r/n)w(\Delta))$. This is easily seen to imply the second inequality. ∎

Theorem 2.10. *The expected size of the cutting generated by* `CutRandomInc` *(with or without merging) is $O(r^2)$, and the expected running time is $O(nr)$.*

Proof: Since the direct work involved in creating the children of a trapezoid is proportional to the number of lines that cross it, we can bound the overall work performed by `CutRandomInc` by the total weight of the active trapezoids that it generates.

Of course, if we perform merging, there is also additional work associated with merging trapezoids. However, the merging stage can be performed in linear time in the total weight of the split trapezoids. This requires a somewhat careful but routine implementation, so that the running time remains linear even when merging the conflict lists of a potentially large number of trapezoids, during the creation of a single new (merged) trapezoid.

Thus, using Lemma 2.9 (a), the expected running time is proportional to

$$E\left[\sum_{i=1}^{n}\sum_{\tau\in\nabla\mathcal{C}_i} w(\tau)\right] = E\left[\sum_{i=1}^{n}\sum_{\Delta\in\nabla\mathcal{VD}_i}\sum_{\tau\in\nabla\mathcal{C}_i(\Delta)} w(\tau)\right].$$

By Lemma 2.9 (b), we have:

$$\begin{aligned} E\left[\sum_{i=1}^{n}\sum_{\tau\in\nabla\mathcal{C}_i} w(\tau)\right] &= E\left[\sum_{i=1}^{n}\sum_{\Delta\in\nabla\mathcal{VD}_i} O\left(w(\Delta)+\sum_{s\in\nabla\mathcal{SP}_i(\Delta)} w(s)\right)\right] \\ &= O\left(E\left[\sum_{\Delta\in\mathcal{T}^{\mathcal{A}}_{\mathcal{VD}}(S)} w(\Delta)+\sum_{s\in\mathcal{T}^{\mathcal{A}}_{\mathcal{SP}}(S)} w(s)\right]\right) = O(nr), \end{aligned}$$

by Lemma 2.6, and Lemma 2.7.

As for the expected size of the cutting,

$$\begin{aligned} E\left[\sum_{i=1}^{n}|\nabla\mathcal{C}_i|\right] &= E\left[\sum_{i=1}^{n}\sum_{\Delta\in\nabla\mathcal{VD}_i}|\nabla\mathcal{C}_i(\Delta)|\right] = O\left(E\left[\sum_{\Delta\in\mathcal{T}^{\mathcal{A}}_{\mathcal{VD}}(S)}\frac{r}{n}w(\Delta)+|\mathcal{T}^{\mathcal{A}}_{\mathcal{SP}}(S)|\right]\right) \\ &= O(r^2), \end{aligned}$$

by Lemma 2.9 (b). ∎

The algorithm `CutRandomInc` works also for planar arrangements of segments and x-monotone curves (such that the number of intersections of any pair of curves is bounded by a constant). This follows by a straightforward adaption of the proof to those cases, which we omit, and it is summarized in the following proposition.

Proposition 2.11. *Let $\hat{\Gamma}$ be a set of x-monotone curves such that each pair intersects in at most a constant number of points. Then the expected size of the $(1/r)$-cutting generated by* `CutRandomInc` *for Γ is $O(r^2)$, and the expected running time is $O(nr)$, for any integer $1 \le r \le n$, in an appropriate model of computation.*[2]

However, the arrangement of a set of n segments or curves might have subquadratic complexity (since the number of intersection points might be subquadratic). This raises the question of whether `CutRandomInc` generates smaller cuttings for such sparse arrangements.

Indeed, an algorithm of [dBS95] generates cuttings of size $O\left(r+\frac{r^2}{n^2}\kappa\right)$, in expected time $O\left(n\log r+\frac{r}{n}\kappa\right)$, where κ is the overall complexity of the arrangement. Using `CutRandomInc` for this case, we obtain the following slightly weaker bounds:

[2] real-RAM model so that the intersection of two curves, and the value of a curve at a certain x-coordinate can be computed in $O(1)$ time.

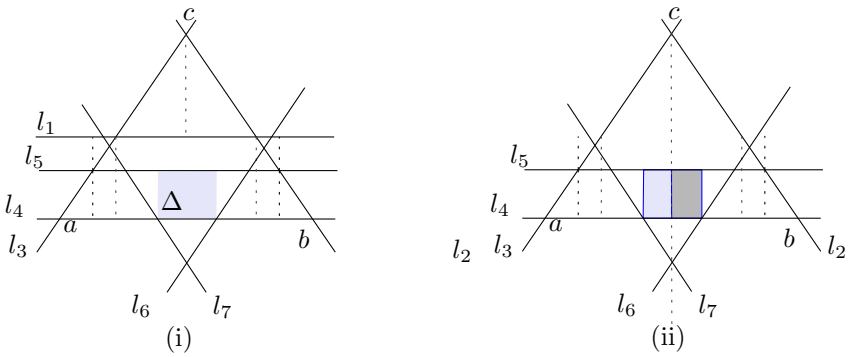

Figure 2.4: If merging is *not* used by `CutRandomInc`, an active trapezoid $\Delta \in \mathcal{C}_i$ might disappear if we skip an insertion of a line, which does not belong to the defining or crossing sets of Δ. Indeed, if `CutRandomInc` inserts the lines in the order $l_1, l_2, \ldots, l_7$, then the trapezoid Δ is created; see (i). However, if we skip the insertion of the line l_1, then the trapezoid Δ is not created, because the ray emanating downward from $l_2 \cap l_3$ intersects it. This implies that there is no 'locality' in the determination of which trapezoids arise in the execution of `CutRandomInc`, so the standard techniques of [CS89, dBDS95, HW87] can not be applied directly in analyzing `CutRandomInc`.

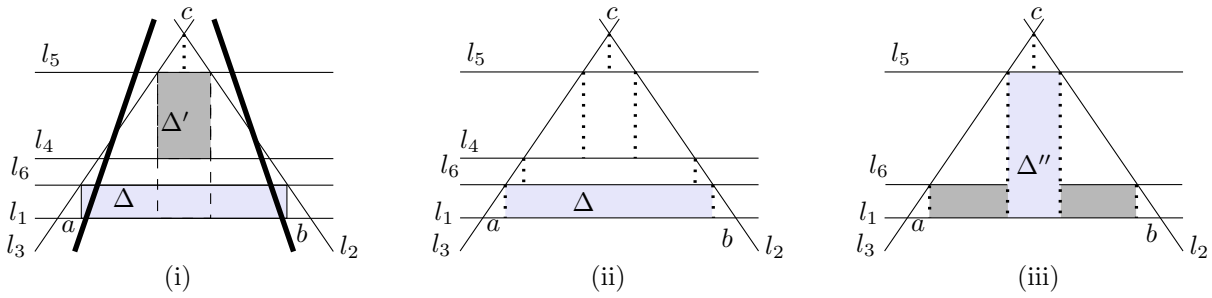

Figure 2.5: Even if merging is used by `CutRandomInc`, an active trapezoid $\Delta \in \mathcal{C}_i$ might disappear if we skip an insertion of a line which is completely unrelated to Δ. The thick lines represent two sets of 100 parallel lines, and we want to compute a (1/10)-cutting. We execute `CutRandomInc` with the first 6 lines $l_1, \ldots, l_6$ in this order. Note that any trapezoid that intersects a thick line is active. The first trapezoid Δ' inside $\triangle abc$ that becomes inactive is created when the line l_5 is being inserted; see parts (i) and (ii). However, if we skip the insertion of the line l_4 (as in part (iii)), the corresponding inactive trapezoid Δ'' will extend downwards and intersect Δ. Since Δ'' is inactive, the decomposition of the plane inside Δ'' is no longer maintained. In particular, this implies that the trapezoid Δ will not be created, since it is being blocked by Δ'', and no merging involving areas inside Δ'' will take place. Here too l_4 belongs neither to the defining nor to the crossing sets of Δ.

Proposition 2.12. *Let $\hat{\Gamma}$ be a set of n curves, such that each pair of curves of $\hat{\Gamma}$ intersect in at most a constant number of points. Then the expected size of the $(1/r)$-cutting generated by* `CutRandomInc`*, when applied to $\hat{\Gamma}$, is*

$$O\left(r \log r + \frac{r^2}{n^2}\kappa\right),$$

and the expected running time is $O(n \log^2 r + r\kappa/n)$, for any integer $1 \leq r \leq n$, where κ is the complexity of $\mathcal{A}(\hat{\Gamma})$.

Since `CutRandomInc` generates superficial splitters, the results for sparse arrangements are slightly worse (by a factor of $\log r$ if κ is relatively small) than those of [dBS95]. We omit any further details. The algorithm of [dBS95] is similar to the algorithm of Chazelle and Friedman [CF90], and we therefore believe that in practice `CutRandomInc` (with merging) will generate much smaller cuttings for $\mathcal{A}(\hat{\Gamma})$, as the results of Section 4 might suggest.

Remark 2.13. An interesting question is whether `CutRandomInc` can be extended to higher dimensions. If we execute `CutRandomInc` in higher dimensions, we need to use a more complicated technique in decomposing each of our "vertical trapezoids" whenever it intersects a newly inserted hyperplane. Chazelle and Friedman's algorithm uses bottom vertex triangulation for this decomposition. However, in our case, it is easy to verify that `CutRandomInc` might generate simplices so that the size of their defining set need not be bounded by a constant, if we use bottom vertex triangulation. This implies that the current analysis can not be extended to this case. We leave the problem of extending `CutRandomInc` to higher (say, three) dimensions as an open problem for further research.

2.1 Cuttings by Vertical Polygons

In this section, we present another variant of `CutRandomInc` that uses 'vertical polygons' instead of vertical trapezoids, and establish similar optimal performance bounds for this variant.

Definition 2.14. Let $\hat{S}$ be a set of n lines. A convex polygon P is a *μ-vertical polygon* of $\hat{S}$, if the boundary of P, except for the two vertical sides of P, if any, is contained in $\bigcup \hat{S}$, and the number of non-vertical edges of P is at most μ, where μ is a small positive constant (the case $\mu = 2$ is the case of vertical trapezoids). We denote the set of all such polygons by $\mathcal{VP}_\mu(\hat{S})$. A μ-vertical polygon P is a *μ-corridor*, if its two splitters (i.e. vertical sides) are defined by vertices of $\mathcal{A}(\hat{S})$ lying on the boundary of P. Note that the size of the defining set of a μ-corridor is $\leq \mu + 2$.

In the following, we consider μ to be a prescribed small constant.

We can use μ-vertical polygons instead of vertical trapezoids in `CutRandomInc`; namely, each region maintained by `CutRandomInc` is a μ-vertical polygon. Whenever a new region is being created, it is split into two subregions if it has more than μ non-vertical edges. This is done by erecting a splitter, from an appropriate vertex of the region. Let `CRIVPolygon` denote this variant of `CutRandomInc`. Here too we have the option of performing *merging*. That is, we merge a sequences long as possible, of adjacent μ-vertical polygons, within the same face of $\mathcal{A}(S_i)$, into a single vertical polygon, as long as the number of its non-vertical edges does not exceed μ.

The gain in using μ-vertical polygons instead of vertical trapezoids is that the number of splitters generated is much smaller, yielding smaller cuttings. See Section 4. The disadvantage is that the regions output by the algorithm are more complex to handle in any subsequent application of the cutting.

Lemma 2.15. *Let $\hat{S}$ be a set of n lines in the plane. The number of μ-corridors of $\mathcal{A}(\hat{S})$ is $O(n^2)$.*

Proof: A vertex p of $\mathcal{A}(\hat{S})$ is the left-bottom vertex of only a constant number ($O(\mu)$) of μ-corridors. The lemma follows since the number of such vertices is $O(n^2)$. ∎

Note that any μ-vertical polygon is contained in only a constant number of μ-corridors. Let S be a random permutation of the lines of $\hat{S}$, and let $\mathcal{CO}_i$ denote the set of active μ-corridors in $\mathcal{A}(S_i)$, for $i = 1, \ldots, n$. Let $\mathcal{T}^{\mathcal{A}}_{\mathcal{CO}} = \bigcup_{i=1}^{n} \mathcal{CO}_i$. Note that those corridors are not necessarily disjoint. Moreover, each active region maintained by `CRIVPolygon`, is contained inside at least one active corridor, and the set of splitters generated by `CRIVPolygon` is a subset of the set of splitters generated by `CutRandomInc`. Thus, Lemma 2.6 holds for the active splitters generated by `CRIVPolygon`. As for the corridors, we have:

Lemma 2.16. *Let S be a random permutation of $\hat{S}$, then*

$$W = E\left[\sum_{\tau \in \mathcal{T}^{\mathcal{A}}_{\mathcal{CO}}(S)} (w(\tau))^c\right] = O\left(n^c r^{2-c}\right),$$

for $c = 0, 1$.

Proof: We follow the proof of Lemma 2.7, using Lemma 2.15 to bound the number of "heavy" μ-corridors. This follows by observing that we can apply the Clarkson-Shor technique [CS89] to bound the number of μ-corridors with weight at most k. Indeed, arguing as in the proof of Lemma 2.7, the expected contribution to total weight of μ-vertical corridors having weight at least $n/(2r)$, created by the algorithm, and defined by d lines is $O(n^c r^{2-c})$, for $d = 1, \ldots, \mu + 4$. ∎

Theorem 2.17. *The expected size of the cutting generated by* `CRIVPolygon` *(with or without merging) is $O(r^2)$, and the expected running time is $O(nr)$.*

Proof: We only sketch the proof, since it is similar to the analysis of `CutRandomInc`. Note that in the proof of Lemma 2.9 can be adapted to handle active corridors, and the active regions maintained by `CRIVPolygon`. Indeed, after the ith iteration of the algorithm, each newly created active μ-vertical polygon is contained in (at least one) newly created active μ-corridor. We assign each such μ-vertical polygon to one of those newly active μ-corridors, in an arbitrary manner.

Now, to bound the work associated with such a μ-corridor X, we apply the same charging scheme used in Lemma 2.9, charging the weight of all the active μ-vertical polygons that were assigned to X (all of them are contained inside X), to the weight of X, and to the weight of the newly created "heavy" splitters inside X.

Since each splitter is contained in a constant number of μ-corridors, this implies that the overall charge made to on a newly created splitter s is $O(w(s))$.

Overall, this implies that the overall expected running time of the algorithm is bounded by the total weight of the active μ-corridors and the heavy splitters that it creates.

By Lemma 2.6 and Lemma 2.16, the expected total weight of the μ-vertical polygons generated by `CRIVPolygon` is $O(nr)$, which implies immediately that the expected running time is $O(nr)$, and the expected size of the cuttings is $O(r^2)$, by following the proof of Theorem 2.10. We omit the details. ∎

Remark 2.18. We can further modify the algorithm `CRIVPolygon`, so that it tries to remove inactive regions from the left and right sides of any newly created active region. We call this variant `PolyVertical`. It is easy to verify that the same proof of correctness, with slight modifications, works also for `PolyVertical`.

3 Generating Small Cuttings

In this section, we present an efficient algorithm that generates cuttings of guaranteed small size. The algorithm is based on Matoušek's construction of small cuttings [Mat98]. We first review this construction, and then show how to modify it for building small cuttings efficiently.

Definition 3.1 ([Mat98]). Let $\hat{S}$ be a set of n lines in the plane in general position, i.e., every pair of lines intersect in exactly one point, no three have a common point, no line is vertical or horizontal, and the x-coordinate of all intersections are pairwise distinct. The *level* of a point in the plane is the number of lines of $\hat{S}$ lying strictly below it. Consider the set E_k of all edges of the arrangement of $\hat{S}$ having level k (where $0 \leq k < n$). These edges form an x-monotone connected polygonal line, which is called the *level* k of the arrangement of $\hat{S}$.

Definition 3.2 ([Mat98]). Let E_k be the level k in the arrangement $\mathcal{A}(\hat{S})$ with edges $e_0, e_1, \ldots, e_t$ (from left to right), and let p_i be a point in the interior of the edge e_i, for $i = 0, \ldots, t$. The *q-simplification* of the level k, for an integer parameter $1 \leq q \leq t$, is defined as the x-monotone polygonal line containing the part of e_0 to the left of the point p_0, the segments p_0p_q, $p_qp_{2q}, \ldots, p_{\lfloor (t-1)/q \rfloor q}p_t$ and the part of e_t to the right of p_t. Let $\mathrm{simp}_q(E_k)$ denote this polygonal line.

Let $\hat{S}$ be a set of n lines in general position, and let $\mathcal{E}_{i,q}$ denote the union of the levels E_i, E_{i+q}, ..., E_{n+i-q}, for $i = 0, \ldots, q-1$. Let $\mathrm{simp}_q(\mathcal{E}_{i,q})$ denote the set of edges of the q-simplifications of the levels of $\mathcal{E}_{i,q}$.

Matoušek showed that the vertical decomposition of the plane induced by $\mathrm{simp}_q(\mathcal{E}_{i,q})$, where $q = n/(2r)$ (we assume that n is divisible by $2r$), is a $(1/r)$-cutting of the plane, for any $i = 0, \ldots, q-1$. Moreover, the following holds:

Theorem 3.3 ([Mat98]). *Let $\hat{S}$ be a set of n lines in general position, let r be a positive integer, and let $q = n/(2r)$. Then the subdivision of the plane defined by the vertical decomposition of $\mathrm{simp}_q(\mathcal{E}_{m,q})$ is a $(1/r)$-cutting of $\mathcal{A}(\hat{S})$, where $0 \leq m < q$ is the index $i \in \{0, \ldots, q-1\}$ for which $|\mathcal{E}_{i,q}|$ is minimized. Moreover, the cutting generated has at most $8r^2 + 6r + 4$ trapezoids.*

Remark 3.4. (i) Matoušek's construction can be slightly improved, by noting that the leftmost and rightmost points in a q-simplification of a level can be placed at "infinity"; that is, we replace the first and second edges in the q-simplification by a ray emanating from p_q which is parallel to e_0. We perform a similar shortcut for the two last edges of the simplified level. We denote this improved simplification by simp'_q. It is easy to prove that using this improved simplification also results in a $(1/r)$-cutting of $\mathcal{A}(\hat{S})$ with at most $8r^2 + 6r + 4 - 4 \cdot 2r = 8r^2 - 2r + 4$ vertical trapezoids.

(ii) Inspecting Matoušek's construction, we see that if we can only find an i such that $|\mathcal{E}_{i,q}| \leq cn^2/q$, where $c > 1$ is a prescribed constant, then the vertical decomposition induced by $\mathrm{simp}'_q(\mathcal{E}_{i,q})$ is a $(1/r)$-cutting having $\leq c(8r^2 - 2r + 4)$ trapezoids.

Let $n_i = |\mathcal{E}_{i,q}|$, for $i = 0, \ldots, q-1$. Matoušek's construction is carried out by computing the numbers $n_0, \ldots, n_q$, and picking the minimal number n_i, which is guaranteed to be no larger than the average n^2/q. Unfortunately, implementing this scheme explicitly requires computing the whole arrangement $\mathcal{A}(\hat{S})$, so the resulting running time is $O(n^2)$. Let us assume for the moment that one can compute any of the numbers n_i quickly. Then, as the following lemma shows, one can compute a number n_i which is $\leq (1+\varepsilon)n^2/q$, without computing all the n_i's.

Lemma 3.5. *Let $n_0, \ldots, n_{q-1}$ be q positive integers, whose sum $m = \sum_{i=0}^{q-1} n_i$ is known in advance, and let $\varepsilon > 0$ be a prescribed constant. One can compute an index $0 \leq k < q$, such that $n_k \leq \lceil (1+\varepsilon)m/q \rceil$, by repeatedly picking uniformly and independently a random index $0 \leq i < q$, and by checking whether $n_i \leq \lceil (1+\varepsilon)m/q \rceil$. The expected number of iterations required is $\leq 1 + 1/\varepsilon$.*

Proof: Let Y_i be the random variable which is the value of n_i picked in the ith iteration. Using Markov's inequality[3], one obtain:

$$Pr\left[Y_i \geq (1+\varepsilon)\frac{m}{q}\right] \leq \frac{E\left[Y_i\right]}{(1+\varepsilon)\frac{m}{q}}.$$

Since, $E[Y_i] = m/q$, we have that the probability for failure in the ith iteration is

$$Pr\left[Y_i \geq (1+\varepsilon)\frac{m}{q}\right] \leq \frac{1}{1+\varepsilon}.$$

Let X denote the number of iterations required by the algorithm. Then $E[X]$ is bounded by the expected number of trials to the first success in a geometric distribution with probability $p \geq 1 - \frac{1}{1+\varepsilon}$. Thus, the expected number of iterations is bounded by

$$E[X] \leq \frac{1}{p} \leq \frac{1}{1 - \frac{1}{1+\varepsilon}} = 1 + \frac{1}{\varepsilon}.$$

■

To apply Lemma 3.5 in our setting, we need to supply an efficient algorithm for computing the level of an arrangement of lines in the plane.

Lemma 3.6. *Let $\hat{S}$ be a set of n lines in the plane. Then one can compute, in $O((n+h)\log^2 n)$ time, the level k of $\mathcal{A}(\hat{S})$, where $h = |E_k|$ is the complexity of the level.*

Proof: The technique presented here is well known (see [BDH99] for a recent example); we include it for the sake of completeness of exposition. Let $e_0, \ldots, e_t$ be the edges of the level k from left to right (where e_0, e_t are rays).

Let e be an edge of the level k. Let f be the face of $\mathcal{A}(\hat{S})$ having e on its boundary and lying above e. In particular, all the edges on the bottom part of ∂f belong to the level k.

Let $f_1, \ldots, f_r$ be the faces of $\mathcal{A}(\hat{S})$ having the level k as their "floor", from left to right. The ray e_0 can be computed in $O(n)$ time since it lies on line l_k of $\hat{S}$, with the kth largest slope. Moreover, by intersecting l_k with the other lines of $\hat{S}$, one can compute e_0 in linear time.

Any face of $\mathcal{A}(\hat{S})$ is uniquely defined as an intersection of half-planes induced by the lines of $\hat{S}$. For the face f_1, we can compute the half-planes and their intersection that represents f_1, in $O(n \log n)$ time; see [dBCKO08]. To carry out the computation of the bottom parts of $f_2, \ldots, f_r$, one can dynamically maintain the intersection representing f_i as we traverse the level k from left to right. To do so, we will use the data-structure of Overmars and Van Leeuwen [OvL81] that maintains such an intersection, with $O(\log^2 n)$ time for each update operation. As we move from f_i to f_{i+1} through a vertex v, we have to "flip" the two half-planes associated with the two lines passing through v, at a cost of $O(\log^2 n)$ time per vertex. Similarly, if we are given an edge e on the boundary of f_i we can compute the next edge in $O(\log^2 n)$ time.

Thus, we can compute the level k of $\mathcal{A}(\hat{S})$ in $O((n+h)\log^2 n)$ time. ■

[3]The inequality asserts that $Pr\left[Y \geq t\right] \leq \frac{E[Y]}{t}$, for a random variable Y that assumes only nonnegative values.

Combining Lemma 3.5 and Lemma 3.6, we have:

Theorem 3.7. *Let $\hat{S}$ be a set of n lines in the plane, and let $0 < \varepsilon \leq 1$ be a prescribed constant. Then one can compute a $(1/r)$-cutting of $\mathcal{A}(\hat{S})$, having at most $(1+\varepsilon)(8r^2 - 2r + 4)$ trapezoids. The expected running time of the algorithm is $O\left(\left(1+\frac{1}{\varepsilon}\right) nr \log^2 n\right)$.*

Proof: By the above discussion, it is enough to find an index $0 \leq i \leq q-1$, such that $|\mathcal{E}_{i,q}| \leq M = (1+\varepsilon)\frac{n^2}{q} \leq 2(1+\varepsilon)nr$, where $q = \lceil n/(2r) \rceil$. By Remark 3.4 (ii), the vertical decomposition of $\mathrm{simp}'_q(\mathcal{E}_{i,q})$, is a $(1/r)$-cutting of the required size.

Picking i randomly, we have to check whether $|\mathcal{E}_{i,q}| \leq M$. We can compute $\mathcal{E}_{i,q}$, by computing the levels $E_i, E_{i+q}, \ldots, E_{i+\lfloor (n-i-1)/q \rfloor q}$ in an output-sensitive manner, using Lemma 3.6. Note that if $|\mathcal{E}_{i,q}| > M$, we can abort as soon as the number of edges we computed exceeds M. Thus, checking if $|\mathcal{E}_{i,q}| \leq M$ takes $O(nr \log^2 n)$ time. By Lemma 3.6, the expected number of iterations the algorithm performs until the inequality $|\mathcal{E}_{i,q}| \leq M$ will be satisfied is $\leq 1 + 1/\varepsilon$. Thus, the expected running time of the algorithm is

$$O\left(\left(1+\frac{1}{\varepsilon}\right) nr \log^2 n\right),$$

since the vertical decomposition of $\mathrm{simp}'_q(\mathcal{E}_{i,q})$ (which is the resulting cutting) can be computed in additional $O(nr)$ time. In fact, one can also compute, in $O(nr)$ time, for each trapezoid in the cutting, the lines of $\hat{S}$ that intersect it. ∎

4 Empirical Results

In this section, we present the empirical results we got for computing cuttings in the plane using `CutRandomInc` and various related heuristics that we have implemented and experimenting with. A program with a GUI demonstrating the algorithms and heuristics presented in the paper, is available on the web in source form [Har98].

4.1 The Implemented Algorithms — Using Vertical Trapezoids

We have implemented the algorithm `CutRandomInc` presented in Section 2, as well as several other algorithms for constructing cuttings. In this section, we report on the experimental results that we obtained.

Most of the algorithms we have implemented are variants of `CutRandomInc`. The algorithms implemented are the following (we denote by $K(\Delta)$ the set of lines that cross a vertical trapezoid Δ):

Classical: This is a variant of the algorithm of Chazelle and Friedman [CF90] for constructing a cutting. We pick a sample $R \subseteq \hat{S}$ of r lines, and compute its arrangement $A = \mathcal{A}_{\mathcal{VD}}(R)$. For each active trapezoid $\Delta \in A$, we pick a random sample $R_\Delta \subseteq K(\Delta)$ of size $6k \log k$, where $k = \lceil r|K(\Delta)|/n \rceil$, and compute the arrangement of $\mathcal{A}_{\mathcal{VD}}(R_\Delta)$ inside Δ. If $\mathcal{A}_{\mathcal{VD}}(R_\Delta)$ is not a $(1/r)$-cutting, then the classical algorithm performs resampling inside Δ until it reaches a cutting. Our implementation is more naive, and it simply continues recursively into the active subtrapezoids of $\mathcal{A}_{\mathcal{VD}}(R_\Delta)$.

Cut Randomized Incremental: This is `CutRandomInc` without merging, as described in Figure 2.1.

Randomized Incremental: This is `CutRandomInc` with merging.

The following four heuristics, for which we currently do not have a proof of any concrete bound on the expected size of the cutting that they generate, also perform well in practice.

Parallel Incremental: Let $\mathcal{C}_i$ be the covering generated in the ith iteration of the algorithm. For each active trapezoid $\Delta \in \mathcal{C}_i$, pick a random line from $K(\Delta)$, and insert it into Δ (i.e., split Δ accordingly). Continue until there are no active trapezoids. Note that unlike `CutRandomInc` the insertion operations are performed locally inside each trapezoid, and the line chosen for insertion in each trapezoid is independent of the lines chosen for other trapezoids.

Greedy Trapezoid: This is a variant of `CutRandomInc` where we try to be "smarter" about the line inserted into the partition in each iteration. Let V_i be the set of trapezoids of $\mathcal{C}_i$ with maximal weight. We pick randomly a trapezoid Δ out of the trapezoids of V_i, and pick randomly a line s from $K(\Delta)$. We then insert s into $\mathcal{C}_i$.

Greedy Line: Similar to **Greedy Trapezoid**, but here we compute the set $\mathcal{U}$ of lines of $\hat{S}$, for which $w'(s)$ is maximal, where $w'(s)$ is the number of active trapezoids in $\mathcal{C}_i$ that intersect the line s. We pick randomly a line from $\mathcal{U}$ and insert it into the current partition of the plane.

Greedy Weighted Line: Similar to **Greedy Line**, but our weight function is:

$$w'(s) = \sum_{\Delta \in \mathcal{C}_i, s \cap \Delta \neq \emptyset, w(\Delta) > n/r} \left\lfloor \frac{w(\Delta)}{\lfloor \frac{n}{3r} \rfloor} \right\rfloor ;$$

namely, we give a higher priority to lines that intersect heavier $(1/r)$-active trapezoids.

4.2 Polygonal Cuttings

In judging the quality of cuttings, the size of the cutting is of major concern. However, other factors might also be important. For example we want the regions defining the cutting to be as simple as possible. Furthermore, there are applications where we are not interested directly in the size of the cutting, but rather in the overall number of vertices defining the cutting regions. This is useful when applying cuttings in the dual plane, and transforming the vertices of the cutting back to the primal plane, as done in the computation of partition trees [Mat92]. A natural question is the following: Can one compute better cuttings, if one is willing to use cutting regions which are different from vertical trapezoids?

For example, if one is willing to cut using non-convex regions having a non-constant description complexity, the size of the cutting can be improved to $4r^2 + 2r + 2$ [Mat98]. On the other hand, if one wishes to cut a collection of lines by triangles, instead of trapezoids, the situation becomes somewhat disappointing, because the smallest cuttings currently known for this case, are generated by taking the cutting of Remark 3.4, and by splitting each trapezoid into two triangles. This results in cuttings having (roughly) $16r^2$ triangles.

In this section, we present a slightly different approach for computing cuttings, suggested to us by Jiří Matoušek, that works extremely well in practice. The new approach, a variant of which has already been presented in Section 2.1, is based on cuttings using polygonal convex regions with a small number

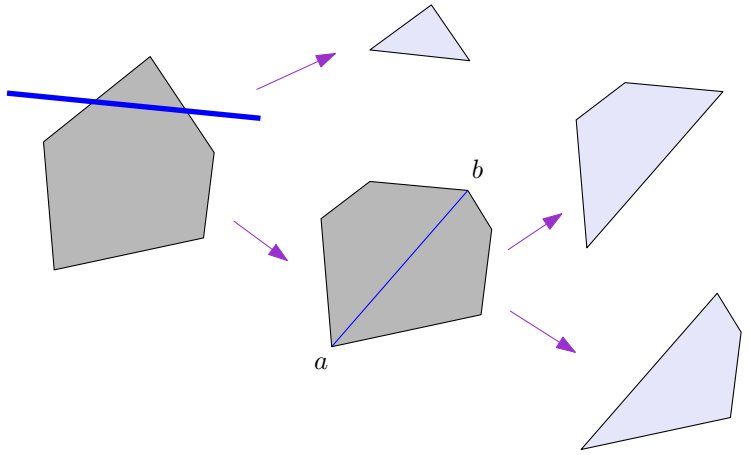

Figure 4.1: In the `PolyTree` algorithm, each time a polygon is being split by a line, we might have to further split it because a split region might have too many vertices.

of sides, instead of vertical trapezoids. Namely, we apply `CutRandomInc`, where each region is a convex polygon (of constant complexity). Whenever we insert a new line into an active region, we split the polygon into two new polygons. Of course, it might be that the number of vertices of a new polygon is too large. If so, we split each such polygon into two subpolygons ensuring that the number of vertices of the new polygons are below our threshold.

Intuitively, the benefit in this approach is that the number of superfluous entities (i.e. vertical walls in the case of vertical trapezoids) participating in the definition of the cutting regions is much smaller. Moreover, since the cutting regions are less restrictive, the algorithm can be more flexible in its maintenance of the active regions.

Here are the different methods we tried:

PolyTree: We use `CutRandomInc` where each region is a convex polygon having at most k-sides. When inserting a new line, we first split each of the active regions that intersect it into two subpolygons. If a split region R has more than k sides, we further split it using the diagonal of R that achieves the best balanced partition of R; namely, it is the pair of vertices a, b realizing the following minimum:

$$\min_{a,b \in V(R)} \max\left(w(R \cap H_{ab}^{+}), w(R \cap H_{ab}^{-})\right),$$

where $V(R)$ is the set of vertices of R, and $w(R)$ is the number of lines intersecting R, and H_{ab}^{+} (resp. H_{ab}^{-}) is the closed halfplane lying to the right (resp. left) of ab. See Figure 4.1.

PolyTriangle: Modified `PolyTree` for generating cuttings by triangles. In each stage, we check whether a newly created region R can be triangulated into a set of inactive triangles. To do so, compute an arbitrary triangulation of the region R and check if all the triangles generated in our (arbitrary) triangulation of R are inactive. If so, replace R in our cuttings by its triangulation.[4]

PolyDeadLeaf: Modified `PolyTree` for generating cuttings by triangles. Whenever a region is being created we check whether it has a leaf triangle (a triangle defined by three consecutive vertices of the

[4]Computing the "best" triangulation (i.e. the weight of the heaviest triangle is minimized) is relatively complicated, and requires dynamic programming. It is not clear that it is going to perform better than `PolyDeadLeaf`, described below.

region) that is inactive. If we find such an inactive triangle, we add it immediately to the final cutting. We repeat this process until the region can not be further shrunk.

PolyVertical: (This is the variant presented in Section 2.1.) We use `PolyTree`, but instead of splitting along a diagonal, we split along a vertical ray emanating from one of the vertices of the region. The algorithm also tries to remove dead regions from the left and right side of the region. Intuitively, each region is now an extended vertical trapezoid having a convex ceiling and floor, with at most two additional vertical walls.

Remark 4.1. Note, that for all the polygonal cutting methods, except `CRIVPolygon` (this is `PolyVertical` without the removal of dead regions, see Section 2.1) and `PolyVertical`, it is not even clear that the number of regions they maintain, in the ith iteration, is $O(i^2)$. Thus, the proof of Theorem 2.10 does not work for those methods.

4.3 Implementation Details

As an underlying data-structure for our testing, we implemented the history-graph data-structure [Sei91]. Our random arrangements were constructed by choosing n points uniformly and independently on the left side of the unit square, and similarly on the right side of the unit square. We sorted the points, and connected them by lines in a transposed manner. This yields a random arrangement with all the $\binom{n}{2}$ intersections inside the unit square.

We had implemented our algorithm in `C++`. We had encountered problems with floating point robustness at an early stage of the implementation, and decided to use exact arithmetic instead, using `LEDA` rational numbers [MN95]. While this solved the robustness problems, we had to deal with a few other issues:

- Speed: Using exact arithmetic instead of floating point arithmetic resulted in a slowdown by a factor 20–40. The time to perform an operation in exact arithmetic is proportional to the bit-sizes of the numbers involved. To minimize the size of the numbers used in the computations, we normalized the line equations so that the coefficients are integer numbers (in reduced form). Using more advanced techniques one can get close to floating-point speed with the "security" provided by exact-arithmetic. See [AHH+99].

- Memory consumption: A `LEDA` rational is represented by a block of memory dynamically allocated on the heap. In order to save, both in the memory consumed and the time used by the dynamic memory allocator, we observe that in a representation of vertical decomposition the same number appears in several places (i.e., an x-coordinate of an intersection point appears in 6 different vertical trapezoids). We reduce memory consumption, by storing such a number only once. To do so, we use a repository of rational numbers generated so far by the algorithm. Whenever we compute a new x-coordinate, we search it in the repository, and if it does not exist, then we insert it. In particular, each vertical trapezoid is represented by two pointers to its x_{left} and x_{right} coordinates, and pointers to its top and bottom lines.

  The repository is implemented using Treaps [SA96].

4.4 Handling Degeneracies

Geometric degeneracy is one of the main obstacles when implementing geometric algorithms. To overcome this problems, we used exact arithmetic. While this ensured that the underlining geometric prim-

itives works correctly, it does not tackle the problem of geometric degeneracies directly. Fortunately, in our case the handling of geometric degeneracies is straightforward.

Indeed, the various algorithms we presented for computing cuttings use only two geometric primitives. The first is *split*, which split a region (i.e., vertical trapezoid/convex polygon) Δ by a line l that crosses it (the line might be an input line, or a line connecting two vertices of the region) into a constant number of regions $R_1, \ldots, R_k$. Here, to overcome degeneracy the algorithm throws away regions having empty interior. If the regions are polygons, an additional step is added to remove superfluous collinear vertices (i.e., a vertex lying on the line induced by its two neighbors) from the newly created regions.

The second primitive, checks whether a line l crosses the interior of a region Δ. This is done by checking for each vertex of Δ, on which side of l it lies. If two vertices of Δ lie on opposite sides of l, then l crosses the interior of Δ. Since the program uses exact arithmetic this primitive gives the correct results.

4.5 Results — Vertical Trapezoids

The empirical results we got for the algorithms/heuristics of Section 4.1, are depicted in Table 5.1–Table 5.5.

For each value of r, and each value of n, we computed a random arrangement of lines inside the unit square, as described above. For each such arrangement, we performed 10 tests for each algorithm/heuristic. The tables present the size of the minimal cutting computed in those tests. Each entry is the size of the output cutting divided by r^2. In addition, each table caption presents a range containing the size of the cutting that can be obtained by Matoušek's algorithm [Mat98].

It is an interesting question whether using merging by `CutRandomInc` results in practice in smaller cuttings. We tested this empirically, and the results are presented in Table 5.6. As can be seen in Table 5.6, using merging does generate smaller cuttings, but the improvement in the cutting size is rather small. The difference in the size of the generated cuttings seems to be less than $2r^2$.

4.6 Implementing Matoušek's Construction

In Table 5.7, we present the empirical results for Matoušek's construction, comparing it with the slight improvement described in Remark 3.4. For small values of r the improved version yields considerably smaller cuttings than Matoušek's construction, making it the best method we are aware of for constructing small cuttings.

We had implemented Matoušek's algorithm naively, using quadratic space and time. Currently, this implementation can not be used for larger inputs because it runs out of memory. Implementing the more efficient algorithm described in Theorem 3.7 is non-trivial since it requires the implementation of the rather complex data-structure of Overmars and van Leeuwen [OvL81]. However, if it is critical to reduce the size of a cutting for large inputs, the algorithm of Theorem 3.7 seems to be the best available option.

Overall, Matoušek's algorithms gives the best results for cuttings by vertical trapezoids. However, for polygonal cuttings, the results we got are even better, as described below.

4.7 Results — Polygonal Cuttings

The results for polygonal cuttings are presented in Table 5.8. As seen in the tables the polygonal cutting methods perform well in practice. In particular, the `PolyTree` method generated cuttings of average size (roughly) $7.5r^2$, beating all the cutting methods that use vertical trapezoids.

As for triangles, the situation is even better: `PolyDeadLeaf` generates cuttings by triangles of size $\leq 12r^2$. (That is better by an additive factor of about $4r^2$ than the best theoretical bound).

To summarize, polygonal cutting methods seems to be the clear winner in practice. They generate cuttings of a small size, with a small number of vertices, and small number of triangles.

5 Conclusions

In this paper, we have presented a new approach, different from that of [CF90], for constructing cuttings in the plane. The new algorithm is rather simple and easy to implement. We have proved the correctness and bounded the expected output size and expected running time of several variants of the new algorithm, and have also demonstrated that the new algorithms perform much better in practice than the algorithm of [CF90]. We believe that the results in this paper show that planar cuttings are practical, and might be useful in practice when constructing data-structures for range searching and related applications.

Moreover, the empirical results show that the size of the cutting constructed by the new algorithms is not considerably larger (and in some cases even smaller) than the cuttings that can be computed by the currently best theoretical algorithm (too slow to be useful in practice due to its $O(n^2)$ running time) of Matoušek [Mat98]. The empirical constants that we obtain are generally between 10 and 13 (for vertical trapezoids). For polygonal cuttings we get a constant of 8 by cutting by convex polygons (using `PolyTree`) having at most 6 vertices. Moreover, the various variants of `CutRandomInc` seem to produce constants that are rather close to each other. As noted above, the method described in Remark 3.4 generates the smallest cuttings by vertical trapezoids.

As for running time, the results we got in practice are the following: In computing a (1/8)-cutting of 1024 lines, the fastest algorithm was `PolyTree`, requiring about half a minute in average. The other polygonal methods lagged slightly behind. `CutRandomInc` was the fastest algorithm that produced cuttings by vertical trapezoids, being several times slower. Matoušek's method required several hours due to our naive (i.e. $O(n^2)$) implementation. This information should be taken with reservation, since no serious effort had gone into optimizing the code for speed, and those measurements tend to change from execution to execution. (Recall also that we use exact arithmetic, which slows down the running time significantly.)

Given these results, we recommend for use in practice one of the polygonal-cutting methods. They perform well in practice, and they should be used whenever possible. If we are restricted to vertical trapezoid, `CutRandomInc` seems like a reasonable algorithm to be used in practice (without merging, as merging is the only "non-trivial" part in the implementation of the algorithm).

There are several interesting open problems for further research:

- Can one obtain provable bounds on the expected size of the cutting generated by, and the running time of the `PolyTree` method (Remember, that in the `PolyTree` method the cutting regions are convex polygons with a constant number of polygons)? The same question for all the other methods we had implemented?
- Can one prove the existence of a cutting smaller than the one guaranteed by the algorithm in Remark 3.4 for specific values of r? For example, Table 5.1 suggests a smaller cutting should exist for $r = 2$. In particular, the test results hint that a cutting made out of 32 vertical trapezoid should exist, while the cutting size guaranteed by Matoušek's algorithm [Mat98] is 48.
- Can one generate smaller cuttings by modifying `CutRandomInc` to be smarter in its decision when to merge trapezoids?

- Is there a simple and practical algorithm for computing cuttings in three and higher dimensions? The current algorithms seems to be far from practical.
- Can the following heuristic improve (substantially) the size of the cuttings in practice? After the cuttings was computed, pass over the cutting regions and merge adjacent regions that are adjacent and compatible, so that the resulting regions are still $(1/r)$-inactive. Inspecting the output of our test program in Figure 1.1 indicates that this indeed the case.

Acknowledgments

The author wishes to thank Pankaj Agarwal, Boris Aronov, Hervé Brönnimann, Bernard Chazelle, Jiří Matoušek, and Joe Mitchell for helpful discussions and suggestions concerning the problems studied in this paper and related problems.

The author also wishes to thank deeply Micha Sharir for his help and guidance in preparing the paper. He contributed several key ideas to the new simplified analysis of `CutRandomInc`.

References

[Aga91] P. K. Agarwal. Geometric partitioning and its applications. In J. E. Goodman, R. Pollack, and W. Steiger, editors, *Computational Geometry: Papers from the DIMACS Special Year*, pages 1–37. Amer. Math. Soc., 1991.

[AHH+99] Y. Aharoni, D. Halperin, I. Hanniel, S. Har-Peled, and C. Linhart. On-line zone construction in arrangements of lines in the plane. In *Proc. 3rd Workshop Alg. Eng.* (WAE), pages 139–153, 1999.

[AM95] P. K. Agarwal and J. Matoušek. Dynamic half-space range reporting and its applications. *Algorithmica*, 13:325–345, 1995.

[BDH99] K.-F. Böhringer, B. Donald, and D. Halperin. The area bisectors of a polygon and force equilibria in programmable vector fields. *Discrete Comput. Geom.*, 22(2):269–285, 1999.

[CF90] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990.

[Cha93] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9(2):145–158, 1993.

[Cla87] K. L. Clarkson. New applications of random sampling in computational geometry. *Discrete Comput. Geom.*, 2:195–222, 1987.

[CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[dBCKO08] M. de Berg, O. Cheong, M. van Kreveld, and M. H. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Santa Clara, CA, USA, 3rd edition, 2008.

[dBDS95] M. de Berg, K. Dobrindt, and O. Schwarzkopf. On lazy randomized incremental construction. *Discrete Comput. Geom.*, 14:261–286, 1995.

[dBS95] M. de Berg and O. Schwarzkopf. Cuttings and applications. *Int. J. Comput. Geom. Appl.*, 5:343–355, 1995.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Heidelberg, 1987.

[Har98] S. Har-Peled. Cuttings — demo program. `http://www.math.tau.ac.il/~sariel/CG/cutting/cuttings.html`, 1998.

[HW87] D. Haussler and E. Welzl. ε-nets and simplex range queries. *Discrete Comput. Geom.*, 2:127–151, 1987.

[Mat92] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8:315–334, 1992.

[Mat98] J. Matoušek. On constants for cuttings in the plane. *Discrete Comput. Geom.*, 20:427–448, 1998.

[MN95] K. Mehlhorn and S. Näher. LEDA: a platform for combinatorial and geometric computing. *Commun. ACM*, 38:96–102, 1995.

[OvL81] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981.

[SA95] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, New York, 1995.

[SA96] R. Seidel and C. R. Aragon. Randomized search trees. *Algorithmica*, 16:464–497, 1996.

[Sei91] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1:51–64, 1991.

Lines	Parallel Inc	Classical	Randomized Inc	Greedy Trapezoid	Greedy Line	Greedy Weighted Line
4	1.50	1.50	1.50	1.50	1.50	1.50
8	2.00	12.50	2.75	2.25	2.25	2.50
16	4.75	25.75	4.25	4.75	4.00	4.50
32	4.75	24.75	7.00	5.50	6.00	6.25
64	6.75	28.25	6.50	7.50	7.50	7.25
128	8.75	26.75	7.75	7.25	8.25	8.50
256	8.75	30.75	6.50	8.00	6.75	7.25
512	6.00	36.50	8.25	9.75	8.50	8.00
1024	9.25	26.00	10.00	7.75	7.75	9.00
2048	7.50	28.50	9.00	7.25	8.75	9.75
4096	7.50	35.25	8.50	8.25	7.75	7.50
8192	8.25	36.75	7.00	7.75	6.25	7.50
16384	9.00	30.25	8.75	8.00	8.50	8.75
32768	10.25	33.00	7.75	9.00	6.25	7.75
65536	10.00	31.25	6.50	6.75	6.75	8.50

Table 5.1: Results for 1/2-cuttings. Each entry is the size of the minimal cutting computed, divided by $2^2 = 4$. The corresponding value in Matoušek's construction [Mat98] is smaller than 12.00

Lines	Parallel Inc	Classical	Randomized Inc	Greedy Trapezoid	Greedy Line	Greedy Weighted Line
8	1.88	2.69	1.56	2.00	1.62	1.62
16	3.69	20.31	3.44	3.62	3.50	3.62
32	6.12	31.12	6.56	5.50	5.75	5.50
64	8.25	35.94	8.06	7.69	7.12	8.06
128	9.19	37.12	9.75	9.94	8.62	7.88
256	11.00	44.06	9.62	9.00	10.25	8.19
512	11.75	48.75	9.81	10.31	9.75	10.12
1024	12.75	46.88	12.12	11.25	10.25	10.62
2048	11.19	37.00	11.50	11.00	10.81	10.50
4096	11.81	44.38	10.94	11.19	10.62	10.50
8192	12.19	52.00	10.19	11.06	10.25	10.00
16384	12.31	43.88	10.94	11.25	10.31	10.88
32768	11.50	42.69	10.69	11.81	11.31	10.25

Table 5.2: Results for 1/4-cuttings. Each entry is the size of the minimal cutting computed, divided by $4^2 = 16$. The corresponding value in Matoušek's construction [Mat98] is smaller than 9.75

Lines	Parallel Inc	Classical	Randomized Inc	Greedy Trapezoid	Greedy Line	Greedy Weighted Line
16	2.23	4.08	2.06	1.89	1.86	1.83
32	4.80	23.17	4.19	3.92	3.91	3.53
64	7.67	37.25	6.64	6.27	6.31	6.12
128	10.44	45.50	8.84	8.83	8.80	8.31
256	11.56	44.12	9.91	10.53	9.91	9.94
512	12.77	50.14	11.36	11.11	10.86	11.20
1024	13.31	47.58	11.61	11.33	10.95	11.31
2048	13.42	54.84	12.36	11.17	12.38	11.17
4096	15.00	53.17	12.08	12.22	12.08	11.98
8192	13.98	51.75	11.73	12.19	12.67	12.27

Table 5.3: Results for 1/8-cuttings. Each entry is the size of the minimal cutting computed, divided by $8^2 = 64$. The corresponding value in Matoušek's construction [Mat98] is smaller than 8.81

Lines	Parallel Inc	Classical	Randomized Inc	Greedy Trapezoid	Greedy Line	Greedy Weighted Line
32	2.70	5.54	2.18	2.09	2.04	2.11
64	5.38	24.27	4.52	4.21	4.36	4.20
128	8.16	44.00	7.16	7.00	6.66	6.55
256	10.88	56.30	9.34	9.25	9.16	8.48
512	12.61	66.60	11.26	10.85	10.37	10.18
1024	14.02	66.64	12.30	11.40	11.23	11.10
2048	14.24	67.25	12.51	11.84	11.98	11.78

Table 5.4: Results for 1/16-cuttings. Each entry is the size of the minimal cutting computed, divided by $16^2 = 256$. The corresponding value in Matoušek's construction [Mat98] is smaller than 8.39

Lines	Parallel Inc	Classical	Randomized Inc	Greedy Trapezoid	Greedy Line	Greedy Weighted Line
64	2.84	5.45	2.26	2.19	2.14	2.14
128	5.48	24.73	4.54	4.44	4.33	4.22
256	8.89	52.62	7.52	7.05	6.68	6.61
512	11.26	67.72	9.48	9.54	8.98	8.87
1024	13.25	74.47	11.63	10.86	10.23	10.34

Table 5.5: Results for 1/32-cuttings. Each entry is the size of the minimal cutting computed, divided by $32^2 = 1024$. The corresponding value in Matoušek's construction [Mat98] is smaller than 8.19

	Value of r									
Number	2		4		8		16		32	
of Lines		Merge		Merge		Merge		Merge		Merge
4	1.50	1.50	—	—	—	—	—	—	—	—
8	2.25	2.75	1.88	1.50	—	—	—	—	—	—
16	4.75	4.00	3.44	3.56	2.27	2.03	—	—	—	—
32	5.00	5.75	6.44	6.25	4.86	4.31	2.56	2.14	—	—
64	7.25	7.75	8.25	9.00	7.36	6.45	5.30	4.54	2.78	2.29
128	8.00	7.50	8.94	9.38	9.56	9.00	8.48	7.19	5.44	4.52
256	8.00	8.00	11.94	9.88	11.86	10.81	10.84	9.73	8.26	7.29
512	5.75	9.00	10.25	10.62	13.05	11.48	12.36	11.29	11.17	9.55
1024	8.25	6.75	12.62	10.12	12.69	11.83	13.80	11.96	13.07	11.24
2048	7.50	8.50	11.31	10.31	13.06	12.66	14.11	13.44	14.04	12.38
4096	8.75	9.25	12.31	11.12	13.81	12.39	13.95	13.13	14.61	12.96
8192	7.50	8.00	12.00	12.12	13.34	12.97	14.67	12.67	14.72	13.18
16384	9.25	8.50	11.38	10.56	12.69	12.33	14.53	13.61	15.02	13.30
32768	7.25	7.50	12.12	10.44	13.00	12.59	13.96	13.00	15.17	13.46

Table 5.6: Comparing the size of cuttings computed by `CutRandomInc`, with or without using merging. Each entry is the size of the minimal cutting computed, divided by r^2.

Number of Lines	Value of r									
	2		4		8		16		32	
		Impr'		Impr'		Impr'		Impr'		Impr'
4	8.75	5.25	—	—	—	—	—	—	—	—
8	10.75	6.25	8.25	6.56	—	—	—	—	—	—
16	10.25	7.25	9.19	7.44	8.14	7.27	—	—	—	—
32	10.00	7.00	9.38	7.56	8.64	7.64	8.06	7.63	—	—
64	10.00	7.25	9.81	7.25	8.66	7.73	8.32	7.82	8.04	7.81
128	10.50	6.50	9.31	7.06	8.72	7.70	8.38	7.90	8.15	7.91
256	11.00	7.50	9.81	7.44	8.81	7.73	8.37	7.90	8.18	7.92
512	11.00	6.75	10.06	7.69	8.81	7.84	8.41	7.88	8.17	7.94

Table 5.7: Comparing the size of cuttings computed by Matoušek's method, to the slightly improved method described in Section 3.

	Size / r^2			Triangles / r^2			Verticess / r^2			Cutting
	min	avg	max	min	avg	max	min	avg	max	Region
ParallelInc	13.64	15.03	16.33	27.28	30.06	32.66	21.84	24.34	26.67	VTrapezoid
ChazelleFriedman	44.30	55.73	79.48	88.59	111.47	158.97	52.78	65.05	89.16	VTrapezoid
CutRandomInc	12.00	12.77	13.52	24.00	25.53	27.03	14.52	15.50	16.30	VTrapezoid
GreedyRandom	11.17	11.69	12.72	22.34	23.38	25.44	13.52	14.19	15.69	VTrapezoid
GreedyLine	10.36	11.02	11.77	20.72	22.05	23.53	12.53	13.19	14.03	VTrapezoid
GreedyWeightedLine	10.33	11.20	12.16	20.66	22.42	24.31	12.39	13.33	14.36	VTrapezoid
Matoušek		8.67			17.34			13.22		VTrapezoid
Matoušek-improved		7.72			15.44			11.78		VTrapezoid
Polytree	9.73	10.83	12.25	14.88	16.20	18.59	9.91	10.55	12.03	4-Polygon
Polytree	7.39	7.89	8.41	12.94	14.08	15.25	8.86	9.61	10.42	5-Polygon
Polytree	6.88	7.47	8.16	13.28	14.42	15.56	9.36	10.14	11.19	6-Polygon
Polytree	6.92	7.38	7.70	13.70	14.61	15.19	9.62	10.30	10.83	7-Polygon
Polytree	6.34	7.28	8.36	12.69	14.53	16.72	8.86	10.41	11.95	**8-Polygon**
Polytree	6.66	7.34	8.12	13.31	14.70	16.22	9.47	10.52	11.81	9-Polygon
Polytree	6.56	7.22	7.91	13.12	14.44	15.81	9.42	10.25	10.98	10-Polygon
Polytree	7.17	7.62	8.14	14.34	15.27	16.28	10.20	11.02	11.64	11-Polygon
PolyTriang (≤ 4)	11.38	15.03	16.97	11.38	15.03	16.97	7.42	9.64	10.75	Triangle
PolyTriang (≤ 5)	13.00	13.72	14.75	13.00	13.72	14.75	8.73	9.27	10.16	Triangle
PolyTriang (≤ 6)	11.91	13.20	14.06	11.91	13.20	14.06	8.19	9.05	9.62	Triangle
PolyTriang (≤ 7)	11.47	13.08	14.94	11.47	13.08	14.94	8.02	9.05	10.25	Triangle
PolyTriang (≤ 8)	11.50	12.89	14.09	11.50	12.89	14.09	8.17	8.97	9.72	Triangle
PolyTriang (≤ 9)	12.06	13.17	15.50	12.06	13.17	15.50	8.47	9.27	11.00	Triangle
PolyTriang (≤ 10)	11.47	12.47	13.47	11.47	12.47	13.47	8.06	8.75	9.30	Triangle
PolyTriang (≤ 11)	12.25	13.28	14.09	12.25	13.28	14.09	8.27	9.23	9.95	Triangle
PolyDeadLeaf (≤ 4)	11.09	11.98	12.81	11.09	11.98	12.81	7.73	8.33	9.08	Triangle
PolyDeadLeaf (≤ 5)	10.38	11.02	11.94	10.38	11.02	11.94	7.33	7.83	8.62	Triangle
PolyDeadLeaf (≤ 6)	10.78	11.50	13.00	10.78	11.50	13.00	7.69	8.22	9.27	Triangle
PolyDeadLeaf (≤ 7)	9.58	11.81	13.78	9.58	11.81	13.78	6.89	8.42	9.61	Triangle
PolyDeadLeaf (≤ 8)	10.59	11.47	12.47	10.59	11.47	12.47	7.64	8.23	8.92	Triangle
PolyDeadLeaf (≤ 9)	11.00	11.62	13.12	11.00	11.62	13.12	7.78	8.31	9.39	Triangle
PolyDeadLeaf (≤ 10)	9.97	10.75	12.47	9.97	10.75	12.47	7.02	7.72	8.70	Triangle
PolyDeadLeaf (≤ 11)	9.97	11.30	12.34	9.97	11.30	12.34	6.98	8.05	8.64	Triangle
PolyVertical	11.61	13.05	14.09	18.94	21.56	23.91	13.81	15.64	16.78	4-Polygon
PolyVertical	8.98	9.67	10.42	14.52	15.61	16.98	11.34	12.31	13.20	5-Polygon
PolyVertical	8.50	9.41	11.16	13.17	14.81	17.55	10.89	12.14	14.30	6-Polygon
PolyVertical	8.25	9.16	10.02	12.88	14.31	15.75	10.61	11.83	13.03	7-Polygon
PolyVertical	8.48	9.33	10.66	13.22	14.55	16.59	11.02	12.05	13.72	8-Polygon
PolyVertical	8.36	9.22	10.05	13.11	14.33	15.58	10.98	11.92	12.86	9-Polygon
PolyVertical	8.39	9.06	10.55	13.25	14.14	16.53	10.98	11.77	13.83	10-Polygon
PolyVertical	8.31	9.03	9.91	12.92	14.08	15.44	10.80	11.73	12.91	11-Polygon

Table 5.8: Results for (1/8)-cutting of 1024 lines. The best results (and fastest) were generated by PolyTree using polygons with at most 8 sides. The execution time for this variant was less than 40 seconds on a Pentium Pro 200MhZ computer.