Algorithmica (2002) 32: 521–539
DOI: 10.1007/s00453-001-0084-9

Algorithmica

Improved Algorithms for Uniform Partitions of Points[1]

P. K. Agarwal,[2] B. K. Bhattacharya,[3] and S. Sen[4]

Abstract. We consider the following one- and two-dimensional bucketing problems: Given a set S of n points in $\mathbb{R}^1$ or $\mathbb{R}^2$ and a positive integer b, distribute the points of S into b equal-size buckets so that the maximum number of points in a bucket is minimized. Suppose at most $(n/b) + \Delta$ points lie in each bucket in an optimal solution. We present algorithms whose time complexities depend on b and Δ. No prior knowledge of Δ is necessary for our algorithms.

For the one-dimensional problem, we give a deterministic algorithm that achieves a running time of $O(b^4(\Delta^2 + \log n) + n)$. For the two-dimensional problem, we present a Monte Carlo algorithm that runs in subquadratic time for small values of b and Δ. The previous algorithms, by Asano and Tokuyama [1], searched the entire parameterized space and required $\Omega(n^2)$ time in the worst case even for constant values of b and Δ. We also present a subquadratic algorithm for the special case of the two-dimensional problem when $b = 2$.

Key Words. Bucketing, Hashing, Random Sampling, Arrangements.

1. Introduction.

We consider geometric optimization problems that do not seem to have any nice properties like convexity and have a large number of distinct global optimal solutions. Consequently, it is hard to develop a search strategy that will avoid examining all the optimum solutions (or more likely near-optimal solutions). However, if there are few optimal solutions, we may be able to prune the search space. This may lead to more efficient algorithms that are "output-sensitive" in the sense that the running time of the algorithm depends on the number of optimal solutions. Since we do not know the optimum solution to begin with, we can try to estimate the optima by some means, say, random-sampling, and then use that to prune the search space. The success of such an approach depends on how effectively we estimate the optima.

In this paper we consider the problem of partitioning a set of points in $\mathbb{R}^1$ or $\mathbb{R}^2$ into equal-size buckets, so that the maximum number of points in a bucket is minimized. These problems were earlier studied in [1] and [4], and they arise in the construction of optimal hash functions; see the aforementioned references for details.

[1] Work by the first author was supported by Army Research Office MURI Grant DAAH04-96-1-0013, by a Sloan fellowship, by NSF Grants EIA-9870724 and CCR-9732787, and by a grant from the U.S.–Israeli Binational Science Foundation. Work by the second author was supported by an NSERC grant. Part of this work was done while the last two authors were visiting the Department of Computer Science, University of Newcastle, Australia.

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

[3] School of Computing Science, Simon Fraser University, Burnaby, British Columbia, Canada V5A 1S6. binay@cs.sfu.ca.

[4] Department of Computer Science and Engineering, IIT Delhi, New Delhi 110016, India. ssen@cs.unc.edu.

Received December 16, 1999; revised September 7, 2000. Communicated by B. Chazelle.
Online publication November 28, 2001.

First we consider the following one-dimensional problem: Given a set S of n real numbers and an integer $1 \leq b \leq n$, partition S uniformly into b equal-size buckets, i.e., each bucket has the same width. The buckets are defined by real numbers $\beta_i = L + i \cdot w$, for $0 \leq i \leq b$ where L is the left endpoint of the leftmost bucket and w is the width (size) of the buckets. The ith bucket B_i, $1 \leq i \leq b$, is defined by the interval $[\beta_i, \beta_{i+1})$ and $S \cap B_i$ is the *content* of the ith bucket (for a fixed choice of L and w). We wish to minimize the maximum size of the contents in buckets. Two version of this problem are studied: (i) the *tight* case in which B_1 and B_b are required to be nonempty, and (ii) the *relaxed* case in which they are allowed to be empty.

Next, we consider the two-dimensional problem. Given a set S of n points in $\mathbb{R}^2$ and an integer $b \leq n$, we again wish to partition S into b *equal-size* buckets so that the maximum number of points in a bucket is minimized. We consider two types of buckets. First, we consider the case in which the buckets are formed by equally spaced $b + 1$ parallel lines, $\ell_0, \ldots, \ell_b$, with orientation θ, for some $\theta \in \mathbb{S}^1$. We require S to lie between ℓ_0 and ℓ_b and both ℓ_0, ℓ_b to contain at least one point of S. The *buckets* are b strips defined by consecutive lines ℓ_{i-1} and ℓ_i $(1 \leq i \leq b)$; see Figure 1(ii). This bucketing problem is known as the *uniform-projection* problem. We next define buckets to be the regions formed by two families of equally spaced $\sqrt{b} + 1$ lines. The extremal lines in both families are required to contain at least one point of S; see Figure 1(iii). This problem is called the *two-dimensional partition* problem.

Asano and Tokuyama [1] describe $O(n^2)$ and $O(b^2 n^2)$-time algorithms for the tight and relaxed cases of the one-dimensional problem. We are able to obtain an $O(b^4(\Delta^2 + \log n) + n)$-time deterministic algorithm for the tight case and an $O(b^5(\Delta^2 + \log n) + bn)$-time algorithm for the relaxed case. The algorithm itself does not require the value of Δ; the value is required only for the analysis. Our algorithm is faster than that of Asano and Tokuyama for small values of b and Δ, e.g., when $b = o(n^{1/3})$ and $\Delta = O(\sqrt{n/b})$, which is the case when points are almost uniformly distributed.

Comer and O'Donnell [4] described an algorithm for the uniform-projection problem that runs in $O(bn^2 \log n)$ time using $O(n^2 + bn)$ space. Asano and Tokuyama [1] gave an $O(n^2 \log n)$-time algorithm, which uses $O(n)$ space, by exploiting the dual transformation of the problem. They also give alternative implementations that could be better for smaller b, but the worst-case running time is $\Omega(n^2)$ even for constant values of b. Bhattacharya [2] also gave an alternate approach for this problem, us-

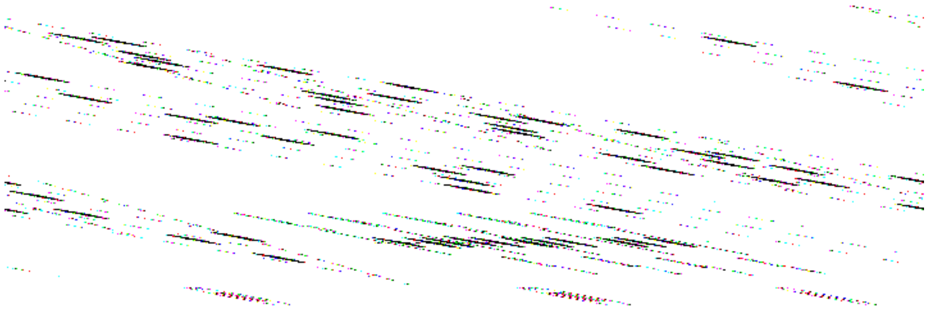

Fig. 1. (i) One-dimensional bucketing problem; (ii) uniform-projection problem; (iii) two-dimensional partitioning problem.

ing the *angle-sweep* method. We first describe a deterministic $O(n^{4/3}\log^{2+\varepsilon} n)$-time algorithm, for any $\varepsilon > 0$, that computes an optimal uniform projection for the special case $b = 2$, thereby improving upon the quadratic upper-bound. For larger values of b, we describe a Monte Carlo algorithm that computes an optimal solution in time $O(\min\{bn^{5/3}\log^{7/3} n + (b^2\Delta)n\log^3 n, n^2\})$, Again, our algorithm is faster for small values of b and Δ. The dependence of running time on Δ is borne out by the fact that the number of possible optimal configurations (having the same value) depends on Δ.

Our overall approach for both one- and two-dimensional problems is similar. Namely, we use a sample to "localize" the search for the global optimum. Although intuitively, this is a good heuristic, analyzing the bound on the number of "potential" candidates for the global optimum, from the optima of the sample, is rather technical. In the one-dimensional problem, we can simply choose a "deterministic" sample because the elements are linearly ordered, but the two-dimensional algorithms rely on random sampling. In both cases we formulate the problem as searching a small portion of a line arrangement. In the one-dimensional case, we localize the search to a few cells of the arrangement while in the two-dimensional case we localize it to a few levels.

The paper is organized as follows. Section 2 describes our one-dimensional algorithm, Section 3 describes the deterministic and Monte Carlo algorithms for the two-dimensional uniform projection problem, and Section 4 describes the two-dimensional partitioning problem in which the buckets are rectangles. We conclude in Section 5 by mentioning a few open problems.

2. Optimal One-Dimensional Cuts.

For a set $S = \{x_1, \ldots, x_n\}$ of real numbers and an integer $1 \le b \le n$, a pair $c = (w, L)$ is called a *cut* if the set of $b + 1$ real numbers $\beta_j = L + j \cdot w$, $0 \le j \le b$, are such that $\beta_0 \le x_1 \le x_n < \beta_b$. The interval $[\beta_{j-1}, \beta_j)$ is called the jth bucket and the set of elements of S lying (strictly) in this interval is the *contents* of the jth bucket. We denote the jth bucket by B_j and the size of its contents $|B_j \cap S|$ by $|B_j^c|$ for a cut c. Let

$$\Phi(c, S) = \max_{1\le j\le b} |B_j^c|$$

denote the *cut value* of c. Let $\mathcal{C}$ be the set of all cuts. The optimal *cut value* $\Phi(S)$ is defined as

$$\Phi(S) = \min_{c\in\mathcal{C}} \Phi(c, S).$$

Any cut that achieves this cut value is an *optimal cut*. If we restrict the cuts to satisfy the condition that $|B_1|, |B_b| \ge 1$, i.e., the first and the last buckets must not be empty, then it is called a *tight* cut. An *optimal tight cut* is defined analogously as above, restricted to the set of tight cuts. We first describe an algorithm for finding an optimal tight cut.

DEFINITION 2.1. Two cuts c_1 and c_2 are *combinatorially distinct* if there is an i, $1 \le i \le b$, such that $|B_i^{c_1}| \ne |B_i^{c_2}|$.

DEFINITION 2.2. The *arrangement* of a set $\mathcal{L}$ of lines in the plane, denoted $\mathcal{A}(\mathcal{L})$, is the planar subdivision induced by the lines of $\mathcal{L}$; that is, $\mathcal{A}(\mathcal{L})$ is a planar map whose *vertices* are the intersection points of lines in $\mathcal{L}$, whose *edges* are maximal (relatively

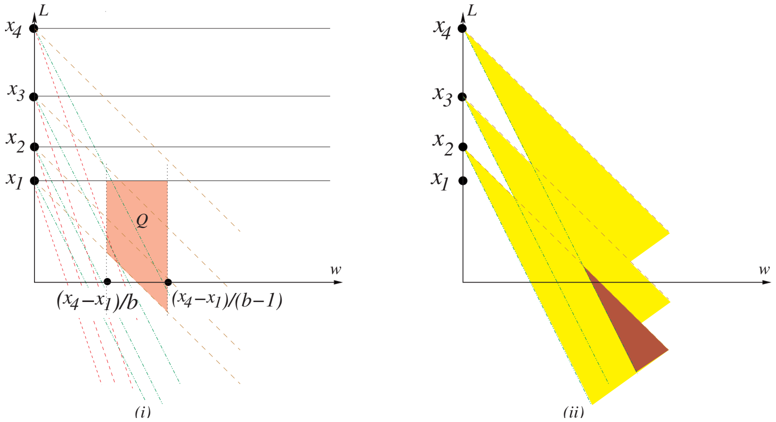

Fig. 2. (i) Set $\mathcal{L}$ and the feasible region Q; (ii) the shaded regions denote C_{22}, C_{23}, and C_{24}, and the dark region denotes $\mathcal{C}(2, 4; 2)$, the set of cuts for which $\{x_2, x_3, x_4\}$ lie in the second bucket B_2.

open) connected portions of the lines that do not contain a vertex, and whose *faces* are the connected components of $\mathbb{R}^2 - \bigcup \mathcal{L}$.

We parameterize the problem as follows. We represent each cut $c = (w, L)$ as a point in the plane. Abusing the notation slightly, we use the term "cut" to denote a point in the (w, L)-plane as well as the set of buckets induced by that cut. Let

$$\mathcal{L} = \{x_i = L + jw \mid 1 \le i \le n, 0 \le j \le b\}$$

be the set of $(b + 1)n$ lines in the (w, L)-plane, which we refer to as the *event* lines.

$\mathcal{L}$ consists of $b + 1$ families of parallel lines (one for each fixed j), each family containing n lines; see Figure 2(i). Hence, every face in $\mathcal{A}(\mathcal{L})$ contains at most $2(b + 1)$ edges. For all cuts $c = (w, L)$ lying in the same face f of $\mathcal{A}(\mathcal{L})$ the cut value remains the same; we denote this value by $\Phi(f, S)$. Let $\Phi_j(f, S) = |B_j^c(S)|$ for any $c \in f$. The nonempty condition of extreme buckets implies that we have to consider only those cuts (w, L) that lie in the quadrilateral Q defined by the intersection of the following four constraints; see Figure 2:

$$Q: \qquad x_1 \ge L > x_1 - w \quad \text{and} \quad \frac{x_n - x_1}{b} < w < \frac{x_n - x_1}{b - 1}. \tag{2.1}$$

The above constraint leads to the following lemma.

Lemma 2.3. *For every point $x_i \in S$, there exists an integer $1 \le j \le b - 1$, so that x_i lies in one of the two buckets B_j or B_{j+1} for any tight cut.*

Proof. A point $x_i \in S$ lies in the bucket B_j of a cut $c = (w, L)$ if and only if

$$L + w \cdot (j - 1) \le x_i < L + w \cdot j.$$

Suppose there are two cuts $c_1 = (w_1, L_1)$ and $c_2 = (w_2, L_2)$ and two integers $1 \le k_1 < k_1 + 1 < k_2 \le b$ such that x_i lies in the bucket B_{k_1} of the cut c_1 and in the bucket B_{k_2} of c_2. Then we have the following two inequalities:

$$x_i - x_1 < k_1 \cdot w_1 \quad \text{and} \quad x_i - x_1 > (k_2 - 1) \cdot w_2 \ge (k_1 + 1)w_2.$$

It follows that $k_1 w_1 > (k_1 + 1)w_2$ and therefore

$$\frac{w_2}{w_1} < \frac{k_1}{k_1 + 1} = 1 - \frac{1}{k_1 + 1}. \tag{2.2}$$

On the other hand, by (2.1),

$$\frac{w_2}{w_1} > \frac{x_n - x_1}{b} \cdot \frac{b - 1}{x_n - x_1} = 1 - \frac{1}{b}. \tag{2.3}$$

Comparing (2.2) and (2.3), we obtain $k_1 > b - 1$, which contradicts the assumption that $k < k_2 - 1 \le b - 2$. Hence, the lemma is true. □

This lemma immediately implies that at most n lines of $\mathcal{L}$ intersect Q, and that Q intersects $O(n^2)$ faces of $\mathcal{A}(\mathcal{L})$. The lines of $\mathcal{L}$ that intersect Q can be determined in $O(bn)$ time. We can therefore search over $Q \cap \mathcal{A}(\mathcal{L})$ in $O(n^2)$ time to find representatives for all classes of combinatorially distinct optimal cuts.

LEMMA 2.4. *For a set of n points, all the combinatorially distinct optimal cuts can be computed in $O(n^2)$ time.*

For an integer $r \ge 1$, let $R \subseteq S$ be the subset of r points obtained by choosing every (n/r)th point of S. Using Lemma 2.4 for directly solving the problem, we can compute the optimal solution for R in $O(r^2)$ time.

LEMMA 2.5. *Let n_o, r_o be the maximum size of a bucket in an optimal solution for S and R, respectively. Then*

$$\left| \frac{n_o}{n} - \frac{r_o}{r} \right| < \frac{1}{r}.$$

PROOF. Let c be an optimal cut for R. Each bucket of c contains at most r_o points. Since R is chosen by selecting every (n/r)th point of S, each bucket of c contains at most $(r_o + 1)n/r - 1$ points of S. Therefore $n_o < (r_o + 1)n/r$, or

$$\frac{n_o}{n} - \frac{r_o}{r} < \frac{1}{r}.$$

Conversely, let c' be an optimal cut for S. Then each bucket of c contains at most n_o points of S, which implies that each bucket contains at most $(n_o + (n/r) - 1)r/n$ points of R. Hence,

$$r_o < \left(n_o + \frac{n}{r}\right)\frac{r}{n} \quad \text{or} \quad \frac{r_o}{r} - \frac{n_o}{n} < \frac{1}{r}.$$

This completes the proof of the lemma. □

We now describe the algorithm for computing an optimal solution for S, assuming that we have already computed the value of r_{o}. Let C_{ij} denote the set of points $c = (w, L)$ in the (w, L)-plane so that the point $x_j \in S$ lies in the bucket B_i of the cut c. Then

$$C_{ij} = \{(w, L) \mid L + (i-1)w \leq x_j < L + iw\}$$

is the cone with apex at $(0, x_j)$; see Figure 2(ii). Given three integers $1 \leq l \leq r \leq n$ and $1 \leq i \leq b$, the set of points in the (w, L)-plane for which the subset $\{x_l, x_{l+1}, \ldots, x_r\}$ of S lies in the ith bucket B_i is $\mathcal{C}(l, r; i) = \bigcap_{j=l}^{r} C_{ij}$. $\mathcal{C}(l, r; i)$ is a cone formed by the intersection of the halfplanes $x_l \geq L + (i-1)w$ and $x_r > L + iw$.

By Lemma 2.5,

$$(2.4) \qquad (r_{\mathrm{o}} - 1)\frac{n}{r} < n_{\mathrm{o}} < (r_{\mathrm{o}} + 1)\frac{n}{r}.$$

Set $m = (r_{\mathrm{o}} + 1)n/r > n_{\mathrm{o}}$. We use this inequality to compute n_{o} efficiently. Define $n_{\mathrm{o}} = (n/b) + \Delta$ and $\delta = m - (n/b)$. Using (2.4), we obtain that $\delta < \Delta + 2n/r$.

If $b^2\delta \geq n$, then we use the $O(n^2)$-time algorithm described earlier to compute an optimal cut, so assume that $b^2\delta < n$. If each bucket B_i in a cut c contains at most m points of S, then, for any $1 \leq i \leq b$, the first i buckets in c contain at most $r_i = mi$ points, therefore $\beta_i < x_{r_i}$. Similarly, the last $b - i$ buckets in c contain at most $(b-i)m$ points, therefore $\beta_i \geq x_{l_i}$, where $l_i = n - m(b-i)$. Hence, $\beta_i \in [x_{l_i}, x_{r_i})$. Set $r_0 = 1$; see Figure 3. Note that $r_i - l_i = b\delta$ for $1 \leq i \leq b$. This implies that the subset $S_i = \{x_j \mid r_{i-1} \leq j < l_i\}$ always lies in the ith bucket B_i (see Figure 3), for all $1 \leq i \leq b$. Hence, if there is a cut $\xi = (w, L)$ so that all buckets in ξ contain at most m points, then ξ lies in the region $P(m) = \bigcap_{i=1}^{b} \mathcal{C}(r_{i-1}, l_i - 1; i)$, which is the intersection of b cones and is thus a convex polygon with at most $2b$ edges. For all cuts $\xi \notin P(m)$, $\Phi(\xi, S) > m$. It thus suffices to search for an optimal cut within $P(m)$.

Let $H_i \subseteq \mathcal{L}$ be a set of $l_i - r_i = b\delta$ lines defined as

$$H_i = \{x_j = L + iw \mid l_i \leq j < r_i\}.$$

Set $H = \bigcup_{i=1}^{b} H_i$; $|H| = b^2\delta$. The same argument as in Lemma 2.3 shows that no line of $H \backslash \mathcal{L}$ intersects the interior of the polygon $P(m)$. We construct the arrangement $\mathcal{A}(H)$ within the polygon $P(m)$ in $O(b^4\delta^2)$ time. Actually, we can clip $\mathcal{A}(H)$ inside $P(m) \cap Q$, where Q is the quadrilateral defined in (2.1). Let $\mathcal{A}_P(H)$ denote this clipped arrangement. By the above discussion, $\mathcal{A}_P(H)$ is the same as $\mathcal{A}(L)$ clipped within $P(m)$. Therefore, for any two points ξ and ξ' in a face $f \in \mathcal{A}_P(H)$, the contents of all buckets in the cuts ξ and ξ' are the same. Let

$$\varphi(f) = \langle \Phi_1(f, S), \ldots, \Phi_b(f, S) \rangle.$$

If f and f' are two adjacent faces of $\mathcal{A}_P(H)$ separated by a line $L + iw = x_j$, then the only difference in the two cuts $\xi \in f$ and $\xi' \in f'$ is that x_j belongs to B_{i-1} in one of

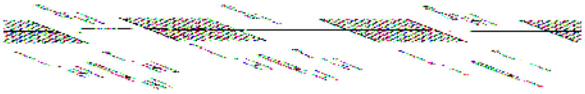

Fig. 3. The boundary β_i can lie in the shaded interval $[l_i, r_i)$.

them and it belongs to B_i in the other. Therefore $\varphi(f')$ and $\Phi(f', S)$ can be computed from $\varphi(f)$ and $\Phi(f, S)$ in $O(1)$ time.

We compute, in time $O(b^4\delta^2)$, a tour $\Pi = \langle f_0, f_1, \dots, f_u\rangle$, where $u = O(b^4\delta^2)$, of the dual graph of $\mathcal{A}_P(H)$ that visits every face of $\mathcal{A}_P(H)$ at least once. We compute $\varphi(f_0)$ and $\Phi(f_0, S)$ in $O(n)$ time. We then visit the faces of $\mathcal{A}_P(H)$ along Π, and for each $i \geq 1$, compute $\varphi(f_i)$ and $\Phi(f_i, S)$ from $\varphi(f_{i-1})$ and $\Phi(f_{i-1}, S)$ in $O(1)$ time. We can thus compute $n_{\mathrm{o}} = \Phi(S) = \min_{f\in\mathcal{A}_P(H)} \Phi(f, S)$ in $O(b^4\delta^2 + n)$ time. The total time spent in computing an optimal cut is

$$O\left(r^2 + b^4\left(\Delta + \frac{n}{r}\right)^2 + n\right).$$

Choosing $r = \lceil b\sqrt{n}\rceil$, we obtain the following.

LEMMA 2.6. *An optimal tight cut for n points into b buckets can be found in $O(b^4\Delta^2 + b^2 n)$ time, assuming that the points are sorted.*

Instead of using the quadratic algorithm for computing r_{o}, we can compute r_{o} recursively. Let $T(r, \Delta')$ denote the maximum running time of the algorithm for computing an optimal cut for a subset $R \subseteq S$ of size r chosen by selecting every (n/r)th point of S, where $r/b + \Delta'$ is the optimal cut value of R. Then we have the following recurrence:

$$T(n, \Delta) = \begin{cases} T(r, \Delta') + O\left(b^4(\Delta + n/r)^2 + n\right) & \text{if} \quad b^2(\Delta + 2n/r) \leq n, \\ O(n^2) & \text{otherwise.} \end{cases}$$

Choosing $r = \lceil n/2\rceil$ and using the fact that $r_{\mathrm{o}} \leq n_{\mathrm{o}} r/n + 1$, we obtain that

$$r_{\mathrm{o}} \leq \frac{r}{b} + \frac{\Delta}{2} + 1, \qquad \text{i.e.,} \quad \Delta' \leq \frac{\Delta}{2} + 1.$$

Hence, we can show that

$$T(n, \Delta) = O(b^4(\Delta^2 + \log n) + n).$$

THEOREM 2.7. *Given a set S of n points in $\mathbb{R}$, sorted in increasing order, and an integer $1 \leq b \leq n$, an optimal tight cut for S with b buckets can be computed in $O(b^4(\Delta^2 + \log n) + n)$ time.*

REMARK 2.8. If we are interested only in computing an ε-approximate solution, for $0 < \varepsilon < 1$, i.e., computing a cut c such that $\Phi(S, c) \leq (1 + \varepsilon)\Phi(S)$, then we can obtain a faster algorithm by choosing a sample R of size $r = \lceil 2b/\varepsilon\rceil$ as described earlier and computing R. Using (2.4) and the fact that $\Phi(S) \geq n/b$, we obtain that $\Phi(R)n/r \leq (1+\varepsilon)\Phi(S)$. The running time of the algorithm is $O(n+(b/\varepsilon)^2)$, assuming that the points in S are sorted. Otherwise, the running time is $O(n\log(b/\varepsilon) + (b/\varepsilon)^2)$.

We can use a similar analysis for finding optimal cuts when relaxed cuts are also allowed. We simply replace n by bn as there are bn event lines. Another way to view this is that the optimal cut can be determined by trying out all nonredundant cuts for η buckets for $2 \leq \eta \leq b$ and selecting the best one.

COROLLARY 2.9. *An optimal (relaxed) cut for a set of n points in $\mathbb{R}$ with b buckets can be found in $O(b^5(\Delta^2 + \log n) + bn)$ time.*

3. The Uniform-Projection Problem. In this section we describe the algorithms for the uniform projection problem. Let $S = \{p_1, \dots, p_n\}$ be a set of n points in $\mathbb{R}^2$ and let $1 \le b \le n$ be an integer. We want to find $b + 1$ equally spaced parallel lines so that all points of S lie between the extreme lines, the extreme lines contain at least one point of S, and the maximum number of points in a bucket is minimized; see Figure 1(ii). If the lines have slope θ, we refer to these buckets as the θ-cut of S. For each θ, there is unique θ-cut of S. We first describe a subquadratic algorithm for $b = 2$. Next, we show how the running time of the algorithm by Asano and Tokuyama can be improved, and then we describe a Monte Carlo algorithm that computes $\Phi(S)$, the optimum value, with high probability, in subquadratic time for small values of b and Δ.

It will be convenient to work in the dual plane. The duality transform maps a point $p = (a, b)$ to the line $p^* : y = -ax + b$ and a line ℓ: $y = \alpha x + \beta$ to the point $\ell^* = (\alpha, \beta)$ [5]; see Figure 4. Let ℓ_i denote the line dual to the point $p_i \in S$, and let $\mathcal{L} = \{\ell_i \mid 1 \le i \le n\}$. The dual of a strip σ bounded by two parallel lines ℓ_1 and ℓ_2 is the vertical segment $\sigma^* = \ell_1^*\ell_2^*$; a point p lies in σ if and only if the line p^* intersects the segment σ^*.

Let $\mathcal{A}(\mathcal{L})$ be the arrangement of $\mathcal{L}$ as defined in Section 2. We define the *level* of a point $p \in \mathbb{R}^2$ with respect to $\mathcal{L}$, denoted by $\lambda(p, \mathcal{L})$, to be the number of lines in $\mathcal{L}$ that lie below p (i.e., the vertical line through p intersects $\mathcal{L}$ below p). The level of all points within an edge or a face of $\mathcal{A}(\mathcal{L})$ is the same. For an integer $0 \le k < n$, we define the k-level of $\mathcal{A}(\mathcal{L})$, denoted by $\mathcal{A}_k(\mathcal{L})$, to be the closure of the set of edges of $\mathcal{A}(\mathcal{L})$ whose level is k. The level $\mathcal{A}_k(\mathcal{L})$ is an x-monotone polygonal chain with at most $O(n(k+1)^{1/3})$ edges [6]. The *lower* and *upper* envelopes of $\mathcal{A}(\mathcal{L})$ are the levels $A_0(\mathcal{L})$ and $\mathcal{A}_{n-1}(\mathcal{L})$, respectively. The total number of vertices on the upper and lower envelopes of $\mathcal{A}(\mathcal{L})$ is n because every such vertex is the dual of the line supporting an edge of the convex hull of S.

Since we require the extreme bucket boundaries to contain a point of S, the points dual to the extreme lines lie on the upper and lower envelopes of $\mathcal{L}$. For a fixed x-coordinate θ, let $s(\theta)$ denote the vertical segment connecting the points on the lower and

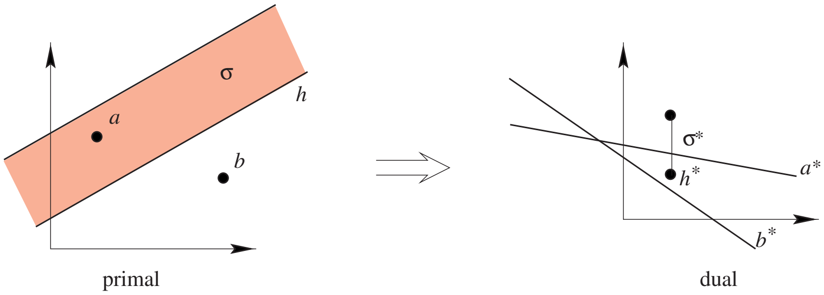

Fig. 4. The duality transform in two dimensions. Vertical segment σ^* is the dual of the strip σ.

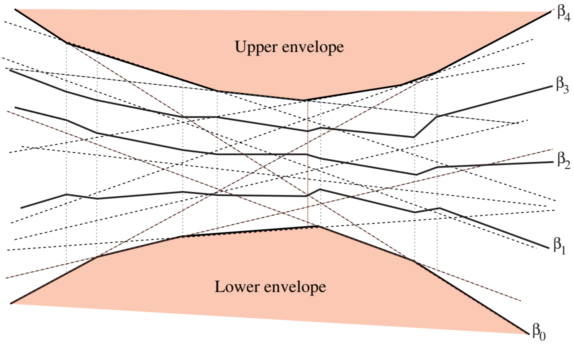

Fig. 5. The uniform-projection problem and the bucket lines in the dual setting.

upper envelopes of $\mathcal{L}$ with the x-coordinate θ. We can partition $s(\theta)$ into b equal-length subsegments $s_1(\theta), \ldots, s_b(\theta)$. Let $\beta_0(\theta), \ldots, \beta_b(\theta)$ be the endpoints of these segments. These endpoints are dual to the bucket boundaries of the θ-cut, and $s_i(\theta)$ is the dual of the jth bucket in the θ-cut. The line ℓ_j intersects $s_i(\theta)$, $i \le b$, if and only if the point p_j lies in the bucket B_i corresponding to the θ-cut. Let β_i denote the path traced by the endpoint $\beta_i(\theta)$ as we vary θ from $-\infty$ to $+\infty$. If we vary θ, $\beta_i(\theta)$, for $0 \le i \le b$, traces along a line segment, as long as the endpoints of $s(\theta)$ do not pass through a vertex of upper or lower envelopes. Therefore each β_i is an x-monotone polygonal chain with at most n vertices; see Figure 5 for an illustration. Since we will be looking at the problem in the dual plane from now, we call the β_i's *bucket lines*. Let $\mathcal{B} = \{\beta_0, \ldots, \beta_b\}$. The intersection of a bucket line β_i with a line ℓ_j is an *event* at which the point p_j switches from B_{j-1} to B_j or vice versa.

For an x-coordinate θ and a subset $A \subseteq \mathcal{L}$, let $\mu_i(A, \theta)$ denote the number of lines of A that intersect the vertical segment $s_i(\theta)$; $\mu_i(A, \theta)$ denotes the set of points dual to A that lie in the ith bucket of the θ-cut. Let $\Phi(A, \theta) = \max_{1 \le i \le b} \mu_i(A, \theta)$. Set $n_o = \Phi(S) = \Phi(\mathcal{L}) = \min_\theta \Phi(\mathcal{L}, \theta)$.

3.1. *Partitioning into Two Buckets.* We first describe a deterministic scheme that finds in subquadratic time an optimal solution for partitioning S into two buckets. By our convention, β_0, β_2 denote the upper and lower envelopes of $\mathcal{L}$, respectively. To determine n_o, we search for an x-coordinate θ_o, where $\beta_1(\theta_o)$ is closest to the $\lceil n/2 \rceil$-level of $\mathcal{A}(\mathcal{L})$. First, we compute $\Lambda = \mathcal{A}_{\lceil n/2 \rceil}$ in $O(n^{4/3} \log^{1+\varepsilon} n)$ time [3], for any $\varepsilon > 0$, and check whether β_1 intersects Λ. If a point $\beta_1(\theta_o)$ lies on Λ, then we return the θ_o-cut. If β_1 lies below Λ, we compute the highest level in the interval $[1, \lceil n/2 \rceil - 1]$ of $\mathcal{A}(\mathcal{L})$ that β_1 intersects, and set λ_o to this level. This can be accomplished in $O(n^{4/3} \log^{2+\varepsilon} n)$ by performing a binary search on the levels. Similarly, if β_1 lies above Λ, we find in $O(n^{4/3} \log^{2+\varepsilon} n)$ time the smallest level in the interval $[\lceil n/2 \rceil + 1, n-1]$ that β_1 intersects and set λ_o to this level. If $\beta_1(\theta_o)$ is an intersection point of β_1 and $\mathcal{A}_{\lambda_o}(\mathcal{L})$, then we return the θ_o-cut. Chan's algorithm computes the edges of a level incrementally from left to right, so we can actually detect whether β_1 intersects the level while computing the level itself in $O(n^{4/3} \log^{1+\varepsilon} n)$ time using $O(n)$ space. Hence, we obtain the following.

LEMMA 3.1. *The optimal uniform projection of n points in $\mathbb{R}^2$ into two buckets can be computed in $O(n^{4/3}\log^{2+\varepsilon} n)$ steps, for any $\varepsilon > 0$, using $O(n)$ space.*

3.2. *A Deterministic Algorithm.* In this section we present a deterministic algorithm for the uniform-projection problem that has $O(bn \log n + K \log n)$ running time and uses $O(n)$ storage, where K denotes the number of event points, i.e., the number of intersection points between $\mathcal{L}$ and $\mathcal{B}$. This improves the running times of $O(n^2 + bn + K \log n)$ for general b and $O(b^{0.610}n^{1.695} + K \log n)$ for $b < \sqrt{n}$ in [1].

As in Asano and Tokuyama's algorithm, we will sweep a vertical line through $\mathcal{A}(\mathcal{L})$, but unlike their approach we will not stop at every intersection point of $\mathcal{L}$ and $\mathcal{B}$. We first compute the lower and upper envelopes of $\mathcal{L}$, which are the bucket lines β_0 and β_b, respectively. We can then compute the rest of the bucket lines $\beta_1, \ldots, \beta_{b-1}$ in another $O(bn)$ time. We preprocess each β_i for answering ray-shooting queries in $O(n \log n)$ time so that a query can be answered in $O(\log n)$ time [9]. The total space used is $O(bn)$.

We sweep a vertical line from $x = -\infty$ to $x = +\infty$, stopping at the intersection points of $\mathcal{L}$ and the bucket lines. At each x-coordinate θ, for $1 \le i \le b$, we maintain $\mu_i(\theta)$, and, for $1 \le j \le n$, the index of the bucket ν_j that contains the line ℓ_j in the θ-cut. These quantities remain the same for all x-coordinates between two consecutive event points. We also maintain an event queue Q that stores some of the event points that lie to the right of the sweep line, but it is guaranteed to contain the next event point. Suppose we are at an event point $\beta_i(\theta) = \beta_i \cap \ell_j$ and ℓ_j lies above β_i to the right of $\beta_i(\theta)$. Then ℓ_j moves from B_i to B_{i+1} at θ. We therefore decrease $\mu_i(\theta)$ by 1, increase $\mu_{i+1}(\theta)$ by 1, and set ν_j to i. The next intersection point of ℓ and $\mathcal{B}$, if it exists, lies on either β_i or β_{i+1}. We compute in $O(\log n)$ time the intersection points of ℓ with β_i and β_{i+1} that lie immediately after $\beta_i(\theta)$, using the ray-shooting data structure and add them to Q.

On the other hand, if ℓ_j lies below β_i to the right of $\beta_i(\theta)$, ℓ_j moves from B_{i+1} to B_i at θ. We decrease $\mu_{i+1}(\theta)$ by 1, increase $\mu_i(\theta)$ by 1, compute the next intersection points of ℓ_j with β_i and β_{i-1}, and add the two intersection points (if they exist) to Q.

We spend $O(\log n)$ time at each event point. Therefore the total running time of the algorithm is $O((bn + K) \log n)$. The event queue Q uses $O(K)$ space and the ray-shooting data structures use $O(bn)$ space. The size of Q can be reduced to $O(n)$ using the standard technique, namely, for each line ℓ_j, store only one intersection point of ℓ_j with the bucket lines [7]. In particular, suppose we want to insert a point $\sigma \in \ell_j$ to Q. We check whether Q already contains a point σ' on ℓ_j. If $x(\sigma) \ge x(\sigma')$, we do not insert σ into Q. Otherwise, we insert σ into Q and delete σ' from it. The total time spent at each event point is still $O(\log n)$, but the size of Q is now $O(n)$. However, the ray-shooting data structure still requires $O(bn)$ space. In order to reduce the overall storage to $O(n)$, we partition the plane into $u \le 2b$ vertical strips $W_1, \ldots, W_u$ so that each W_i contains at most n vertices of the bucket lines. Note that each β_j contains at most n/b vertices inside W_i. We now run the above sweep-line algorithm in each W_i separately. While sweeping a vertical line through W_i, we have to preprocess only $\beta_i \cap W_i$ for ray shooting, for each $0 \le i \le b$. Since each β_i has at most n/b vertices inside W_i, the total space used by the ray-shooting data structures is $O(n)$. The asymptotic running time is still $O((bn + K) \log n)$. Hence, we obtain the following.

THEOREM 3.2. *An optimum partitioning in the tight case can be determined in $O((bn+K)\log n)$ time using $O(n)$ storage, where K is the number of event points.*

3.3. *A Monte Carlo Algorithm.* We now present a Monte Carlo algorithm that runs in subquadratic time, with high probability, for small values of b and Δ, where $n_{\rm o} = (n/b) + \Delta$. The overall idea is quite straightforward and similar to Section 2. From the given set $\mathcal{L}$ of n lines, we choose a random subset R of size $r > 20\log n$ (a value that we will specify more precisely in the analysis). Let Θ_R be the x-coordinates of all the intersection points of R and $\mathcal{B}$, the set of bucket lines with respect to $\mathcal{L}$. We compute $r_{\rm o} = \min_{\theta\in\Theta_R}\Phi(R,\theta)$. Note that we are not computing $\Phi(R)$ since we are considering buckets lines with respect to $\mathcal{L}$. $\mathcal{B}$ can be computed in $O(n\log n + bn)$ time, and $r_{\rm o}$ can be computed in additional $O(r(b+n)) = O(rn)$ time. We use $r_{\rm o}$ to estimate the overall optimum $n_{\rm o}$ with high likelihood. In the next phase we use this estimate and the ideas used in the one-dimensional algorithm to sweep only those regions of $\mathcal{B}$ that "potentially" contain the optimal solution. In our analysis, we will show that the number of such event points is $o(n^2)$ if b and Δ are small. This approach is similar to the randomized selection algorithm of Floyd and Rivest.

We choose two parameters r and $\mathrm{Var} = \mathrm{Var}(r)$ whose values will be specified in the analysis below. An *event point* with respect to $\mathcal{L}$ (resp. R) is a vertex of $\mathcal{B}$ or an intersection point of a line of $\mathcal{L}$ (resp. R) with a chain in $\mathcal{B}$. The event points with respect to R partition the chains of $\mathcal{B}$ into disjoint segments, which we refer to as *canonical intervals*. Before describing the algorithm we state a few lemmas, which are crucial for our algorithm.

Random sampling. In the following we assume that R is a random subset of $\mathcal{L}$ of size $r > 20\log n$. Our first lemma establishes a relation between the event points of $\mathcal{A}(\mathcal{L})$ and those of $\mathcal{A}(R)$.

LEMMA 3.3. *Let $\alpha > 0$ be a constant and let $1 \le i \le b$ be an integer. With probability at least $1 - 1/n^{\alpha}$, at most $O((n/r)\log n)$ event points of $\mathcal{A}(\mathcal{L})$ lie on any canonical interval of β_i.*

PROOF. The proof follows along the lines of a standard random-sampling argument. Consider any event point of $\mathcal{A}(\mathcal{L})$. The probability that more than $c(n/r)\log n$ lines of $\mathcal{L}$ are not chosen before the first line is chosen to its right is no more than $(1 - r/n)^{cn\log n/r} \le n^{-c}$. The probability that this holds for *any* event point of $\mathcal{A}(\mathcal{L})$ (and hence for $\mathcal{A}(R)$) is less than $K\cdot n^{-c}$. Since $K = O(n^2)$, by choosing $c = \alpha + 2$, the lemma follows. □

Using a classical result by Vapnik and Chervonenkis on ε-approximations (see, e.g., Chapter 16 of [12]), which can also be proved using Chernoff's bound, we can establish a relationship between the number of lines of $\mathcal{L}$ and of R intersecting a vertical segment.

LEMMA 3.4. *Let e be a vertical segment and let $\mathcal{L}_e \subseteq \mathcal{L}$ be the subset of n_e lines that intersect e. There is a constant c such that with probability exceeding $1 - 1/n^2$,*

$$\left|\frac{n_e}{n} - \frac{|\mathcal{L}_e \cap R|}{r}\right| \le c\sqrt{\frac{\log n}{r}}.$$

An immediate corollary of the above lemma is the following.

COROLLARY 3.5. *There is a constant c so that, with probability exceeding* $1 - 1/n$,

$$\left|\frac{n_o}{n} - \frac{r_o}{r}\right| \leq c\sqrt{\frac{\log n}{r}}.$$

PROOF. Suppose the θ-cut is an optimal cut for R. Apply Lemma 3.4 to the segments $s_1(\theta), \ldots, s_b(\theta)$. Since $b \leq n$ and each segment $s_i(\theta)$ intersects less than n lines of $\mathcal{L}$, the claim follows. □

COROLLARY 3.6. *Let ξ be a θ-cut so that every bucket of ξ contains at most m points of S. For* $1 \leq i \leq b - 1$, *let*

$$l_i = r - (b - i)m\frac{r}{n} - c\sqrt{r \log n} \quad \textit{and} \quad r_i = im\frac{r}{n} + c\sqrt{r \log n},$$

where c is an appropriate constant. Then with probability exceeding $1 - 1/n$,

$$l_i \leq \lambda(\beta_i(\xi), R) \leq r_i. \tag{3.1}$$

PROOF. If each bucket of ξ contains at most m points, then the first i buckets of ξ contain at most mi points of S and the last $(b - i)$ buckets of ξ contain at most $(b - i)m$ points of S. The lemma now follows by an application of Lemma 3.4 to the segments $\beta_0(\xi)\beta_i(\xi)$ and $\beta_i(\xi)\beta_b(\xi)$. □

We also need the following result by Matoušek on simplex range searching.

LEMMA 3.7 [10]. *Given a set P of n points in $\mathbb{R}^2$ and a parameter m, $n \leq m \leq n^2$, one can preprocess P for triangle range searching in time $O(m \log n)$, to build a data-structure of $O(m)$ space and then report queries in $O((n \log^2 n)/\sqrt{m} + K)$ time, for output size K, where K is number of points in the query triangle.*

REMARK. If $m = \Omega(r^2 \log^2 n)$ and $K \geq (n/r) \log n$, then the output size dominates the query time, so the query time becomes $O(K)$ in this case.

First phase. We now describe the algorithm in detail. We first compute in $O(n \log n + bn)$ time the upper and lower envelopes of $\mathcal{L}$ and the bucket lines $\beta_0, \ldots, \beta_b$. Next, we choose a random sample R of size r, where $r > 20 \log n$ is a parameter to be fixed later, and compute $r_o = \min_\theta \Phi(R, \theta)$, where θ varies over the x-coordinates of all the event points of $\mathcal{B}$ with respect to R. As mentioned earlier, we are not computing an optimal solution for R, since the bucket lines are defined by $\mathcal{L}$. We can compute r_o in $O(rn)$ time as described in [1]. This completes the first phase of the algorithm. The total time required by this phase is

$$O(n \log n + bn) + O(rn) = O((r + b)n). \tag{3.2}$$

Second phase. In the following we assume that the set R satisfies Lemmas 3.3 and 3.4 and Corollaries 3.5 and 3.6. This holds with probability exceeding $1 - 1/n$. By Corollary 3.5,

$$r_{\mathrm{o}}\frac{n}{r} - cn\sqrt{\frac{\log n}{r}} \le n_{\mathrm{o}} \le r_{\mathrm{o}}\frac{n}{r} + cn\sqrt{\frac{\log n}{r}}.$$

Set

$$m_L = \max\left\{r_{\mathrm{o}}\frac{n}{r} - cn\sqrt{\frac{\log n}{r}}, \frac{n}{b}\right\}.$$

By testing for $i = 0, 1, \ldots$ in increasing order, we first find the smallest $0 \le i \le \lceil \log n \rceil$ such that $m_L + 2^i < n_{\mathrm{o}} \le m_L + 2^{i+1}$. We then perform a binary search in the interval $[m_L + 2^i, m_L + 2^{i+1}]$ to compute the optimal value n_{o}. We thus need a procedure that, given an integer $m \in [m_L + 2^i, m_L + 2^{i+1}]$, can determine whether $n_{\mathrm{o}} \le m$ or $n_{\mathrm{o}} > m$. Suppose $n_{\mathrm{o}} = (n/b) + \Delta$ and $\delta = m - n/b$. Since $m_L \ge n/b$ and $m_L + 2^i < n_{\mathrm{o}}$, we have $\Delta > 2^i$. Therefore

$$m \le m_L + 2^{i+1} \le n_{\mathrm{o}} + 2^i < \frac{n}{b} + 2\Delta. \tag{3.3}$$

We run the decision algorithm $O(\log n)$ times.

We now describe the decision algorithm. If each bucket of a θ-cut contains at most m points of S, then, by Corollary 3.6, $l_i \le \lambda(\beta_i(\theta), R) \le r_i$. For each $1 \le i < b$, let

$$X_i = \{\theta \mid l_i \le \lambda(\beta_i(\theta), R) \le r_i\}.$$

Let $X = \bigcap_{i=1}^{b-1} X_i$, and let $|X|$ be the number of connected components in X. For any $\theta \notin X$, at least one of the β_i does not satisfy (3.1), so $\Phi(\mathcal{L}, \theta) > m$ for any such θ-cut. We therefore restrict our search to the θ-cuts for which $\theta \in X$ and compute $m_{\mathrm{o}} = \min_{\theta \in X} \Phi(\mathcal{L}, \theta)$. If $m_{\mathrm{o}} \le m$, then $n_{\mathrm{o}} \le m$. Otherwise, we conclude that $n_{\mathrm{o}} > m$. Hence, it suffices to describe an algorithm for computing m_{o}.

For each $0 \le i \le b$, let $\mathcal{I}_i$ be the set of canonical intervals of β_i whose x-projections intersect X (see Figure 6), and let

$$\Theta_i = \{\theta \in X \mid \beta_i(\theta) \text{ is an event point with respect to } \mathcal{L}\}.$$

Set $\mathcal{I} = \bigcup_{i=0}^{b} \mathcal{I}_i$, $\nu = |\mathcal{I}|$, and $\Theta = \bigcup_{i=0}^{b} \Theta_i$. Since every event point whose x-coordinate is in Θ_i lies on a canonical interval in $\mathcal{I}_i$, by Lemma 3.3, $|\Theta| = O(\nu(n/r)\log n)$.

Since the contents of buckets change only at the event points,

$$m_{\mathrm{o}} = \min_{\theta \in X} \Phi(\mathcal{L}, \theta) = \min_{\theta \in \Theta} \Phi(\mathcal{L}, \theta).$$

It thus suffices to compute $\Phi(\mathcal{L}, \theta)$ for all $\theta \in \Theta$. We describe later how to compute X and $\mathcal{I}$, but we first describe how to compute Θ and an optimal cut from X and $\mathcal{I}$.

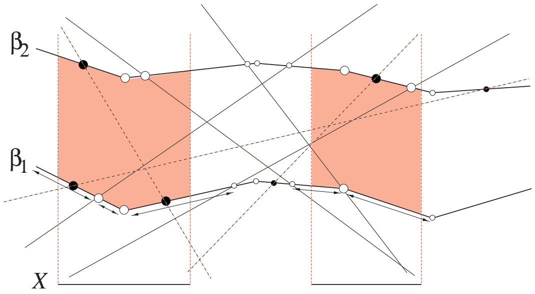

Fig. 6. X, β_1, and β_2. Solid lines belong to R and dashed lines belong to $\mathcal{L}\backslash R$. Shaded regions denote the segments $s_1(\theta)$ for $\theta \in X$. Large (small) bullets are the intersection points of $\mathcal{L}$ with the bucket lines that lie (resp. do not lie) inside $X \times \mathbb{R}$. Arrowed segments represent the canonical intervals in $\mathcal{I}_1$.

Computing Θ. We preprocess S in $O(r^2 \log^2 n)$ time into a data structure of size $O(r^2 \log n)$ for answering triangle range queries using Lemma 3.7. For each canonical interval $I \in \mathcal{I}_i$, we compute the subset $\mathcal{L}_I \subseteq \mathcal{L}$ of lines that intersect I in $O((n/r) \log n)$ time using the range-searching data structure, because, in the primal plane, I corresponds to a double-wedge and it contains a point of $p_i \in S$ if and only if I intersects ℓ_i. We then compute the intersection points of I and $\mathcal{L}_I$—these are the event points with respect to $\mathcal{L}$ that lie on I. We repeat this step for all intervals in $\mathcal{I}$. The total time spent in computing these intersection points is $O(r^2 \log^2 n + \nu(n/r) \log n)$. We discard those event points whose x-projections do not lie in X. Let Θ denote the set of the remaining event points. We sort Θ in increasing order. The total time spent in computing and sorting Θ is

$$O(r^2 \log^2 n + \nu(n/r) \log n) + O(|\Theta| \log n) = O(r^2 \log^2 n + \nu(n/r) \log^2 n). \tag{3.4}$$

We sweep a vertical line over X from left to right, stopping at the x-values in Θ. For $\theta \in X$, we maintain

$$\mu(\theta) = \langle \mu_1(\mathcal{L}, \theta), \ldots, \mu_b(\mathcal{L}, \theta) \rangle.$$

The vector $\mu(\theta)$ remains the same for all x-values in X lying between two consecutive values in Θ. Suppose we are at a point $\theta \in \Theta$, which belongs to Θ_i. Let I be the connected component of X that contains θ. If θ is the leftmost event point in I, we compute the number of lines in $\mathcal{L}$ intersecting the vertical segment $s_i(\theta)$ (i.e., the points of S lying in the ith bucket of the θ-cut), for $1 \le i \le b$, using the range-searching data structure in time $O((n/r) \log n)$, and set $\mu_i(\mathcal{L}, \theta)$ to this value. We can therefore compute $\mu(\theta)$ for such an event point in $O(b(n/r) \log n)$ time. If θ is not the first event point in I, then we update $\mu(\theta)$ as follows. Suppose $\beta_i(\theta) = \beta_i \cap \ell_j$ and ℓ_j lies above β_i after $\beta_i(\theta)$. Then the point p_j moves from the bucket B_i to B_{i+1} at θ. We decrease $\mu_i(\mathcal{L}, \theta)$ by 1 and increase $\mu_{i+1}(\mathcal{L}, \theta)$ by 1. Similarly, if ℓ_j lies below β_i to the right of $\beta_i(\theta)$, we increase $\mu_i(\mathcal{L}, \theta)$ by 1 and decrease $\mu_{i+1}(\mathcal{L}, \theta)$ by 1. The total time spent by the sweep-line algorithm is

$$O\left(\frac{bn}{r} \log n\right) \cdot |X| + O(|\Theta|) = O\left((b|X| + \nu)\frac{n}{r} \log n\right). \tag{3.5}$$

Computing X and I. Finally, we describe how to compute X and $\mathcal{I}_i$. Set

$$\begin{aligned} l_i &= r - (b-i)m\frac{r}{n} - c\sqrt{r\log n} \quad \text{and} \\ r_i &= im\frac{r}{n} + c\sqrt{r\log n}, \end{aligned}$$

and define

$$\begin{aligned} \sigma &= r_i - l_i = bm\frac{r}{n} - r + 2c\sqrt{r\log n} \\ &\le \frac{br}{n}\left(\frac{n}{b} + 2\Delta\right) - r + 2c\sqrt{r\log n} \\ &\quad \text{(by (3.3))} \\ &\le 2b\Delta\frac{r}{n} + 2c\sqrt{r\log n}. \end{aligned}$$

Recall that X_i is the x-projection of the portion of β_i that lies between $\mathcal{A}_{l_i}(R)$ and $\mathcal{A}_{r_i}(R)$. We compute $\mathcal{A}_{l_i}(R)$ and $A_{r_i}(R)$ and clip the portion of β_i between these two levels; see Figure 7. $\mathcal{A}_{l_i}(R)$ and $A_{r_i}(R)$ have $O(r^{4/3})$ vertices. Since β_i has n vertices, X_i consists of $O(n + r^{4/3})$ connected components and can be computed within this bound. We set $X = \bigcap_{i=1}^{b-1} X_i$; $|X| = O(b(n + r^{4/3}))$. Next, we compute the levels $\mathcal{A}_j(R)$, $l_i \le j \le r_i$. Let M_i be the resulting planar subdivision induced by the edges and vertices of $\mathcal{A}_{l_i}(R), \ldots, \mathcal{A}_{r_i}(R)$. By a result of Dey [6],

$$|M_i| = O(r^{4/3}(r_i - l_i)^{2/3}) = O(r^{4/3}\sigma^{2/3}).$$

Clearly, M_i can be computed in time $O(r\log r + |M_i|) = O(r^{4/3}\sigma^{2/3})$ [8]. Since β_i is an x-monotone polygonal chain and M_i consists of σ edge-disjoint x-monotone polygonal chains, the number of intersection points between β_i and M_i is $O(n\sigma + |M_i|) = O(n\sigma + r^{4/3}\sigma^{2/3})$, and they can be computed within that time bound. We can thus compute the set $\mathcal{I}_i'$ of all canonical intervals of β_i whose x-projections intersect X_i in time $O(n\sigma + r^{4/3}\sigma^{2/3})$. We discard those canonical intervals of $\mathcal{I}_i'$ whose x-projections do not intersect X. The remaining intervals of $\mathcal{I}_i'$ gives the set $\mathcal{I}_i$. Therefore

$$\nu \le \sum_i |\mathcal{I}_i'| = O(b(n\sigma + r^{4/3}\sigma^{2/3})).$$

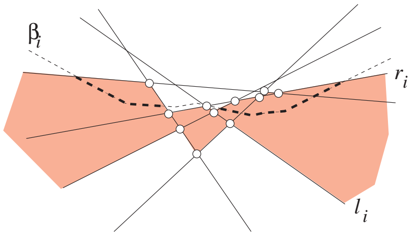

Fig. 7. The bucket line β_i and the planar subdivision M_i. The shaded region denotes M_i.

Repeating this procedure for all bucket lines, the total time in computing $|X|$ and $\mathcal{I}$ is

$$O(b(n\sigma + r^{4/3}\sigma^{2/3})). \tag{3.6}$$

Summing up (3.2), (3.4), (3.5), and (3.6); substituting the values of ν and X; and using the fact that we run the decision algorithm $O(\log n)$ times, the total time in computing n_o is thus

$$\begin{aligned} T(n) &= O((r+b)n) + O(r^2 \log^3 n) + b(n\sigma + r^{4/3}\sigma^{2/3})\frac{n}{r}\log^3 n \\ &\quad + O\left(b^2(n + r^{4/3}) \cdot \frac{n}{r}\log^2 n + b(n\sigma + r^{4/3}\sigma^{2/3})\right) \cdot \frac{n}{r}\log^2 n \\ &\quad + O(b(n\sigma + r^{4/3}\sigma^{2/3}))\log n \\ &= O(rn) + O(b(n\sigma + r^{4/3}\sigma^{2/3}))\frac{n}{r}\log^3 n + O\left(b^2(n + r^{4/3}) \cdot \frac{n}{r}\log^2 n\right). \end{aligned}$$

Substituting the value of σ, we obtain

$$\begin{aligned} T(n) &= O(rn) \\ &\quad + O\left(\frac{bn^2}{r}\left(b\Delta\frac{r}{n} + \sqrt{r\log n}\right)\log^3 n + br^{1/3}n\log^3 n\left(\Delta\frac{br}{n} + \sqrt{r\log n}\right)^{2/3}\right) \\ &\quad + O\left(b^2\left(\frac{n^2}{r} + nr^{1/3}\right) \cdot \log^2 n\right) \\ &= O((b^2\Delta)n\log^3 n) + O\left(rn + \frac{bn^2}{\sqrt{r}}\log^{7/2} n + br^{2/3}n\log^{10/3} n\right) \\ &\quad + O\left(b^2\left(\frac{n^2}{r} + nr^{1/3}\right) \cdot \log^2 n\right). \end{aligned}$$

Setting $r = \lceil b^{2/3}n^{2/3}\log^{7/3} n\rceil$, we obtain the following.

THEOREM 3.8. *There is a Monte Carlo algorithm to compute the optimal uniform projection of a set of n points in* $\mathbb{R}^2$ *onto b equal-size buckets in time*

$$O(\min\{bn^{5/3}\log^{7/3} n + (b^2\Delta)n\log^3 n, n^2\}),$$

with probability at least $1 - 1/n$, *where the optimal value is* $(n/b) + \Delta$. *In particular, our algorithm can detect in* $O(\min\{bn^{5/3}\log^{7/3} n, n^2\})$ *time whether* $\Delta = 0$.

REMARK 3.9. As in Remark 2.8, we can obtain a fast ε-approximation algorithm. We choose a random subset R of size $r = \alpha\,\lceil b/\varepsilon\rceil^2 \log n$, where α is a sufficiently large constant, and compute $r_0 = \min_\theta \Phi(R, \theta)$. Corollary 3.5 and the fact that $n_\mathrm{o} \geq n/b$ implies that $r_\mathrm{o} n/r \leq (1+\varepsilon)\Phi(S)$. From (3.2), the running time of the algorithm is $O((r+b)\cdot n)$ which is $O((b\cdot(1+\varepsilon)/\varepsilon)^2 n\log n)$ for the above choice of r.

4. Two-Dimensional Partitioning. In this section we consider the problem of partitioning a set S of n points in $\mathbb{R}^2$ into "rectangular" buckets. More precisely, given S and an integer $b \geq 1$, we want to compute two families of equally spaced $\sqrt{b}+1$ lines $\mathcal{L} = \{\ell_0, \ldots, \ell_{\sqrt{b}}\}$ and $\mathcal{L}' = \{\ell'_0, \ldots, \ell'_{\sqrt{b}}\}$, so that the following conditions hold:

(i) If the orientation of the lines in $\mathcal{L}$ is $\theta \in [0, \pi/2)$, then the orientation of the lines in $\mathcal{L}'$ is $\pi/2 + \theta$.
(ii) S lies between ℓ_0 and $\ell_{\sqrt{b}}$ as well as between ℓ'_0 and $\ell'_{\sqrt{b}}$.
(iii) Each of the extreme lines $\ell_0, \ell'_0, \ell_{\sqrt{b}}, \ell'_{\sqrt{b}}$ contains at least one point of S.
(iv) The buckets are rectangles B_{ij} defined by $\ell_{i-1}, \ell_i, \ell'_{j-1}, \ell'_j$, for any pair $1 \leq i, j \leq \sqrt{b}$. The maximum number of points in a bucket is minimum.

See Figure 1(iii) for an example. If the slope of lines in $\mathcal{L}$ is θ (and of lines in $\mathcal{L}'$ is $-1/\theta$), we refer to the resulting buckets as the θ-cut. Let $\mu_{ij}(\mathcal{L}, \theta)$ be the number of points in the bucket B_{ij} of the θ-cut.

In the dual setting, the strip formed by the lines ℓ_{i-1} and ℓ_i of the θ-cut is the vertical segment $s_i(\theta)$ as defined in the previous section. Similarly, the dual of the strip formed by ℓ'_{j-1} and ℓ'_j is the segment $s_j(-1/\theta)$. Hence, a point p_k belongs to the bucket B_{ij} of the θ-cut if ℓ_k intersects both $s_i(\theta)$ and $s_j(-1/\theta)$. Let $\mathcal{B} = \{\beta_0, \ldots, \beta_{\sqrt{b}}\}$ be the set of bucket lines as defined in Section 3.2 (a vertical segment $s(\theta)$ whose endpoints lie on the lower and vertical envelopes of $\mathcal{A}(\mathcal{L})$ is partitioned into $\sqrt{b}$ equal segments $s_1(\theta), \ldots, s_{\sqrt{b}}(\theta)$).

As noted by Asano and Tokuyama, we can still compute an optimal solution by a sweep-line algorithm. We sweep two vertical lines L and L'. The line L sweeps the plane from $x = 0$ to $x = +\infty$. When L is at $x = \theta$, L' is at $x = -1/\theta$. We stop when either L or L' crosses an intersection point of $\mathcal{L}$ and $\mathcal{B}$. At each θ, we maintain, for every $1 \leq i, j \leq \sqrt{b}$, the number of points of S that lie in the bucket B_{ij} of the θ-cut, and for each line $\ell_k \in \mathcal{L}$, the pair (i, j) if $p_k \in B_{ij}$. If L passes through an event point lying on β_i, then a line moves from a bucket B_{ij} to $B_{(i+1)j}$ at θ, or vice versa. Similarly, if L' passes through an event point lying on β_j, then a line moves from a bucket B_{ij} to the bucket $B_{i(j+1)}$ at θ, or vice versa. As in Section 3.2, we can update the invariant and the event queue at each event point in $O(\log n)$ time. Hence, we conclude the following:

THEOREM 4.1. *An optimum two-dimensional partitioning in the tight case can be determined in $O((bn+K)\log n)$ time using $O(n)$ storage, where K is the number of event points.*

We can also extend the Monte Carlo algorithm to this problem. If $\Phi(S, \theta) \leq m$, then the strips defined by two consecutive lines of $\mathcal{L}$ (or $\mathcal{L}'$) contain at most $\sqrt{bm}$ points. If we choose a random sample R as in Section 3.3 and define $r_o = \min_\theta \max_{i,j} \mu_{ij}(R, \theta)$ and compute it using the deterministic algorithm, then Lemmas 3.3 and 3.4 and Corollary 3.5 still hold. Corollary 3.6 can now be restated as follows.

COROLLARY 4.2. *Let ξ be a θ-cut so that every bucket of ξ contains at most m points of S. For $1 \leq i \leq \sqrt{b}-1$, let*

$$l_i = r - (\sqrt{b} - i)\sqrt{bm}\frac{r}{n} - c\sqrt{r\log n} \quad \textit{and} \quad r_i = i\sqrt{bm}\frac{r}{n} + c\sqrt{r\log n},$$

where c is an appropriate constant. Then with probability exceeding $1 - 1/n$,

$$l_i \leq \lambda(\beta_i(\xi), R), \lambda(\beta_i(-1/\xi), R) \leq r_i. \tag{4.1}$$

We can now proceed along the same lines as in Section 3.3. In order to determine whether $n_o \leq m$ for a given integer m, we first define the set

$$X = \{\theta \mid l_i \leq \lambda(\beta_i(\theta), R), \lambda(\beta_i(-1/\theta), R) \leq r_i, \ \forall 1 \leq i \leq \sqrt{\beta}\}.$$

We sweep two vertical lines through X as in the deterministic algorithm, but using the ideas from Section 3.3 to compute event points, to move directly from one connected component of X to another, and to compute X and $\mathcal{I}$. Since there are $\sqrt{b}+1$ bucket lines in this case, we have $\nu = \sum_{i=0}^{\sqrt{b}} |\mathcal{I}_i| = O(\sqrt{b}(n\sigma + r^{4/3}\sigma^{2/3}))$, where $\sigma = r_i - l_i \leq 2b\Delta(r/n) + 2c\sqrt{r \log n}$. Carrying out the analysis of Section 3.3 with the new value of ν, we can conclude the following.

THEOREM 4.3. *Given a set of n points in* $\mathbb{R}^2$ *and an integer b, there exists a Monte Carlo algorithm to find an optimal two-dimensional partition in time* $O(\min\{b^{1/2}n^{5/3}\log^{7/3} n + (b^{3/2}\Delta)n\log^3 n, n^2\})$, *with probability at least* $1 - 1/n$, *where the optimal value is* $(n/b) + \Delta$.

5. Conclusions.

We presented bucketing algorithms in one and two dimensions whose running times depend on how "nonuniform" the optimal partition is. Intuitively, the algorithm searches in a small neighborhood of an optimal solution, and the size of this neighborhood depends on the maximum size of a bucket in an optimal partition. We conclude by mentioning a few interesting open problems:

- Can the dependence on b and Δ in the running time of the one-dimensional algorithm be improved?
- Can the $n^{5/3}\log^{O(1)} n$ term in the running time of the uniform projection algorithm be removed?
- We assume in Sections 3 and 4 that the extremal lines contain at least one of the input points. Can this assumption be relaxed without affecting the running time of the algorithms?

References

[1] T. Asano and T. Tokuyama. Algorithms for projecting points to give the most uniform distribution with applications to hashing. *Algorithmica*, 9, (1993), 572–590.

[2] B. Bhattacharya. Usefulness of angle sweep over line sweep. *Proc. Foundations of Software Technology and Theoretical Computer Science*, 1991, pp. 390–419.

[3] T. Chan. Remarks on computing the level in line arrangements. Manuscript, 1999.

[4] D. Comer and M. J. O'Donnell. Geometric problems with applications to hashing. *SIAM Journal on Computing*, 11 (1982), 217–226.

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

[6] T. K. Dey. Improved bounds for planar k-sets and related problems. *Discrete and Computational Geometry*, 19 (1998), 373–382.

[7] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, Heidelberg, 1987.

[8] S. Har-Peled. Taking a walk in a planar arrangement. *Proc.* 40*th IEEE Annual Symposium on Foundations of Computer Science*, 1999, pp. 100–110.

[9] J. Hershberger and S. Suri, A pedestrian approach to ray shooting: Shoot a ray, take a walk, *Journal of Algorithms*, 18 (1995), 403–431.

[10] J. Matoušek. Efficient Partition trees. *Discrete and Computational Geometry*, 8 (1992), 315–334.

[11] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, 1995.

[12] J. Pach and P. K. Agarwal. *Combinatorial Geometry*. Wiley, New York, 1995.