International Journal of Computational Geometry & Applications © World Scientific Publishing Company

AN EFFICIENT DIVIDE-AND-CONQUER APPROXIMATION ALGORITHM FOR PARTITIONING INTO D-BOXES*

TEOFILO F. GONZALEZ,

MOHAMMADREZA RAZZAZI[†],

Department of Computer Science, University of California, Santa Barbara, California 93106-5110, USA

and

SI-QING ZHENG

Computer Science Department, Louisiana State University, Baton Rouge, Louisiana 70803-4020, USA

> Received (received November 23, 1990) Revised (revised March 2, 1992) Communicated by K. Mehlhorn

ABSTRACT

Let P be a set of n points located inside a d-box (rectangle if d = 2) R. We study the problem of partitioning R into d-boxes by introducing a set of hyperplane (line if d = 2) segments of least total (d-1)-volume (length if d = 2). Each of the resulting d-boxes in a valid partition cannot contain points from P as interior points. Since this problem is computationally intractable (NP-hard) even when d = 2, we present an efficient approximation algorithm for its solution. The partition generated by our approximation algorithm is guaranteed to be within 2d times the optimal solution value. We also present a problem instance for each $d \ge 2$ for which the approximation bound is tight for the algorithm. The time complexity for our algorithm is $O(dn \log n)$.

Keywords: Approximation algorithms, d-box partitions, minimize (d-1)-volume, multidimensional computational geometry.

1. Introduction

We present efficient approximation algorithms for the $RG-P_2$ problem and its generalization to *d*-dimensional Euclidean space (E^d) . The input to the $RG-P_2$ problem is a set of points, P, in E^2 inside a rectangle R. The problem consists of introducing a set of (orthogonal) line segments with least total length to partition

^{*}A condensed version of this paper appears in the Proceedings of the Second Canadian Conference on Computational Geometry, August 1990, pp. 214 - 217.

[†]Present Adress: Department of Computer Science, Amir Kabir University of Technology, Tehran, Iran.

R into rectangles without elements from P as interior points. The $RG-P_2$ problem is an NP-hard problem.¹ Several approximation algorithms for the $RG-P_2$ problem have been developed.^{2,3,4,5,6} Table 1 summarizes the currently best approximation algorithms for the $RG-P_2$ problem.

| Approximation Bound | Time Complexity Bound | Method |
|---------------------|-----------------------|-----------------------------------|
| $3 + \sqrt{3}$ | $O(n \ log \ n)$ | divide and conquer ^{2,7} |
| 3 | $O(n^4)$ | ${ m transformation}^3$ |
| 1.75 | $O(n^5)$ | dynamic programming ⁴ |

Table 1. Currently best approximation algorithms for partitioning a rectangle

A more general version of the problem is when R has holes instead of points. This problem arises in VLSI design where it models the problem of partitioning a routing region into channels.⁸ Approximation algorithms for this more general problem exist.^{9,10,11,7} Levcopoulos' algorithms^{11,7} are the ones with the smallest approximation bound. His fastest algorithm⁷ invokes as a sub-procedure the procedure for the RG- P_2 problem given in Ref. 2. One may improve Levcopoulos algorithm⁷ by applying instead the algorithm presented in this paper.

Let P be a set of points located inside a d-box R. The $RG-P_d$ problem, which is a generalization of the $RG-P_2$ problem to d dimensions, consists of partitioning Rinto d-boxes by introducing a set of hyperplane segments of least total (d-1)-volume. Each of the resulting d-boxes in a valid partition cannot contain points from Pas interior points. A dynamic programming approximation algorithm based on guillotine partitions has been analyzed.⁶ The algorithm takes $O(dn^{2d+1})$ time and generates solutions within $2d-4+\frac{4}{d}$ times the optimal solution value. An application for the $RG-P_3$ is discussed in Ref. 6.

We use V(I) to represent the total (d-1)-volume of the hyperplane segments in the solution for problem instance I generated by our algorithm, and opt(I) to represent the corresponding one in an optimal solution. In this paper we present a divide-and-conquer approximation algorithm for the $RG-P_d$ problem that takes $O(dn \ log \ n)$ time and generates solutions within 2d times the optimal solution value, i.e. for every I, $V(I) \leq 2d \ opt(I)$. For d = 2, the solutions generated by our algorithm are similar to the ones generated by previous algorithms.². The main difference between the result in the paper and the one in Ref. 2 is that the new approximation bound is smaller (4 instead of $3 + \sqrt{3}$ for d = 2), the new algorithm is simpler (we only introduce one cut at each step), the proof is much simpler than the previous one (there are fewer cases and the proof for each case is simpler), and the new algorithm is more general in the sense that it generates solutions within 2d for all $d \geq 2$, rather than restricting to problems in 2-dimensional space. With respect to the results in Ref. 6, the algorithm in this paper is faster $(O(dn \log n))$ instead of $O(dn^{2d+1})$, but the algorithm in Ref.6 always generates solutions which are not farther from optimal than the ones generated by our algorithm.

2. The Algorithm and its Analysis

An $RG-P_d$ problem instance is given by I = (o, X, P), where o and X define a d-box boundary R ($o = (o_1, o_2, \ldots, o_d)$ is the "lower-left" corner of the boundary (origin of I), and $X = (X_1, X_2, \ldots, X_d)$ are the dimensions of the boundary) in d-dimensional Euclidean space (E^d) , and $P = \{p_1, p_2, \ldots, p_n\}$ is a nonempty set of points inside d-box R. We define $f_i(I)$, the (d-1)-volume of a facet of R orthogonal to the i-axis, as $\prod_{j \neq i} X_j$, and we define $X_{i,j}$ as $\prod_{i \leq k \leq j} X_k$. We shall refer to the d dimensions (or axes) of E^d by the integers $1, 2, \ldots, d$ (in 2-space we have the first dimension (x-dimension, x-axis, or 1-axis) and the second dimension (y-dimension, y-axis, or 2-axis)).

Assume without loss of generality that there is at least one point in P. Procedure PARTITION, formally defined below, introduces a *mid-cut* or an *end-cut*. A *mid-cut* is a hyperplane segment orthogonal to the 1-axis that intersects the center of the *d*-box (i.e., it includes point $(o_1 + \frac{X_1}{2}, o_2 + \frac{X_2}{2}, \ldots, o_d + \frac{X_d}{2}))$ and an *end-cut* is a hyperplane segment orthogonal to the 1-axis that contains either one of the "leftmost" or the "rightmost" points in P. By "leftmost" ("rightmost") we mean a point with smallest (largest) first coordinate value. A *mid-cut* is introduced when the two resulting subproblems have at least one point each. Otherwise, an *end-cut* is introduced. The *end-cut* contains the leftmost point if such a point is not located to the left of the center of the *d*-box, otherwise the *end-cut* contains the rightmost point. The procedure is then applied recursively to the nonempty resulting subproblems.

Fig. 1. Subinstances generated by procedure PARTITION for d = 3.

PROCEDURE PARTITION(o, X, P);

\mathbf{begin}

Relabel the dimensions so that $X_1 \ge X_2 \ge \cdots \ge X_d$; /* Extreme care must be taken after this step since the axes have been */ /* relabeled. For clarity we do not include all the details needed because */ /* of the relabeling. As we shall see later on, relabeling is not required, */ /* we only need the largest X_i . The relabeling step was introduced to */ */ /* simplify the correctness proof. $P_1 \leftarrow \{p_k \mid p_k \in P \text{ and } q_1 < o_1 + \frac{X_1}{2}, \text{ where } p_k = (q_1, q_2, \dots, q_d)\}; \\ P_2 \leftarrow \{p_k \mid p_k \in P \text{ and } q_1 > o_1 + \frac{X_1}{2}, \text{ where } p_k = (q_1, q_2, \dots, q_d)\};$ \mathbf{case} $:P_1 \neq \emptyset$ and $P_2 \neq \emptyset$: /* introduce a *mid-cut* */ Introduce the hyperplane segment orthogonal to the 1-axis that partitions R through its center; $\begin{array}{l} \text{PARTITION}(o, (\frac{X_1}{2}, X_2, \dots, X_d), P_1);\\ \text{PARTITION}((o_1 + \frac{X_1}{2}, o_2, \dots, o_d), (\frac{X_1}{2}, X_2, \dots, X_d), P_2);\\ :\textbf{else:} \ /^* \ \text{introduce an } end-cut \ */ \end{array}$ Let c be the coordinate value along the 1-axis of a point in P with smallest $| c - (o_1 + \frac{X_1}{2}) |;$ Introduce the hyperplane segment orthogonal to the 1-axis that partitions R through the points with 1-coordinate value equal to c; Delete from P_1 and P_2 all the points located along the *end-cut*; if $P_1 \neq \emptyset$ then PARTITION($(o_1, o_2, \ldots, o_d), (c - o_1, X_2, \ldots, X_d), P_1$); if $P_2 \neq \emptyset$ then PARTITION $((c, o_2, \ldots, o_d), (X_1 - (c - o_1), X_2, \ldots, X_d), P_2);$ endcase

 \mathbf{end}

Ì

It is easy to verify that Figure 1 represents all the possible outcomes of one step in the recursive process of our algorithm for d = 3. A region labeled EMPTY represents a subinstance without interior points. We use X'_1 and X''_1 to represent the length along the 1-axis of the two resulting subinstances $(I_1 \text{ and } I_2)$, respectively.

Our lower bound function, LB(I), is defined by taking a "portion" of the (d-1)-volume at each step of our recursive algorithm. We define recursively the function LB(I) as follows:

| ĺ | $f_1(I)$ | (a) An end-cut is introduced, |
|------------------|--|---|
| | $LB(I_1) + LB(I_2)$ | $P_1 = \emptyset$ and $P_2 = \emptyset$ (b) A mid-cut is introduced, |
| $B(I) = \langle$ | | $P_1 \neq \emptyset \text{ and } P_2 \neq \emptyset$ |
| | $LB(I_1) + min\{f_1(I), X_1 X_{3,d}\}$ | (c) An end-cut is introduced, $P_1 \neq \emptyset$ and $P_2 = \emptyset$ |
| | $LB(I_2) + min\{f_1(I), X'_1X_{3,d}\}$ | (d) An end-cut is introduced, $P_1 = \emptyset$ and $P_2 \neq \emptyset$ |

Lemma 1 For any problem instance I, $LB(I) \leq opt(I)$.

Proof. To prove our lower bound we construct a *d*-box partition by following a procedure similar to procedure PARTITION, which we call MOD-PARTITION. The two differences between these procedures are: when an end-cut is introduced by PARTITION and $P_1 = P_2 = \emptyset$, MOD-PARTITION returns without introducing a cut; and when PARTITION introduces an end-cut and $P_1 \cup P_2 \neq \emptyset$, MOD-PARTITION associates the points in the end-cut with the empty d-box generated at this step. As a result of this modification, MOD-PARTITION constructs a d-box partition in which each d-box either contains a set of points inside it, all of which are located on a hyperplane orthogonal to one of the axes; or the d-box contains no interior points, but there is at least one point on its boundary associated with the d-box. In the former case it is simple to see that any d-box partition must have a hyperplane segment inside that d-box including the interior points with (d-1)-volume at least equal to the lower bound given in case (a) of the definition of LB; and in the latter case any d-box partition must have a hyperplane segment inside or on the facets of the d-box that includes the point associated with it with (d-1)-volume at least equal to the lower bound given in case (c) or (d) in the definition of LB. Therefore, for any problem instance I any d-box partition has (d-1)-volume at least equal to LB(I). This completes the proof of the lemma. \Box

We define USE(I) to be the (d-1)-volume of the hyperplane segments introduced during the first call to procedure PARTITION(I). I.e., if $P = \emptyset$ then USE(I) = 0; otherwise, $USE(I) = V(I) - V(I_1) - V(I_2)$. Let I_1, I_2, \ldots, I_m be the problem instances encountered at all levels of the recursive process (including instance I) when invoking PARTITION(I). Clearly, $V(I) = \sum_{j=1}^{m} USE(I_j)$.

Assume $X_1 \ge X_2 \ge \cdots \ge X_d > 0$ (or equivalently, $f_1(I) \le f_2(I) \le \cdots \le f_d(I)$). For convenience we define $X_0 = X_1$ and $X_{d+1} = \frac{X_d}{3}$. Note that $X_1 \le 2X_1$ and $X_d > 2X_{d+1}$ for all I. A problem instance I = (o, X, P) is said to be of type i $(0 \le i < d)$ if $2X_{i+1} \ge X_1 > 2X_{i+2}$ (or equivalently, $\frac{f_{i+1}(I)}{2} \le f_1(I) < \frac{f_{i+2}(I)}{2}$). By the definition of type, it is simple to show that each problem instance is of exactly one type. We define the carry function for a problem instance I of type i as:

$$C(I) = (d-i) f_{i+1}(I) + \sum_{j=1}^{i} f_j(I).$$

One may visualize the analysis of our algorithm as follows. Whenever a hyperplane segment is introduced by the algorithm (*mid-cut* or *end-cut*) it is colored red, and when a lower bound from LB(I) is "identified" a hyperplane segment with such (*d*-1)-volume is marked blue. Our approach is to bound the sum of the (*d*-1)-volume of all the red segments by 2*d* times the sum of the (*d*-1)-volume of the blue segments. The carry function *C* corresponds to the (*d*-1)-volume of some red segments introduced at previous steps which have not yet been accounted for by blue segments. To establish our approximation bound we prove that for all problem instances *I*, $V(I) + C(I) \leq 2d \cdot LB(I)$. Clearly, this statement is stronger than what we need to prove. Let us establish some preliminary bounds before proving our algorithm's approximation bound.

Lemma 2 For any problem instance I

(i) If $X_k \leq 2X_j$, then $f_k(I) \geq \frac{f_j(I)}{2}$; and (ii) $C(I) < (2d-1) f_1(I)$.

Proof. The proofs follow directly from our definitions. \Box

Lemma 3 For any problem instance I, $V(I) + C(I) \le 2d \cdot LB(I)$.

Proof. The proof is by induction on the number of points in P (remember that I = (o, X, P)).

Basis: Set P contains exactly one point.

Clearly, in this case the procedure introduces a hyperplane segment that intersects the single point in P (this corresponds to the introduction of an *end-cut* with $P_1 = P_2 = \emptyset$). By this observation, Lemma 2(ii), and the definition of LB(I), we know that $V(I) = USE(I) = f_1(I)$, $C(I) \leq (2d-1) f_1(I)$, and $LB(I) = f_1(I)$. Hence, $V(I) + C(I) \leq 2d LB(I)$.

Induction hypothesis: Assume the lemma holds when the number of points in P is less than m.

Induction step: Prove the lemma holds when the number of points in P is m > 1. There are three cases depending on the type of cut introduced by the algorithm.

Let i be the type of I.

Case 1: The algorithm introduces a *mid-cut*, and both I_1 and I_2 contain at least one point (Figure 1(b)).

We know that $V(I) = V(I_1) + V(I_2) + USE(I)$, $LB(I) = LB(I_1) + LB(I_2)$, and $USE(I) = f_1(I)$. Applying the induction hypothesis (note that both I_1 and I_2 have less than m points each) the proof is reduced to showing that

$$f_1(I) + C(I) \le C(I_1) + C(I_2)$$

Since I is of type i, we only need to show that

$$f_1(I) + \sum_{j=1}^i f_j(I) + (d-i) f_{i+1}(I) \le C(I_1) + C(I_2).$$

The dimensions of I_1 are $(X'_1 = \frac{X_1}{2}, X_2, X_3, \ldots, X_d)$, before the relabeling step. Since $X_1 \leq 2X_{i+1}$ and $X'_1 = \frac{X_1}{2}$, we know that $X'_1 \leq X_{i+1}$; and since $X_1 > 2X_{i+2}$ and $X'_1 = \frac{X_1}{2}$, we know that $X'_1 > X_{i+2}$. Therefore, I_1 has ordered dimensions $X_2 \geq X_3 \geq \cdots \geq X_i \geq X_{i+1} \geq X'_1 > X_{i+2} \geq \cdots \geq X_d$. Since $X''_1 = X'_1 = \frac{X_1}{2}$, a similar relation holds for X''_1 . Since $X_2 \leq X_1$ and $X'_1 = X''_1 = \frac{X_1}{2}$, we know that $X_2 \leq 2X'_1 = 2X''_1$. Therefore, I_1 and I_2 are both of type k, for some $k \geq i$. It is simple to verify that:

$$\begin{array}{rcl}
f_{i+1}(I_1) &=& f_1(I); & (1) \\
f_j(I_1) &=& \frac{f_{j+1}(I)}{2}, & \text{for } j < i+1; & (2) \\
f_j(I_1) &=& \frac{f_j(I)}{2}, & \text{for } j > i+1. & (3)
\end{array}$$

We now show that $f_1(I) + \sum_{j=1}^{i} f_j(I) + (d-i)f_{i+1}(I) \leq C(I_1) + C(I_2)$. There are two cases depending on the relative values of i and k.

Subcase 1.1: k = i.

Since $X_1 \leq 2X_{i+1}$, we know by Lemma 2(i) that $f_1(I) \geq \frac{f_{i+1}(I)}{2}$. Therefore,

$$f_1(I) + \sum_{j=1}^i f_j(I) + (d-i) \ f_{i+1}(I) \le f_1(I) + \sum_{j=1}^i f_j(I) + f_{i+1}(I) + 2(d-i-1) \ f_1(I)$$

$$= \sum_{j=1}^{i} f_{j+1}(I) + 2(d-i) f_1(I).$$

By (1) and (2), this is equivalent to $2\sum_{j=1}^{i} f_j(I_1) + 2(d-i) f_{i+1}(I_1)$, which is equal to $C(I_1) + C(I_2)$, since both I_1 and I_2 are of type k = i. Hence,

$$f_1(I) + \sum_{j=1}^i f_j(I) + (d-i) f_{i+1}(I) \le C(I_1) + C(I_2).$$

This completes the proof of this case.

Subcase 1.2: k > i.

Since k > i and $f_j(I) \leq f_k(I)$ for $j \leq k$,

$$f_1(I) + \sum_{j=1}^i f_j(I) + (d-i) \ f_{i+1}(I) \le f_1(I) + \sum_{j=1}^i f_j(I) + \sum_{j=i+1}^k f_j(I) + (d-k) \ f_{k+1}(I) \le f_1(I) + \sum_{j=1}^i f_j(I) + (d-k) \ f_{k+1}(I) \le f_1(I) + \sum_{j=1}^i f_j(I) + \sum_{j=1}^k f_j(I) + (d-k) \ f_{k+1}(I) \le f_1(I) + \sum_{j=1}^i f_j(I) + \sum_{j=1}^k f_j$$

This is equal to $\sum_{j=1}^{i} f_{j+1}(I) + 2f_1(I) + \sum_{j=i+2}^{k} f_j(I) + (d-k) f_{k+1}(I)$. By (1)-(3), the above bound becomes

$$2\sum_{j=1}^{i} f_j(I_1) + 2f_{i+1}(I_1) + 2\sum_{j=i+2}^{k} f_j(I_1) + 2(d-k) f_{k+1}(I_1),$$

which by definition is equal to $C(I_1) + C(I_2)$. Hence,

$$f_1(I) + \sum_{j=1}^i f_j(I) + (d-i) \ f_{i+1}(I) \le C(I_1) + C(I_2).$$

This completes the proof of this case.

Case 2: The algorithm introduces an *end-cut*, and exactly one of the two resulting subproblems has no interior points (Figure 1 (c) and (d)).

Assume without loss of generality that I_2 has no interior points. From the lower bound function and the algorithm we know that

$$LB(I) = LB(I_1) + min\{f_1(I), X_1''X_{3,d}\}$$
 and $V(I) = V(I_1) + USE(I)$

By the induction hypothesis (note that $|P_1| < m$) the proof is reduced to showing that $USE(I) + C(I) \leq C(I_1) + 2d \min\{f_1(I), X_1''X_{3,d}\}$. Clearly, $USE(I) = f_1(I)$, $C(I) \leq (2d-1)f_1(I)$ (Lemma 2(ii)) and $C(I_1) > 0$. Therefore, the above relation holds when $f_1(I) \leq X_1''X_{3,d}$. To complete the proof we show that

$$USE(I) + C(I) \le C(I_1) + 2dX_1''X_{3,d}$$

when $X_1'' X_{3,d} \leq f_1(I)$. Since $X_1'' X_{3,d} \leq f_1(I)$, we know that $X_1'' \leq X_2$. Clearly, instance I_2 has dimensions $X_1'' \leq X_2 \geq \cdots \geq X_d$, and $X_1'' X_{3,d} = f_2(I_2)$. So we need to show that $f_1(I) + C(I) \leq C(I_1) + 2d f_2(I_2)$. From the algorithm it is simple to verify that $X_1' \leq \frac{X_1}{2}, X_1'' \geq \frac{X_1}{2}$ and $USE(I) = f_1(I)$.

Since $\frac{X_1}{2} \leq X_1''$ and $X_1'' \leq X_2$, it must be that $X_1 \leq 2X_2$. So *i*, the type of *I*, must be greater than zero. Since $C(I) = (d-i)f_{i+1}(I) + \sum_{j=1}^{i} f_j(I)$, our proof reduces to showing that

$$f_1(I) + (d-i)f_{i+1}(I) + \sum_{j=1}^i f_j(I) \le C(I_1) + 2d f_2(I_2).$$

Since $X_1 \leq 2X_{i+1}$ and $X'_1 \leq \frac{X_1}{2}$, we know that $X'_1 \leq X_{i+1}$. Therefore, the ordering for the dimensions of I_1 is

$$X_2 \geq X_3 \geq \cdots \geq X_l \geq X'_1 > X_{l+1} \geq \cdots \geq X_d,$$

for some $i + 1 \le l \le d$. Since $X_2 \le X_1$ and $X_1 \le 2X_{i+1}$, we know that $X_2 \le 2X_{i+1}$ and I_1 is type k, for some $k \ge i - 1 \ge 0$.

Let I'_1 be the instance with dimensions (X'_1, X_2, \ldots, X_d) . Note that the difference between I'_1 and I_1 is that the dimensions are not sorted in I'_1 . Since $f_1(I) \leq f_2(I) \leq f_2(I'_1) + f_2(I_2) \leq f_{i+1}(I'_1) + f_2(I_2)$; $f_j(I) = f_j(I'_1) + f_j(I_2)$ for $j \neq 1$ as $X'_1 + X''_1 = X_1$; and $f_1(I) \leq f_2(I) \leq 2f_2(I_2)$ because $X''_1 \geq \frac{X_1}{2}$, we know that

$$f_{1}(I) + (d-i)f_{i+1}(I) + \sum_{j=1}^{i} f_{j}(I)$$

$$\leq f_{i+1}(I_{1}') + f_{2}(I_{2}) + (d-i)(f_{i+1}(I_{1}') + f_{i+1}(I_{2})) + 2f_{2}(I_{2}) + \sum_{j=2}^{i}(f_{j}(I_{1}') + f_{j}(I_{2}))$$

$$= f_{i+1}(I_{1}^{'}) + (d-i)f_{i+1}(I_{1}^{'}) + \sum_{j=2}^{i} f_{j}(I_{1}^{'}) + 3f_{2}(I_{2}) + (d-i)f_{i+1}(I_{2}) + \sum_{j=2}^{i} f_{j}(I_{2})$$

Since $X_2 \leq 2X_j$, for $3 \leq j \leq i-1$, then by Lemma 2(i), we know that $f_2(I_2) \geq \frac{f_j(I_2)}{2}$. Therefore, the sum of the terms involving I_2 is less than or equal to $2d f_2(I_2)$. The sum of the remaining terms is equal to $\sum_{j=1}^{i-1} f_j(I_1) + f_i(I_1) + (d-i)f_i(I_1)$. Since $k \geq i-1$, and $f_i(I_1) \leq f_j(I_1)$, for j > i, the summation is at most equal to $C(I_1)$. Hence, $C(I) + f_1(I) \leq C(I_1) + 2d f_2(I_2)$. This completes the proof for case 2.

Case 3: The algorithm introduces an *end-cut*, and I_1 and I_2 contain no points (case (a) in Figure 1).

The proof for this case is omitted since it is similar to the proof for the basis case. This completes the proof of the lemma. \Box

Let us now establish our main result (Theorem 1), show that the approximation bound is tight (Theorem 2), and explain implementation details needed to establish our time complexity bound (Theorem 3).

Theorem 1 For any instance of the RG-P_d problem, algorithm PARTITION generates a solution such that $V(I) \leq 2d$ opt(I).

Proof. The proof follows from the definition of the carry function, and Lemmas 1 and 3. \Box

Theorem 2 For any $\epsilon > 0$, there exists an RG-P_d problem instance I such that $V(I) = (2d - \epsilon) opt(I)$.

Proof. Let $k = \lceil \log(\frac{d}{\epsilon}) \rceil$. The origin of the *d*-box *R* is at point $(0, 0, \ldots, 0)$, and each side has length 2^k . Partition *d*-box *R* into 2^{dk} identical interior *d*-boxes. In each interior *d*-box there are 2^{d-1} points. These points are located at

$$(i_1 + q_1, i_2 + q_2, \dots, i_{d-1} + q_{d-1}, i_d + \frac{1}{2})$$

for integer $r, 0 \le r \le 2^{d-1} - 1$; where q_j is defined as $\frac{1}{3}$ if the leftmost *jth* bit in the *d*-bit binary representation of r is 0, and $\frac{2}{3}$ otherwise, for $1 \le j \le d-1$; and (i_1, i_2, \ldots, i_d) is the "lower left" corner of the interior *d*-box. Therefore, the total number of points is $n = 2^{d-1} \cdot 2^{dk}$.

A problem instance for k = 2 and d = 2 is given in Figure 2a. The solution generated by procedure PARTITION is given in Figure 2b and an optimal solution is given in Figure 2c. For any $k \ge 0$ and $d \ge 2$, our algorithm introduces hyperplane segments for each grid segment, and in each interior *d*-box one hyperplane orthogonal to each axis is introduced. Therefore,

$$V(I) = d \cdot (2^k - 1) \cdot 2^{(d-1)k} + d \cdot 2^k \cdot 2^{(d-1)k}.$$

It is simple to show that a partition with 2^k hyperplane segments orthogonal to the *d*-axis covers all points. Therefore, $opt(I) = 2^k \cdot 2^{(d-1)k}$. Hence, $\frac{V(I)}{opt(I)} = 2d - \frac{d}{2^k}$. The proof of the theorem now follows from the fact that $k = \lceil \log(\frac{d}{2}) \rceil$. \Box

Note that algorithm PARTITION can be trivially modified so that when a problem instance has all points along the same hyperplane and the (d-1)-volume of such hyperplane segment is not too large compared to $f_1(I)$, then such segment is introduced. On the above example such modified algorithm will outperform the previous algorithm and the approximation bound will be smaller. However, there are examples for which the above modified algorithm will achieve the same worst case approximation bound as PARTITION. Let us consider the case when d = 2. Instead of all points having y-coordinate value $\frac{1}{2}$, they have $i_2 + \epsilon$ if i_2 is odd and $i_2 + 1 - \epsilon$ if i_2 is even. In this case the ratio between the approximation solution generated by the modified algorithm and the optimal solution value is similar to the one in the previous case. Examples achieving a bound of $2d - \epsilon$ for all $d \geq 3$ can be constructed by following a similar strategy.

Fig. 2. Problem instance in proof of Theorem 2 when k = 2.

A straight forward implementation of algorithm PARTITION requires $O(dn^2)$ time, where *n* is the number of points in *P*. Let us show that procedure PARTITION can be implemented to take $O(dn \log n)$ time. In what follows a point *p* in E^d is represented by $p = (x_1(p), x_2(p), \ldots, x_d(p))$. In Ref. 7, two implementations that take $O(n \log n)$ time for the rectangular partition algorithm given in Ref. 2 are presented. These techniques can be easily extended to procedure PARTITION so that it takes $O(dn \log n)$ time. Let us briefly describe the generalization of one of the methods given in Ref. 7. To avoid trivial details, we assume as in Ref. 7 that $x_k(p) \neq x_k(q), 1 \leq k \leq d$, for any two distinct points *p* and *q* in *P*. In the preprocessing step, a multi-linked data structure *L* for *P* is constructed. Each node in *L* is a point record, which represents a point *p* in *P* and consists of 4*d* fields: $x_k(p), i_k(p), \ llink_k(p), \ rlink_k(p), \ 1 \leq k \leq d$. The k-th coordinate value of p is $x_k(p)$, and $i_k(p) = j$ if $x_k(p)$ is the j-th smallest value in the set $\{x_k(q) \mid q \in P\}$. The pointers $llink_k$ and $rlink_k$ are used to implement an ordered doubly linked list L_k for all $x_k(p)$. The point records are linked in increasing (decreasing) order on x_k by the $rlink_k$ ($llink_k$) fields. Clearly, such a data structure can be constructed in $O(dn \log n)$ time.

Let us consider the operations performed at each step in algorithm PARTITION. The purpose behind the relabeling of X_i , $1 \le i \le d$, is to simplify the presentation of the algorithm and the analysis of approximation bound. The real concern is finding $X_k = \min\{X_j \mid 1 \le j \le d\}$. This can be easily determined in O(d) time. Then, the major operation is to partition the points in P into two subsets P_1 and P_2 , where P_1 contains points p such that $x_k(p) < o_k + \frac{X_k}{2}$, and P_2 contains points p such that $x_k(p) > o_k + \frac{X_k}{2}$. This can be done in $O(\min\{|P_1|, |P_2|\})$ time by scanning L_k from its left end and right end toward the middle of L_k in such a way that a one step advance from the left to the right is accompanied by one step advance from the right to the left. Assume that $n \ge |P| = m \ge |P_1| = m - u \ge |P_2| = u$, and $u \leq m/2$. From the above stated scanning procedure it is simple to show that P can be partitioned into P_1 and P_2 in O(u) time. Since the point records for P are doubly-linked in each L_i , the point records corresponding to points in P_2 can be removed from L in O(du) time, so the remaining records form the multi-linked structure L^1 for P_1 . What remains to be constructed is the multi-linked structure L^2 for points in P_2 . By the radix sort algorithm given in Ref. 12, u integers in the range [1, n] can be sorted in $O(u \log \log_u n)$ time. Hence, constructing L^2 can be done in $O(du \log \log_u n)$ time by sorting on i_k , $1 \le k \le d$. Let T(n, n) be the total time required by algorithm PARTITION. From the above observations we know that

$$T(m,n) = \begin{cases} cd, & m = 1 \\ max\{cdu \ log \ log_u \ n + T(u,n) + T(m-u,n) \ | \ 1 \le u \le \frac{m}{2}\}, & m > 1, \end{cases}$$

where c is a constant. Using analysis similar to that in Ref. 7, $T(n) = O(dn \log n)$. Therefore, we have the following result.

Theorem 3 The above implementation of algorithm PARTITION (including preprocessing) takes $O(dn \log n)$ time.

Proof. By the above discussion. \Box

3. Discussion

In practical situations one could execute several variations of the algorithm presented in this paper (e.g., the one in Ref. 2) and then select a solution with least (d-1)-volume. We conjecture that an approximation algorithm based on this approach generates solutions that are very close to optimal. However, proving a smaller bound for this approach seems to be difficult. A postprocessing procedure that transforms the solution generated by PARTITION into a feasible solution in which each hyperplane segment includes at least one point can be easily constructed.

In general it will not generate better solutions, but in many cases the solution generated by our algorithm will be improved.

Acknowledgements

The authors which to thank an anonymous referee for fine-tuning our carry function so as to simplify the proof of Lemma 3. This research was supported in part by the National Science Foundation under Grant DCR-8503163

References

- A. Lingas, R. Y. Pinter, R. L. Rivest, and A. Shamir, "Minimum Edge Length Partitioning of Rectilinear Polygons," Proc. 20th Annual Allerton Conference on Communication, Control, and Computing, Monticello, Illinois, Oct. 1982.
- T. F. Gonzalez, and S. Q. Zheng, "Bounds for Partitioning Rectilinear Polygons", Proc. Symp. Computational Geometry, June 1985, pp. 281-287, (also appears as Technical Report #85-22, CS Department, UCSB, Dec. 1985).
- 3. T. F. Gonzalez, and S. Q. Zheng, "Approximation Algorithms for Partitioning Rectilinear Polygons with Interior Points," *Algorithmica*, 5, January 1990, 11-42.
- 4. T. F. Gonzalez, and S. Q. Zheng, "Improved Bounds for Rectangular and Guillotine Partitions," *Journal of Symbolic Computation*, 7, 1989, pp 591-610.
- D. Z. Du, L. Q. Pan and M. T. Shing, "Minimum Edge Length Guillotine Rectangular Partition," Technical Report, MSRI 02418-86, Jan. 1986.
- T. F. Gonzalez, M. Razzazi, M. Shing and S. Q. Zheng, "On Optimal d-Guillotine Partitions Approximating Hyperrectangular Partitions," Technical Report TR-89-25, CS Department, UCSB, October 1989.
- 7. C. Levcopoulos, "Fast Heuristics for Minimum Length Rectangular Partitions of Polygons," Proceedings of the 2nd Computational Geometry Conference, June 1986.
- 8. R. L. Rivest, "The "PI" (Placement and Interconnect) System," Proc. 19th Design Automation Conference, June 1982.
- A. Lingas, "Heuristics For Minimum Edge Length Rectangular Partitions of Rectilinear Figures," 6th GI-Conference, Dortmund, 1983, Lecture Notes in Computer Science 195 (Springer-Verlag).
- D. Z. Du and Chen Y. M., "On Fast Heuristics for Minimum Edge Length Rectangular Partition," Technical Report, MSRI 03618-86, Feb. 1986.
- 11. C. Levcopoulos, "Minimum Length and Thickest-First Rectangular Partitions of Polygons," Proceedings of the 23rd Allerton Conference on Communication, Control and Computing, Monticello, Illinois, Oct. 1985.
- Kirkpatrick, D. G., "An Upper Bound for Sorting Integers in Restricted Ranges", Proc. 18th Allerton Conference on Communication, Control and Computing, Monticello, Illinois, Oct. 1980.



Fig. 1. Subinstances generated by procedure PARTITION for d= 3.



(a) Problem instance with ${\bf k}$ = 2.







(c) Optimal solution.

Fig. 2. Problem instance in proof of Theorem $\ref{eq:kernel}$ when k=2.