54

Minimum-Edge Length Rectangular Partitions

Teofilo F. Gonzalez
University of California, Santa Barbara

Si Qing Zheng
University of Texas at Dallas

54.1 Introduction

In this chapter we discuss approximation algorithms for partitioning a rectangle with interior points. This problem is denoted by *RG-P*, where *RG* stands for *Rectangle* and *P* stands for *Points*. An instance of the *RG-P* problem is a set P of n points inside a rectangle R in the plane. A feasible solution is a *rectangular partition*, which consists of a set of (orthogonal) line segments that partition R into rectangles such that each of the n points in P is on a partitioning line segment. The objective of the *RG-P* problem is to find a rectangular partition whose line segments have least total length. The *RG-P* problem is an NP-hard problem [1].

A more general version of the problem is when R has interior rectilinear holes instead of points. This problem arises in VLSI design where it models the problem of partitioning a routing region into channels [2]. Approximation algorithms for this more general problem exist [3–6]. Levcopoulos' algorithms [4,5] are the ones with the smallest approximation ratio. His fastest algorithm [5] invokes as a subprocedure the divide-and-conquer algorithm for the *RG-P* problem developed by Gonzalez and Zheng [7], which is a preliminary version of the one discussed in this chapter.

The structure of an optimal rectangular partition can be very complex (Figure 54.1 [a]). A restricted version of the *RG-P* problem limits the set of feasible solutions to recursively defined partitions that at each level partition the instance into two subinstances of the *RG-P* problem by introducing a single line segment. A recursive partition with this property is given in Figure 54.1(b). For a given rectangle, such a line segment is called a *cut* of the rectangle in Ref. [7], and a *guillotine cut* in Ref. [8]. A rectangular partition such that at every level there is a guillotine cut is called a *guillotine partition*. By definition every guillotine partition is a rectangular partition, but the converse is not true. For example, the rectangular partition given in Figure 54.1(a) is not a guillotine partition. For the same instance, the structure of an optimal guillotine partition can be much simpler than the structure of an optimal rectangular partition. Furthermore, the concept of guillotine cut is useful for developing approximation algorithms for the *RG-P* problem using divide-and-conquer and dynamic programming techniques. The algorithm given in Ref. [7] finds an approximation to the *RG-P* problem by generating a suboptimal guillotine partition via a divide-and-conquer algorithm, whereas the algorithm given by Du et al. [8] finds an approximation to the *RG-P* problem by generating an optimal guillotine partition via dynamic programming. Gonzalez and Zheng's [7] algorithm takes $O(n \log n)$ time. Du et al.'s [8] algorithm takes $O(n^5)$ time to construct an optimal guillotine partition.

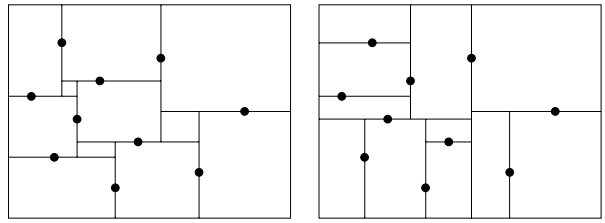

FIGURE 54.1 (a) Optimal rectangular partition. (b) Optimal guillotine partition.

On the other hand, Du et al. [8] showed that the length of an optimal guillotine partition is at most twice the length of an optimal rectangular partition for the *RG-P* problem, but the approximation ratio for the divide-and-conquer algorithm given in Ref. [7] is $3\sqrt{3}$. This ratio was reduced to 4 by using a slightly different divide-and-conquer procedure. A complex proof showing that the length of an optimal guillotine partition is within 1.75 times the length of an optimal rectangular partition is given by Gonzalez and Zheng [9]. Clearly, neither algorithm dominates the other when one takes into account both the time complexity and the approximation ratio.

It is important to note that optimal and near-optimal guillotine partitions are not the only way one can generate solutions with a constant approximation ratio for the *RG-P* problem. Gonzalez and Zheng [10] developed an algorithm with approximation ratio of 3 for the *RG-P* problem that generates a rectangular partition that is not necessarily a guillotine partition. Both the time and approximation ratio for this algorithm are between the ones of the two algorithms mentioned above.

The *RG-P* problem was generalized to d-dimensional Euclidean space. In this generalized problem, R is a d-box and P is a set of points in d-dimensional space, and the objective is to find a set of orthogonal hyperplane segments of least total $(d-1)$-volume that includes all points of P. This is the *d-dimensional RG-P problem*. Approximating an optimal d-box partition by suboptimal and optimal guillotine partitions based on divide-and-conquer and dynamic programming techniques has been established in Refs. [11,12], respectively.

In this chapter, we present two approximation algorithms for the *RG-P* problems. For an *RG-P* instance I, we use $E(I)$ to represent the set of line segments in the solution generated by our algorithm, and $E_{opt}(I)$ the set of line segments in an optimal rectangular partition. We use $L(S)$ to represent the length of the line segments in set S. We first present an $O(n \log n)$-time divide-and-conquer approximation algorithm that generates suboptimal guillotine partitions with total edge length within four times the one in an optimal rectangular partition, that is, for every I, $L(E(I)) \leq 4L(E_{opt}(I))$. We then present a simple proof that shows that an optimal guillotine partition has total edge length that is within two times the one in an optimal rectangular partition, that is, for every I, $L(E(I)) \leq 2L(E_{opt}(I))$. This proof is much simpler than the proof of Ref. [8] for the same bound. We also outline the main ideas of the complex proof in Ref. [9] that establishes that an optimal guillotine partition has edge length that is within 1.75 times the optimal rectangular partition value. One may improve Levcopoulos' [5] algorithm by replacing the algorithm in Ref. [7] by any of the above algorithms.

54.2 A Divide-and-Conquer Algorithm

An *RG-P* problem instance is given by $I = ((X, Y), P)$, where (X, Y) defines the rectangle R (with height Y and width X), and $P = \{p_1, p_2, \ldots, p_n\}$ is a nonempty set of points located inside R. Before we present our algorithm we define some terms and define a way to establish lower bounds for the length of rectangular partitions.

A *partial rectangular partition* is a partition of R into rectangles where not all the points in P are located along the partitioning line segments. Let Q be any partial rectangular partition for problem instance I. A

subset of the points in P is said to be *assigned* to Q if the following three conditions are satisfied:

1. Every rectangle r in Q with at least one point inside it has one such point assigned to it.
2. A rectangle r in Q without points inside it may be assigned at most one point on one of its sides.
3. Two rectangles with a common boundary cannot have both their assigned points on their common boundary.

An assignment of the points in P to Q is denoted by $A(Q)$.

Every rectangle r with a point assigned to it is said to have *value,* $v(r)$, equal to the minimum of the height of r and the width of r. The assignment $A(Q)$ of the partial rectangular partition Q has value equal to the sum of the values of the rectangles that have an assigned point. This value is denoted by $v(A(Q))$.

We now establish that the edge length of a rectangular partition is at least equal to the value of any assignment for any partial rectangular partition of R.

Lemma 54.1

For every problem instance I, partial rectangular partition Q, and assignment $A(Q)$, a lower bound for $L(E(I))$ is given by $v(A(Q))$, that is, $v(A(Q)) \leq L(E(I))$. In particular, $v(A(Q)) \leq L(E_{OPT}(I))$.

Proof

Consider any rectangle r with one or more points in it. By definition one of these points is assigned to the rectangle. Clearly, any partition into rectangles must include line segments inside r with length greater or equal to the minimum of the height of r or the width of r. This is equal to the value $v(r)$.

Consider now any rectangle r without points inside, but with an assigned point. The point assigned to r must be located on one of its sides. Let us say it is on side s. By definition any rectangle with a common boundary to side s of rectangle r does not have its assigned point on this common boundary. Therefore, any rectangular partition that covers the assigned point of r must include line segments in r with length at least equal to the minimum of the height of r or the width of r. This is equal to the value $v(r)$.

Since $v(A(Q))$ is equal to the sum of the values of the rectangles that have been assigned a point, the above arguments establish that $v(A(Q)) \leq L(E(I))$, that is, $v(A(Q))$ is a lower bound for $L(E(I))$. □

We now define our divide-and-conquer procedure to generate a rectangular partition. From this rectangular partition we identify a partial rectangular partition and an assignment of a subset of the points P. Applying Lemma 54.1 we have a lower bound for an optimal rectangular partition. Then we show that the length of the edges in the solution generated is within four times the lower bound provided by the assignment.

Assume without loss of generality that we start with a nonempty problem instance such that $Y \leq X$. Procedure *DC* introduces a *mid-cut* or an *end-cut*, depending on whether or not the rectangle has points to the left and also to the right of the center of R. The cut is along a vertical line that partitions R into R_l and R_r. The set of points in P that are not part of the cut are partitioned into P_l and P_r depending on whether they are inside of R_l or R_r. A *mid-cut* is a vertical line segment that partitions R and includes the center of the rectangle. An *end-cut* is a vertical line segment that partitions R and includes either the "rightmost" or the "leftmost" points in P, depending whether all the points are located to the left or right of the center of R. Procedure *DC* is then applied recursively to the nonempty resulting subproblems, that is, $I_l = ((X_l, Y), R_l)$ and $I_r = ((X_r, Y), R_r)$, where X_l and X_r represent the length along the x-axis of the two resulting subinstances (I_l and I_r), respectively. Note that when a mid-cut is introduced it must be that both P_l and P_r end up being nonempty. For an end-cut at least one of these sets will be empty. When both sets of points are empty the cut is called *terminal end-cut.*

It is easy to verify that Figure 54.2 represents all the possible outcomes of one step in the recursive process of our procedure. A subinstance without interior points is represented by a rectangle filled with diagonal line segments.

Our lower bound function, $LB(I)$, is defined from a partial rectangular partition of the rectangular partition generated by Procedure *DC*. The partial rectangular partition is the rectangular partition generated

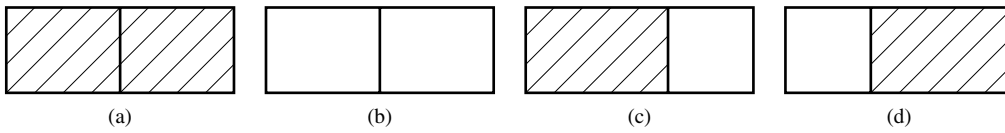

FIGURE 54.2 Subinstances generated by Procedure *DC*.

by Procedure *DC*, except that all the terminal end-cuts are not included. The association for the partial rectangular partition is defined as follows. Every rectangle with points inside it is assigned one of the points inside it. Rectangles without points inside them resulting from a nonterminal end-cut have one of the points along the end-cut assigned to them. By Lemma 54.1, the value of this assignment is a lower bound for the length of an optimal rectangular partition. A recursive definition for the value of the above assignment is given by the following recurrence relation:

$$LB(I) = \begin{cases} Y & \text{(a) A terminal end-cut is introduced, } P_l = \emptyset \text{ and } P_r = \emptyset. \\ LB(I_l) + LB(I_r) & \text{(b) A mid-cut is introduced, } P_l \neq \emptyset \text{ and } P_r \neq \emptyset. \\ \min\{Y, X_l\} + LB(I_r) & \text{(c) A nonterminal end-cut is introduced, } P_l = \emptyset \text{ and } P_r \neq \emptyset \\ LB(I_l) + \min\{Y, X_r\} & \text{(d) A nonterminal end-cut is introduced, } P_l \neq \emptyset \text{ and } P_r = \emptyset. \end{cases}$$

Let $L(E_{DC}(I))$ denote the total length of the set $E_{DC}(I)$ of line segments introduced by Procedure *DC*. We define $USE(I)$ to be the length of the line segments introduced directly by Procedure $DC(I)$, but not by the recursive invocations, that is, when $P = \emptyset$ then $USE(I) = 0$; otherwise, $USE(I) = L(E_{DC}(I)) - L(E_{DC}(I_l)) - L(E_{DC}(I_r))$.

Assume $X \geq Y$. A problem instance $I = ((X, Y), P)$ is said to be *regular* if $X \leq 2Y$, and *irregular* otherwise (i.e., $X > 2Y$). We define the *carry function* C for a problem instance I as

$$C(I) = \begin{cases} 3Y & \text{if } I \text{ is irregular} \\ X + Y & \text{if } I \text{ is regular} \end{cases}$$

One may visualize the analysis of our approximation algorithm as follows. Whenever a line segment (*mid-cut* or *end-cut*) is introduced by Procedure *DC* it is colored red, and when a lower bound from $LB(I)$ is "identified" we draw an "invisible" blue line segment with such length. Our budget is four times the length of the blue line segments, which we must use to pay for all the red line segments. In other words, the idea is to bound the sum of the length of all the red segments by four times the one of the blue segments. The length of the red line segments introduced at previous recursive invocations of Procedure *DC* which have not yet been accounted by previously identified blue segments is bounded above by the carry function C, defined above. In other words, the carry value is the maximum edge length (length of red segments) we could possibly owe at this point. Before proving our result, we establish some preliminary bounds.

Lemma 54.2

For any problem instance I such that $X \geq Y$, $2Y \leq C(I) \leq 3Y$.

Proof

The proof follow from the definition of the carry function. □

Lemma 54.3

For every problem instance I, $L(E_{DC}(I)) + C(I) \leq 4LB(I)$.

Proof

The proof is by contradiction. Let I be a problem instance with the least number of points P that does not satisfy the lemma, that is,

$$L(E_{DC}(I)) + C(I) > 4LB(I) \tag{54.1}$$

Assume without loss of generality that $Y \leq X$. There are three cases depending on the cut introduced by Procedure *DC* when it is initially invoked with I.

Case 1. The procedure introduces a terminal end-cut.
Since $Y \leq X$, and $P_l = P_r = \emptyset$, we know that $LB(I) = Y$. From the way the procedure operates we know $L(E_{DC}(I)) = USE(I) = Y$. Substituting into Eq. (54.1), we know that $C(I) > 3Y$. But this contradicts Lemma 54.2. So it cannot be that the algorithm introduces a terminal end-cut when presented with instance I.

Case 2. The procedure introduces a mid-cut.
Since a *mid-cut* is introduced, both P_l and P_r must be nonempty and since both I_l and I_r have fewer points than P, we know they satisfy the conditions of the lemma. Combining the conditions of the lemma for I_l and I_r we know that

$$L(E_{DC}(I_l)) + L(E_{DC}(I_r)) + C(I_l) + C(I_r) \leq 4LB(I_l) + 4LB(I_r)$$

By definition, $L(E_{DC}(I)) = L(E_{DC}(I_l)) + L(E_{DC}(I_r)) + USE(I)$ and $LB(I) = LB(I_l) + LB(I_r)$. Since $Y \leq X$ we know $USE(I) = Y$. Substituting these equations into Eq. (54.1) we have

$$Y + C(I) > C(I_l) + C(I_r) \tag{54.2}$$

There are two cases depending on whether I is regular or irregular.

Subcase 2.1. I is regular.
By definition $C(I) = X + Y$. Substituting into Eq. (54.2),

$$Y + C(I) = X + 2Y > C(I_l) + C(I_r)$$

Since I is regular and the procedure introduces a mid-cut, both I_1 and I_2 must also be regular. Therefore, $X + 2Y = C(I_l) + C(I_r)$. A contradiction. So it cannot be that I is regular.

Subcase 1.2. I is irregular.
Since $Y \leq X$ and I is irregular, we know that $Y + C(I) = 4Y$. Substituting into Eq. (54.2) we have $4Y > C(I_l) + C(I_r)$. Since I is irregular and the algorithm introduces a mid-cut, it must be that $X_l = X_r \geq Y$. But by Lemma 54.2, $C(I_l) \geq 2Y$ or $C(I_r) \geq 2Y$. A contradiction. So it cannot be that the procedure introduces a min-cut.

Case 3. The procedure introduces a nonterminal end-cut.
When a nonterminal end-cut is introduced, exactly one of the two resulting subproblems has no interior points (Figure 54.2 [c] and [d]). Assume without loss of generality that I_r has no interior points. From the lower bound function and the procedure we know that

$$LB(I) = LB(I_l) + \min\{Y, X_r\} \text{ and } L(E_{DC}(I)) = L(E_{DC}(I_l)) + USE(I)$$

Since instance I_l has fewer points than I, it then follows that $L(E_{DC}(I_l)) + C(I_l) \leq 4LB(I_l)$. Clearly, $USE(I) = Y$. Substituting these inequalities into Eq. (54.1) we know that

$$Y + C(I) > C(I_l) + 4\min\{Y, X_r\} \tag{54.3}$$

By Lemma 54.2 we know $C(I) \leq 3Y$ and $C(I_l) > 0$. So Eq. (54.3) is $4Y > 4\min\{Y, X_r\}$. Therefore, it cannot be that $Y \leq X_r$ as otherwise there is a contradiction. It must then be that $X_r < Y$. Substituting into Eq. (54.3)

$$Y + C(I) > C(I_l) + 4X_r \tag{54.4}$$

Instance I is regular because $X_r < Y$ and $X_r \geq \frac{X}{2}$ implies $X < 2Y$. Substituting $C(I) = X + Y$ into Eq. (54.4) we know

$$X + 2Y > C(I_l) + 4X_r \tag{54.5}$$

If I_l is regular, then substituting $C(I_l)$ into Eq. (54.5) we know that $X + 2Y > X_l + Y + 4X_r$. Since $X_l + X_r = X$, Eq. (54.5) becomes $Y > 3X_r$. But we know that $X_r \geq X/2$ and $X \geq Y$. So $X_r \geq Y/2$. A contradiction. So it must be that I_l is irregular.

In contrast, if I_l is irregular, then since $X_l \leq Y$ substituting $C(I_l)$ into Eq. (54.5) we know that $X + 2Y > 3X_l + 4X_r$. Since $X_l + X_r = X$, we know that $2Y > 2X + X_r$. But we know that $X \geq Y$ and $X_r > 0$. A contradiction.

This completes the proof of this case and the lemma. □

We establish the main result (Theorem 54.1), which shows that the approximation bound is tight (Theorem 54.2), and explain implementation details needed to establish the time complexity bound for procedure *DC* (Theorem 54.3).

Theorem 54.1

For any instance of the RG-P problem, algorithm DC generates a solution such that $L(E_{DC}(I)) \leq 4\, L(E_{opt}(I))$.

Proof

The proof follows from Lemmas 54.2 and 54.3. □

We now show that the approximation bound is asymptotically tight, that is, $L(E_{DC}(I))$ is about $4L(E_{opt}(I))$. The problem instance we use to establish this result has the property that $LB(I) = L(E_{opt}(I))$. Our approach is a standard one that begins with small problem instances and then combines them to build larger ones. As the problem instances become larger, the approximation ratio for the solution generated by Procedure *DC* increases. The analysis just needs to take into consideration a few steps performed by the procedure and the previous analysis for the smaller problem instances.

We define problem instances P_i, for $i \geq 0$ as follows: Problem instance P_i consists of a rectangle of size 2^i by 2^i. The instance P_1 contains two points. Figure 54.3(a) and Figure 54.3(b) depicts $E_{DC}(P_1)$ and $E_{opt}(P_1)$, respectively. Clearly, $L(E_{DC}(P_1)) = 4$ and $L(E_{opt}(P_1)) = 2$. In this case the approximation ratio is 2. Instance P_2 is a combination of four instances of P_1. Figure 54.3(c) and Figure 54.3(d) depicts $E_{DC}(P_2)$ and $E_{opt}(P_2)$, respectively. Clearly, $L(E_{DC}(P_2)) = 2^3 + 4L(E_{DC}(P_1)) = 24$ and $L(E_{opt}(P_2)) = 4L(E_{opt}(P_1)) = 8$. The ratio is 3. Problem instance P_3 combines four instances of P_2 as shown in Figure 54.3(c). Figure 54.3(e) and Figure 54.3(f) depicts $E_{DC}(P_3)$ and $E_{opt}(P_3)$, respectively. Clearly, $L(E_{DC}(P_3)) = 2^4 + 4L(E_{DC}(P_2)) = 112$ and $L(E_{opt}(P_3)) = 4L(E_{opt}(P_2)) = 32$. The ratio is $112/32 = 3.5$. To construct P_i we apply the same combination using P_{i-1}. Note that when applying our procedure to P_i it always introduces mid-cuts, except when presented P_0 in which case it introduced a terminal end-cut. It is simple to see that our approximation algorithm introduces

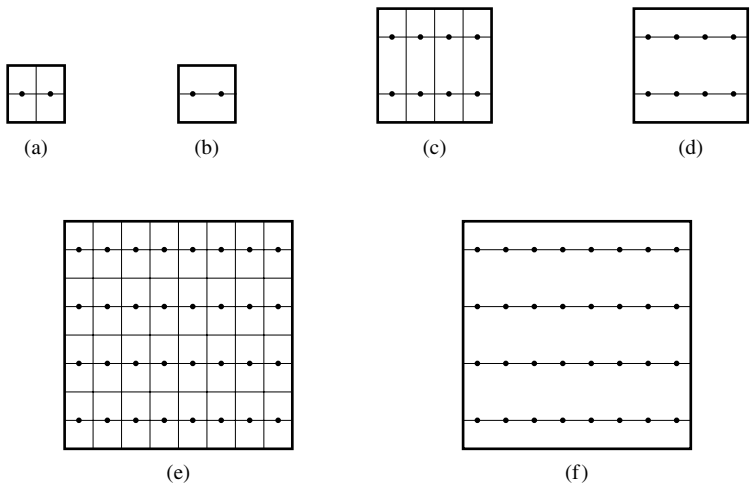

FIGURE 54.3 (a) $E_{DC}(P_1)$, (b) $E_{OPT}(P_1)$, (c) $E_{DC}(P_2)$, (d) $E_{OPT}(P_2)$, (e) $E_{DC}(P_3)$, and (f) $E_{OPT}(P_3)$.

cuts with length $L(E_{DC}(P_i)) = 2^{i+1} + 4L(E_{DC}(P_{i-1}))$, for $i > 1$ and $L(E_{DC}(P_1)) = 4$. An optimal solution, in this case, is identical to the lower bound function which is $L(E_{opt}(P_i)) = 4L(E_{opt}(P_{i-1}))$, for $i > 1$ and $L(E_{opt}(P_1)) = 2$.

Solving the above recurrence relations we know that

$$L(E_{DC}(P_i)) = \sum_{j=0}^{i-2} 2^{2i-j-1} + 2^{2i-2}L(E_{DC}(P_1)) = 2^{2i+1} - 2^{i+1}$$

and

$$L(E_{OPT}(P_i)) = 2^{2(i-1)}L(E_{OPT}(P_1)) = 2^{2i-1}$$

Therefore the approximation ratio is $L(E_{DC}(P_i))/L(E_{opt}(Pi)) = 4 - 1/2^{i-2}$.

Theorem 54.2

There are problem instances for which procedure DC generates a solution such that $L(E_{DC}(I))$ tends to $4L(E_{opt}(I))$.

Proof

By the above discussion. □

Procedure *DC* can be easily modified so that for problem instances with "many" points along the same line it will introduce a line to cover all the points provided that the line segment has length close to Y. For the problem instance given above the modified algorithm will generate a better solution decreasing substantially the approximation ratio. However, as pointed out in Ref. [11], there are problem instances for which the modified procedure will generate solutions with edge length of about $4LB(I)$. The idea is to perturb the points slightly. This way no two points will belong to the same vertical or horizontal line.

The time complexity $T(n)$ for Procedure *DC* when operating on an instance with $n = |P|$ points is given by the recurrence relation $T(n) \leq T(n - i - 1) + T(i) + cn$, for $1 \leq i < n$ and some constant c. However, it is possible to implement the procedure so that it takes $O(n \log n)$ time [5]. In what follows, we briefly describe one of the two implementations given in Ref. [5] with the $O(n \log n)$ bound. To simplify the presentation assume that no two points can be covered by the same vertical or horizontal line. The idea is to change the procedure so that the time complexity term cn becomes $cf(\min\{i, n - i - 1\})$, for some function $f()$ that we specify below. Note that this requires a preprocessing step that takes $O(n \log n)$ time. For certain functions $f()$ this reduces the overall time complexity bound to $O(n \log n)$.

In the preprocessing step we create a multilinked structure in which there is a record (data node) for each point. The record contains the x- and y-coordinate values of the point. We also include the rank of each point with respect to their x and y values. For example, if the rank of a point is (i, j) then it is the ith smallest point with respect to its x value, and the jth smallest point with respect to its y value. Note that this ranking is with respect to the initial set of points. We also have all of these records in two circular lists, one sorted with respect to the x values and the other sorted with respect to the y values. It is simple to see that this multilinked structure can be constructed in $O(n \log n)$ time.

In each recursive invocation of procedure *DC* we need to construct P_l and P_r from P, and assume that $X \geq Y$. This construction can be implemented to take $O(\min\{|P_l|, |P_r|\})$ by scanning the doubly linked lists for the x values from both ends (alternating one step from each end) until all the points at one end are P_l and the other ones are on P_r. Assume that $n \geq |P| = m > |P_l| = m - q \geq |P_r| = q$, and $1 < q \leq m/2$. Clearly, the above procedure can be used to identify the points in P_r and then remove the points in P_r from P in $O(q)$ time. The remaining points are P_l and are represented according to our structure. But now the problem is that we need to construct a multilinked data structure for the points in P_l. These points are already sorted by their x values, but not by their y values. The algorithm given in Ref. [13] can sort any q integers in the range $[1, n]$ in $O(q \log \log_q n)$ time. Once we have sorted them we can construct the multilinked data structure for P_l. This takes $O(q \log \log_q n)$ time. Let $T_n(m)$ be the total time required by procedure *DC* when the initial invocation involved a problem with n points and

we now have a problem with m points. Clearly, $T_n(1) = O(1)$ and $T_n(m) = \max\{O(q \log \log_q n) + T_n(q) + T_n(m-q) | 1 \leq q \leq m/2\}$ for $m > 1$. By the analysis of Ref. [5], $T_n(n) = O(n \log n)$. This result is summarized in the following theorem:

Theorem 54.3

Procedure DC and its preprocessing step can be implemented to take $O(n \log n)$ time.

Proof

By the above discussion. □

54.3 Dynamic Programming Approach

The divide-and-conquer algorithm *DC* presented in the previous section introduces guillotine cuts by following a set of simple rules, which makes the algorithm run very efficiently. But such guillotine cuts do not form an optimal guillotine partition. It seems natural that using an optimal guillotine partition would generate a better solution. But how fast can one generate an optimal guillotine partition? By the recursive nature of guillotine partitions, it is possible to construct an optimal guillotine partition in polynomial time. In this section we analyze an approximation algorithm based on this approach. As we shall see the algorithm has a smaller approximation ratio, but it takes longer to generate its solution.

54.3.1 Algorithm

Let $I = (R = (X, Y), P)$ be any problem instance and let the x-interval and y-interval define the rectangle $R_{x,y}$, which is part of rectangle R. We use $g(R_{x,y})$ to represent the length of an optimal guillotine partition for $R_{x,y}$.

First it is important to establish that there is always an optimal guillotine partition such that all its line segments include at least one point from P inside them (excluding those at its endpoints). This is based on the observation that given any optimal partition that does not satisfy this property it can either be transformed to one that does satisfy the property or one can establish that it is not an optimal guillotine partition. The idea is to move each horizontal (vertical) guillotine cut without points from P inside them either to the left or right (up or down) without increasing the total edge length. Note that when the above operation is perfomed some vertical (horizontal) segments need to be extended and some need to be retracted to preserve a guillotine partition. If the total edge length decreases then we know that it is not an optimal guillotine partition and when it remains unchanged after making all the transformations it becomes an optimal guillotine partition in which all its guillotine cuts include at least one point from P.

By applying the above argument we know that for any rectangle $R_{x,y}$ one can easily compute $g(R_{x,y})$ recursively by selecting the best solution obtained by trying all $2n$ guillotine cuts (that include a point from P) and then solving recursively the two resulting problem instances. It is simple to show that there are $O(n^4)$ different g values that need to be computed. By using dynamic programming and the above recurrence relation the length of an optimal guillotine partition can be constructed in $O(n^5)$ time. By recording at each step a guillotine cut forming an optimal solution and using the g values, an optimal guillotine partition can be easily constructed within the same time complexity bound. The following theorem follows from the above discussion:

Theorem 54.4

An optimal guillotine partition for the RG-P problem can be constructed in $O(n^5)$ time.

54.3.2 Approximation Bound

The set of line segments in a feasible rectangular partition of I is denoted by $E(I)$, and a set of line segments in a minimum-length guillotine partition is denoted by $E_G(I)$). We use $L(E)$ to represent the length of the edges in a rectangular partition E. In what follows we show that $L(E_G(I)) \leq 2L(E(I))$ by introducing a set

of vertical and horizontal line segments $A(I)$ such that $E(I) \cup A(I)$ is a guillotine partition and $L(E(I) \cup A(I)) \leq 2L(E(I))$. The bound follows from the fact that $L(E_G(I)) \leq L(E(I) \cup A(I))$. Our main result follows from the application of this result to $E(I) = E_{opt}(I)$. The set $A(I)$ of additional segments are introduced by Procedure *TR1*. Before we present this procedure and in its analysis we define some terms.

A horizontal (vertical) line segment that partitions rectangle R into two rectangles is called a *horizontal (vertical) cut*. We say that $E(I)$ has a *horizontal guillotine cut,* if there is a horizontal cut, l, such that $L(E(I) \cap l) = X$. A rectangular partition $E(I)$ has a *half horizontal overlapping cut* if there is a horizontal cut l such that $L(E(I) \cap l) \geq 0.5X$. *Vertical guillotine cuts* are defined similarly.

Suppose R is partitioned by a vertical or horizontal cut into two rectangles, R_l and R_r. With respect to this partition we define $E(I_l)$ and $E(I_r)$ as the set of line segments of $E(I)$ inside R_l and R_r, respectively. We use $E_h(I)$ ($E_v(I)$) to denote all the horizontal (vertical) line segments in $E(I)$. With respect to $A(I)$ we define similarly $A_h(I)$ and $A_v(I)$.

Assume that $E(I)$ is nonempty. The idea behind Procedure *TR1* is to either introduce horizontal or vertical line segments at each step and then apply recursively the procedure to the nonempty problem instances. When a horizontal line segment is introduced, it is added along a half horizontal overlapping cut. The two resulting problems will not have any of these segments inside them. So at this step the added horizontal segments have length at most equal to the ones of the half horizontal overlapping cut.

Since there are problem instances without half horizontal overlapping cuts, Procedure *TR1* checks to see if there is a vertical guillotine cut. In this case we just partition the rectangle along this cut. Clearly, there are no additional line segments introduced in this case.

There are rectangular partitions without a half horizontal overlapping cut or a vertical guillotine cut. In this case, Procedure *TR1* introduces a *mid-cut*, which is just a vertical line that partitions the rectangle along its center. The problem now is that this mid-cut does not necessarily overlap with any of the segments in $E_v(I)$. Our approach is to remember this fact and later on identify a set of line segments in $E_v(I)$ that will account for the length of this mid-cut. To remember this fact we will color the right side of rectangle R_l. As we proceed in the recursive process different parts of this colored side will be inherited by smaller rectangles that will get their right side colored. At some point later on when we pay for the segment represented by a colored right side of a rectangle, it will no longer appear in recursive calls resulting from the one for this rectangle. The budget in this case is two times the total length of the vertical line segments in $E_v(I)$. This budget should be enough to pay for the total length of the vertical line segments introduced during the transformation.

To be able to show that $A_v(I) \leq E_v(I)$ it must be that the rectangles in the terminal recursive calls will not have their right side colored. Before we establish this result, we formally present Procedure *TR1* that defines the way colors are inherited in the recursive calls.

Procedure *TR1*$(I = (R = (X, Y), E(I)))$;
  case
    :$E(I)$ is empty: **return**;
    :There is a half horizontal overlapping cut in $E(I)$:
      Partition the rectangle R along one such cut;
      If the right side of R is colored, the right sides of rectangles R_l and R_r remain colored;
    :There is a vertical guillotine cut in $E(I)$:
      Partition R along one such cut;
      Remove the color, if any, of the right side of rectangle R_r;
    :**else**:
      Partition R by a vertical cut intersecting the center of R;// introduce a mid-cut
      If the right side of R is colored, the right side of R_r remains colored;
      Color the right side of R_l;
  endcase
  Apply *TR1* recursively to $(I_l = (R_l, E(I_l)))$;
  Apply *TR1* recursively to $(I_r = (R_r, E(I_r)))$;
end of Procedure TR1

It is important to remember that Procedure *TR1* is only used to establish our approximation bound. The right side of rectangle R is not colored in the initial invocation to Procedure *TR1*. Before establishing the approximation ratio of 2 in Theorem 54.5, we prove the following lemmas:

Lemma 54.4

Every invocation of Procedure TR1 $(I, E(I))$ *satisfies the following conditions:*

(a) if $E(I)$ *is empty, then the right side of* R *is not colored, and*
(b) if the right side of R *is colored, then* $E(I)$ *does not have a horizontal guillotine cut.*

Proof

Initially the right side of R is not colored so the the first invocation $(I, E(I))$ satisfies conditions (a) and (b). We now show that if upon entrance to the procedure the conditions (a) and (b) are satisfied, then the invocations made directly from it will also satisfy (a) and (b). There are three cases depending on the type of cut introduced by procedure *TR1*.

Case 1. Procedure *TR1* partitions R along a half horizontal overlapping cut.
First consider the subcase when $E(I)$ has a horizontal guillotine cut. From (b) we know that the right side of R is not colored, and the algorithm does not color the right side of R_l or R_r. Therefore the two invocations made directly from this call satisfy properties (a) and (b). In contrast, when $E(I)$ does not have a horizontal guillotine cut, then the right side of R may be colored. Since $E(I)$ does not have a horizontal guillotine cut, we know that there is at least one vertical line segment on each side of the half horizontal overlapping cut, so $E(I_l)$ and $E(I_r)$ must be nonempty. Since neither of these two partitions has a horizontal guillotine cut, each invocation made directly by our procedure satisfies properties (a) and (b). This completes the proof for this case.

Case 2. Procedure *TR1* partitions R along a vertical guillotine cut.
Since the right side of R_l and R_r end up uncolored; then the invocations made directly by the procedure satisfy (a) and (b). This completes the proof of this case.

Case 3. Procedure *TR1* introduces a mid-cut.
Since $E(I)$ does not have a half horizontal overlapping cut and there is no vertical guillotine cut, then each of the resulting rectangular partitions has at least one vertical line segment and there are no horizontal guillotine cuts in the two resulting problems. Therefore, both of the resulting problem instances are not empty and do not have a horizontal guillotine cut. This implies that both problem instances satisfy (a) and (b). This completes the proof for this case and the lemma. □

Lemma 54.5

For any nonempty rectangular partition $E(I)$ *of any instance* I *of the RG-P problem, procedure TR1 generates a set* $A(I)$ *of line segments such that* $L(A_h(I)) \leq L(E_h(I))$, *and* $L(A_v(I)) \leq L(E_v(I))$.

Proof

First we show that $L(A_h(I)) \leq L(E_h(I))$. This is simple to prove because horizontal cuts are only introduced over half horizontal overlapping cuts. Each time a horizontal cut is introduced the segments added to $A_h(I)$ have length that is at most equal to the length of the segments in the half horizontal cut in $E_h(I)$. Since these line segments are located on the boundary of the two resulting instances, these line segments will not account for other segments in $A_h(I)$ and thus $L(A_h(I)) \leq L(E_h(I))$.

Let us now establish that $L(A_v(I)) \leq L(E_v(I))$. It is simple to verify that Procedure *TR1* does not color a side more than once, and all empty rectangular partitions do not have their sides colored (Lemma 54.4). Every invocation of Procedure *TR1* with a nonempty rectangular partition R generates two problem instances whose total length of their colored right sides is at most the length of the right side of R, if it is colored, plus the length of the vertical line segment introduced. The only exception is when the procedure introduces a cut along an existing vertical guillotine cut. In this case if the right side of R is colored, the right sides of two resulting partitions will not be colored. So the length of a line segment previously introduced

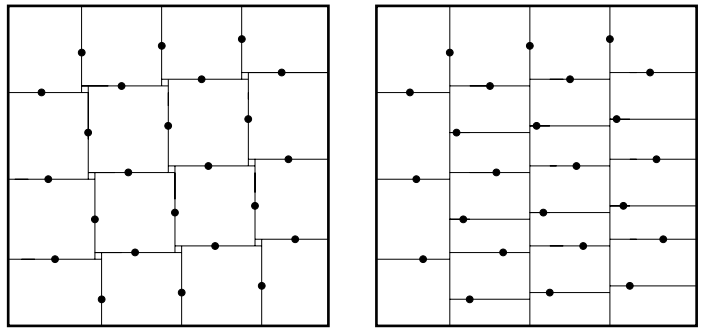

FIGURE 54.4 (a) Optimal rectangular partition. (b) Optimal guillotine partition.

in $A(I)$, which is recorded by the fact that the right side of R is colored, is charged to the existing guillotine cut and such cut will not be charged another segment again. Therefore, $L(A_v(I)) \leq L(E_v(I))$. This concludes the proof of the lemma. □

Theorem 54.5

The length of an optimal guillotine partition is at most twice the length of an optimal rectangular partition, that is, $L(E_G(I)) \leq 2L(E_{opt}(I))$.

Proof

Apply procedure *TR1* to any optimal rectangular partition $E(I) = E_{opt}$. By Lemma 54.5 we know that $L(E(I) \cup A(I)) \leq 2L(E_h(I)) + 2L(E_v(I)) = 2L(E(I))$. Since $E(I) \cup A(I)$ is a guillotine partition, $L(E_G(I)) \leq L(E(I) \cup A(I))$. Hence, $L(E_G(I)) \leq 2L(E(I)) = 2L(E_{opt}(I))$. □

It is simple to find a problem instance I such that $L(E_G(I))$ is about $1.5L(E_{opt}(I))$ [8]. One of such problem instances has the distribution of points shown in Figure 54.4. As the number of points increases, the ratio $\frac{L(E_G(I))}{L(E_{opt}(I))}$ approaches 1.5.

54.3.3 Improved Approximation Bound

In this section, we describe the idea behind the complex proof given in Ref. [9] that establishes the fact that $L(E_G(I)) \leq 1.75L(E_{opt}(I))$. The proof is based on a recursive transformation procedure *TR2* that, when performed on any rectangular partition $E(I)$, generates a set $A(I)$ of line segments such that $E(I) \cup A(I)$ forms a guillotine partition (of course $A(I) \cap E(I) = \emptyset$). Without loss of generality, assume that $L(E_v(I)) \leq L(E_h(I))$. The transformation is performed in such a way that

$$L(A_v(I)) \leq L(E_v(I)) \tag{54.6}$$

and

$$L(A_h(I)) \leq 0.5L(E_h(I)) \tag{54.7}$$

Then, we have

$$\begin{aligned} L(E_G(I)) &\leq L(A(I) \cup E(I)) \\ &= L(A_h(I)) + L(A_v(I)) + L(E(I)) \\ &\leq 0.5L(E_h(I)) + L(E_v(I)) + L(E(I)) \\ &= 0.5L(E_v(I)) + 1.5L(E(I)) \\ &\leq 1.75L(E(I)) \end{aligned}$$

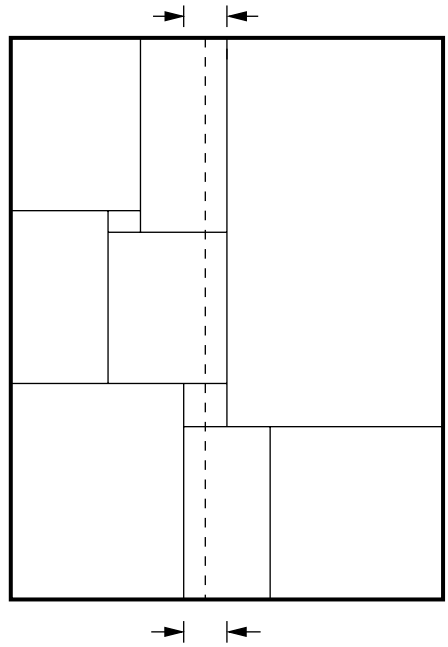

FIGURE 54.5 A vertically separable $E(I)$. $E(I)$ has no guillotine cut. The broken line segment is a vertical through cut of $E(I)$. In fact, any vertical cut in the region marked by the vertical segments outside of rectangular boundary is a vertical through cut for $E(I)$.

To satisfy Eq. (54.6), the notion of *vertical separability* is introduced. Denote the x-coordinate of a vertical segment l by $x(l)$. A vertical cut l is *left* (*right*) *covered by* $E_v(I)$ if for every point p on l there exists a line segment l' in $E_v(I)$ such that $x(l') \leq x(l)$ ($x(l') \geq x(l)$), and there is a point p' on l' with $x(p') = x(p)$. A vertical cut is called a *vertical through cut* if it is both left and right covered by $E_v(I)$. The set of segments $E(I)$ is said *vertically separable* if there exists at least one vertical through cut. Note that a vertical guillotine cut is also a vertical through cut, but the converse is not necessarily true. Figure 54.5 shows a vertically separable $E(I)$ and a vertical through cut. When a new vertical line segment is introduced by *TR2*, it is ensured to be a portion of a vertical through cut.

To satisfy Eq. (54.7), horizontal segments are carefully introduced by *TR2* according to the structure of $E(I)$. When there is neither guillotine cut and nor vertical through cut in $E(I)$, a three-step subprocedure *HVH_CUT* is invoked to introduce new segments. Procedure *TR2* is given below

Procedure *TR2*($E(I)$);
  case
    :$E(I)$ is empty:
      return;
    :$E(I)$ has a guillotine cut l:
      Partition $E(I)$ along l into $E(I_1)$ and $E(I_2)$;
      Recursively apply *TR2* to $E(I_1)$ and $E(I_2)$;
    :$E(I)$ is vertically separable:
      Let l be any vertical through cut;
      Partition $E(I)$ along l into $E(I_1)$ and $E(I_2)$;
      Recursively apply *TR2* to $E(I_1)$ and $E(I_2)$;
    :**else**:
      Use Procedure *HVH_CUT* to partition $E(I)$ into $E(I_1)$, $E(I_2)$, $\cdots$, $E(I_{q+1})$;
      Recursively apply *TR2* to each $E(I_i)$;

endcase
end of Procedure TR2

Procedure *HVH_CUT* is outlined below:

Procedure *HVH_CUT*($E(I)$);

1. Introduce a carefully selected set of q horizontal cuts that partition $E(I)$ into $q + 1$ vertically separable rectangular subpartitions;
2. In each resulting partition of Step 1, introduce either the leftmost or the rightmost vertical through cut;
3. Divide each resulting partition of Step 2 into two rectangular subpartitions by a horizontal guillotine cut if such a cut exists;

end of Procedure HVH_CUT

Let $H(I)$ be the set of horizontal cuts introduced in Step 1, $V(I)$ the set of vertical through cuts chosen in Step 2, and $H_3(I)$ the horizontal guillotine cuts found in Step 3. Define $H_1(I) = H(I) \cap E_h(I)$ and $H_1'(I) = H(I) - E_h(I)$. Note that $H_3(I) \subset E_h(I)$. The sets $H(I)$ and $V(I)$ are selected in such a way that

$$L(H_1'(I)) \leq 0.5qX \tag{54.8}$$

and

$$L(H_1(I) \cup H_3(I)) \geq qX \tag{54.9}$$

It is quite complex to establish that such $H(I)$ and $V(I)$ always exist [9]. We omit additional details of the procedure *HVH_CUT* and the related proofs. Vertical though cuts introduced by *TR2* are carefully selected to satisfy Eq. (54.6) and ensure Eq. (54.9). Since all horizontal cuts introduced by *HVH_CUT* satisfy Eq. (54.8) and Eq. (54.9), and all horizontal cuts that are not introduced by invocations to *HVH_CUT* are horizontal guillotine cuts in their respective subrectangular boundaries, Eq. (54.7) is satisfied. Consequently, $L(E_G(I)) \leq 1.75L(E(I))$. Since $E(I)$ is any arbitrary rectangular partition, we have $L(E_G(I)) \leq 1.75L(E_{opt}(I))$. The next theorem sums up the discussion.

Theorem 54.6

The length of an optimal guillotine partition is at most 1.75 times the length of an optimal rectangular partition, that is, $L(E_G(I)) \leq 1.75L(E_{opt}(I))$.

Proof

The full details of the proof, whose outline is given above, appears in Ref. [9]. □

54.4 Concluding Remarks

In this chapter, we considered the *RG-P* problem. We presented a fast divide-and-conquer approximation algorithm with approximation ratio 4. This ratio is smaller than the $3 + \sqrt{3}$ ratio of a similar divide-and-conquer approximation algorithm given [7]. We also examined in detail the approach of approximating optimal rectangular partitions via optimal guillotine partitions. Optimal guillotine partitions can be constructed in polynomial time using dynamic programming [3]. For this approach we presented a proof that the approximation ratio is at most 2. Our proof is simpler than the proof for the same bound given in Ref. [3]. We presented the idea behind a complex proof given in Ref. [9], which establishes that the approximation ratio is at most 1.75 when approximating optimal rectangular partitions via optimal guillotine partitions. Both proofs are based on the technique of recursively transforming a rectangular

partition into a guillotine partition by introducing additional line segments. The difference is that the additional segments in the transformation for the approximation ratio of 1.75 are selected more carefully than those introduced by the transformation for the bound of 2. Whether or not one can further reduce this bound remains a challenging open problem.

For the *RG-P* problem, the partitions obtained by the divide-and-conquer algorithms (the one of Ref. [7] and the one presented in this chapter) and the dynamic programming algorithm are guillotine partitions formed by recursive guillotine cuts. The approach examined in this chapter can be referred to as approximating optimal rectangular partitions via optimal and suboptimal guillotine partitions. The first approximation algorithm for this problem based on suboptimal guillotine partitions appeared in Ref. [7]. Subsequently, optimal guillotine partitions were used in Ref. [8]. Both of these approaches were generalized to multidimensional space in Refs. [11,12].

The term "guillotine cut" was introduced in the 1960s in the context of cutting stock problems. In the context of rectangular partitions it was first used in Ref. [8]. As pointed out in Ref. [14], the concept of guillotine partition has been generalized into a powerful general approximation paradigm for solving optimization problems in different settings. The guillotine partition algorithms for the *RG-P* problem were among the first that manifested the power of this paradigm.

References

[1] Lingas, A., Pinter, R. Y., Rivest, R. L., and Shamir, A., Minimum edge length partitioning of rectilinear polygons, in *Proc. 20th Allerton Conf. on Comm., Cont., and Comput.,* Monticello, Illinois, 1982.

[2] Rivest, R. L., The "PI" (placement and interconnect) system, in *Proc. Design Automation Conf.*, 1982.

[3] Du, D. Z. and Chen Y. M., On Fast Heuristics for Minimum Edge Length Rectangular Partition, Technical report, MSRI 03618–86, 1986.

[4] Levcopoulos, C., Minimum length and thickest–first rectangular partitions of polygons, *Proc. 23rd Allerton Conf. on Communication, Control and Computing,* Monticello, Illinois, 1985.

[5] Levcopoulos, C., Fast heuristics for minimum length rectangular partitions of polygons, *Proc. 2nd ACM Symp. on Computational Geometry,* 1986.

[6] Lingas, A., Heuristics for minimum edge length rectangular partitions of rectilinear figures, *Proc. 6th GI–Conf.,* Lecture Notes in Computer Science, Springer, Dortmund, p. 195, 1983.

[7] Gonzalez, T. F., and Zheng, S. Q., Bounds for partitioning rectilinear polygons, *Proc. ACM Symp. Computational Geometry,* 1985, p. 281.

[8] Du, D. Z., Pan, L. Q., and Shing, M. T., Minimum Edge Length Guillotine Rectangular Partition, Technical report, MSRI 02418–86, 1986.

[9] Gonzalez, T. F., and Zheng, S. Q., Improved bounds for rectangular and guillotine partitions, *J. Symbolic Comput.*, **7**, 591, 1989.

[10] Gonzalez, T. F., and Zheng, S. Q., Approximation algorithms for partitioning rectilinear polygons with interior points, *Algorithmica*, 5, 11, 1990.

[11] Gonzalez, T. F., Razzazi, M., and Zheng, S. Q., An efficient divide-and-conquer algorithm for partitioning into d-boxes", *Int. J. Comput. Geometry Appl.,* 3(4), 1993, 417 (condensed version appeared in *Proc. 2nd Canadian Conf. on Computational Geometry,* 1990, p. 214).

[12] Gonzalez, T. F., Razzazi, M., Shing, M., and Zheng, S. Q., On optimal d-guillotine partitions approximating hyperrectangular partitions, *Comput. Geometry: Theor. Appl.,* 4(1), 1, 1994.

[13] Kirkpatrick, D. G., An upper bound for sorting integers in restricted ranges, in *Proc. 18th Allerton Conf. on Communication, Control and Computing,* Monticello, Illinois, 1980.

[14] Cardei, M., Cheng, X., Cheng, X., and Du, D. Z., A tale on guillotine cut, in *Proc. Novel Approaches to Hard Discrete Optimization,* Ontario, Canada, 2001.