# Bounds for Partitioning Rectilinear Polygons

Teofilo Gonzalez and Si-Qing Zheng
Department of Computer Science
The University of California
Santa Barbara, CA 93106

*Abstract:* We study the problem of partitioning a rectilinear polygon with interior points into rectangles by introducing a set of line segments. All points must be included in at least one of the line segments introduced and the objective function is to introduce a set of line segments such that the sum of their lengths is minimal. Since this problem is computationally intractable, we present efficient approximation algorithms for its solution. The solutions generated by our algorithms are guaranteed to be within a fixed constant of the optimal solution value. Even though the constant approximation bound is not so small, we conjecture that in general the solutions our algorithms generate are close to optimal.

*Keywords:* Approximation algorithms, partition of rectilinear polygons, polynomial time complexity.

## *Introduction.*

Computational geometry is becoming more important because new problems in areas like pattern recognition, artificial intelligence, graphics, VLSI design and robotics are being identified as inherently geometric. For many geometric problems the algorithmic method of solution appropriate to a computer appears to be quite complex. One of the fundamental problems in computational geometry is to partition a polygon into parts. Traditionally the objective function is to obtain a convex partition with minimal number of components, and fortunately polynomial time algorithms for this problem exist ( [CD], [GJPT], [LLMP], [LLMPL], [S] ). Lingas *et. al.* ( [LPRS] ) investigated the partition problem for rectilinear polygons, the goal was to obtain a partition with minimum length of partitioning edges. In VLSI design, the problem of dividing routing regions into channels can be reduced to this new partition problem ( [R] ). Some versions of this partition problem were shown NP-complete, but the computational complexity of some other related problems is not known ( [LPRS] ). In this paper we investigate the problem of finding approximate solutions to some of these partition problems. We show that generating a solution within a "small" constant factor of the optimal solution value is a polynomially solvable problem.

A rectilinear *boundary* is a simple polygon with the additional constraint that all of its sides are either parallel or perpendicular to each other. A *hole* is a simple rectilinear polygon whose sides are parallel or perpendicular to the sides of the rectilinear boundary and located on the inside of a rectilinear boundary. There can be no holes inside a hole. A single point inside the boundary is called an *degenerate hole*. A *figure* is a rectilinear boundary which may contain an arbitrary number of nonoverlapping holes. A *rectangular partition* of a figure is a set of line segments lying within its boundary and not crossing any non-degenerate hole so that when drawn into the figure the area not enclosed by holes is partitioned into rectangles which do not contain degenerate holes. The partitioning line segments are called *edges*. A minimum edge length partition of a figure is a rectangular partition such that the sum of the length of its edges is the smallest among all feasible partitions. For every set of edges, E( I ), in a feasible solution for problem instance I we use the function L( E( I ) ) to represent the sum of the length of the edges in E( I ). For more details about these definitions see [LPRS].

Known results about the computational complexity of minimum edge rectangular partitioning of rectilinear polygons is displayed in the following table.

| Problem | Boundary | Points and Holes | Complexity |
|---|---|---|---|
| RP-HF | rectilinear polygon | hole free | $O(n^4)$ |
| RG-NLP | rectangle | non-corectilinear points | NP |
| RP-NLP | rectilinear polygon | non-corectilinear points | NP |
| RG-P | rectangle | points | NP-complete |
| RP-P | rectilinear polygon | points | NP-complete |
| RG-RP | rectangle | rectilinear polygons | NP-complete |
| RP-RP | rectilinear polygon | rectilinear polygons | NP-complete |
| RG-RPP | rectangle | rectilinear polygons and points | NP-complete |
| RP-RPP | rectilinear polygons | rectilinear polygons and points | NP-complete |

In the above table the RP-NLP problem is interpreted as "minimum edge length rectangular partitioning of a Rectilinear Polygon with Non-corectiLinear Points inside it", and similarly other abbreviations are self-explanatory. The computational complexity of the problems in the above table is displayed in figure 1.1, where " problem A ——> problem B " means that problem A is polynomially reducible to problem B, i.e. problem A is not computationally "harder" than problem B.

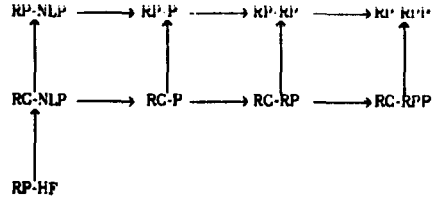



figure 1.1

The computational complexity of the RG-NLP and RP-NLP has not yet been established. Since an optimal solution for a given problem instance of RG-NLP could be very complex ( see figure 1.2 for an example ) and it seems that there is no way to obtain an optimal solution without exhaustive search, we strongly conjecture that both of these problems are NP-complete.

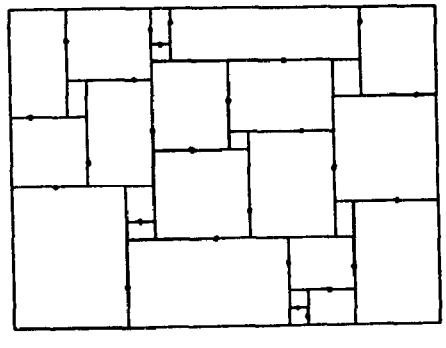


figure 1.2

In this paper we present approximation algorithms for all the problems included in the RP-P problem. Along with presenting our algorithms, we show that the solutions generated by our algorithms are optimal to within a constant factor. That is, if we denote the set of edges in an optimal solution by $E_{opt}(I)$, then $L(E_{apx}(I)) \leq c\, L(E_{opt}(I))$, where c is a constant, and $E_{apx}(I)$ is the set of edges in the solution generated by our algorithm. Although the approximation bounds we have obtained are not so small, we believe that the actual performance of our algorithms is considerably better than what we have been able to show.

## *II. Preliminaries.*

A problem instance is formally defined as $I = (B, P, H)$, where B is a set of ( corner ) points which defines a rectilinear polygonal boundary, P is a set of points as degenerate holes inside the boundary and H is a set of point sets, each defines a rectilinear hole inside the boundary. In the rest of this paper we call the polygon defined by B the global ***boundary***. Certainly B, P and H must satisfy a set of constrains so that "I" is a valid problem instance. A polygon can be represented by a list of corner points, $p_0, p_1, \ldots, p_{n-1}$, such that the line segment from $p_i$ to $p_{(i+1) mod (n)}$ is a side of the boundary for $0 \leq i \leq n-1$. A line segment is represented by the pair of end points of the segment, i.e. $[(x_i, y_i), (x_j, y_j)]$ represent the line segment with end points $(x_i, y_i)$ and $(x_j, y_j)$.

We define the ***grid*** induced by a problem instance as the set of horizontal and vertical line segments introduced by extending all sides of the figure and the holes in both directions until either the line segment intersects the side of a hole or the line segments intersects the boundary. Figures 2.1 and A.1 illustrate the grid induced ( by broken lines ) for two problem instances.

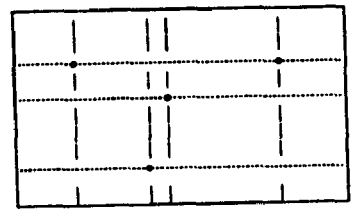


figure 2.1: ( Non-solid lines are grid lines ).

It can be easily shown that in any optimal solution for the RP-RPP problem, all partitioning edges lie on the induced grid. The proof of this fact uses arguments similar to the ones in the proof of lemma 1 in [LPRS]. The solutions that our algorithms generate also have this property.

*III.* Divide-and-Conquer Approximation Algorithm for the RG-P Problem.

In section II we mentioned that a problem instance is defined as I = ( B , P , H ). For the RG-P problem, H is an empty set and B contains four corner points defining a rectangular global boundary. In our algorithm we use ( $x_0$ $y_0$ ), X, and Y to define the boundary, where ( $x_0$ , $y_0$ ) is the bottom-left corner of the boundary ( origin of I ), X and Y are the width and height of the boundary.

We use $E_{apx}$ to denote the solution generated by our algorithm. In the algorithm $E_{apx}$ is a set of pairs of points, each pair representing a line segment in the solution. Initially $E_{apx}$ is empty.

```
procedure PARTITION( x0 , y0 , X , Y , P )
 begin
 Let n be the cardinality of set P; if n = 0 then return;
 if X < Y then rotate the rectangle 90 degrees and let ( x0 , y0 ) be the bottom-left point;
 Rearrange the points so that x0 ≤ x1 ≤ ... ≤ xn; Let X' = X/(1+√3);
 Let c = min{ i | |( xi - x0 ) - X/2| = min{ |( xj - x0 ) - X/2| | 1 ≤ j ≤ n } }
 Let u = max{ i | |( xi - x0 ) - X/2| = min{ |( xj - x0 ) - X/2| | 1 ≤ j ≤ n } }
 case
  :x0 is located to the left of the center of the rectangle, xu - x0 ≤ X' and u < n:
   begin {compound_cut step}
    Eapx = Eapx ∪ { [ ( x0 , y0 + Y ) , ( x0 , y0 ) ] , [ ( xu+1 , y0 + Y ) , ( xu+1 , y0 ) ] }
    P1 = { pk ∈ P | and xk < xu }; P2 = { pk | ∈ P and xk > xu };
    X1 = xu - x0; X2 = X - ( xu+1 - x0 );
    PARTITION( x0 , y0 , X1 , Y , P1 ); PARTITION( xu+1, y0 , X2 , Y , P2 );
   end
  :x0 is located to the right of the center of the rectangle, X - ( xu - x0 ) ≤ X' and c > 1:
   begin {compound_cut step}
    Eapx = Eapx ∪ {[( xu , y0 + Y ) , ( xu , y0 ) ] , [ ( xc-1 , y0 + Y ) , ( xc-1 , y0 ) ] }
    P1 = { pk | pk ∈ P and xk < xc-1}; P2 = { pk | pk ∈ P and xk > xu };
    X1 = xc-1 - x0; X2 = X - ( xu - x0 );
    PARTITION( x0 , y0 , X1 , Y , P1 ); PARTITION( xu , y0 , X2 , Y , P2 );
   end
  :else: { simple_cut step }
   begin
    Eapx = Eapx ∪ { [ ( xu , y0 + Y ) , ( xu , y0 ) ] }
    P1 = { pk | pk ∈ P and xk < xu }; P2 = { pk | pk ∈ P and xk > xu };
    X1 = xu - x0; X2 = X - ( xu - x0 );
    PARTITION( x0 , y0 , X1 , Y , P1 ); PARTITION( xu , y0 , X2 , Y , P2 );
   end
 end
end
```

Since the rules used to do the "horizontal cuts" are the same as what we use to do the "vertical cuts", we only consider "horizontal cuts". It is easy to see that figure 3.1 below includes all possible situations of one step in the recursive process of our algorithm. A shadowed rectangle represents a sub-instance without interior points from set P.

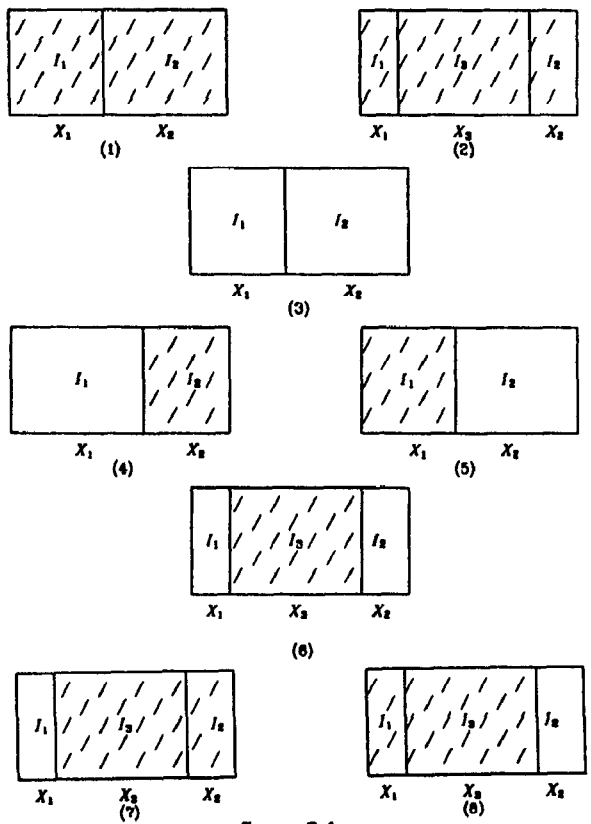



figure 3.1:

In figure 3.1, (1), (3), (4) and (5) correspond the "simple_cut step" in the algorithm, and the remaining ones correspond the "compound_cut step" in the algorithm.

The basic idea behind recursive definition of our lower bound function, LB, is to take a "portion" of edge length from L( $E_{apx}$ ) in such a way that LB( I ) ≤ L( $E_{opt}$( I ) ). Now with reference to figure 3.1 we define LB( I ) recursively as follows:

$$LB(I) = \begin{cases} 0 & \text{I is empty} \\ Y & (1) \\ \min\{ X , 2Y \} & (2) \\ LB(I_1) + LB(I_2) & (3) \\ LB(I_1) + LB(I_2) + \min\{ X_3 , 2Y \} & (4) \\ LB(I_1) + \min\{ X_2 , Y \} & (5) \\ LB(I_2) + \min\{ X_1 , Y \} & (6) \\ LB(I_1) + \min\{ X_2 + X_3 , 2Y \} & (7) \\ LB(I_2) + \min\{ X_1 + X_2 , 2Y \} & (8) \end{cases}$$

*Lemma 3.1:* For any instance I, LB( I ) ≤ L( $E_{opt}$( I ) ).

*Proof:* The proof appears in [GZ].

For any instance I, we define the function USE( I ) to be the sum of the length of the edges introduced by the algorithm when presented instance I and without including the edges introduced in any of its recursive calls. For problem instance I, let $I_1$, $I_2$, ..., $I_m$ be the subinstances which

were used when calling PARTITION( I ). Then L( $E_{apx}$ ) = $\sum_{j=1}^{m}$ USE( $I_j$ ).

For a problem instance I = ( $x_0$, $y_0$, X, Y, P ), if max{ X, Y } ≤ (1+√3) min{ X, Y }, we call it a *regular ( R )* instance, otherwise we call it an *irregular ( IR )* instance. We define the CARRY function as follows:

$$CARRY(I) = \begin{cases} 0 & \text{if } I \text{ is } empty \\ X + Y & \text{if } I \text{ is } R \text{ and } non-empty \\ (2+\sqrt{3})\min\{X, Y\} & \text{if } I \text{ is } IR \text{ and } non-empty \end{cases}$$

For non-empty instance I, we have, equivalently, CARRY( I ) = min{ X + Y, (2+√3) min{ X, Y } }. Function CARRY( I ) indicates the maximum amount of edge length introduced by the algorithm in previous ( ancestor ) calls that have not yet been accounted for by (3+√3) * lower bound. For any subinstance I we must show that L( $E_{apx}$( I ) ) + CARRY( I ) ≤ (3+√3) * LB( I ) ≤ (3+√3) * L( $E_{opt}$( I ) ).

*Lemma 3.2:* For any problem instance I,

(i) L( $E_{apx}$( I ) ) + CARRY( I ) ≤ (3+√3) * LB( I )

(ii) L( $E_{apx}$( I ) ) + CARRY( I ) ≤ (3+√3) * L( $E_{opt}$( I ) ).

*Proof:* The proof appears in [CZ].

∎

*Theorem 3.1:* For any instance of the RG-P problem, algorithm PARTITION generates a solution $E_{apx}$( I ) such that L( $E_{apx}$( I ) ) ≤ (3+√3) L( $E_{opt}$ ( I ) ).

*Proof:* The proof follows from lemma 3.2.

∎

Let us now determine the tightness of our approximation bound. In example 3.1 we give an instance such that L( $E_{apx}$ ) = 3.5 L( $E_{opt}$ ) - ∈ for any small ∈ > 0. Since we could not find examples whose behavior is worse than this, we conjectured that the analysis of our approximation bound could be improved. However, this is not possible unless one redefines the lower bound function. In example 3.2 we show that there are instances for which L( $E_{apx}$ ) = (3+√3) LB( I ) - ∈ for any small ∈ > 0.

*Example 3.1:*

The boundary is the rectangle with origin (0, 0), X = $2^k$, Y = 1. To ease the arguments we assume that there are 3m points inside the boundary, where m = $2^{k+3}$-1. The points are defined as follows:

$p_i$ = (i/8, 1/2 + ∈), 1 ≤ i ≤ m;

$p_{2i}$ = (i/8, 1/2 - ∈), 1 ≤ i ≤ m;

$p_{3i}$ = (i/8, 1/2), 1 ≤ i ≤ m.

For small ∈ the solution generated by our algorithm and the optimal solution are depicted in figure 3.2.

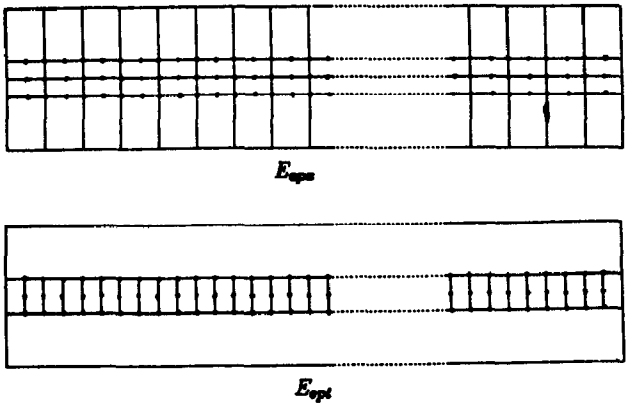



figure 3.2

Clearly,

$L(E_{apx}(I)) = 3X + (m + 1)/2 - 1 = 3*2^k + 2^{k+2} - 1$, and

$L(E_{opt}((I)) = 2X + 2m\epsilon = 2^k + (2^{k+4} - 2)\epsilon$.

Thus, $\lim_{k \to \infty} \lim_{\epsilon \to 0} L(E_{apx}(I))/L(E_{opt}(I)) = 3.5$

*Example 3.2:*

The boundary is a rectangular with origin (0, 0) and X = Y = $2^k$. There are ($2^{2k}$-1) + 4*$2^{2k}$ points inside the boundary. Among them $2^{2k}$-1 points are arranged in such a way that the algorithm will partition the instance into $2^{2k}$ subinstances of size one by one. For k = 2 the partition generated by the algorithms is given in figure 3.3. Applying the definition of our lower bound function, LB, we know that if each of these subinstances is not empty none of the lengths of these edges will be accounted for in LB. In other words, the lower bound function for the problem is equal to the sum of the lower bounds for each of the squares.

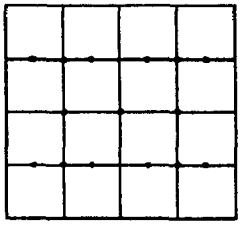


figure 3.3

The remaining $4*2^{2k}$ points are distributed into $2^{2k}$ subinstances each with four points. Each subinstance is such that algorithm PARTITION performs a compound_cut followed by two simple_cuts, in further recursive steps. This is illustrated in figure 3.4.

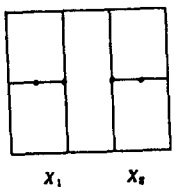



figure 3.4

Let $X_1 = X_2 = 1/(1+\sqrt{3})$. Then by the definition of LB, it is easy to see that for each square I, $LB(I) = 2^{2k}$.
Clearly,

$$L(E_{aps}(I)) = 2*2^k*(2^k-1) + 2^{2k}*(2+2/(1+\sqrt{3})).$$

Thus, $\lim_{k\to\infty} L(E_{aps}(I))/LB(I) = \lim_{k\to\infty} (4+2/(1+\sqrt{3})-2^{1-k}) = 3+\sqrt{3}$.

In the above examples we only considered the RG-P problem, if we relocate the points we can show that the above bounds are satisfied by an instance of the RG-NLP problem. Example 3.1 is also the worst case we could find for the partition algorithm which only makes a simple-cut at each recursive step. The above examples shows that the bound we obtained is not so far away from the real bound.

With respect to the time complexity bound for algorithm PARTITION we have the following result.

*Theorem 3.2:* The time complexity of algorithm PARTITION is $O(n^2)$.
*Proof:* The proof of this theorem is straight forward. For brevity it will be omitted.

If compound cuts are not defined in algorithm PARTITION, we obtain another algorithm which in some cases generates better solutions than the ones generated by algorithm PARTITION. The new algorithm always makes a vertical cut along $x_2$ and recursively solves the two remaining problems. The approximation bound is 5 and the time complexity is the same as before, $O(n^2)$. The proof for the approximation bound is much more elaborate than the one for PARTITION, but the constant associated with the time complexity bound is smaller than the one for algorithm PARTITION. We cannot claim that the solution generated by one of these algorithms is always better than the solution generated by the other, because there are instances for which any of these two algorithms outperforms ( with respect to the objective function value ) the other. We can also show that for some problem instances the solutions generated by both of these algorithms is equally "bad" ( see example 1 ). In summary, the approximation algorithms discussed in this section have approximation bounds that are large because of the lower bound function. The main problem with the lower bound function is that it only takes into account local relationships between the points and ignores their global relation. In the next section we try to circumvent this problem.

*IV. Approximation Algorithms Using Problem Transformation.*

In this section we present another approximation algorithm for the RG-P problem. This new algorithm tries to avoid the drawback of the previous algorithms. The new algorithm makes decisions based on the global relationship between the points and the lower bound function used in this case is closer to the optimal solution. Our approach consists of transforming the instance of the RP-G problem into an instance of a generalized RP-HF problem for which we can find an optimal partition in polynomial time (see appendix). Such a partition is our solution to the original problem. The first transformation is performed by scanning the points inside the boundary one by one and introducing jogging lines to make them directly or indirectly connected to the global boundary. Let $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$..... $p_n = (x_n, y_n)$ be the set of points inside the rectangle and assume that they have been reordered in such a way that:

1) $x_i \le x_j$, for $1 \le i < j \le n$; and
2) if $x_i = x_j$ then $y_i > y_j$, for $1 \le i < j \le n$.

The scanning traverses the points in the order $p_1$, $p_2$, ..., $p_n$. During the ith iteration point $p_i$ is connected directly or indirectly to the global boundary by a vertical and a horizontal line. Point $p_i$ is connected to (x,y), for some $x \le x_i$ and $y \ge y_i$, by a shortest path. The path consists of a most one vertical line segment and at most one horizontal line segment. The vertical line segment ( if present ) must be adjacent to $p_i$ and (x,y) must be a boundary point, a line segment introduced before the ith iteration or one of the first i - 1 points. Let C( I ) be the set of lines introduced by the above scanning rule. Figure 4.1 shows the lines introduced for some instance I.

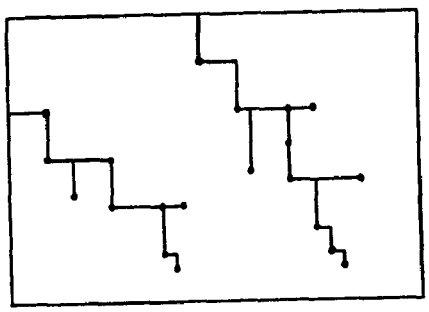


figure 4.1

The instance displayed in figure 4.1 is an instance of the generalized RP-HF problem. In such a problem we do not have holes nor points. We only have a rectilinear boundary with jogging lines connected to it. These jogging lines should be viewed as boundaries. Each side of a line segment is a boundary. Since the algorithm in [LPRS] can be trivially modified to find an optimal solution to this problem we do not include the result in the main text. An explanation on how to generalize the results in [LPRS] is given in the appendix.

*Algorithm TRANS*

Step 1: Construct an instance of the generalized RP-HF problem by performing scanning procedure;

Step 2: Use the algorithm in the appendix to find an optimal solution to the RP-HF problem constructed by step 1.

*end* of algorithm

*Theorem 4.1:* The time complexity for algorithm TRANS is $O(n^4)$.

*Proof:* For brevity the proof is omitted.

Let $E_{apx}(I)$ be the solution generated by algorithm TRANS. Let $E(I) = C(I) \cup E_{opt}(I) \cup D(I)$, where $E_{opt}(I)$ is an optimal solution for I and D( I ) is a set of line segments (to be defined below) needed to make $C(I) \cup E_{opt}(I)$ a rectangular partition. Clearly $L(E(I)) \geq L(E_{apx}(I))$. If we can show that $L(E(I)) \leq 3\,L(E_{opt}(I))$, then $L(E_{apx}(I)) \leq 3\,L(E_{opt}(I))$. Therefore, to prove our result it is only required to show that $E(I) = C(I) \cup E_{opt}(I) \cup D(I)$ is a rectangular partition for the generalized RP-HF problem constructed by step 1 in procedure TRANS and to prove that $L(E(I)) \leq 3\,L(E_{opt}(I))$. Before proving these results, we make some definition and reduce further our proofs.

Let $P(I) = C(I) \cup E_{opt}(I)$ for some optimal solution $E_{opt}(I)$. Clearly $E_{opt}(I)$ is an rectangular partition. Every time we add a path from C( I ) to it we will partition some rectangles and perhaps introduce a joint (see figure 4.2) in a rectangle. After introducing all the lines in C( I ), we can easily partition P( I ) into rectangles with and without joints ( see figure 4.2 ).

figure 4.2

To produce a rectangular partition for P( I ), we must eliminate the joints by introducing one line segment and thus partition the rectangle with a joint into three rectangles. All of these lines that we introduce are the elements of a set that we define as D( I ). Clearly all joints consist of only line segments from $E_{apx}(I)$. Each of the sides of a rectangle is either a boundary, a line segment from $E_{opt}$ or a line segment from $E_{apx}$. Note that in some situations a side of a rectangle is a line segment from $E_{opt}$ and a line segment from $E_{apx}$. The lines in $C(I) - E_{opt}(I)$ are labeled A. What we will show is that $L(D(I) \cup (C(I) - E_{opt}(I))) \leq 2\,L(E_{opt}(I))$. We prove this result by showing that for each rectangle R in P( I ), LEN( R ) ≤ OPT( R ), where LEN( R ) is the sum of the length of the edges from D( I ) inside rectangle R plus the sum of the length of the edges labeled A that belong to the bottom and right sides of the rectangle plus the length of the lines that form the joint (if present); and OPT( R ) is the sum of the length of the edges in $E_{opt}(I)$ that belong in rectangle R. Clearly, in order to complete the proof of our approximation bound it is only required to show that LEN( R ) ≤ OPT( R ) for all R. For rectangle R, let X be its width and Y be its height. Before proving our result we prove some useful properties that are satisfied by every rectangle R.

*Lemma 4.1:* For every rectangle R,

(a) If the joint is present it can only be located on the bottom left corner of R;

(b) If the joint is present then neither the left nor the bottom side of R can be labeled A nor can they be boundaries;

(c) No side labeled A can be adjacent to a boundary located on the right or the bottom side of R;

(d) It is impossible for both the bottom and right side of R to be labeled A;

(e) There cannot be a path all of it labeled A that joins two boundaries; and

(f) If the left and right side of R is labeled A, then X ≥ Y.

(g) If the top and bottom side of the rectangle is labeled A, then $Y \geq X$.

*Proof:* For brevity the proof will be omitted.

•

*Lemma 4.2:* For all R, $LEN(R) \leq OPT(R)$.

*Proof:* The proof appears in [GZ].

•

Note that if algorithm PARTITION is used then c is $(4+\sqrt{3})$, and if algorithm TRANS is employed then c is only 4.

*Theorem 5.2:* Algorithm REP has time complexity $O(n^4)$.

*Proof:* For brevity the proof is omitted.

•

*V. Discussion.*

Our algorithms have an approximation bound that is far from optimal, however we believe that the solutions generated by our algorithms are usually very close from optimal. Presently we are working on approximation algorithms for the RP-RPP problem. We believe that the techniques used by our algorithms can be generalized to solve these other problem with an approximation bound which is very close to the previous bounds. Note that this result would imply the solution of all the partition problems given in figure 1.

*VI. References.*


[AHU] Aho, A. V., J. E. Hopcroft and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

[AT] Avis, D. and G. T. Toussaint, An Efficient Algorithm for Decomposing a Polygon into Star-shaped Polygons, *Pattern Recognition*, Vol. 13, 1981.

[CD] Chazelle, B. and D. Dobkin, Decomposing a Polygon into its Convex Parts; Proc. 11th ACM Symp. on Theory of Comput., 1979.

[GJPT] Garey, M. R., D. S. Johnson, F. P. Preparata and R. E. Tarjan, Triangulating a Simple Polygon; *Information Processing Letters*, Vol. 7, No. 4, (June 1978).

[GZ] Gonzalez, T. and S-Q. Zheng, Approximation Algorithms for Partitioning Rectilinear Polygons, Technical Report, University of California, Santa Barbara, March 1985.

[LLMP] Lodi, E., F. Luccio, C. Mugnai, and L. Pagli, On Two-dimensional Data Organization I; *Fundamenta Informaticae*, Vol. 2, No. 2 (1979).

[LLMPL] Lodi, E. F. Luccio, C. Mugnai, L. Pagli and W. Lipski, Jr., On Two-dimensional Data Organization II; *Fundamenta Informaticae*, Vol. 2, No. 3 (1979).

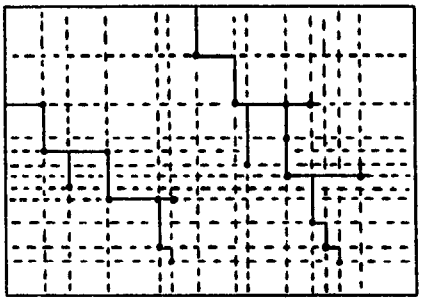
[LPRS] Lingas, A., R. Y. Pinter, R. L. Rivest, and A. Shamir, Minimum Edge Length Partitioning of Rectilinear Polygons, Proc. 20th Annual Allerton Conference on Communication, Control, and Computing, Monticello, Illinois, Oct. 1982.

[R] Rivest, R. L., The "PI" (Placement and Interconnect) System, Proc. 19th Design Automation Conference, June 1982.

[S] Sack, J. R., An O( n log n ) Algorithm for Decomposing Simple Rectilinear Polygons into Convex Quadrilaterals, Proc. 20th Annual Allerton Conference on Communication, Control and Computing, Oct. 1982.


*Appendix:* An $O(n^4)$ Optimal Algorithm for the Generalized RP-HF Problem.

In this section, we introduce an algorithm for the generalized RP-HF problem. The instance displayed in figure 4.1 is an instance of the generalized RP-HF problem. In the RP-HF problem do not have holes nor points. We only have a rectilinear boundary with jogging lines connected to it. These jogging lines should be viewed as boundaries. Each side of a line segment is a boundary. The grid induced for the problem instance given in figure 4.1 is displayed in figure A.1.



From these observations it is simple to see that the algorithm given in [LPRS] can be used to solve the generalized RP-HF problem. The time complexity of the algorithm is $O(n^4)$.