GEBZE TECHNICAL UNIVERSITY

Computer Engineering Department

CSE 495 Graduation Project Minimum Rectangle Cover Problem

Tugay Apaydın
1801042081

Project Consultant
Assoc. Prof. Didem GÖZÜPEK

January 2021

GEBZE TECHNICAL UNIVERSITY

Computer Engineering Department

CSE 495 Graduation Project Minimum Rectangle Cover Problem

Tugay Apaydın
1801042081

Project Consultant
Assoc. Prof. Didem GÖZÜPEK

January 2021

This study was accepted as Undergraduate Graduation Project in Computer Engineering Department by the jury below on …/…/202...

Graduation Project Jury

Consultant Name		
University		
Faculty		

Jury Name		
University		
Faculty		

Jury Name		
University		
Faculty		

TABLE OF CONTENTS

TABLE OF CONTENTS 4
FIGURE LIST 5
SUMMARY 6
1. INTRODUCTION 7
1.1. DEFINITIONS 8
2. ALGORITHM INSPECTION 11
2.1. ANALYSING THE POLYGON 11
2.2. MINIMUM COVER ALGORITHM OF T. OHTSUKI [3] 11
3.2.2. Maximum Independent Set of Chords 13
3. ALGORITHM ANALYSIS 18
4. RESULT 20
5. RESOURCES 21

FIGURE LIST

Figure 1: VLSI chip with thousands of integrated transistors [1] 7
Figure 2.1: Ex. a rectilinear polygon P with edges in black 9
Figure 2.2: Ex. The rectilinear polygon P covered by rectangles in green. 9
Figure 3.1: Ex. The rectilinear polygon P with chords in blue dotted line. 11
Figure 3.2: Rectilinear polygon P with independent chords in black line. 12
Figure 3.3: Rectilinear polygon P after Step 2. 12
Figure 4: Bipartite graph that represents intersection of chords of the polygon P.... 13
Figure 5: A bipartite graph converted to a network 14
Figure 6: Theorem, E. L. Lawler, "Combinatorial Optimization: Networks and Matroids", Holt, Rinehart and Winston, New York, 1976.
..... 16

SUMMARY

In this project the minimum rectilinear polygon cover problem, i.e., the problem of covering rectilinear polygons with the minimum number of rectangles is considered.

This problem is commonly studied for some important applications like VLSI chip design or reducing waste of material in production etc. The most common approach to solving the problem of covering rectilinear polygons is to partition the polygon into smaller sub polygons. This part is usually the most complex part of the general algorithm and so, defines the general time complexity of the algorithm.

The other approaches to minimum cover problem may be more specific. Some of the works done by others more focused on the VLSI chips. That's why, their algorithms work on simple polygons. Compared to their algorithms, in this project, my implementation is more general which can be used for any type of rectilinear polygon. Experimental results have shown that the algorithm that is discussed in this project works with $O(n^2)$ time complexity.

1) INTRODUCTION

In this project, we consider the minimum rectilinear polygon cover problem, i.e., the problem of covering rectilinear polygons with the minimum number of rectangles. This problem finds applications in VLSI design, in order to minimize the cost of the fabrication it is desirable to cover the polygonal area of each layer of the circuit with as few rectangle as possible, fabrication of DNA chip arrays [Hannenhalli et al. 2002]. Other applications are cutting a set of given polygons for manufacturing, image compression etc.

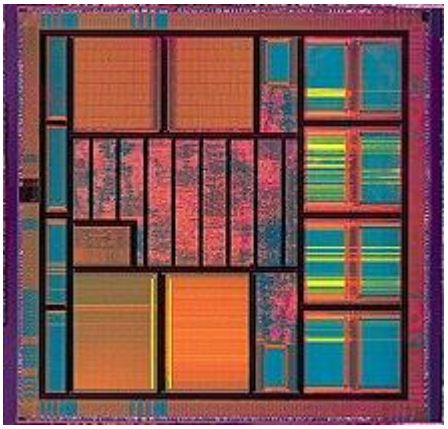

Fig. 1: VLSI chip with thousands of integrated transistors [1].

A rectangle cover for a rectilinear polygon P is a collection of rectangles contained within R, whose union exactly covers R. A minimum cover is one with the minimum number of rectangles. The functions to be performed on rectilinear polygons are often more easily performed using either a rectangle partition or a rectangle cover. In either case the rectilinear polygon is decomposed into a set of rectangles whose union is the original rectilinear polygon [2].

In order to find partition algorithm for a rectilinear polygon P with n vertices, Ohtsuki has developed an $O(n^{5/2})$ algorithm [3]. Imai and Asano have developed an $O(n^{3/2} \log n)$ algorithm [4] and Liout, Tan, and Lee have an O(nloglogn) algorithm for hole-free rectilinear polygons. [2]

Nahar and Sahni introduced a complexity measure for hole-free rectilinear polygons. In their work, they defined a variable k to measure the simplicity of the polygon. They examined 2869 polygons from VLSI mask data provided by UNISYS and found that 85% of these have $k = 1$ and 95% have $k \leq 2$ (simple polygons). They took advantage of this observation and developed a fast algorithm to partition hole-free rectilinear polygons [2].
San-Yuan Wu and Sartaj Sahni has extended the work of Nahar and Sahni and obtain an O(kn) and an O(n logk) algorithm to find the minimum cover of a hole-free rectilinear polygon. [2]

In this paper, my approach for solving this problem is similar to the algorithm developed by T. Ohtsuki.

1.1) DEFINITIONS

1. Rectilinear Polygon: A rectilinear polygon is a polygon that all horizontal and vertical edges (sides) are parallel to the axes of Cartesian coordinates.
2. Rectangle: A rectangle is a 4-sided flat shape with straight sides where all interior angles are 90° angle.
3. Vertex: A vertex is the intersection of a horizontal and vertical edge of a rectilinear polygon.
A **concave vertex** is the intersection of two edges which form a 270° angle inside the polygon.
A **convex vertex** is the intersection of two edges which form a 90° angle inside the polygon.
Two concave vertices are **cohorizontal** if there is a horizontal line from one to the other that lies in polygon.
Two concave vertices are **covertical** if there is a vertical line from one to the other that lies in polygon.

4. Rectangle Cover: A rectangle cover for a given rectilinear polygon P is to find a collection m of rectangles whose union is exactly equal to the polygon P.

A **minimum rectangle cover** is a rectangle cover such that the size of the collection m is minimized.

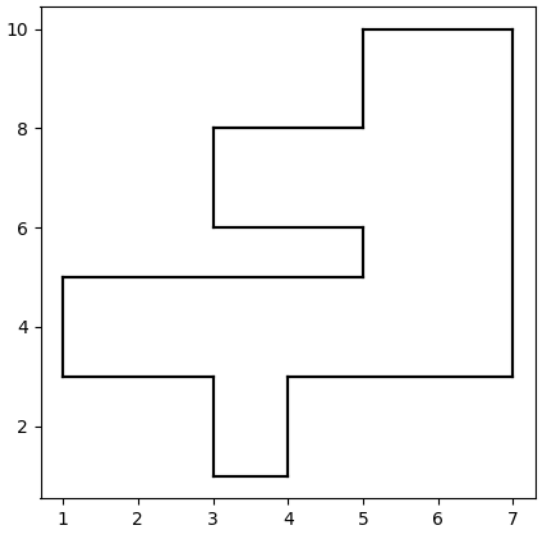

Fig. 2.1: Ex. a rectilinear polygon P with edges in black.

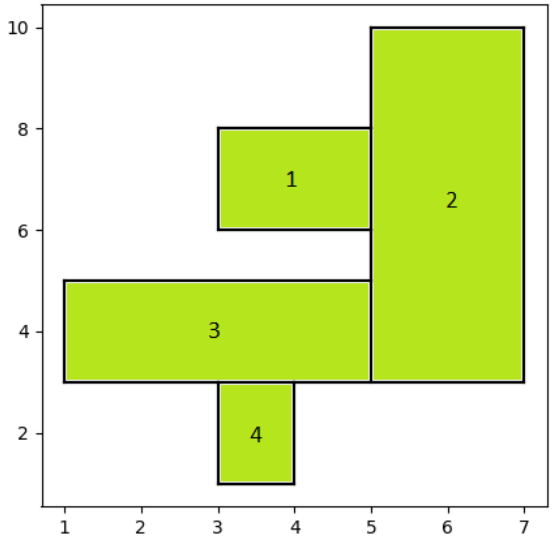

Fig. 2.2: Ex. Rectilinear polygon P covered by rectangles in green.

5. Chord: A chord is a line segment connecting two vertices.

A **horizontal chord** is a line segment that lies wholly within the polygon and joins two cohorizontal concave vertices.

A **vertical chord** is a line segment that lies wholly within the polygon and joins two covertical concave vertices.

6. Bipartite Graph: A bipartite graph G is a graph whose vertex set V can be partitioned into two nonempty subsets A and B (i.e., $A \cup B = V$ and $A \cap B = \emptyset$) such that each edge of G has one endpoint in A and one endpoint in B. The partition $V = A \cup B$ is called a bipartition of G.

7. Independent Set: An independent set in a given graph $G = (V, E)$ is a subset $I \subseteq V$ where none of the vertices are adjacent.

8. Matching: A matching, M, of a graph $G = (V, E)$ is a subset of the edges E, such that no vertex in V is incident to more than one edge in M.

A matching M is said to be **maximum matching** if for any other matching M', $|M| \geq |M'|$.

In the following sections, the general approach of the algorithm is discussed. After that, algorithms explained detailed and analyses are done.

2) ALGORITHM INSPECTION

2.1) ANALYSING THE POLYGON

As we know; in a rectilinear polygon, vertical edges are parallel to the y-axis and horizontal edges are parallel to the x-axis. That's why, we will focus on horizontal and vertical chords.

We will first find the concave vertex set and horizontal and vertical chord sets.

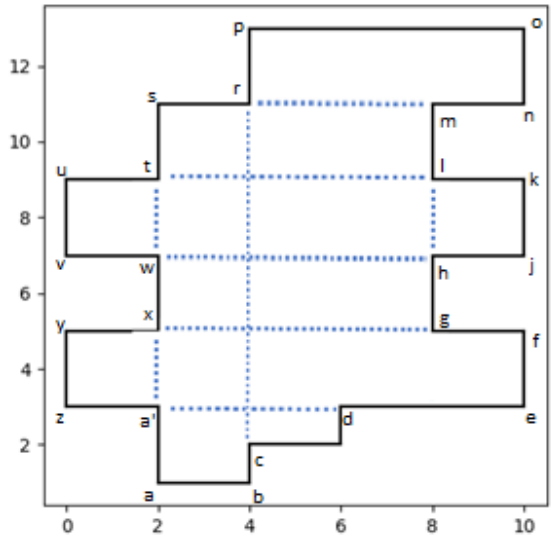

Fig. 3.1: Ex. The rectilinear polygon P with chords in blue dotted line.

In figure 3.1, the set of the concave vertex are (a', d, g, h, l, m, r, t, w, x). Some of the vertical chords are {tw, lh, xa', rc} and some of the horizontal chords are {rm, wh, a'd}. In a rectilinear polygon P, concave vertices can be calculated in O (1) time and chord sets can be found by using a simple sweep line algorithm which has O(n) time complexity.

2.2) Minimum Cover Algorithm of T. Ohtsuki [3]

The minimum cover algorithm of T. Ohtsuki uses three steps:

Step 1: Find the maximum independent chord set M for the polygon P (a set of chords is independent if none of the chords in the set intersect).

By drawing the chords in the maximum independent chord set, polygon is partitioned into smaller rectilinear polygons.

Step 2: From each of the concave vertices that a chord was not drawn in Step 1, draw a vertical or horizontal line that lies wholly within the smaller rectilinear polygon created in Step 1.

I explain finding maximum independent chord set in a polygon P in section 2.1.1.

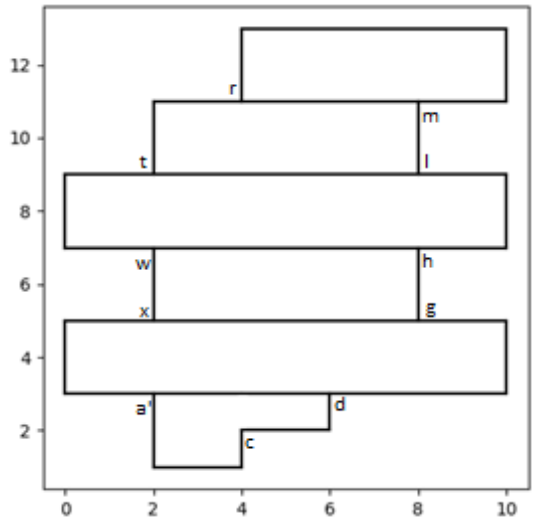

Fig. 3.2: Rectilinear polygon P with independent chords in black line.

As shown in figure 3.2, the maximum independent chord set M of the polygon P is {rm, tl, wh, xg, a'd}. By drawing the chords in the M, the polygon P is partitioned into smaller polygons.

We can simply find the set of the concave vertices from which a chord is not drawn ({c}) by using a sweep line algorithm.

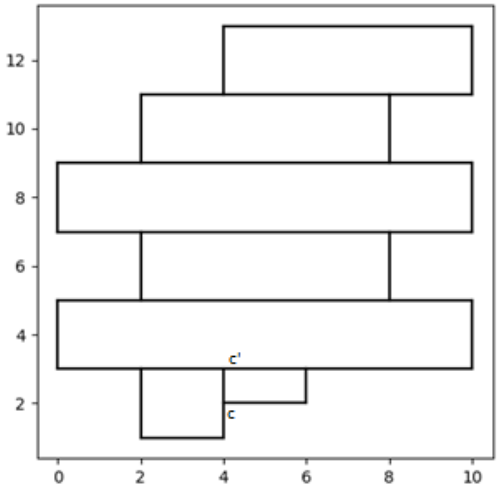

Fig. 3.3: Rectilinear polygon P after Step 2.

2.2.1) Maximum Independent Set of Chords

T. Ohtsuki shows that the problem of finding a maximum independent chord set can be converted to finding a maximum independent vertex set in a bipartite graph and a maximum independent vertex set of a bipartite graph is related to a maximum matching.

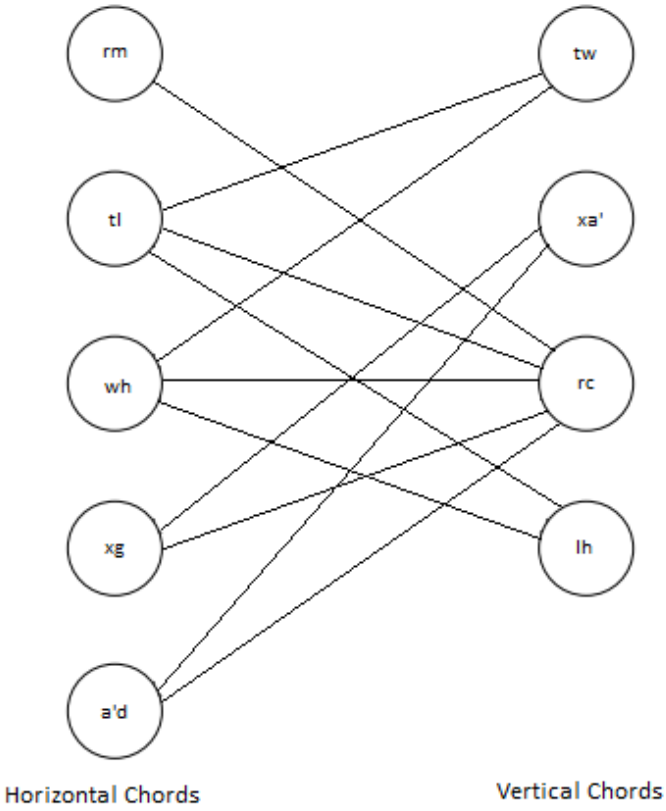

Fig. 4: Bipartite graph that represents intersection of chords of the polygon P.

Let H be the horizontal chord set and V be the vertical chord set of the polygon P. Firstly, a bipartite graph G = (H ∪V, E) is created. In this graph, there is an edge between two vertices if the chords they represent intersect. Figure 4 shows the bipartite graph G that represents polygon P also shown in figure 3.

Finding a maximum matching in a graph G can be done by reducing the problem to finding maximum flow in a network.

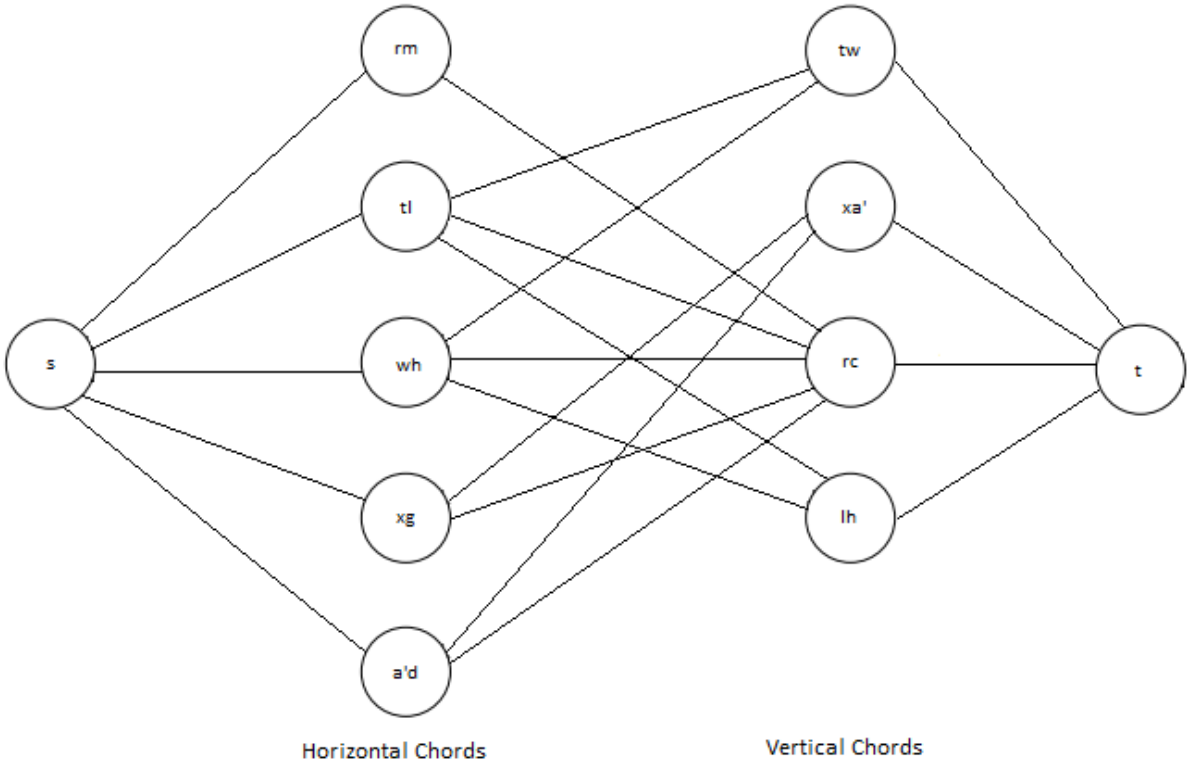

Fig. 5: Bipartite graph converted to a network.

Using network flow to solve bipartite matching:

1) Given bipartite graph G = (H ∪V, E), direct the edges from H to V.
2) Add new s (source) and t (sink) vertices.
3) Add an edge from s to every vertex in H.
4) Add an edge from every vertex in V to t.
5) Make all the capacities 1.
6) Solve maximum network flow problem on this new graph.

Figure 5 shows the graph G after reducing the problem to a finding maximum flow in a network problem.

H. Imai, and T. Asano [4] shows that a maximum matching of g-bipartite graph G can be found in O(√n N) time where m is number of pairs of intersecting segments and N = min{m, nlog n}.

In this paper, I use Ford-Fulkerson algorithm for maximum flow problem. The algorithms are discussed detailed in section 3.

After finding a maximum matching M for a polygon P, maximum independent set of chords can be found by using the following theorem [2].

Theorem 1: [11] Let G = (H∪V, E) be a bipartite graph. Let M be a maximum matching of G and let F be the set of free vertices relative to this matching (i.e., F contains all vertices of G that are not matched in M).

(a) Every MIS of G has cardinality H + V - M.

(b) There is an MIS S of G such that F ⊆ S and for every edge (v, h) ∈ M exactly one of v and h is

Using the above theorem, the algorithm shown in Figure 6 can be used to construct maximum independent chord set by using a maximum matching M [2].

```
Procedure MaxInd(G,M,S);
{Given a bipartite graph G = ( H∪V, E) and a maximum matching M ⊆ E.  Find an MIS S such that
S ⊆ H∪V}
begin
      S := ∅;
      F := {u | u ∈ H∪V and (u, x) ∉ M for any x}; {Free vertices}
      while (F ≠ ∅ ) or (M ≠ ∅) do
      begin
            if F ≠ ∅ then
            begin {add a free vertex to S}
                  Let u ∈ F;  F := F - {u};  S := S ∪ {u};
            end
            else {add a vertex in M to S}
            begin
                  Let (u, v) ∈ M;  M := M - {(u, v)};
                  E := E - {(u, v)};  S := S ∪ {u};
            end;
            { Process vertex u }
            for all (u, v) ∈ E do
            begin
                  E := E - {(u, v)};
                  if there is an h such that (v, h) ∈ M then
                  begin
                        M := M - {(v, h)};  F := F ∪ {h}; {h is free}
                  end;
            end; {of for}
      end;{of while}
end;{of MaxInd}
```

Fig. 6: Theorem, E. L. Lawler, "Combinatorial Optimization: Networks and Matroids", Holt, Rinehart and Winston, New York, 1976.

T. Ohtsuki used the O ($n^{5/2}$) bipartite graph matching algorithm of Hopcroft and Karp [7]. Since the remaining steps take less time than this, Ohtsuki's implementation has complexity O ($n^{5/2}$). H. Imai, and T. Asano [4] have developed an O ($n^{3/2}$ log n) algorithm by improving the execution time of the Ford Fulkerson algorithm to find a maximum matching of the intersection graph of horizontal and vertical line segments.

3) ALGORITHM ANALYSIS

Until this part, I discussed the general algorithms of the project. In this part, I will explain some of the algorithms and their analysis more precisely.

In order to create a rectilinear polygon P, a simple algorithm can be used to create a graph G and read all vertices and edges into G. In my project, I used hashing methods to create the graph G. That's why, since in a rectilinear polygon every vertex can have at most 4 edges connected to it, finding an edge in the graph G takes constant (O(1)) time in worst case.
By using the algorithm that is used to find an edge and calculating concave vertices, chord sets in a graph G may be found in O(n) time.

Now, we have horizontal and vertical chord sets. By using these sets, a bipartite graph is created and converted to a network in order to find maximum matching.

Basically, a simple algorithm may be used to create a network G' in O(n) time.
Now we will discuss maximum flow algorithm.

Ford Fulkerson Algorithm for Maximum Flow in a graph:
Since we only need the edges that are used in flow, there is no need to calculate total flow.

- Repeat until there is no path from s to t:
- Run Breadth First Search from source vertex s to sink vertex t to find a flow path
- Let f be the minimum capacity value on the path
- For each edge H → V on the path
    - Decrease capacity of the edge e (H → V) by f
    - Add the chord (edge e) that is used in flow to an array M.

At the end, M will be maximum matching set of the graph G.
In our graph G, there will be no duplicate edges and each edge has unit capacity (1 or infinite). Then, the maximal flow f will not be more than the number of vertices.

That’s why the time complexity of the Ford Fulkerson Algorithm will be O(V * E) (where V number of vertices and E is number of edges in a graph). But we can also think that, a vertex in a rectilinear polygon can have at most 4 edges (vertical edges must be parallel to y-axis and horizontal edges must be parallel to x-axis). By using this approach, we can say that the Ford Fulkerson Algorithm takes O(V* 4V) which is equal to $O(V^2)$.

Finding Maximum Independent Set of Chords (MISC):

In a set of maximum matching M, we will assume that there are n chords. Then, the algorithm will be executed for O(n + number of free vertices) times which is O(n). For each iteration over a chord, All of the edges in the polygon will be searched and some of the edges that are related to the edge that is searched, will be removed. That’s why, The total time complexity T(n) = O(n) * O(logn) = O(nlogn).

Since finding the maximum matching set in a graph G has the worst case time complexity compared to all other algorithms, the total time complexity of the minimum cover problem will be $O(n^2)$ where n is the number of vertices in a rectilinear polygon P.

RESULT

In this project, an algorithm for covering rectilinear polygons with minimum number of rectangles is considered. A common approach for covering a rectilinear polygon is to reduce the polygon into sub polygons. It is pointed that the most important part of the algorithm is to find a maximum matching for a bipartite graph. H. Imai, and T. Asano [4] created the most efficient algorithm for this part of the algorithm.

Experimental results show that the average and worst case run time of the algorithm is $O(n^2)$ (where n is the number of vertices). While some algorithms focused on some specific (simple) polygons, this algorithm works generally for any type of rectilinear polygons.

RESOURCES

[1] Jaromił Najman,“Covering Rectilinear Polygons by Rectangles”
[2] San-Yuan Wu and Sartaj Sahni, “Fast Algorithms to Partition Simple Rectilinear Polygons”
[3] T. Ohtsuki., "Minimum Dissection of Rectilinear Regions", Proceedings 1982 International Symposium on Circuits and Systems (ISCAS), 1982, pp. 1210-1213
[4] H. Imai, and T. Asano, "Efficient Algorithms for Geometric Graph Search Problems", SIAM
[5] W. T. Liou, J. J. M. Tan, and R. C. T. Lee, "Minimum Partitioning Simple Rectilinear Polygons in O(nloglogn) time", Proceedings of the Fifth annual symposium on Computational Geometry, pp344-353, 1989.
[6] S. Nahar and S. Sahni, "A Fast Algorithm for Polygon Decomposition", IEEE Trans. on
[7] J. E. Hopcroft and R. M. Karp, "An n 5/2 Algorithm for Maximum matchings in bipartite graphs", SIAM J. Comput. 2, 1973, pp. 225-231.