Partitioning of Rectilinear Polygons

Design Project

Submitted in fulfillment of the requirements of
MATH F376

By

Govind Mittal
ID No. 2014B4A7530P

Under the supervision of:

Dr. Krishnendra Shekhawat
Mathematics

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, PILANI CAMPUS
April 2017

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, PILANI CAMPUS

Abstract

Bachelor of Engineering (Hons.) Computer Science & Master of Science (Hons.) Mathematics

Partitioning of Rectilinear Polygons

by Govind MITTAL

In this report, I review two algorithms used in partitioning rectilinear polygons, containing no holes in them. One has complexity of **O(kn)** and is used for partitioning the polygon into maximum number of non-overlapping rectangles and other has **O(nlogk)**, which is used for partitioning the polygon into the fewest number of non-overlapping rectangles, where n is the number of vertices in the polygon and k is a hyper parameter which is decided by the arrangements of vertices in the polygons. The algorithm is simulated and tested on various inputs. The major portion of the project is studied from the paper by San-Yuan Wu and Sartaj Sahni[1]. The algorithm is improvised and better implementation patterns have been generated.

Keywords: Rectilinear hole-free polygons, algorithm design, computational geometry. . .

Acknowledgements

I would like to express my deepest appreciation to Dr. Balram Dubey, Head of Department of Mathematics at Birla Institute of Technology and Science Pilani, Pilani Campus for giving me an opportunity to enroll in **Design Project – MATH F376**. I express my special gratitude to my supervisor, **Dr. Krishnendra Shekhawat** who gave me the golden opportunity to do this interesting project on the topic, 'Partitioning of Rectilinear Polygons' which also helped me in doing a lot of self-motivated research and resulted in expansion of my knowledge frontiers. With his feedbacks and patience in letting me choose my topic, the project would not have been what it is.
Furthermore, I would also like to acknowledge with much appreciation the crucial role of my friends. Without their valuable feedbacks and discussions of the applications of crucial concepts in this field, the project would not have been done this well.

Govind Mittal

Contents

List of Figures

Chapter 1

Introduction

Rectilinear Polygons are a simple connected single-cyclic graph in $\mathbb{R}\times\mathbb{R}$, such that each of its edge is perpendicular or in-line with another one of its edge(s).

We encounter rectilinear polygons in various fields and applications such as network desgining, computer graphics, databases, VLSI Layout and image processing. The functions to be performed on rectilinear polygons are often more easily performed using either a rectangle partition or a rectanglular cover. In either case the rectilinear polygon is decomposed into a set of rectangles whose union is the original rectilinear polygon. If every two elements in the set of rectangles are disjoint, then it the set is called as a *partition.*

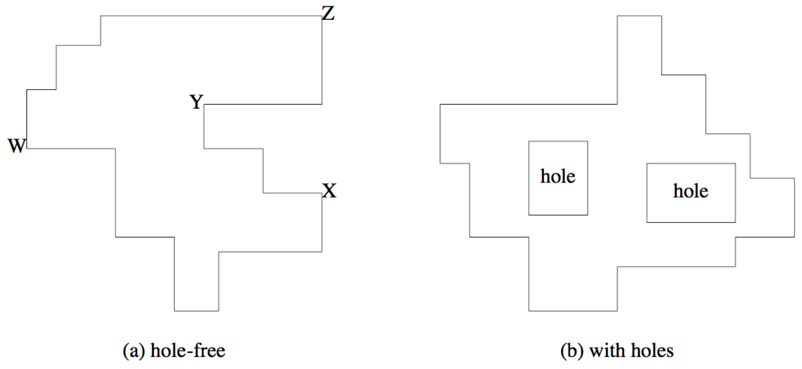


Figure 1.1: Rectilinear Polygons [1]

A ***minimal non-overlapping cover*** of a rectilinear polygon P is a rectangle partition of P that contains the fewest possible number of rectangles. The paper discusses a new complexity measure for the hole free rectilinear polygons. The paper defines the number of *horizontal inversions,* k_H to be twice the minimum number of changes in horizontal motion while traveling around the polygon once. Similarly, the number of *vertical inversions,* k_V to be twice the minimum number of changes in vertical motion while traveling around the polygon once.

The main complexity factor is k which is given by

$$k = \min\{k_H, k_V\}$$

This complexity measure covers a wide variety of polygons. The paper analysed 2869 polygons of which 85% had $k = 1$ and 95% had $k \leq 2$. Thus, for most practical reasons the *O(kn)* algorithm is linear in time.

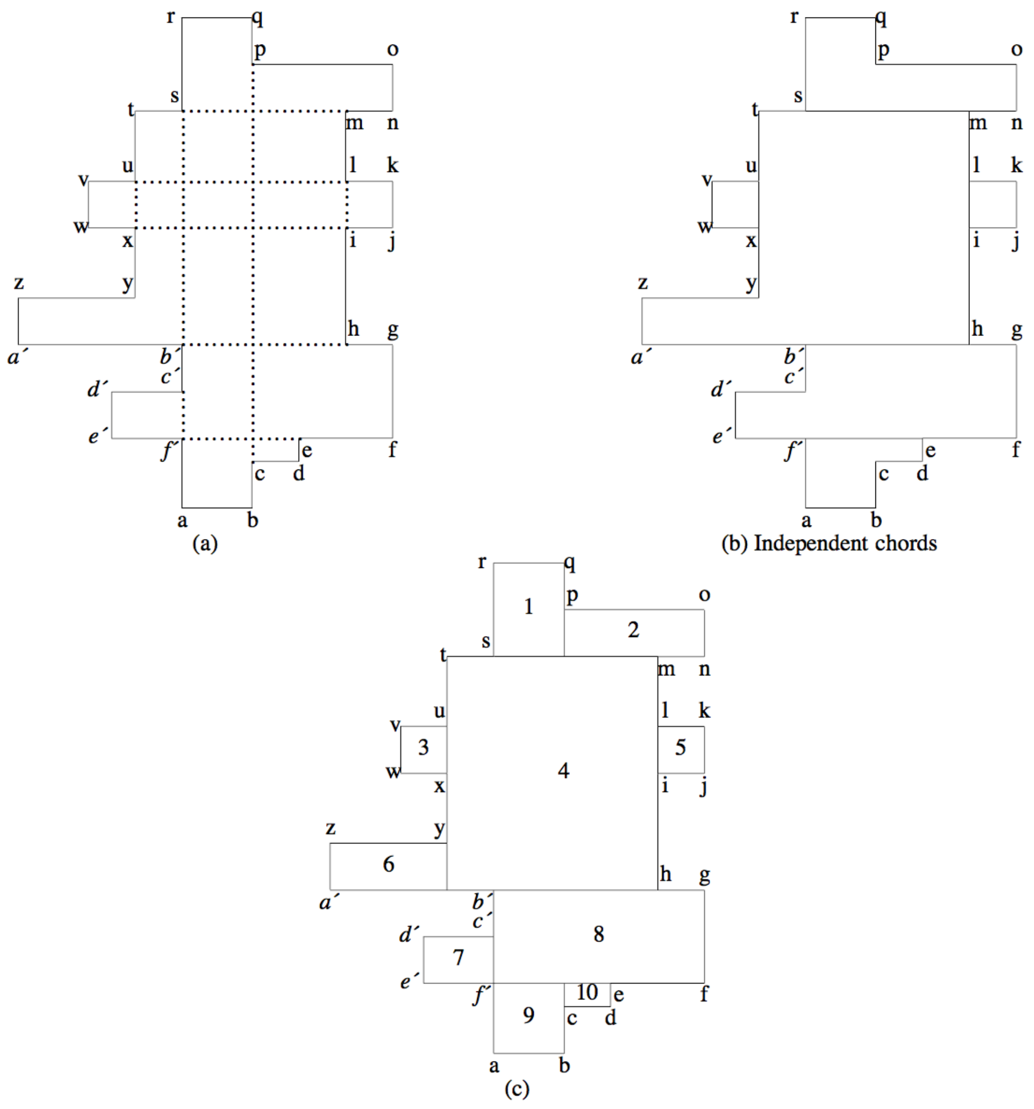


FIGURE 1.2: Maximum Non-overlapping cover of a rectilinear polygon [1]

Chapter 2

Definitions

1. **Concave Vertices:** A vertex is said to be concave if the two edges intersecting at it makes an angle of 270° with the interior of the polygon. For example, in Figure 1.2, the set of concave vertices = {c, h, i, l, m, p , s}

2. **Convex Vertices:** A vertex is said to be concave if the two edges intersecting at it makes an angle of 90° with the interior of the polygon. For example, in Figure 1.2, the set of convex vertices = {a, b, e, f}

3. **Chords:** A chord is a line joining any two co-horizontal concave vertices.

4. **NEB(v):** Neighbourhood of a vertex v of a graph is defined as the set of all vertices, that have an edge incident from that vertex.

5. **Convex Bipartite Graph:** A bipartite graph is said to be convex in V, if ∀ v ∈V, we can write it as a closed interval of the form [FIRST(v), LAST(v)]. In a convex bipartite graph labeling is crucial and needs to be defined for all problems together.

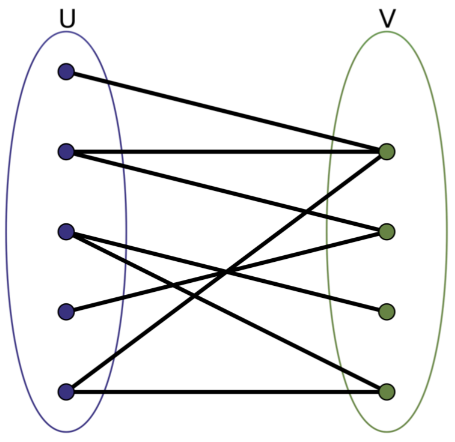


FIGURE 2.1: Convex Bipartite Graph on V [4]

6. **Matchable vertex:** A vertex is matchable iff $\exists$y $\in$ *NEB(x)* such that,*NEB(y)* $\subseteq$*NEB(z)*, $\forall$z $\in$*NEB(x)*.

7. **Maximum Partition:** Partition of given rectillinear polygon into maximum number of non-overlapping rectangles.

8. **Minimum Partition:** Partition of given rectillinear polygon into minimum number of non-overlapping rectangles, such that any two rectangles obtained , if merged will not form a rectangle.

Chapter 3

The Algorithm

The algorithm is divided into multiple steps. The report reviews each step and the way of accepting input from the user, with a simple example.

3.1 Method of Labelling the graph

We take input as a rectilinear polygon from cursor keys, i.e., up(↑), left(←), and right (→). As input is read, the pointer proceeds forward and draws a rectilinear polygon with its trail. The labelling of the vertices starts from v_0 to v_{n-1}, and $v_0 = v_n$, where n is the number of vertices in the polygon.

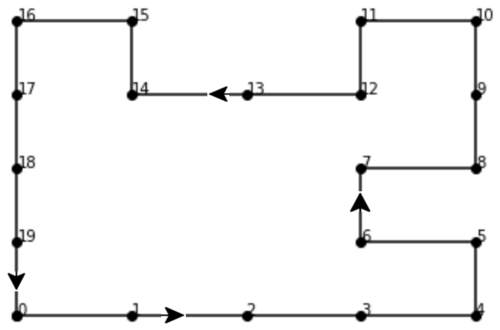


FIGURE 3.1: A Rectillinear polygon consisting of 20 vertices with direction of construction [5]

3.2 INPUTS

G = Rectilinear Graph
X = Set of Abscissa of vertices
Y = Set of Ordinates of vertices
Collinear_Vertices = Set of Collinear Vertices
Concave_Vertices = Set of Concave Vertices
Horizontal_Chords = Set of Horizontal Chords
Vertical_Chords = Set of Vertical Chords

Important points to note

1. Left and Right operations changes the direction the pointer faces.
2. Vertices that are induced after going forward consecutively. Although in the example, they are not explicitly shown, but they do exist and at a distance of one unit from its previous vertex.
3. If the interior angle made by the two edges incident at this vertex is 270 degree.
4. Chords are lines joining two vertices which are not already part of the polygon.
5. As, the way of labelling is defined, there is unique labelling of each rectillinear polygon.

EXAMPLE:
In Figure 3.1, the pointer is shown by an arrow.
Total number of vertices = 20
Collinear_Vertices = [v_1, v_2, v_3, v_9, v_{13}, v_{17}, v_{18}, v_{19}]
Concave_Vertices = [v_6, v_7, v_{12}, v_{14}]

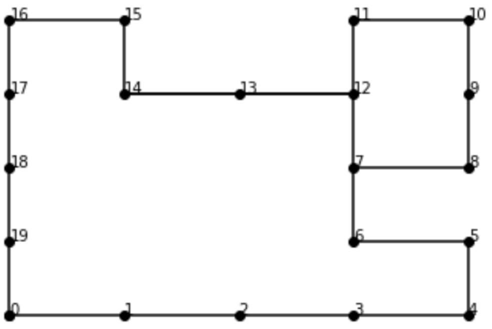
3.3 Steps of Finding Maxiumum Partition

3.3.1 Step I

```
max_partition(G):
    for u in Concave_Vertices:
        for v in Concave_Vertices and v > u+1:
            if exists a chord joining v & u and ~exists another concave
             vertex on chord joining v & u:
                if chord is horizontal:
                    add (v, u) to Horizontal_Chords
                else if chord is vertical:
                    add (v, u) to Vertical_Chords
            else :
                loop_back
```

Task Achieved: All the edges that exist between *any two concave vertices* are being added to their *respectful categories.*

EXAMPLE:



Horizontal_Chords = ϕ

Vertical_chords = $[(v_7, v_{12})]$

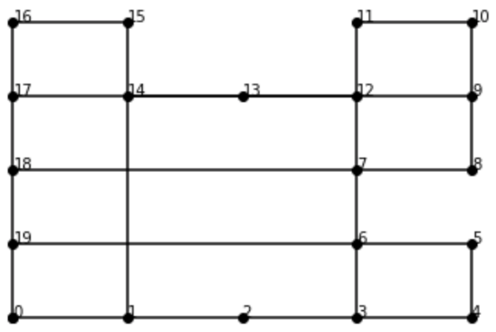
Explanation: **u** > **v** :Comparison between two vertices is done on the basis of their respective vertex indices. Here **v-u** should be greater than unity, because this assures the vertex v is not consecutive to u and has a higher index than u. Thus, iteration through each pair of vertex is done only once, making it more efficient.

In the above code, we iterate through all (concave vertex, concave vertex') pairs, and check for existence of vertical and horizontal chords, that are not intersected by any other vertex. We observe that, v_7 and v_{12} are the only two concave vertices and between whom, there exists a vertical chord. Therefore, it is added to the set of *Vertical_Chords*. Also, there does not exist any horizontal chord between any two concave vertices and therefore, set of *Horizontal_Chords* is empty.

3.3.2 Step II

```
for u in Collinear_Vertices:
     for v in Concave_Vertices:
         if exists a chord joining v & u and ~exists another concave
             or collinear vertex on chord joining v & u:
             if chord is horizontal:
                 add (v, u) to Horizontal_Chords
             else if chord is vertical:
                 add (v, u) to Vertical_Chords
         else :
             loop_back
```

Task Achieved: All the chords between *collinear vertices and concave vertices* are being added to their *respective categories.*



Horizontal_Chords = $[(v_9, v_{12}), (v_{17}, v_{14}), (v_{18}, v_7), (v_{19}, v_6)]$
Vertical_Chords = $[(v_7, v_{12}), (v_1, v_4), (v_3, v_6)]$

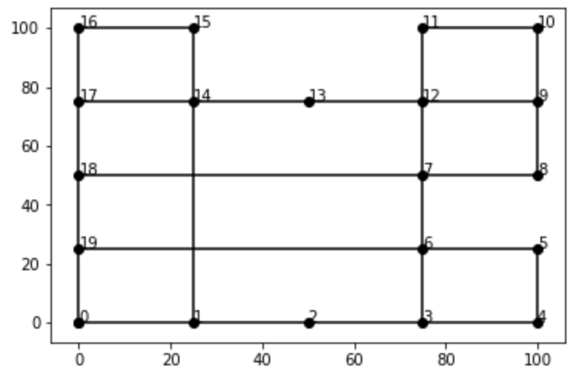
Explanation: In the above code, we iterate through all (collinear vertex, concave vertex) pairs, and check for existence of vertical and horizontal chords between them, that are not intersected by any other vertex. If any chord is found, it is added to set of *Vertical_Chords or Horizontal_Chords*, depending on its orientation.

3.3.3 Step III

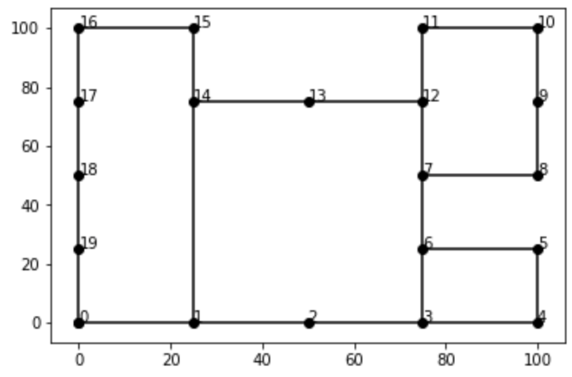
Thus, we have found all the chords, and only need to plot them now.

```
    plot(X,Y)
    plot(Horizontal_Chords)
    plot(Vertical_Chords)
    display(plot)
```

```
The maximum partitioned rectillinear polygon
<matplotlib.figure.Figure at 0x11c7b5ef0>
```



```
The mimumum partitioned rectillinear polygon
<matplotlib.figure.Figure at 0x11c7f5ef0>
```



3.3.4 Step IV

Now we have found the maximum partition, but to find the minimum partition the following needs to be done

1. Find a maximum independent set of chords (i.e., a maximum cardinality set of independent chords).
2. Draw the chords in this maximum independent set. This partitions the polygon into smaller rectilinear polygons.

3.3.5 Step V

From each of the concave vertices from which a chord was not drawn in *Step IV* draw a maximum length vertical line that is wholly within the smaller rectilinear polygon created in *Step III* that contains this vertex.

3.3.6 Step VI

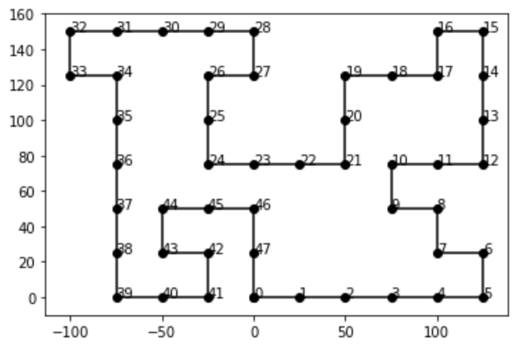
Thus, we have found all the chords, and only need to plot them now.

```
    plot(X,Y)
    plot(Horizontal_Chords)
    plot(Vertical_Chords)
    plot(Nearest_Partial_Chords)
    display(plot)
```

3.4 Test Cases

INPUT 1

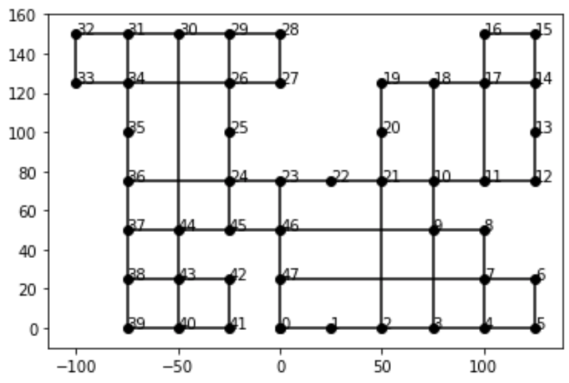
```
Initial input rectillinear graph
```



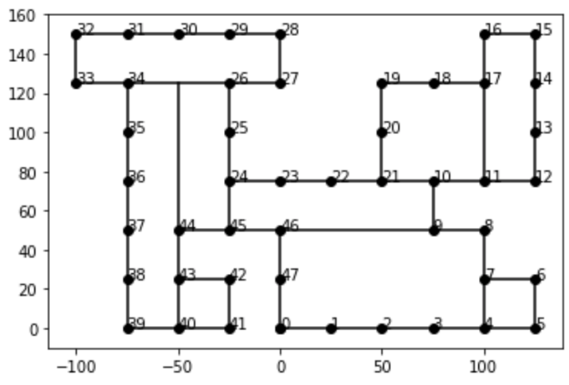
```
collinear_vertices =  [1, 2, 3, 4, 11, 13, 14, 18, 20, 22, 23, 25, 29, 30, 31, 35, 36, 37, 38, 40, 45, 47]
concave_vertices = [7, 9, 10, 17, 21, 24, 26, 34, 43, 44, 46]
horizontal_chords =  [(9, 46), (10, 21), (26, 34), (14, 17), (36, 24), (37, 44), (38, 43), (47, 7)]
vertical_chords =  [(2, 21), (3, 9), (4, 7), (11, 17), (18, 10), (23, 46), (29, 26), (30, 44), (31, 34), (40, 43), (45
, 24)]
```

OUTPUT 1

```
The maximum partitioned rectillinear polygon

<matplotlib.figure.Figure at 0x111e2b5c0>
```

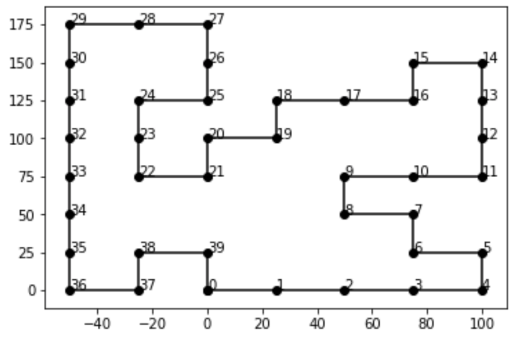


```
The minimum partitioned rectillinear polygon

<matplotlib.figure.Figure at 0x111e58da0>
```



INPUT 2

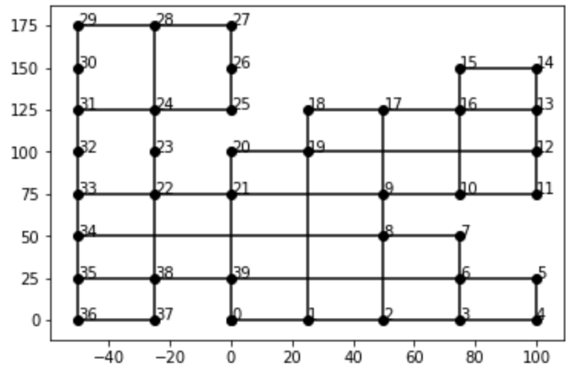
```
Initial input rectillinear graph
```



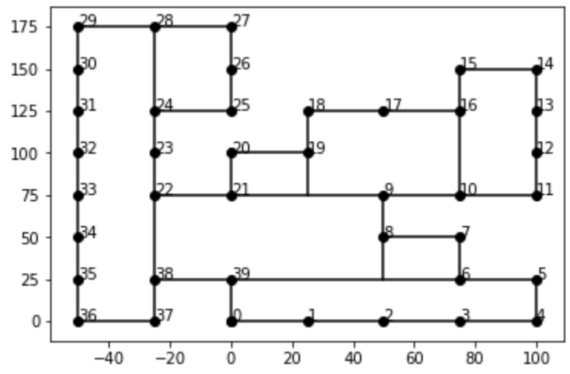
```
collinear_vertices =  [1, 2, 3, 10, 12, 13, 17, 23, 26, 28, 30, 31, 32, 33, 34, 35]
concave_vertices = [6, 8, 9, 16, 19, 21, 22, 24, 38, 39]
horizontal_chords =  [(6, 39), (9, 21), (12, 19), (13, 16), (31, 24), (33, 22), (34, 8), (35, 38)]
vertical_chords =  [(21, 39), (22, 38), (1, 19), (2, 8), (3, 6), (10, 16), (17, 9), (28, 24)]
```

OUTPUT 2

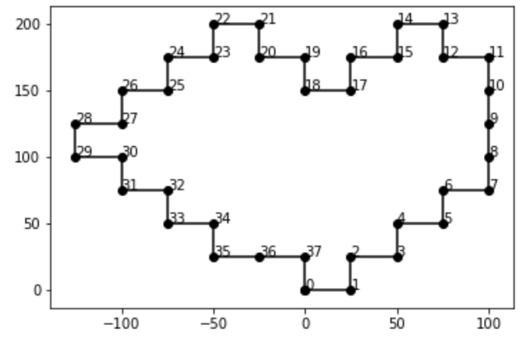
```
The maximum partitioned rectillinear polygon

<matplotlib.figure.Figure at 0x11bff2748>
```



```
The minimum partitioned rectillinear polygon

<matplotlib.figure.Figure at 0x1187b0048>
```



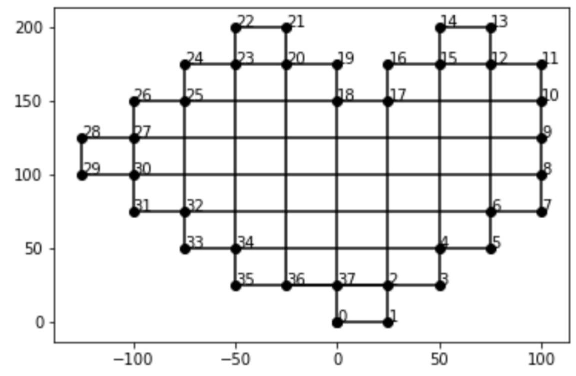
INPUT 3



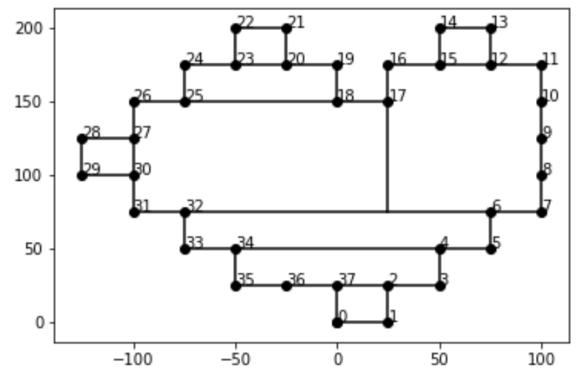
```
collinear_vertices =  [8, 9, 10, 36]
concave_vertices = [2, 4, 6, 12, 15, 17, 18, 20, 23, 25, 27, 30, 32, 34, 37]
horizontal_chords =  [(2, 37), (4, 34), (6, 32), (12, 15), (18, 25), (20, 23), (8, 30), (9, 27), (10, 17), (
vertical_chords =  [(2, 17), (4, 15), (6, 12), (18, 37), (23, 34), (25, 32), (27, 30), (36, 20)]
The maximum partitioned rectillinear polygon

<matplotlib.figure.Figure at 0x11cff98d0>
```

OUTPUT 3

```
The maximum partitioned rectillinear polygon

<matplotlib.figure.Figure at 0x11cff98d0>
```



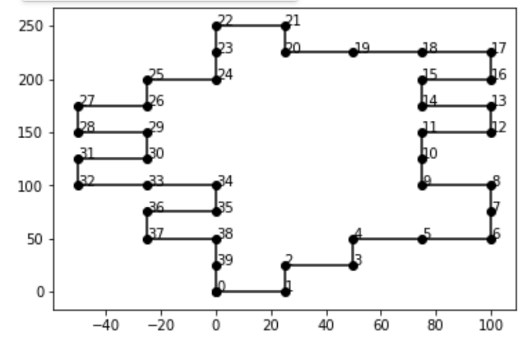
```
The mimumum partitioned rectillinear polygon

<matplotlib.figure.Figure at 0x11bbfafd0>
```



INPUT 4

```
Initial input rectillinear graph
```

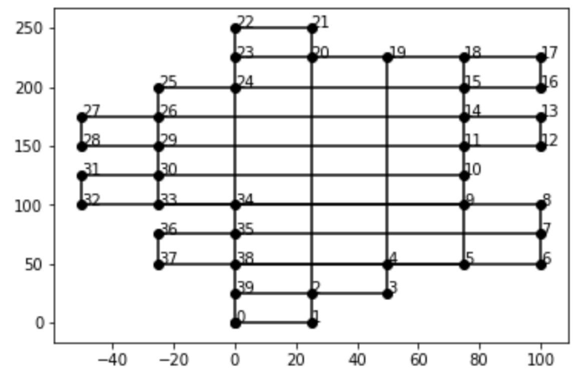
click to scroll output; double click to hide



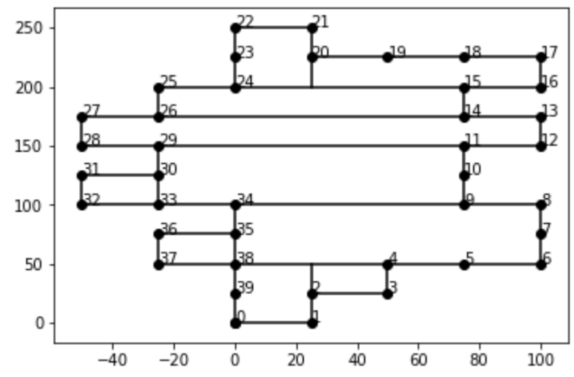
```
collinear_vertices =  [5, 7, 10, 18, 19, 23, 33, 39]
concave_vertices = [2, 4, 9, 11, 14, 15, 20, 24, 26, 29, 30, 34, 35, 38]
horizontal_chords =  [(4, 38), (9, 34), (11, 29), (14, 26), (15, 24), (5, 38), (7, 35), (10, 30), (23, 20), (33, 9), (
39, 2)]
vertical_chords =  [(2, 20), (11, 14), (24, 34), (26, 29), (35, 38), (5, 9), (18, 15), (19, 4), (33, 30)]
```

OUTPUT 4

```
The maximum partitioned rectillinear polygon

<matplotlib.figure.Figure at 0x116fc6d68>
```



```
The mimumum partitioned rectillinear polygon

<matplotlib.figure.Figure at 0x11b575390>
```



Chapter 4

Conclusion

The algorithm reviewed does not give unique solutions, as the maximum independent set is not itself unique. Since, the horizontal and vertical sets of chords themselves make up two different set of maximum independent set of chords, if the number of horizontal chords and vertical chords are equal. Also there can be other maximum independent set of chords.

My future work will involve working on problems related to applications of graph theory, involving similar but more complicated and self-motivated problems.

Thus, this implementation and design with the newest tools will help architects, in preparing their own architectural arrangements, according to their choice.

Appendix A

Working Code in Python

```
'Please make sure that the INPUT is A CLOSED RECTILINEAR POLYGON,'
'CONSTRUCTED WHILE GOING IN ANTI-CLOCKWISE ONLY'
# importing libraries
import networkx as nx
import turtle
import matplotlib.pyplot as plt
import warnings
from networkx.algorithms import bipartite
import math

warnings.filterwarnings("ignore")
# declaration list
wn = turtle.Screen()
# a turtle called gopu
gopu = turtle.Turtle()
# co-ordinate points
x = []
y = []
# vertex-type := rectilinear = -1; convex = 0; concave = 1
vertex_type = []
# store the bipartite graph of chords
G = nx.Graph()
# the line below can be commented if one needs animation
gopu.speed(0)
# starts recording keys being pressed
gopu.begin_poly()

# Event handlers
stride = 25
```

```
# Up  = up() - vertex to be deleted = -1
def up():
    vertex_type.append(-1)
    gopu.forward(stride)

# left = left() ; make vertex - convex = 0
def left():
    vertex_type.append(0)
    gopu.setheading(gopu.heading()+90)
    gopu.forward(stride)

# right = right() ; make vertex - concave = 1
def right():
    vertex_type.append(1)
    gopu.setheading(gopu.heading()-90)
    gopu.forward(stride)

# back = back() -- doing undo is allowed only once.
def back():
    vertex_type.pop()
    gopu.undo()

# quits the screen and outputs the plots the partitioned polygon
def partition_polygon():
    # closing the screen
    wn.bye()
    # stopped recording the polygon
    gopu.end_poly()
    p = gopu.get_poly()
    p = list(p)
    compute_partition(p)

'''
The movements of the turtle are recorded and the rectilinear polygon thus
obtained is converted into bipartite graph of chords
'''
def compute_partition(p):
    # p now contains the list of coordinates of vertices
    # last one is same as the origin
    # and the origin is always going to be a convex vertex
    p.pop()
    vertex_type[0] = 0

    # x and y contain list of x and y coordinates respectively
```

```
for i,j in p:
    x.append(i)
    y.append(j)

# this is done as there are some very small errors in recording the
# position by the turtle library
for i in range(len(x)):
    x[i] = int(x[i])
    y[i] = int(y[i])

for i in range(len(x)):
    if x[i] % stride != 0:
        x[i] += (stride - x[i] % stride)
    if y[i] % stride != 0:
        y[i] += (stride - y[i] % stride)

# separating concave and collinear vertices
collinear_vertices = [i for i,val in enumerate(vertex_type) if val == -1]
concave_vertices = [i for i,val in enumerate(vertex_type) if val == 1]

# finding the chords inside the polygon
horizontal_chords = []
vertical_chords = []

# middles is used because, there are cases when there is a chord between vertices
# and they intersect with external chords, hence if there is any vertex in between
# two vertices then skip that chord.
for i in range(len(concave_vertices)):
    for j in range(i+1,len(concave_vertices)):
        if concave_vertices[j] != concave_vertices[i] + 1:
            middles = []
            if y[concave_vertices[i]] == y[concave_vertices[j]]:
                for k in range(len(x)):
                    if y[concave_vertices[i]] == y[k] and (x[concave_vertices[i]]
< x[k] and x[concave_vertices[j]] > x[k] \
                                                              or
x[concave_vertices[i]] > x[k] and x[concave_vertices[j]] < x[k]):
                        middles.append(k)
                if len(middles) == 0:

horizontal_chords.append((concave_vertices[i],concave_vertices[j]))
            middles = []
            if x[concave_vertices[i]] == x[concave_vertices[j]]:
                for k in range(len(x)):
                    if x[concave_vertices[i]] == x[k] and (y[concave_vertices[i]]
< y[k] and y[concave_vertices[j]] > y[k] \
                                                                 or
y[concave_vertices[i]] > y[k] and y[concave_vertices[j]] < y[k]):
                        middles.append(k)
                if len(middles) == 0:

vertical_chords.append((concave_vertices[i],concave_vertices[j]))



temp_hori = horizontal_chords[:]
temp_verti = vertical_chords[:]

for i in range(len(collinear_vertices)):
    for j in range(len(concave_vertices)):
        middles = []
        if y[collinear_vertices[i]] == y[concave_vertices[j]]:
            if collinear_vertices[i] < concave_vertices[j]:
                for k in range(len(x)):
                    if y[k] == y[collinear_vertices[i]] and (x[k] <
x[concave_vertices[j]] \
                        and x[k] > x[collinear_vertices[i]] or x[k] >
x[concave_vertices[j]] \
                        and x[k] < x[collinear_vertices[i]]):
                        middles.append(k)
                if collinear_vertices[i]+1 == concave_vertices[j]:
                    middles.append(0)
            else:
                for k in range(len(x)):
                    if y[k] == y[collinear_vertices[i]] and (x[k] >
x[concave_vertices[j]] \
                        and x[k] < x[collinear_vertices[i]] or x[k] <
x[concave_vertices[j]] \
                        and x[k] > x[collinear_vertices[i]]):
                        middles.append(k)
                if collinear_vertices[i] == concave_vertices[j]+1:
                    middles.append(0)
            if len(middles) == 0:

horizontal_chords.append((collinear_vertices[i],concave_vertices[j]))
        middles = []
        if x[collinear_vertices[i]] == x[concave_vertices[j]]:
            if collinear_vertices[i] < concave_vertices[j]:
                for k in range(len(x)):
                    if x[k] == x[collinear_vertices[i]] and (y[k] <
y[concave_vertices[j]] \
                        and y[k] > y[collinear_vertices[i]] or y[k] >
y[concave_vertices[j]] \
                        and y[k] < y[collinear_vertices[i]]):
                        middles.append(k)
                if collinear_vertices[i]+1 == concave_vertices[j]:
                    middles.append(0)
            else:
                for k in range(len(x)):
                    if x[k] == x[collinear_vertices[i]] and (y[k] >
y[concave_vertices[j]] \
                        and y[k] < y[collinear_vertices[i]] or y[k] <
y[concave_vertices[j]] \
                        and y[k] > y[collinear_vertices[i]]):
                        middles.append(k)
                if collinear_vertices[i] == concave_vertices[j]+1:
                    middles.append(0)
            if len(middles) == 0:

vertical_chords.append((collinear_vertices[i],concave_vertices[j]))
# displaying all attributes and important parameters involved
# plotting the initial input given
print ("Initial input rectillinear graph")
fig, ax = plt.subplots()
ax.plot(x+[0], y+[0], color='black')
ax.scatter(x+[0], y+[0], color='black')
for i in range(len(x)):
    ax.annotate(i, (x[i],y[i]))
plt.show()
plt.clf()

print("collinear_vertices = ", collinear_vertices)
print("concave_vertices =", concave_vertices)
print("horizontal_chords = " ,horizontal_chords)
print("vertical_chords = ",vertical_chords)

# drawing the maximum partitioned polygon
print("The maximum partitioned rectillinear polygon")
fig, ax = plt.subplots()
ax.plot(x+[0], y+[0], color='black')
ax.scatter(x+[0], y+[0], color='black')
for i in range(len(x)):
    ax.annotate(i, (x[i],y[i]))
```

```
for i,j in horizontal_chords:
    ax.plot([x[i],x[j]],[y[i],y[j]],color='black')
for i,j in vertical_chords:
    ax.plot([x[i],x[j]],[y[i],y[j]],color='black')
plt.show()
plt.clf()
# MAXIMUM PARTITION CODE ENDS ---------------------------------

# MINIMUM PARTITION CODE STARTS -------------------------------
horizontal_chords = temp_hori[:]
vertical_chords = temp_verti[:]

# Creating a bipartite graph from the set of chords
for i,h in enumerate(horizontal_chords):
    y1 = y[h[0]]
    x1 = min(x[h[0]] ,x[h[1]] )
    x2 = max(x[h[0]] ,x[h[1]])
    G.add_node(i, bipartite=1)
    for j,v in enumerate(vertical_chords):
        x3 = x[v[0]]
        y3 = min(y[v[0]],y[v[1]])
        y4 = max(y[v[0]],y[v[1]])
        G.add_node(j + len(horizontal_chords),bipartite=0)
        if x1 <= x3 and x3 <=x2 and y3 <= y1 and y1 <= y4:
            G.add_edge(i, j + len(horizontal_chords))

if len(horizontal_chords) == 0:
    for j,v in enumerate(vertical_chords):
        x3 = x[v[0]]
        y3 = min(y[v[0]],y[v[1]])
        y4 = max(y[v[0]],y[v[1]])
        G.add_node(j,bipartite=0)

# finding the maximum matching of the bipartite graph, G.
M = nx.Graph()
maximum_matching = nx.bipartite.maximum_matching(G)
maximum_matching_list = []
for i,j in maximum_matching.items():
    maximum_matching_list += [(i,j)]
M.add_edges_from(maximum_matching_list)
maximum_matching = M.edges()
# breaking up into two sets
H, V = bipartite.sets(G)
free_vertices = []
```

```
for u in H:
    temp = []
    for v in V:
        if (u,v) in maximum_matching or (v,u) in maximum_matching:
            temp += [v]
    if len(temp) == 0:
        free_vertices += [u]
for u in V:
    temp = []
    for v in H:
        if (u,v) in maximum_matching or (v,u) in maximum_matching:
            temp += [v]
    if len(temp) == 0:
        free_vertices += [u]

# finding the maximum independent set
max_independent = []
while len(free_vertices) != 0 or len(maximum_matching) != 0:
    if len(free_vertices) != 0 :
        u = free_vertices.pop()
        max_independent += [u]
    else:
        u, v = maximum_matching.pop()
        G.remove_edge(u,v)
        max_independent += [u]

    for v in G.neighbors(u):
        G.remove_edge(u, v)
        for h in G.nodes():
            if (v,h) in maximum_matching:
                maximum_matching.remove((v,h))
                free_vertices += [h]
            if (h,v) in maximum_matching:
                maximum_matching.remove((h,v))
                free_vertices += [h]


# drawing the partitioned polygon
independent_chords = []
for i in max_independent:
    if (i >= len(horizontal_chords)):
        independent_chords += [vertical_chords[i-len(horizontal_chords)]]
    else:
        independent_chords += [horizontal_chords[i]]
```

```
unmatched_concave_vertices = [i for i in concave_vertices]
for i,j in independent_chords:
    if i in unmatched_concave_vertices:
        unmatched_concave_vertices.remove(i)
    if j in unmatched_concave_vertices:
        unmatched_concave_vertices.remove(j)

nearest_chord = []
for i in unmatched_concave_vertices:
    dist = 0
    nearest_distance = math.inf
    for j in max_independent:
        if j < len(horizontal_chords):
            temp1, temp2 = horizontal_chords[j]
            if abs(y[i] - y[temp1]) < nearest_distance and \
            (x[i] <= x[temp1] and x[i] >= x[temp2] or x[i] >= x[temp1] and x[i]
<= x[temp2]) \
            and abs(temp1 - i) != 1 and abs(temp2 - i) != 1:
                middles = []
                for u in range(len(x)):
                    if x[i] == x[u] and (y[i] < y[u] and y[u] < y[temp1] or
y[temp1] < y[u] and y[u] < y[i]):
                        middles.append(u)
                if len(middles) == 0:
                    nearest_distance = abs(y[i] - y[temp1])
                    dist = y[temp1] - y[i]

    if nearest_distance != math.inf:
        nearest_chord.append((i,dist))
    else:
        for k in collinear_vertices:
            if x[k] == x[i] and abs(y[k] - y[i]) < nearest_distance and abs(k-i)
!= 1:
                middles = []
                for u in range(len(x)):
                    if x[i] == x[u] and (y[i] < y[u] and y[u] < y[k] or y[k] <
y[u] and y[u] < y[i]):
                        middles.append(u)
                if len(middles) == 0:
                    nearest_distance = abs(y[i] - y[k])
                    dist = y[k] - y[i]
        nearest_chord.append((i,dist))

print("The minimum partitioned rectillinear polygon")
```

```
    fig, ax = plt.subplots()
    ax.plot(x+[0], y+[0], color='black')
    ax.scatter(x+[0], y+[0], color='black')
    for i in range(len(x)):
        ax.annotate(i, (x[i],y[i]))
    for i,j in independent_chords:
        ax.plot([x[i],x[j]],[y[i],y[j]],color='black')
    for i,dist in nearest_chord:
        ax.plot([x[i],x[i]],[y[i], y[i]+dist],color='black')
    plt.show()
    # MAXIMUM PARTITION CODE ENDS
# Defining the keyboard keys function
wn.onkey(up, "Up")
wn.onkey(left, "Left")
wn.onkey(right, "Right")
wn.onkey(back, "Down")
wn.onkey(partition_polygon, "Escape")
wn.listen()
wn.mainloop()
```

References

1. Wu, San-Yuan, and Sartaj Sahni. "Fast algorithms to partition simple rectilinear polygons." VLSI Design 1.3 (1994): 193-215.
2. Networkx Library(Accessed on: $23^{rd} March, 2017$)
3. Python(Accessed on: $23^{rd} March, 2017$)
4. Wikipedia (Accessed on: $23^{rd} February, 2017$)
5. Author's Github Repository