SW Assignment-1: Gate Packing

Given:

- a set of rectangular logic gates g1, g2...gn
- width and height of each gate gi

Write a program to assign locations to all gates in a plane so that:

- no two gates are overlapping
- the bounding box of the entire circuit (smallest rectangle that encloses all gates) has minimum area.

Design Decisions:

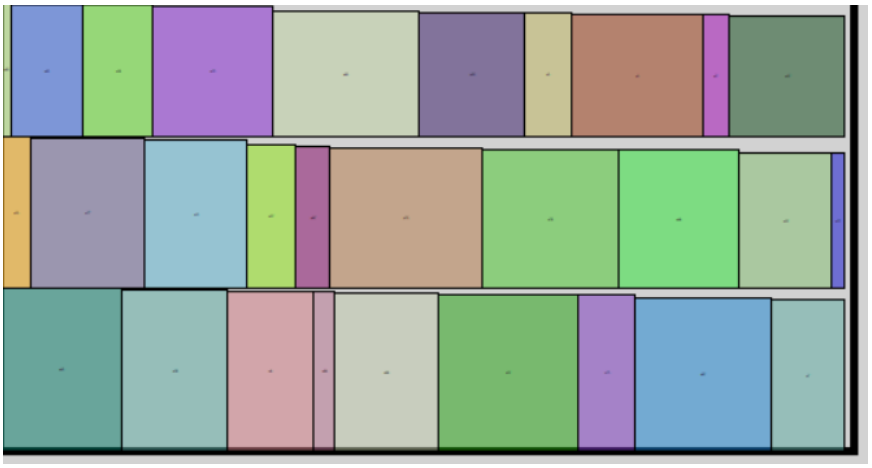
Data structures:

1. Dictionary for storing (width, height) of each gate, i.e., 'g{}' : (width, height). When sorted by decreasing height dictionary is called gh and gw when sorted by decreasing width.
2. Dictionary for storing bottom-left coordinates of each gate, i.e., 'g{}' : (x,y). When sorted by decreasing height, dictionary is called gxyh and gxyw for decreasing width.
3. Class gate is defined to store all the information of specific gate,
   i.e., final x-coordinate = x + width of gate; final y-coordinate = y + height of gate.
4. Class bdd is defined to dynamically create bounding box as gates are added. It computes final coordinates (by taking the maximum of final coordinates of gates) and area of the bounding box. Since the first gate is placed at (0, 0) and all gates are placed either to the right or at the top of the previous gate, final coordinates (x, y) signify width and height of bounding box.
5. Class Level is defined to keep track of the level at which the gate is placed. It stores x0 and y0 (starting of level), h (height of level) and w (width of level). We also added function add_rect() to store gates at each level.
6. We defined function vertical_space that checks if the next gate can be added above the previous gate.
7. We defined function horiz_space that checks if the next gate can be added to the right of the previous gate.

In our implementation, the algorithm sorts the rectangle in decreasing height/width.

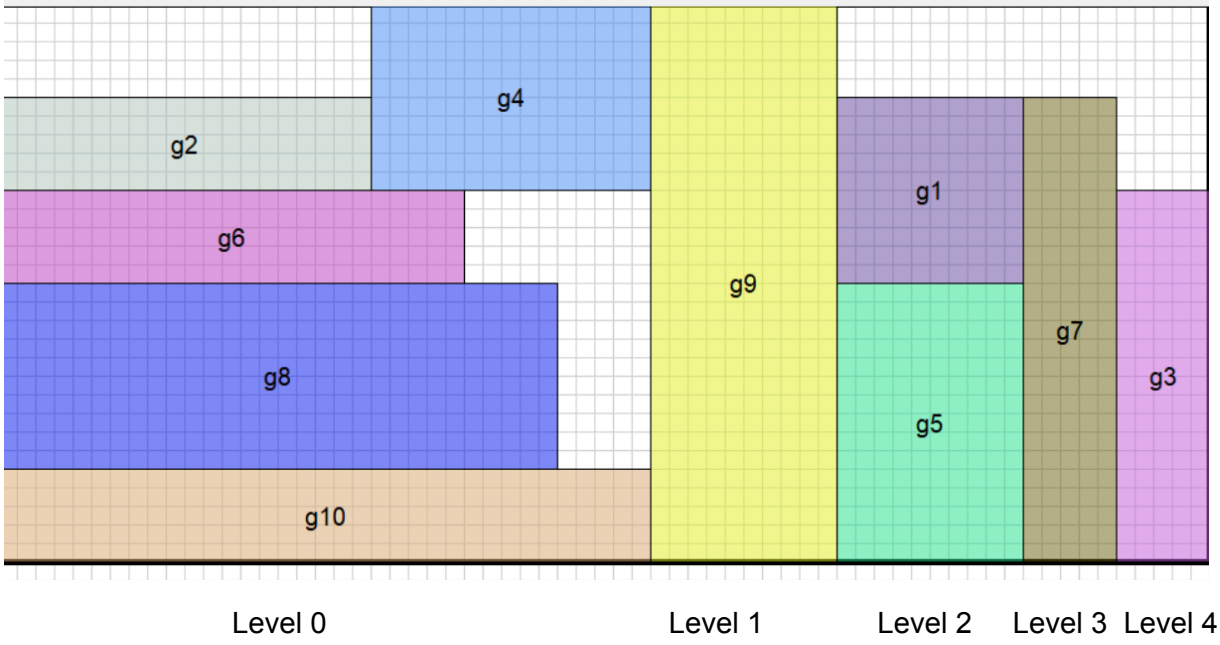
1. Sort by decreasing height (in case heights of gates are equal, sorted by decreasing width)
   The gate with the maximum height is placed at (0, 0). The first level begins at lh= 0. Hence, height of level lh is taken to be the maximum height of gates at that level. The next gate is placed to the right and so on as the coordinates of the next gate are fixed at the final x-coordinate and initial y-coordinate of the previous gate. The vertical_space function checks if a gate can be placed at the top of the previous gate. When no more gates can be fitted at that level, a new level lh+1 is created at the top of the previous level and the process restarts.

Capacity of each level: A new level is created when the final x-coordinate of the gates exceeds the square root of the minimum possible area of bounding box i.e. the sum of all the gate areas. We chose this value after many guesses for optimal level capacity, e.g. max(gate widths). For a large number of gates it tries to 'square' the bounding box area, hence adjusting both height and width of the final bounding box.



2. Sort by decreasing width (in case widths of gates are equal, sorted by decreasing height)
   The gate with the maximum width is placed at (0, 0). The first level begins at lw = 0. Hence, width of level lw is taken to be the maximum width of gates at that level. The next gate is placed above and so on as the coordinates of the next gate are fixed at the final y-coordinate and initial x-coordinate of the previous gate. The horizontal space function checks if a gate can be placed to the right of the previous gate.
   When no more gates can be fitted at that level, a new level lw+1 is created to the right of the previous level and the process restarts.

Capacity of each level: A new level is created when the final y-coordinate of the gates exceeds the square root of the minimum possible area of bounding box i.e. the sum of all the gate areas. We chose this value after many guesses for optimal level capacity,
e.g. max(gate heights). For a large number of gates it tries to 'square' the bounding box area, hence adjusting both height and width of the final bounding box, though there may be more optimal values that can fixed for capacity of each level.



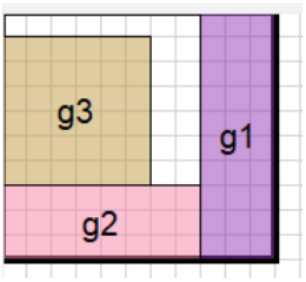
At the end we compared areas of the two bounding boxes formed and chose the one with the least area.

Time Complexity Analysis

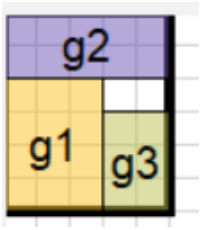
Since we have sorted the gates at first (nlog n), and iterated for all gates in the dictionary (n), the time complexity of the algorithm is O(nlogn).

Test cases

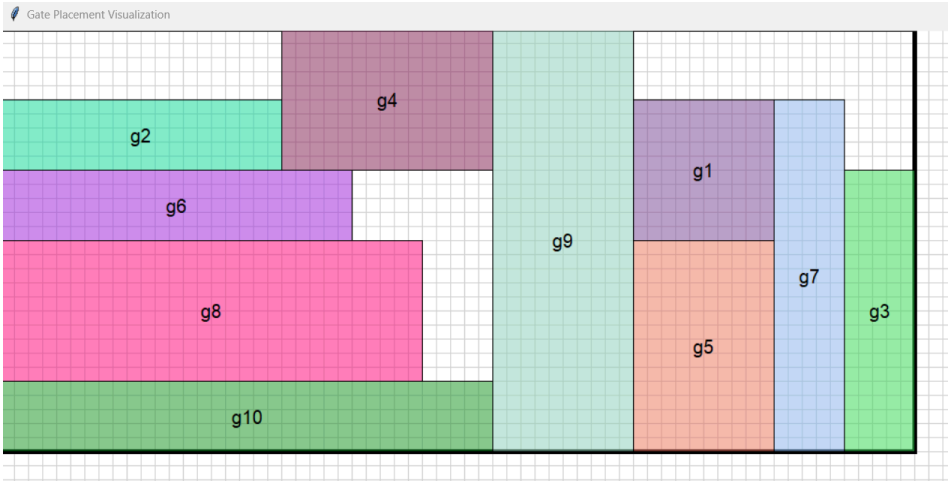
For sample_test_case_1:
bounding_box 11 10
g2 0 0
g3 0 3
g1 8 0
Packing efficiency: 81.818%



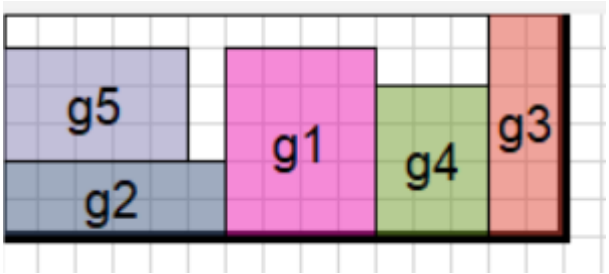
For sample_test_case_2:
bounding_box 5 6
g1 0 0
g3 3 0
g2 0 4
Packing efficiency: 93.333%



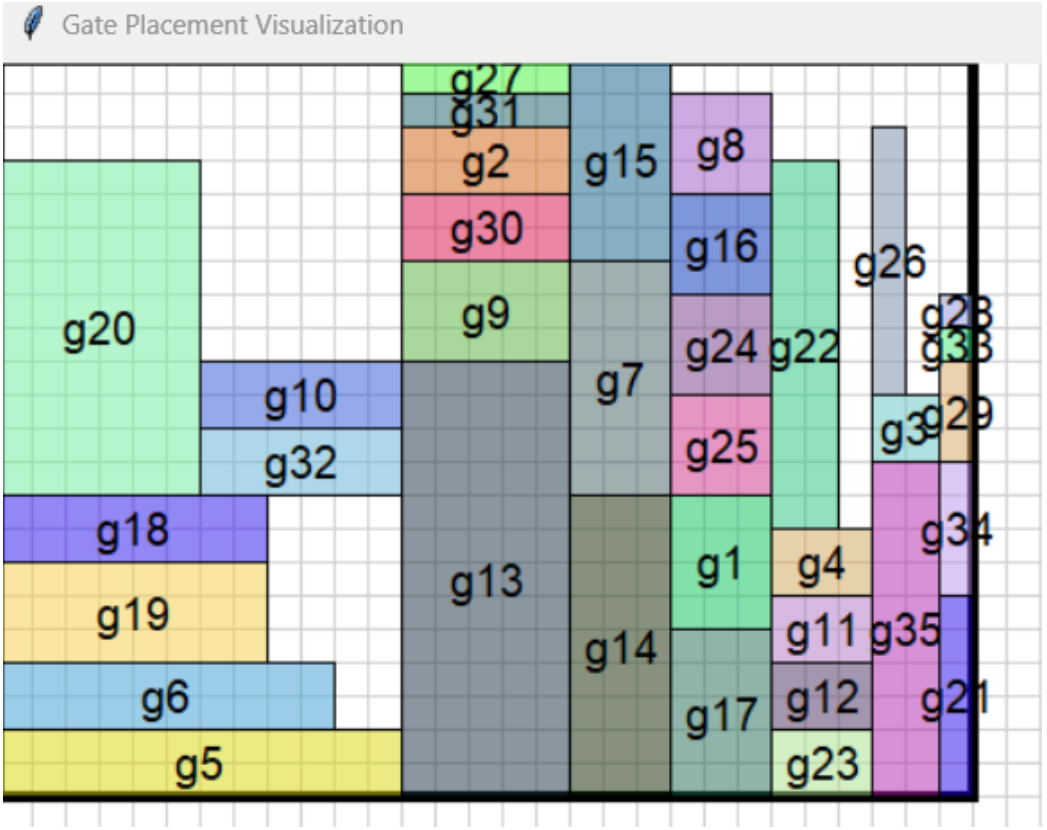
For sample_test_case_3:
bounding_box 65 30
g10 0 0
g8 0 5
g6 0 15
g2 0 20
g4 20 20
g9 35 0
g5 45 0
g1 45 15
g7 55 0
g3 60 0
Packing efficiency: 83.333%



For sample_test_case_4:
bounding_box 15 6
g2 0 0
g5 0 2
g1 6 0
g4 10 0
g3 13 0
Packing efficiency: 78.88888888888889 %



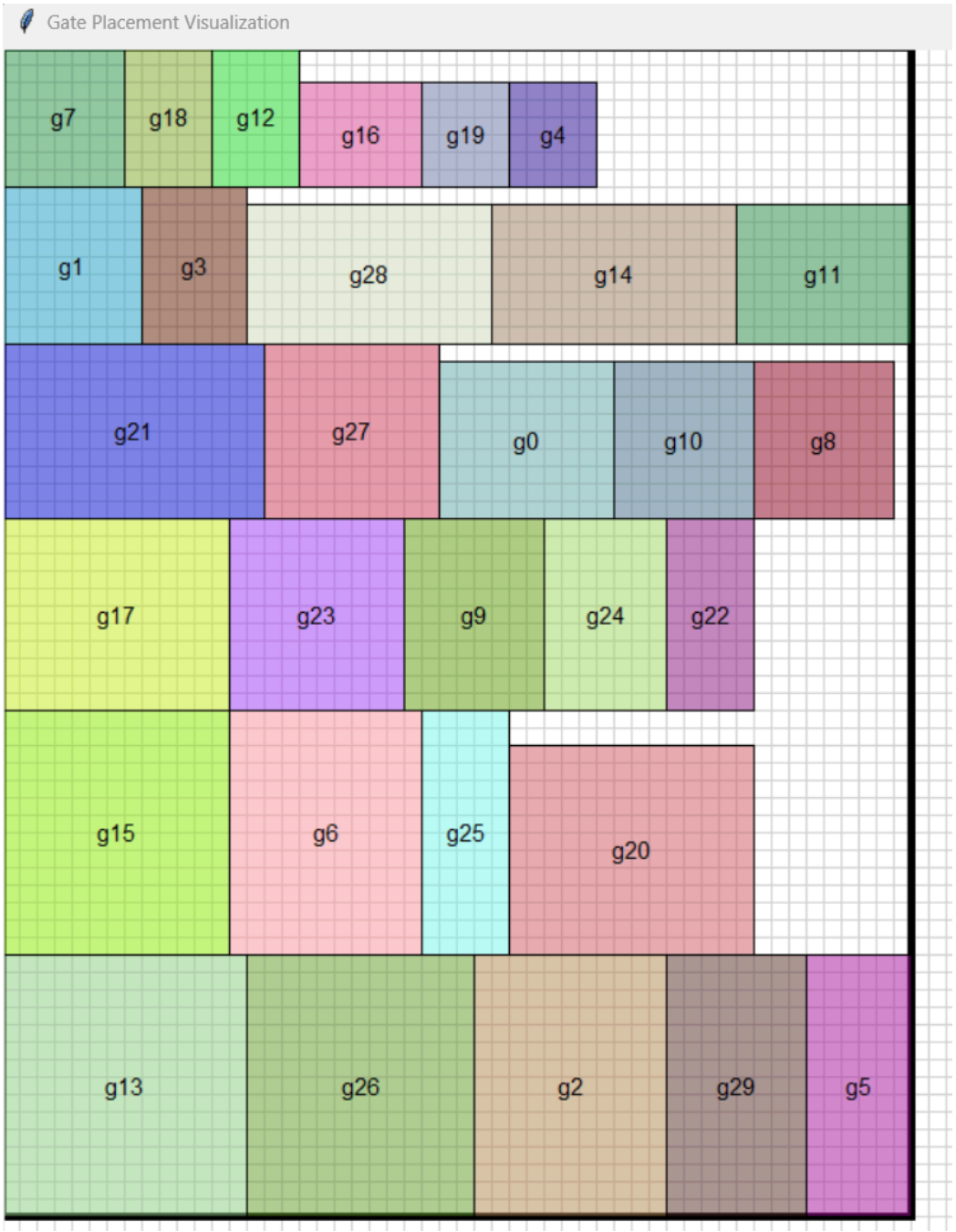
For sample_test_case_5:
bounding_box 29 22
g5 0 0
g6 0 2
g19 0 4
g18 0 7
g20 0 9
g32 6 9
g10 6 11
g13 12 0
g9 12 13
g30 12 16
g2 12 18
g31 12 20
g27 12 21
g14 17 0
g7 17 9
g15 17 16
g17 20 0
g1 20 5
g25 20 9
g24 20 12
g16 20 15
g8 20 18
g23 23 0
g12 23 2
g11 23 4
g4 23 6
g22 23 8
g35 26 0
g3 26 10
g26 26 12
g21 28 0
g34 28 6
g29 28 10
g33 28 13
g28 28 14
Packing efficiency: 78.36990595611285 %



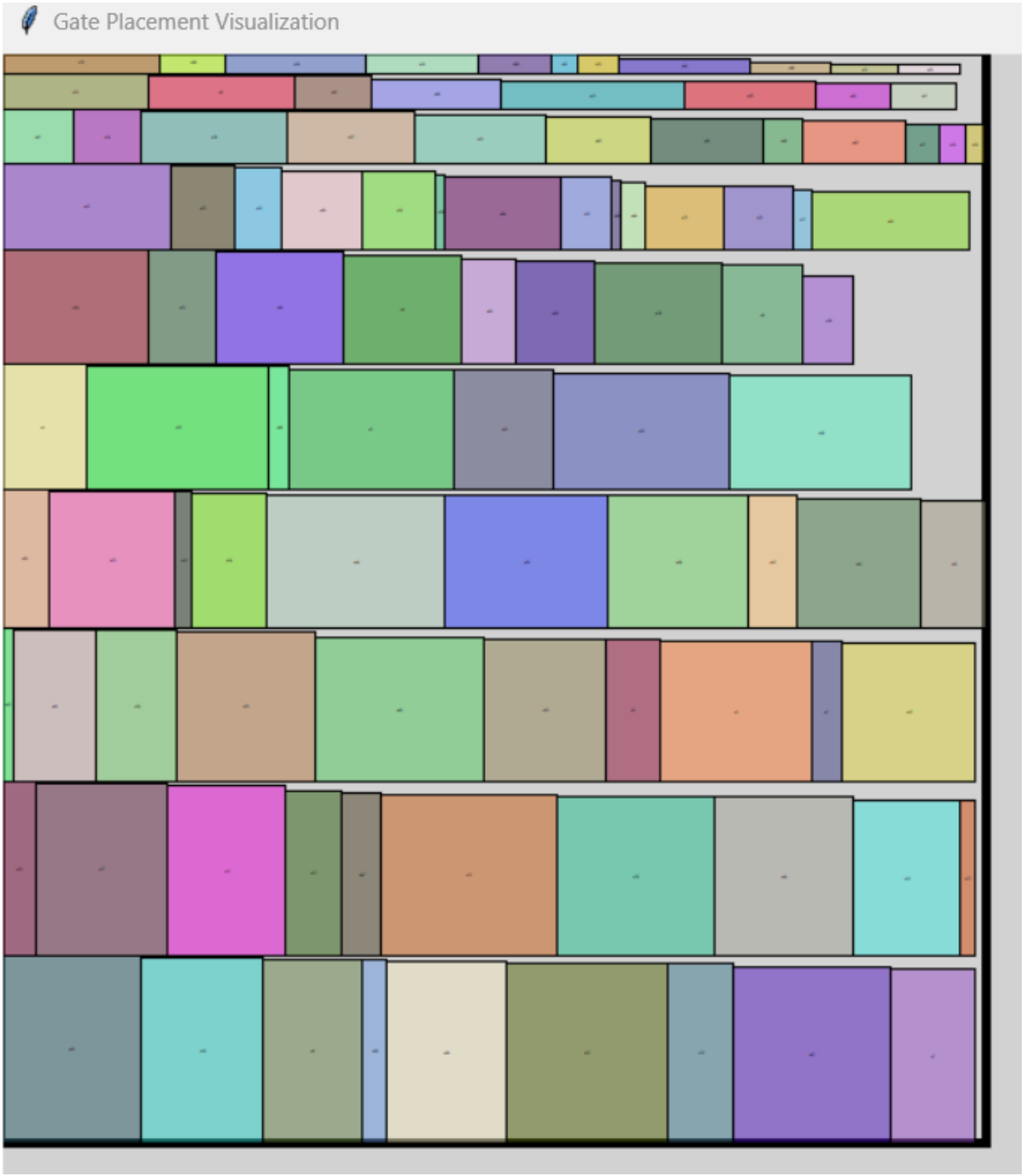
We generated test cases of 30, 40, 100, 300, 500 and 1000 gates of range(1, 100) for both height and width.

E.g.
Input7.txt (30 gates):
g0 10 9
g1 8 9
g2 11 15
g3 6 9
g4 5 6
g5 6 15
g6 11 14
g7 7 8
g8 8 9
g9 8 11
g10 8 9
g11 10 8
g12 5 8
g13 14 15
g14 14 8
g15 13 14
g16 7 6
g17 13 11
g18 5 8
g19 5 6
g20 14 12
g21 15 10
g22 5 11
g23 10 11
g24 7 11
g25 5 14
g26 13 15
g27 10 10
g28 14 8
g29 8 15

bounding_box 52 67

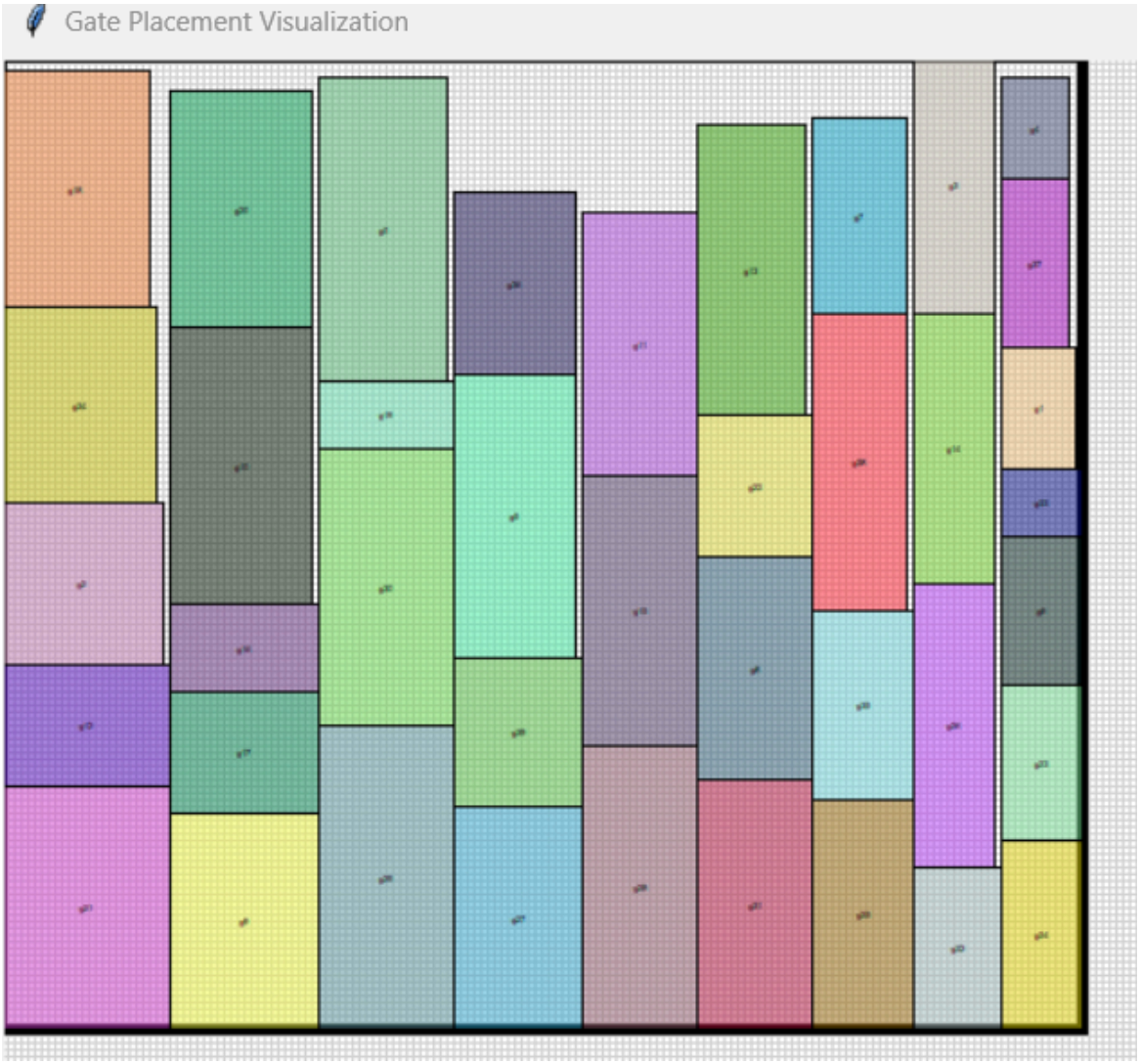


Packing efficiency:
85.50516647531573 %

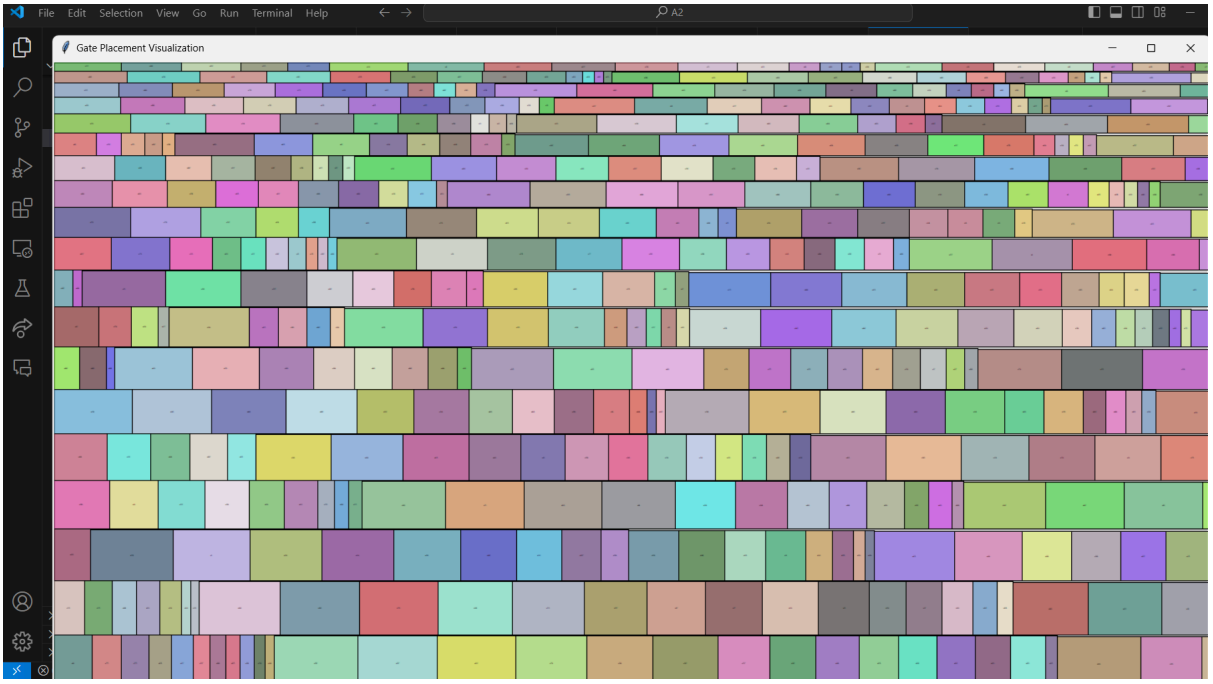


For input6.txt (100 gates)
bounding_box 526 583
Packing efficiency: 90.33027020328835 %

input8.txt (40 gates)
bounding_box 160 144
g21 0 0
g12 0 36
g2 0 54
g34 0 78
g18 0 107
g9 25 0
g17 25 32
g16 25 50
g10 25 63
g20 25 104
g29 47 0
g30 47 45
g19 47 86
g0 47 96
g27 67 0
g39 67 33
g5 67 55
g36 67 97
g28 86 0
g15 86 42
g11 86 82
g31 103 0
g8 103 37
g22 103 70
g13 103 91
g25 120 0
g35 120 34
g38 120 62
g7 120 106
g32 135 0
g26 135 24
g14 135 66
g3 135 106
g24 148 0
g23 148 28
g6 148 51
g33 148 73
g1 148 83
g37 148 101
g4 148 126



Packing efficiency: 91.1328125 %



input9.txt (1000 gates, beyond grid size of canvas)
bounding_box 1776 1867
Packing efficiency: 95.63099856685824 %

For generated test cases we created a random text file generator randomtxt.py

```
import random
with open('input12.txt', 'w') as f:
    for i in range(300):
                f.write('g'+str(i)+'   '+str(random.randint(1,   50))+'
'+str(random.randint(10, 100))+'\n')
```

Citations:
https://www.csc.liv.ac.uk/~epa/surveyhtml.html#bib.9