COL215 SW Assignment 3 - Delay Optimised Gate Packing

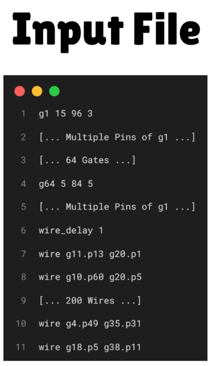
Submission By :

Yash Rawat 2023CS50334

Priyanshi Gupta 2023CS10106

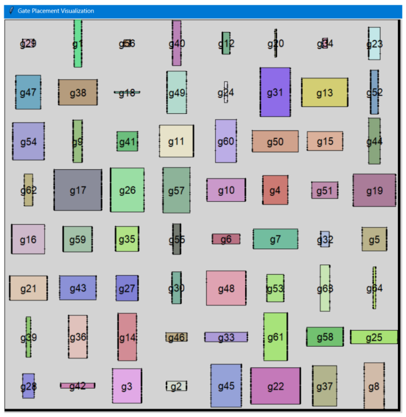
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

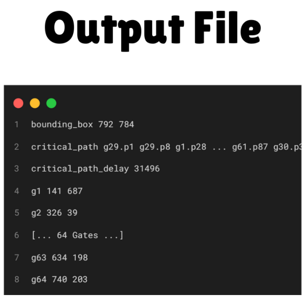
October 19, 2024



Input File

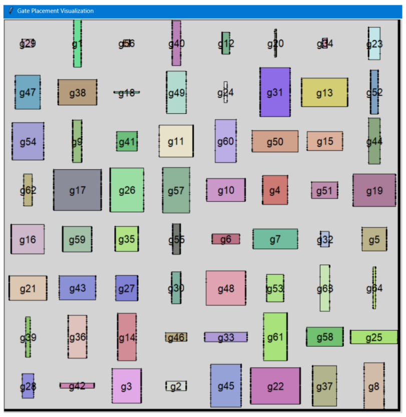
```
g1 15 96 3
[... Multiple Pins of g1 ...]
[... 64 Gates ...]
g64 5 84 5
[... Multiple Pins of g1 ...]
wire_delay 1
wire g11.p13 g20.p1
wire g10.p60 g20.p5
[... 200 Wires ...]
wire g4.p49 g35.p31
wire g18.p5 g38.p11
```





Output File

```
bounding_box 792 784
critical_path g29.p1 g29.p8 g1.p28 ... g61.p87 g30.p3
critical_path_delay 31496
g1 141 687
g2 326 39
[... 64 Gates ...]
g63 634 198
g64 740 203
```



1 Modelling Gate Packing & Circuit Delays

1.1 Understanding the Problem Statement

This problem statement builds upon the last two assignments by introducing gate and wire delays as follows :

1. The gates are a set of n rectangles (provided as input for each test case): $\{g_1, g_2, ..., g_n\}$ each represented by a pair of integers: $g_i = (w_i, h_i, D_i)$, where w_i, h_i,D_i are the width, height and the delay of the i^{th} gate.
2. The input and output pin locations (x and y coordinates relative to bottom left origin) on the boundary of each gate represented by its gate and PIN: $\{g_i.p_1, g_i.p_2, ..., g_i.p_m\}$ (where gate $\{g_i\}$ has m pins).
3. The wires (pin-level connections), represented by $\{(g_i.p_x, g_j.p_y)\}$ where x^{th} pin of i^{th} gate is connected to y^{th} pin of j^{th} gate. Apart from this, `wire_delay` D_{wire} has been provided as delay per unit length wire of all the wires.

A given set of gates is said to be "correctly assigned" if no two gates have overlapping areas. We aim to pack the gates and establish pin-level connections to minimise the delay of the critical path. We assume that gates cannot be reoriented and all wiring is horizontal and vertical.

1.2 Critical Path

We have considered the following

1. All input pins are located on the left side of the gate and output pins are located on the right boundary of the gate.
2. An input can take input from another gate or through external source.Here, we refer to an input pin of a gate that is not connected to any output pin of any other gate as primary input.
3. An output could be transmitted to another gate or can be output externally. Primary output refers to an output pin from a gate that is not connected to the input pin to some other gate (it is an external output).
4. The path delay is given by the sum of the gate delays and wire delays (the wire delay depends on the length of wire).
5. The critical path delay for a circuit is defined as the longest path delay from any primary input to primary output of the circuit. Our aim is minimize the maximum critical path delay.

1.3 Mathematical Formulation

Consider a structure consisting of gates and wires with length L_{w_m}, each connecting a set of pins, gate delay as D_{g_i} and wire delay as D_{wire}. A path P is a sequence of gates and wires from the input pin to the output pin.
For a path P, delay is calculated as

$$T_P = \textstyle\sum_{(g_i,w_m)\in P}(D_{g_i} + D_{wire}.L_{w_m})$$

where, g_i and w_m are the gates and wires in Path P. Further, critical path delay is represented as:

$$T_{cp} = \underset{p\in P}{max} T_P$$

Finally, the objective function is

$$T_{mincp} = minT_{cp} = min(\underset{allpaths}{max} \textstyle\sum_{(g_i,w_m)\in P}(D_{g_i} + D_{wire}L_{w_m})$$

For estimating wire length Semi-Perimeter method is used as instructed.

1.4 Constraints on the Problem

The input constraints are as follows:

1. Corners of gate have integral coordinates
2. Pins will have integral coordinates relative to the corresponding gate
3. $0 <$ Number of gates ≤ 1000
4. $0 <$ Width of gate ≤ 100
5. $0 <$ Height of gate ≤ 100
6. $0 <$ Number of pins on one side of a gate $\leq$ Height of Gate
7. $0 <$ Total Number of pins ≤ 40000
8. There is atleast 1 wire connecting a gate

2 Project Structure

The project directory consistes of the following files

1. **main.py:** This files contains the Input parser and the Output Parser. It calls all the required functions and gives us the final output.
2. **oops.py:** This file contains all the classes we have implemented in the project. It consists of our implementation of the Gate_Data, Gate_Env, Pin, Heap and dp_state objects.

3. **utils.py:** This file contains all utilities including code for test case generation.

4. **input.txt and output.txt:** These files contain the input and output data of our code respectively.

3 Algorithm Design and Implementation

There are two parts in our algorithm design. The first part is the generation of gate packing.The second part is the calculation of the critical path and the maximum delay of the generated packing. We use randomized algorithm to generate packing for the gates and finally choose the packing which minimizes the critical path delay.
Before carrying out any packing or calculation, we carry out a check on our test cases for the presence of loops. In case a loop is formed, an error message is displayed and we do not proceed further.

3.1 Gate Packing Generation

1. An upper bound is established on the dimensions of the bounding box by using the number of gates to be packed.

2. Since each gate is always enclosed in its envelope (initially and ensured throughout the implementation, explained here). The envelopes are in a square-shaped grid in a row-major order.

3. Thus for each gate the `set_coord_env`, `set_coord_rel_env` are called and each gate is initially assigned to be in equidistant from both the pair of boundaries, centered to the envelope.

4. After this a call is given to a method of `Gate_Data` class to initialize and calculate the value of wire delays present throughout the wire groups in the circuit, which is critical for our DP implementation of calculating critical path delay.

3.2 Calculation of Critical Path Delay

1. We have implemented **dynamic programming** to identify the critical path, which is the path with the largest delay.

2. To use DP we have created the dp_state class. Every gate has a dp_state attribute which stores the information about the previous gate and pin, its input pin, max gate delay from any primary input till current gate, length of this path etc.

3. The dp_state of an gate gives the path with maximum delay out of all possible paths from a gate with primary input to our current gate. This is calculated by comparing the (max delay of previous gates + semi-perimeter of bounding box of wire group between the previous and current gate) of all previous gates.

4. We implement the bottom-up approach for DP as it is the better choice. We do not exceed the recursion depth with this approach. So, we iterate over all the gates and calculate their DP states. We start with those gates whose all input pins are primary inputs and then iterate over the gates connected to such gates.

5. Following this we iterate over all the gates with primary output pins and find the gate with max delay. We output the maximum/critical delay and the critical path for the packing.

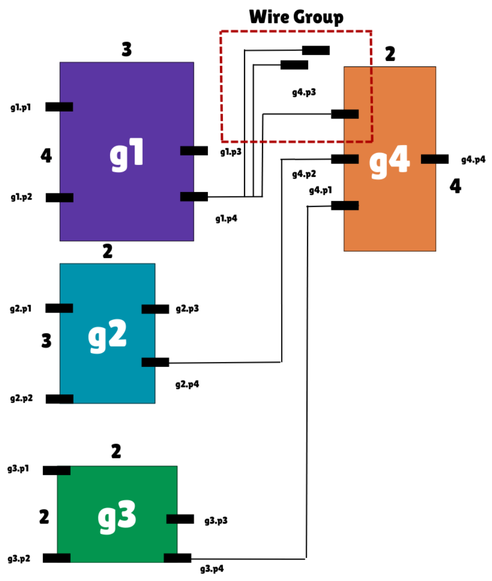


Figure 1: Diagram

6. For the given example, suppose we know the dp_state of gates 1, 2 and 3. To calculate dp_state of gate 4, we iterate over gates 1, 2 and 3 and take sum of their max_delay + semi perimeter of the bounding box of wire group. A wire group is the collection of all wires emanating from an output pin. We set max_delay of gate 4 as sum of the gate_delay D_4 and the maximum resultant value for the gates 1, 2 and 3.

We fix a number of iterations (depending on the size of our test case and time limit) suppose N. We generate N random packings using our Gate Packing Generation algorithm , such that any two packings have sufficient differences. For each packing, we calculate the critical delay. We finally pack the gates according to the packing which outputs minimum critical delay.

4 Correctness of Algorithm

4.1 No overlapping of gates

Since we have implemented gate envelopes, each gate is always packed inside a unique envelope, and envelopes are placed non-intersecting during the initial packing. Thus two gates can never overlap.

4.2 Randomised Packing

We have sampled sufficient number of random arrangement of gates from all possible packings. We have ensured that each packing differs sufficiently from others. Thus there is high possibility that we would also sample a packing which is quite close to the best packing.

4.3 Dynamic Programming for Calculation of Critical Delay

Dynamic programming is an inductive technique where an algorithm relies on putting together smaller parts to create a larger solution. The correctness then follows by induction on problem size. Since, our calculation is correct for a single gate (base case) and correctness for any gate implies correctness of all gates which take input from it, we can say that our algorithm is correct for all gates. Thus, correctness is ensured.

5 Time Complexity Analysis

5.1 Analysis for the Packing

The packing method iterates through the complete set of gates, updating both their envelope coordinates and relative coordinates. The time complexity of this procedure is O(G), where G is the number of gates. This method includes the preliminary wire length computation performed by the wire_cost function.

5.2 Analysis for the init_wire_group method

This method initializes the information of the current packing with the calculation of all the necessary wire group delays which takes O(W) time since initialising a wire group object for a given output pin requires O(w) [w is the number of wires in that wire group] time complexity and the fact that every wire can only belong to Wire Group of one output pin. It also takes O(W) time to initialize the valid wire groups between two gates. After that it takes O(P) [P is the number of total Pins] to establish the primary input and output pin conditions.

5.3 Analysis for the DP functions

1. **find_max_delay_routine**
   This function implements our bottom-up DP approach. For every gate dp_state is updated only once. The rest of the times it is only accessed which is $O(1)$.Thus time complexity of the function is $O(G)$ where G is the total number of gates.

2. **find_max_delay**
   This function iterates over all gates with output pins and checks their max_delay. The comparison and assignment operations are $O(1)$. Thus the function has a time complexity of $O(G)$ where G is the total number of gates.

Since the time complexity of packing and critical path calculation is $O(G)$ where G is the total number of gates, the overall time complexity is also $O(G)$.

5.4 Experimental Analysis for the Code

To test our implementation, we created a sophisticated test case generated, which adheres to the guidelines of the problem statement. Some of the findings are shared below : [Note the experimental data involves time of one packing IO, since the upper bound is $\approx$ 30 seconds, which allows us to run multiple iterations for randomised packing]. Also theoretical time complexity is expected to be $\approx$ O(P+G+W), although constants of implementation matter for practical purposes.Some of these cases have been attached in the submission.

5.4.1 Runtime Variation with wire

As expected, the expected runtime of single packing grows linearly with the number of wires as is visible below, largely influenced due to the calculation of wire-groups logic in the implementation :

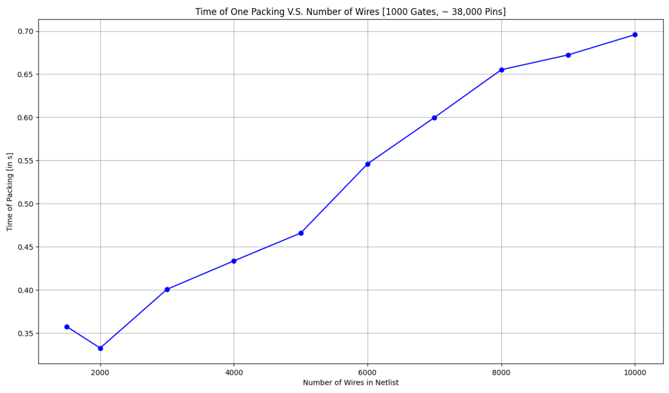


Figure 2: Analysis of Time Complexity as a function of wires

5.4.2 Runtime Variation with pins

As expected the runtime varies linearly with the number of pins, mainly due to primary input and output pin calculations as is visible below

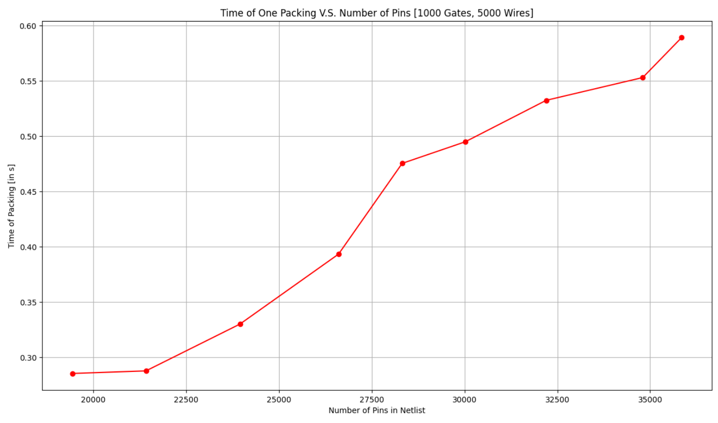


Figure 3: Analysis of Time Complexity as a function of pins

5.4.3 Runtime Variation with gates

As expected the runtime varies linearly with the number of gates. Due to the way test case generation was implemented, the number of pins grows linearly in these test cases, which doesn't affect the fact that we expect a linear variation as shown below.

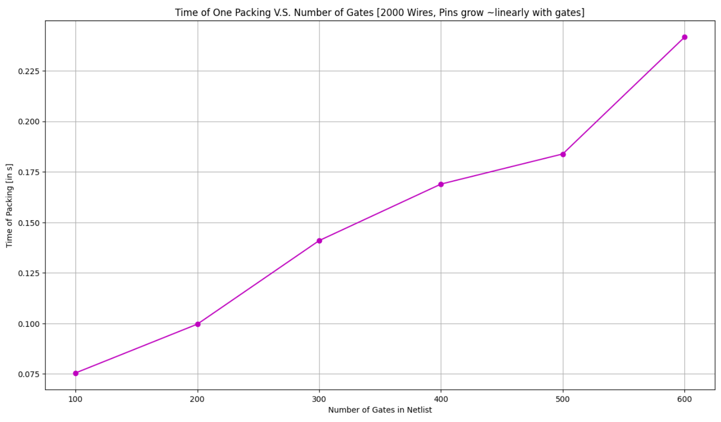


Figure 4: Analysis of Time Complexity as a function of gates

6 Test Cases

6.1 Moodle Test Cases

6.1.1 Test Case 1

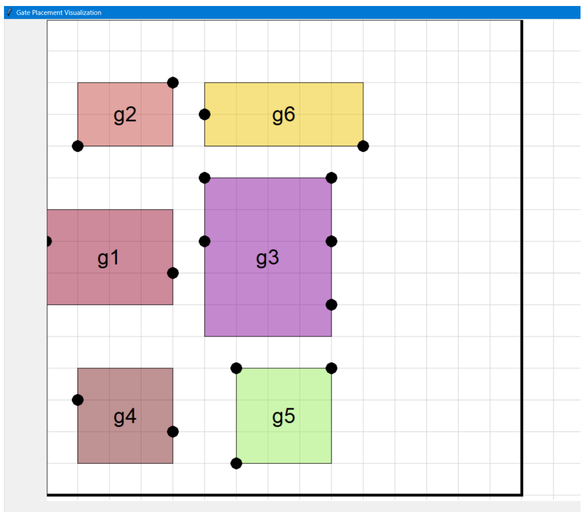
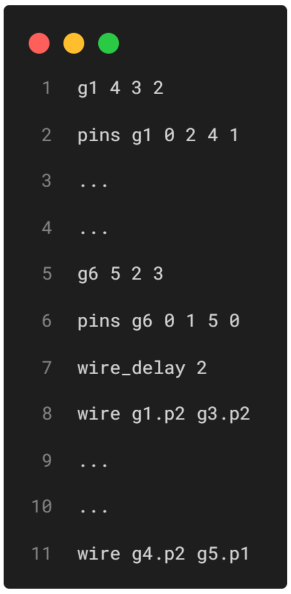
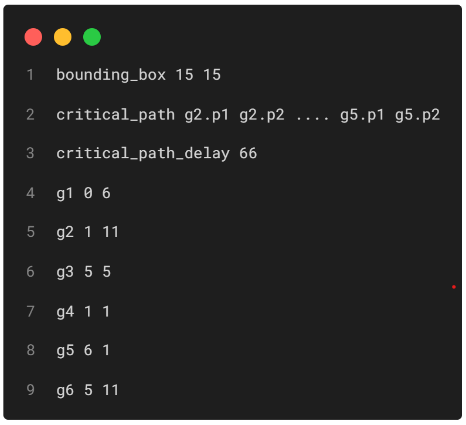


Figure 5: Visualization Test Case 1



```
g1 4 3 2
pins g1 0 2 4 1
...
...
g6 5 2 3
pins g6 0 1 5 0
wire_delay 2
wire g1.p2 g3.p2
...
...
wire g4.p2 g5.p1
```

Figure 6: Input



```
bounding_box 15 15
critical_path g2.p1 g2.p2 .... g5.p1 g5.p2
critical_path_delay 66
g1 0 6
g2 1 11
g3 5 5
g4 1 1
g5 6 1
g6 5 11
```

Figure 7: Output

6.1.2 Test Case 2

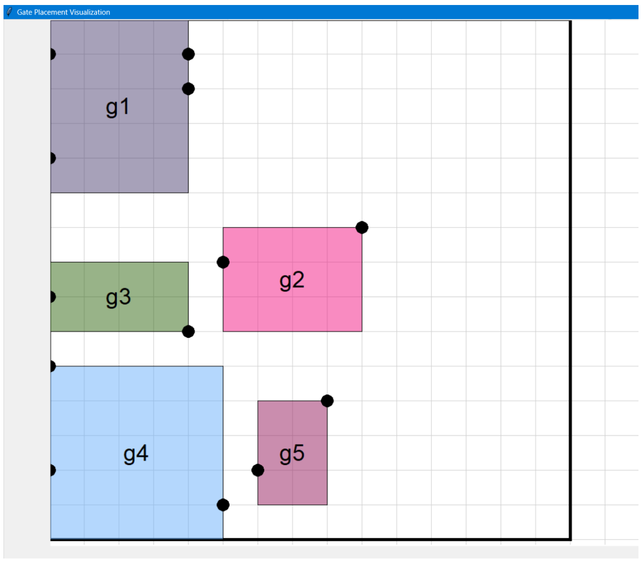
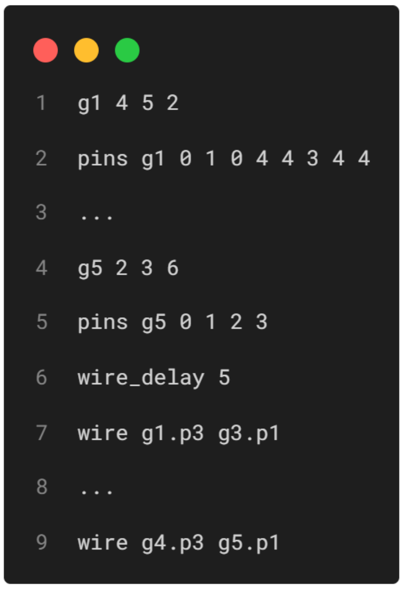
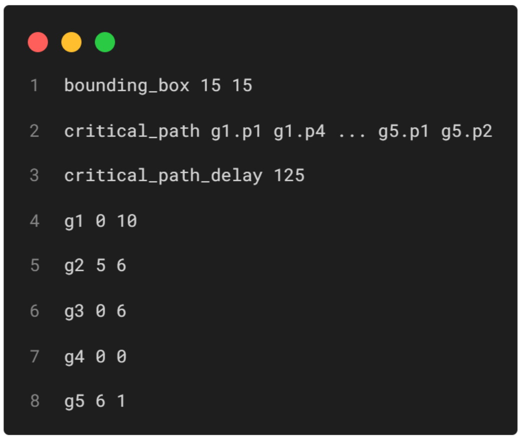


Figure 8: Visualization Test Case 2



```
g1 4 5 2
pins g1 0 1 0 4 4 3 4 4
...
g5 2 3 6
pins g5 0 1 2 3
wire_delay 5
wire g1.p3 g3.p1
...
wire g4.p3 g5.p1
```

Figure 9: Input



```
bounding_box 15 15
critical_path g1.p1 g1.p4 ... g5.p1 g5.p2
critical_path_delay 125
g1 0 10
g2 5 6
g3 0 6
g4 0 0
g5 6 1
```

Figure 10: Output

6.2 Self-generated Test Cases

6.2.1 Edge Case: Two Gates

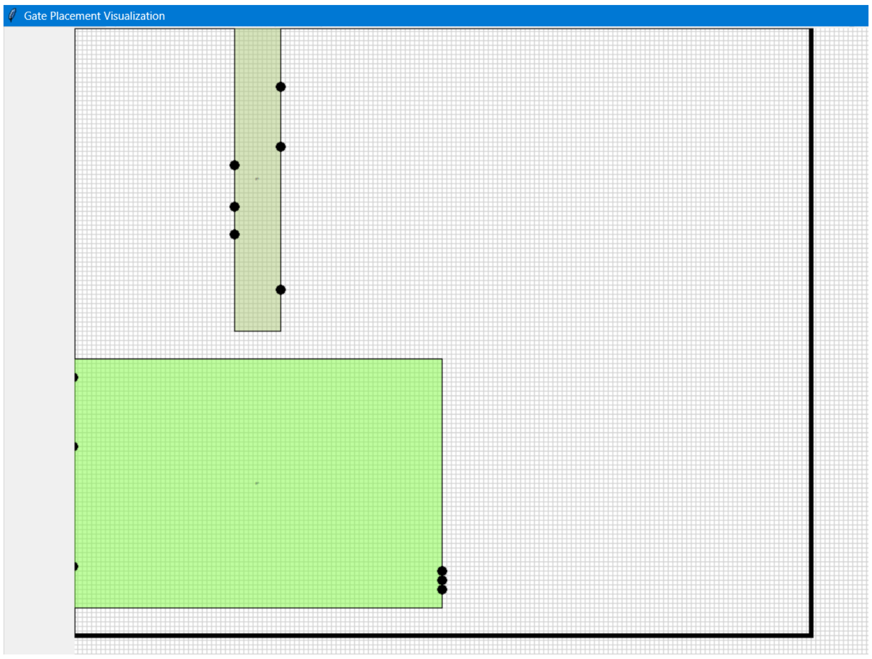
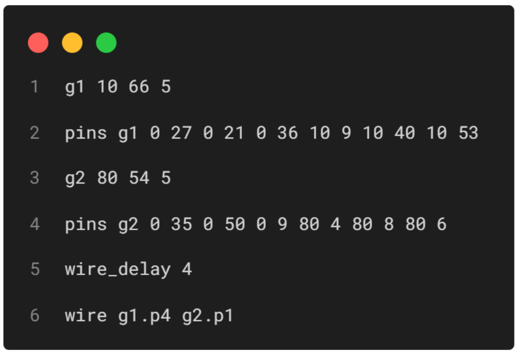
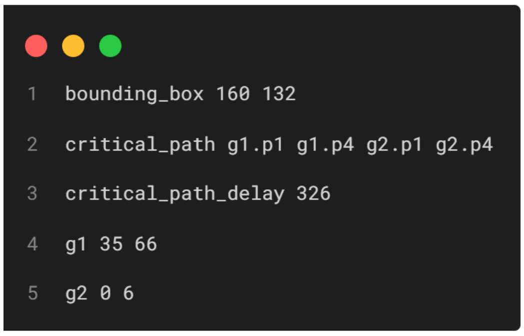


Figure 11: Visualization Edge Case 1



```
g1 10 66 5
pins g1 0 27 0 21 0 36 10 9 10 40 10 53
g2 80 54 5
pins g2 0 35 0 50 0 9 80 4 80 8 80 6
wire_delay 4
wire g1.p4 g2.p1
```

Figure 12: Input



```
bounding_box 160 132
critical_path g1.p1 g1.p4 g2.p1 g2.p4
critical_path_delay 326
g1 35 66
g2 0 6
```

Figure 13: Output

6.2.2 Edge Case: Chain of Gates

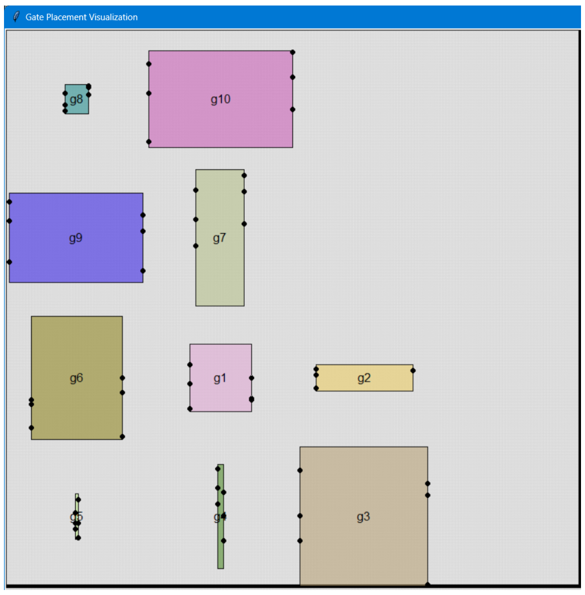
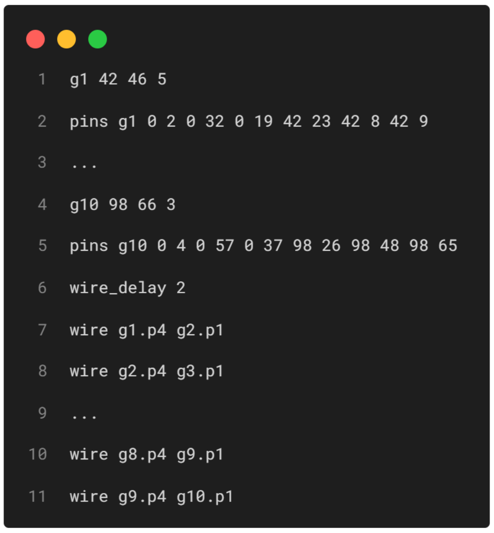
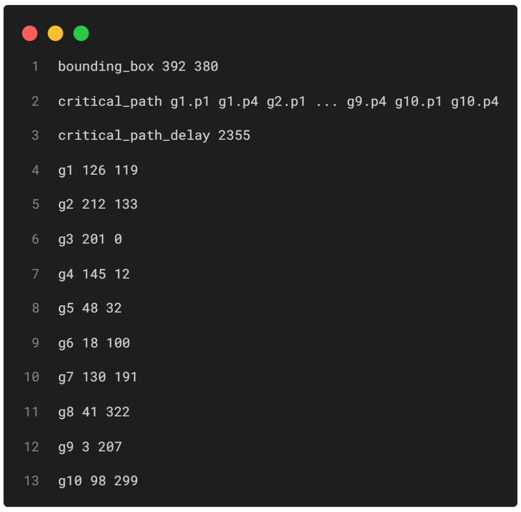


Figure 14: Visualization Edge Case 2



```
g1 42 46 5
pins g1 0 2 0 32 0 19 42 23 42 8 42 9
...
g10 98 66 3
pins g10 0 4 0 57 0 37 98 26 98 48 98 65
wire_delay 2
wire g1.p4 g2.p1
wire g2.p4 g3.p1
...
wire g8.p4 g9.p1
wire g9.p4 g10.p1
```

Figure 15: Input



```
bounding_box 392 380
critical_path g1.p1 g1.p4 g2.p1 ... g9.p4 g10.p1 g10.p4
critical_path_delay 2355
g1 126 119
g2 212 133
g3 201 0
g4 145 12
g5 48 32
g6 18 100
g7 130 191
g8 41 322
g9 3 207
g10 98 299
```

Figure 16: Output

6.2.3 Test Case 1

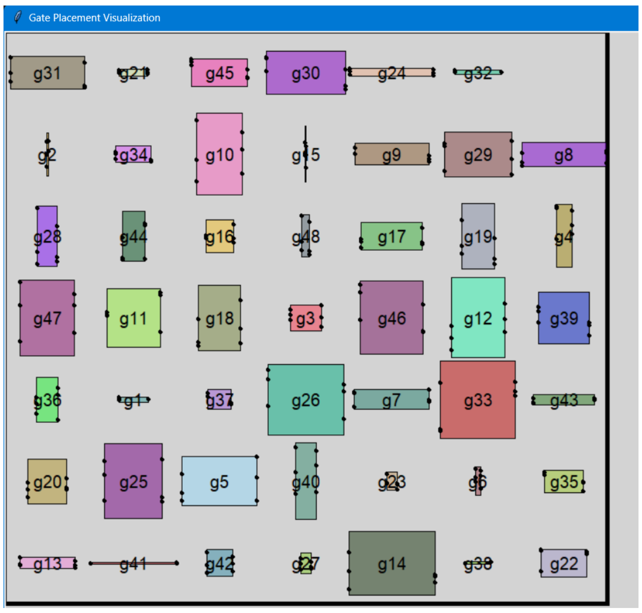
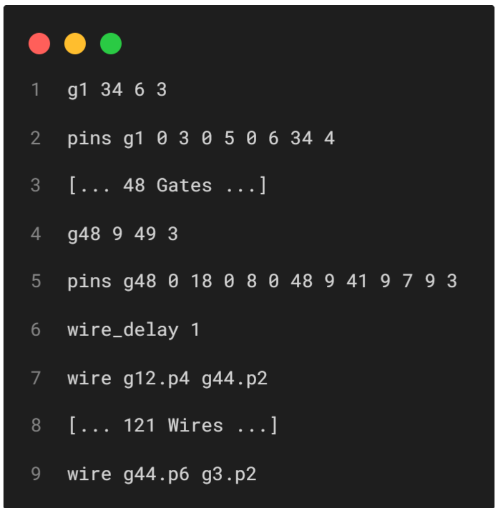
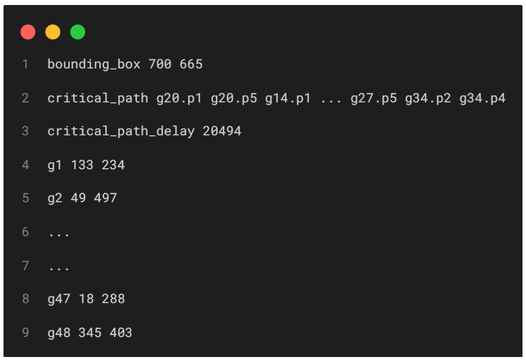


Figure 17: Visualization Test Case 1



```
g1 34 6 3
pins g1 0 3 0 5 0 6 34 4
[... 48 Gates ...]
g48 9 49 3
pins g48 0 18 0 8 0 48 9 41 9 7 9 3
wire_delay 1
wire g12.p4 g44.p2
[... 121 Wires ...]
wire g44.p6 g3.p2
```

Figure 18: Input



```
bounding_box 700 665
critical_path g20.p1 g20.p5 g14.p1 ... g27.p5 g34.p2 g34.p4
critical_path_delay 20494
g1 133 234
g2 49 497
...
...
g47 18 288
g48 345 403
```

Figure 19: Output

6.2.4 Test Case 2

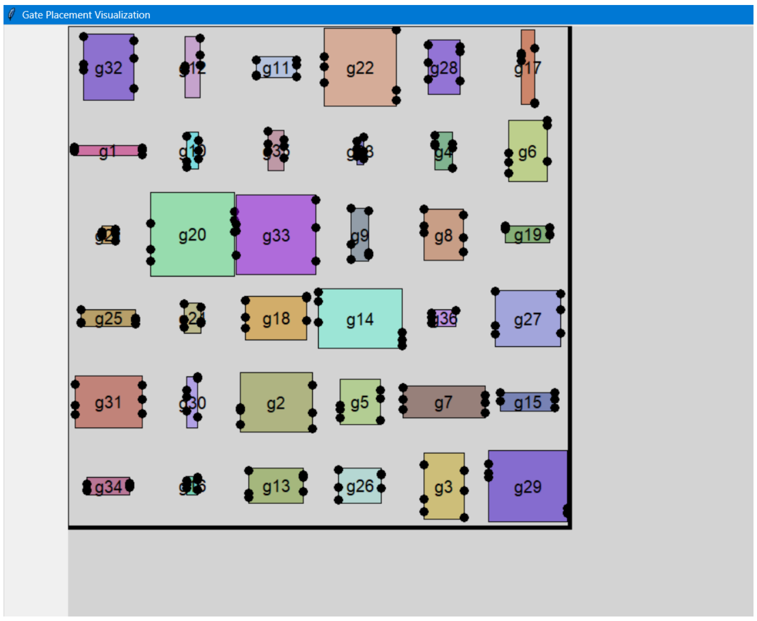
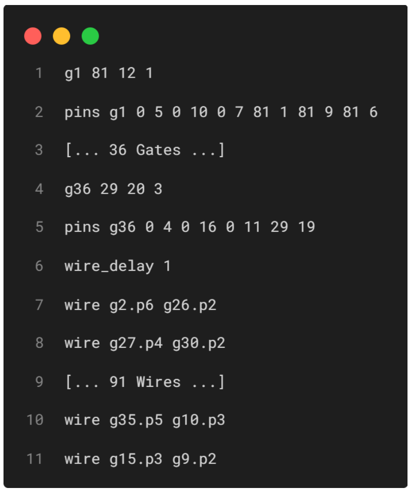
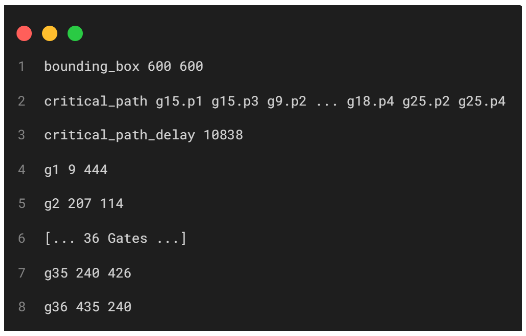


Figure 20: Visualization Test Case 2



```
g1 81 12 1
pins g1 0 5 0 10 0 7 81 1 81 9 81 6
[... 36 Gates ...]
g36 29 20 3
pins g36 0 4 0 16 0 11 29 19
wire_delay 1
wire g2.p6 g26.p2
wire g27.p4 g30.p2
[... 91 Wires ...]
wire g35.p5 g10.p3
wire g15.p3 g9.p2
```

Figure 21: Input



```
bounding_box 600 600
critical_path g15.p1 g15.p3 g9.p2 ... g18.p4 g25.p2 g25.p4
critical_path_delay 10838
g1 9 444
g2 207 114
[... 36 Gates ...]
g35 240 426
g36 435 240
```

Figure 22: Output

6.2.5 Test Case 3

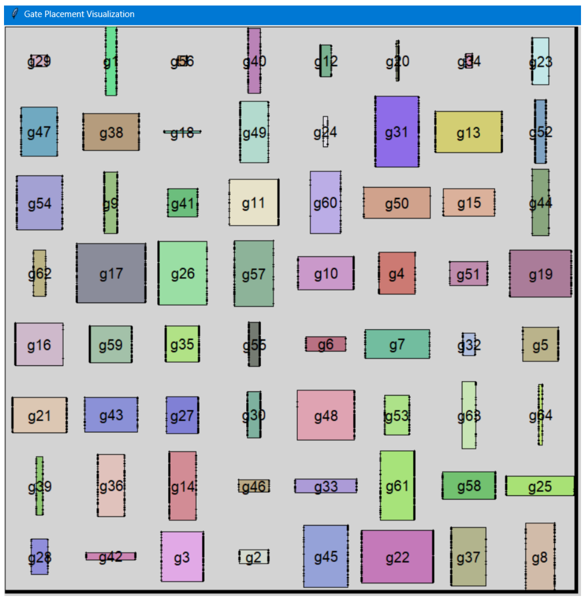
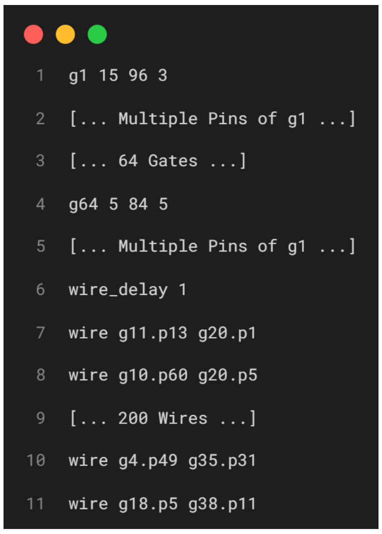
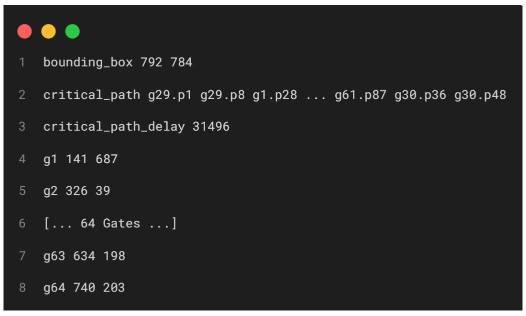


Figure 23: Visualization Test Case 3



```
g1 15 96 3
[... Multiple Pins of g1 ...]
[... 64 Gates ...]
g64 5 84 5
[... Multiple Pins of g1 ...]
wire_delay 1
wire g11.p13 g20.p1
wire g10.p60 g20.p5
[... 200 Wires ...]
wire g4.p49 g35.p31
wire g18.p5 g38.p11
```

Figure 24: Input



```
bounding_box 792 784
critical_path g29.p1 g29.p8 g1.p28 ... g61.p87 g30.p36 g30.p48
critical_path_delay 31496
g1 141 687
g2 326 39
[... 64 Gates ...]
g63 634 198
g64 740 203
```

Figure 25: Output