Software Assignment – 3 Report

Parth Agrawal, Nishchay Patil

20 October, 2024

1. Introduction:

In this assignment, we are asked to minimize the critical path delay which is the maximum delay for any path in a placement from any primary input to any primary output. Gates will have pins and wire connections between them. Primary input pin is the input pin of the gate which is not connected to any other output pin of any other gate. Primary output pin is the output pin of the gate which is not connected to any other input pin of any other gate.

For each testcase mentioned in the report, input and output files are uploaded. And some random testcases are also uploaded along with their output files.

2. Design Decisions:

(Failed attempt: We first started by first storing all the possible paths, and worked a lot on that code and it worked perfectly. But we realised it is not at all efficient for bigger test cases with large input because of space. So we then implemented another approach to calculate critical delay. But for that new code we worked our old code as checker, as this checker could print all the possible paths. It helped a lot in debugging.)

Identify the critical path and critical delay:

For this we first when processing the input file, created a gate class, and stored the information of pins of the gate (number and location with respect to gate) in the class. Then we stored the wire connections and after that we identified all the connections one particular pin can have with any other pin. We made a list of connections for that. Any input pin can be connected to all the output pins of the same gate, and any output pin can be connected to the input pins with which it has wire connections.

After doing this we now have a way of identifying that from which pin we can go to any other pin.

Now we are using dynamic programming to traverse path, so that we don't have to go to all the possible paths which saves time. We assign delay at primary output as 0, and add up the delays till we get to the primary input, but we consider the max delay and pin corresponding to that max delay and store it. This avoids repetition as we will then use this state for further scan. Now we got till the primary inputs and identify at which primary input total delay is maximum. Once we identified it, we just have to identify the critical path, we do this by identifying which connected pin is having max further delay (remember we have stored this).

Gate Placement:

We have used our SW1 code for initial placement which we will further optimise to minimize the critical delay. In the last assignment, we followed the clustering idea, but in this assignment, no clusters will be formed as cleared by the TAs. So we have directly arranged them by area-optimisation logic. Now this heuristic approach ensures that gates are tightly packed and wire length is less (but not least possible, as we will optimize critical delay later).

Modifying Gate placement to optimize critical path delay:

After placing the gates, we apply simulated annealing in which we randomly choose a gate and then move it by a random amount in a direction, and then check if the critical delay was more optimised or not. Here storing all the paths helps as we don't have to find them again for each iteration. Now when we are shifting randomly we have to ensure that they are not overlapped. So we made a checker function for this purpose which checks if the gate we are shifting is not getting overlapped with other gates. Now we accept this shift if it does not overlap and it reduces the cost. A high number of iterations ensures that we get an optimised critical delay.

Complexity Analysis:

First finding the critical path. This depends on the total number of possible connections. So O(number of connections). Note that here connections include all the wires as well as the intra-gate connections. By using dynamic programming, we are avoiding repeated calculations, so we can do it in O(number of connections). Note here, for output pins connected to multiple input pins, we have to calculate the semi-perimeter of the bounding box of those pins, this will add an extra factor in the complexity per output pin.

After that, we place the gates according to the algorithm of SW1, that we do in the complexity of O(m^2logm), where m is the number of gates.

Now after that calculating max delay is simply the number of pins in critical path, so it is O(len(critical path)).

After this simulated annealing in which we randomly move a gate and then check overlap of the gates and then calculate the max delay again.

Check overlap is done in O(n), where n is the number of gates.
Now the simulated annealing runs for say “num” iterations. So total complexity is O(num*max(n, X)). Here X is just denoted to show that it is the complexity of re-calculating the max delay, so first we have to find the critical path, so it takes O(number of connections) + O(len(critical path)). The dominant term will be O(number of connections).

We have set number of iterations in such a way that we get an optimal result in optimal time. If number of gates, pins, wires are less, it is possible to get a very good minimization, so we have increased number of iterations. We go on decreasing the number of iterations as number of gates/pins/wires increase. For eg. we have done 10000 iterations for gates < 50, and 1000 iterations for gates> 500

3. **Testing Strategy**:

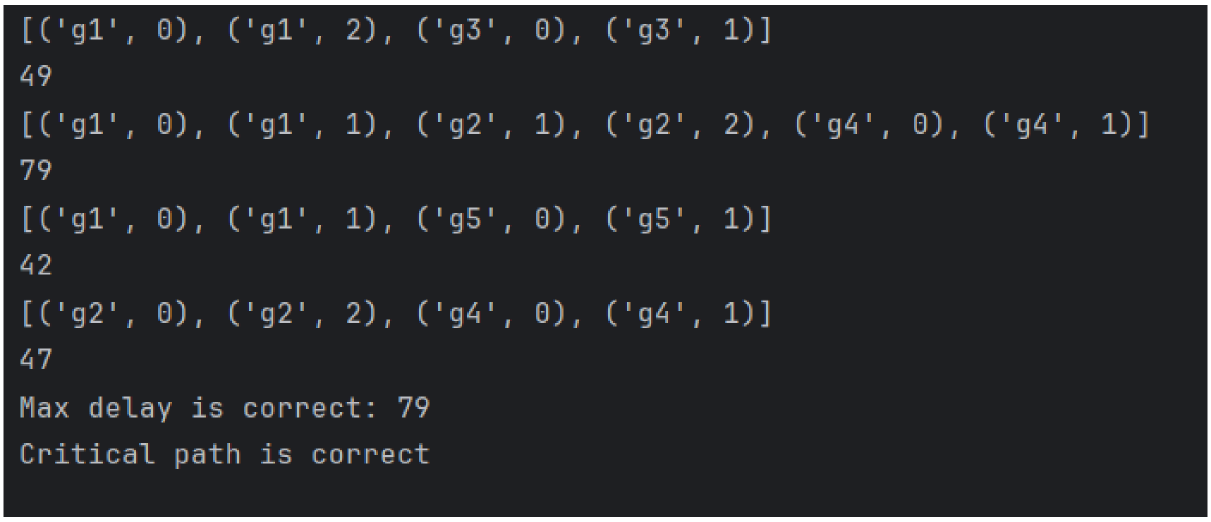
We implemented a checker, which identifies all the possible path and calculates the delay, we have included the checker code also. We first checked the checker manually for moodle testcases, and then we have used it for checking implementation of our final code.

Testcases to check implementation step by step:

3.0 Path printing of checker:

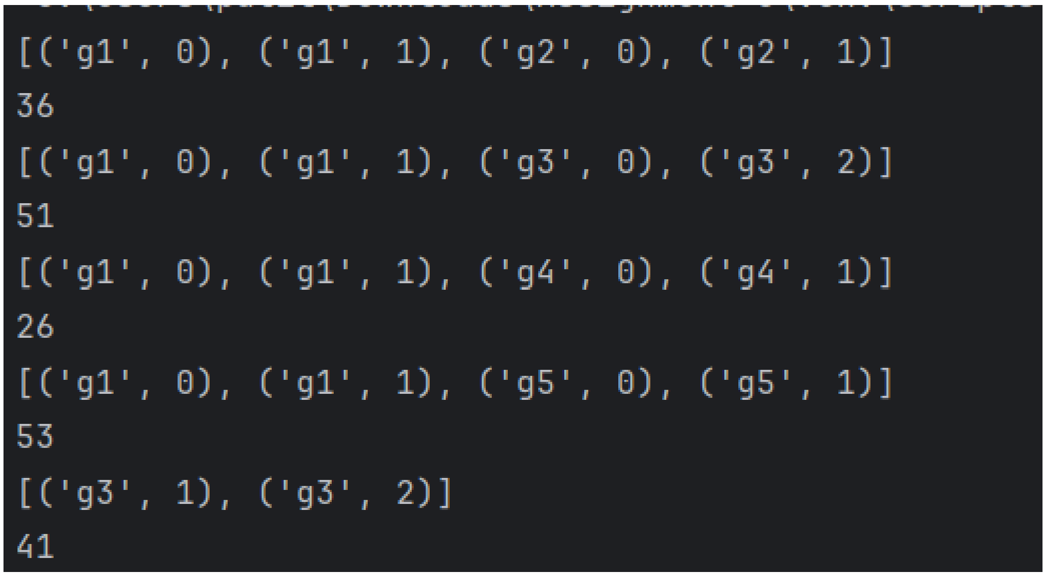
For this we generated testcases of 5 gates and manually checked whether all possible paths are printed and whether they are correct or not.

Checker_check_ip1:



```
[('g1', 0), ('g1', 2), ('g3', 0), ('g3', 1)]
49
[('g1', 0), ('g1', 1), ('g2', 1), ('g2', 2), ('g4', 0), ('g4', 1)]
79
[('g1', 0), ('g1', 1), ('g5', 0), ('g5', 1)]
42
[('g2', 0), ('g2', 2), ('g4', 0), ('g4', 1)]
47
Max delay is correct: 79
Critical path is correct
```

Checker_check_ip2:



```
[('g1', 0), ('g1', 1), ('g2', 0), ('g2', 1)]
36
[('g1', 0), ('g1', 1), ('g3', 0), ('g3', 2)]
51
[('g1', 0), ('g1', 1), ('g4', 0), ('g4', 1)]
26
[('g1', 0), ('g1', 1), ('g5', 0), ('g5', 1)]
53
[('g3', 1), ('g3', 2)]
41
```

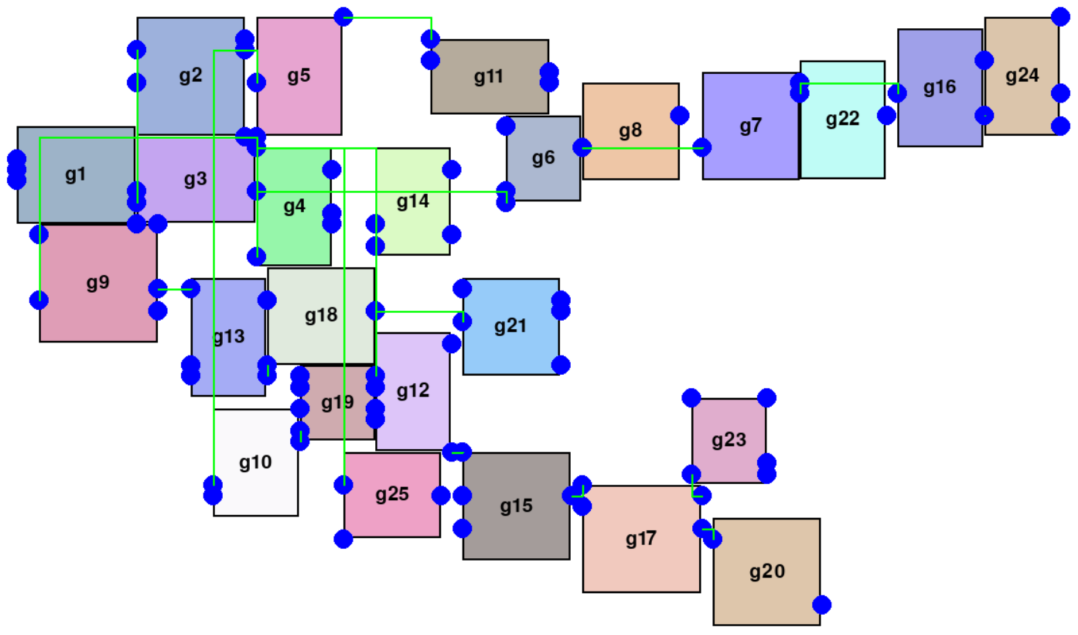
We have included the above testcases.

Once we had ensured that our checker was working correctly, we used the checker to check the implementation of our final code. (Also here we have kept wire delay to be 0 so that correct paths are calculated).

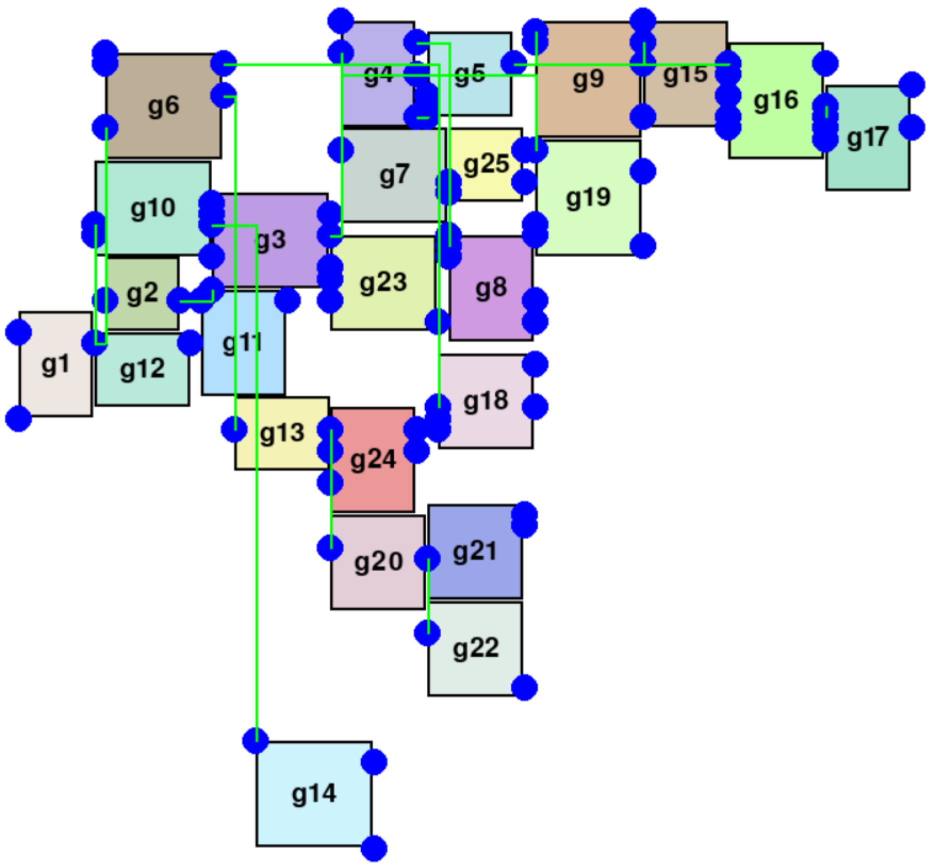
3.1 Checking overlap:

As we are randomly shifting the gates, we have to ensure that gates are not overlapped, for this we have implemented a function in our code, but to check whether this function is satisfied, we made 2 testcases of 25 gates and visualised them to see any overlap occurs or not.

Overlap_checker_ip1



Overlap_checker_ip2

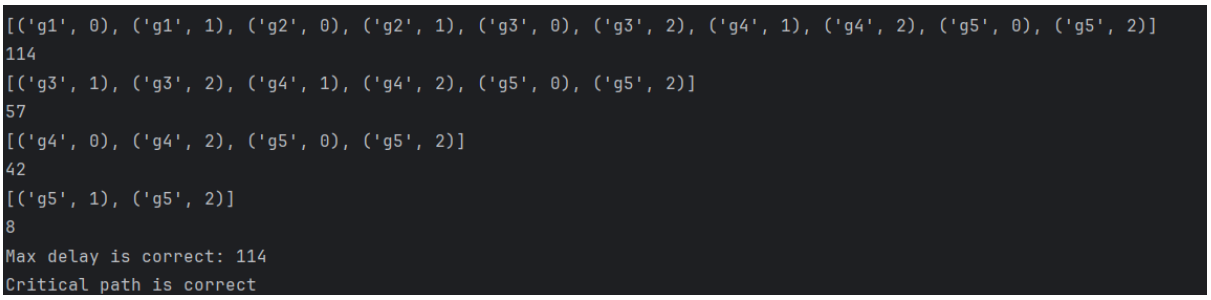


So we ensured that no overlap exists after running simulated annealing repeated times.

3.2 Correct critical path and delay:

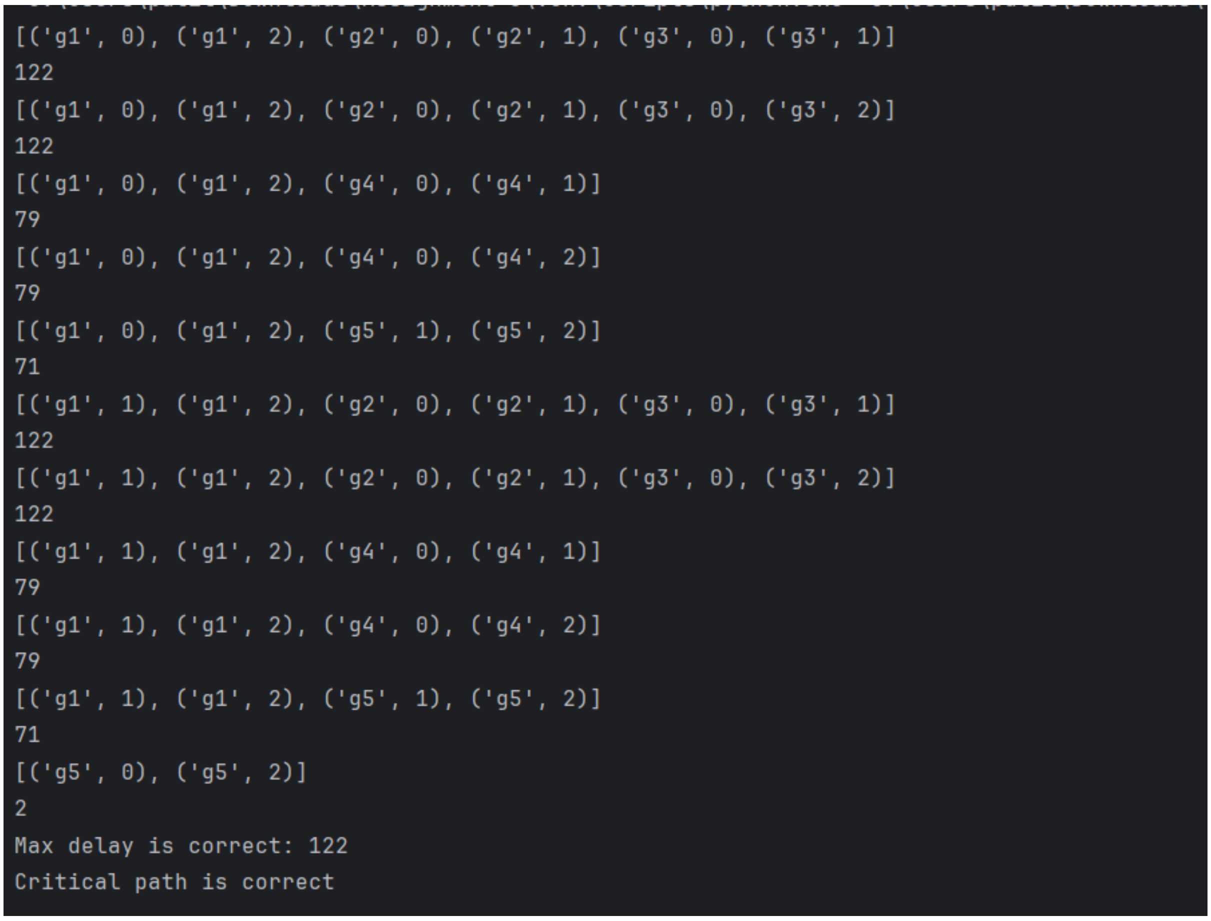
For this we used our checker to first print all the possible paths and then we ran out code on the testcase, then we manually verified that the correct path and delay is calculated. We generated a testcase of 5 gates.

Correct_criticalpath_and_delay_ip1



```
[('g1', 0), ('g1', 1), ('g2', 0), ('g2', 1), ('g3', 0), ('g3', 2), ('g4', 1), ('g4', 2), ('g5', 0), ('g5', 2)]
114
[('g3', 1), ('g3', 2), ('g4', 1), ('g4', 2), ('g5', 0), ('g5', 2)]
57
[('g4', 0), ('g4', 2), ('g5', 0), ('g5', 2)]
42
[('g5', 1), ('g5', 2)]
8
Max delay is correct: 114
Critical path is correct
```

Correct_criticalpath_and_delay_ip2

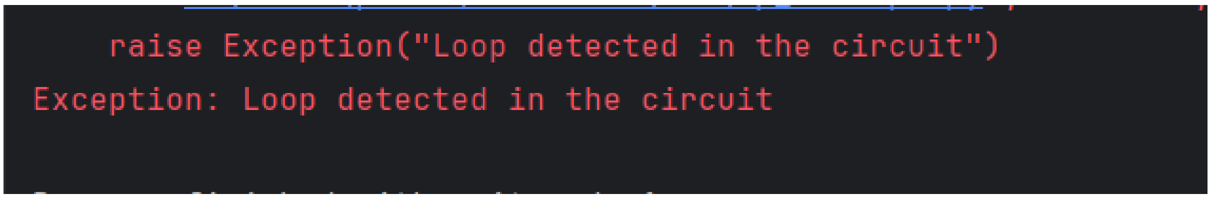


```
[('g1', 0), ('g1', 2), ('g2', 0), ('g2', 1), ('g3', 0), ('g3', 1)]
122
[('g1', 0), ('g1', 2), ('g2', 0), ('g2', 1), ('g3', 0), ('g3', 2)]
122
[('g1', 0), ('g1', 2), ('g4', 0), ('g4', 1)]
79
[('g1', 0), ('g1', 2), ('g4', 0), ('g4', 2)]
79
[('g1', 0), ('g1', 2), ('g5', 1), ('g5', 2)]
71
[('g1', 1), ('g1', 2), ('g2', 0), ('g2', 1), ('g3', 0), ('g3', 1)]
122
[('g1', 1), ('g1', 2), ('g2', 0), ('g2', 1), ('g3', 0), ('g3', 2)]
122
[('g1', 1), ('g1', 2), ('g4', 0), ('g4', 1)]
79
[('g1', 1), ('g1', 2), ('g4', 0), ('g4', 2)]
79
[('g1', 1), ('g1', 2), ('g5', 1), ('g5', 2)]
71
[('g5', 0), ('g5', 2)]
2
Max delay is correct: 122
Critical path is correct
```

We have uploaded these testcases.

3.3 Loop detection and exit:

We created 2 testcases where loop exists, our code successfully detects the loop and exits.

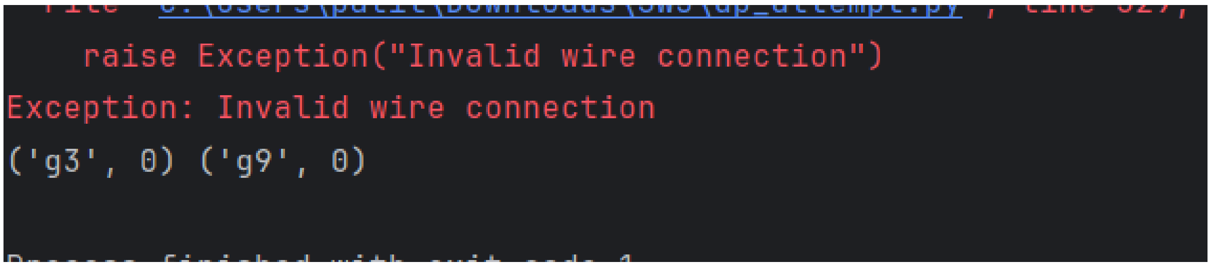


```
    raise Exception("Loop detected in the circuit")
Exception: Loop detected in the circuit
```

We have included 2 testcases which contains loop.

3.4 Flagging invalid input:

To make sure that the input file is correct, we implemented an if statement, which checks that in a wire output pin of a gate is getting connected to input pin of a gate.



```
    raise Exception("Invalid wire connection")
Exception: Invalid wire connection
('g3', 0) ('g9', 0)
```

We have included this testcase as invalid_ip.

3.5 Edge cases and moodle, pdf testcases:

Only 1 gate (edge_case_ip1), no paths from primary input to primary output (edge_case_ip2), in second edge case(pins are only primary input/output) delay comes out to be 0 which suggests that no path exists.

Assignment pdf testcases:

For the first testcase, we got delay as 13
For the second testcase, we got delay as 25

Moodle testcases:

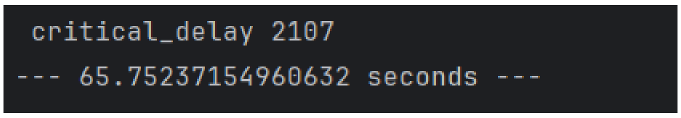
For the first testcase, we got delay as 35.

For the second testcase, we got delay as 30.

We have included the output files for the above testcases.

3.6 Difference on changing the no. of gates:

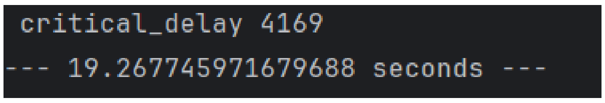
On changing the number of gates and accordingly changing the number of total pins, we observed a linear growth in the critical path delay.
1. For 50 gates, max total pins = 500



```
 critical_delay 2107
--- 65.75237154960632 seconds ---
```

50gates500pins.txt

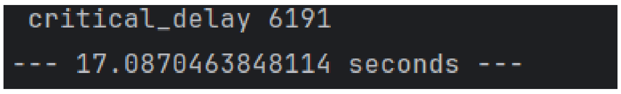
2. For 100 gates, max total pins = 1000



```
 critical_delay 4169
--- 19.267745971679688 seconds ---
```

100gates1000pins.txt

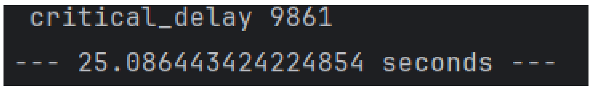
3. For 150 gates, max total pins = 1500



```
 critical_delay 6191
--- 17.0870463848114 seconds ---
```

150gates1500pins.txt

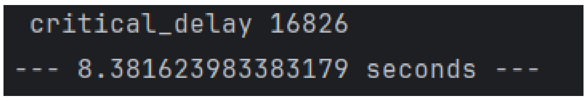
4. For 250 gates, max total pins = 2500



```
 critical_delay 9861
--- 25.086443424224854 seconds ---
```

250gates2500pins.txt

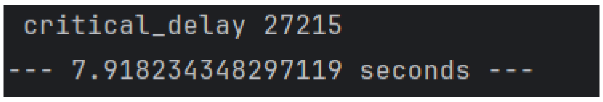
5. For 500 gates, max total pins = 5000



```
 critical_delay 16826
--- 8.381623983383179 seconds ---
```

500gates5000pins.txt

6. For 1000 gates, max total pins = 10000



```
 critical_delay 27215
--- 7.918234348297119 seconds ---
```

1000gates10000pins.txt

It is observable that on changing the no. of gates, critical delay increases linearly, although much change does not come in the time taken.

Apart from that we have included 50 random testcases with their input and output files.

4. Conclusion

In this assignment, we used our strategy of both our previous assignments SW1 and SW2. We placed the gates according to SW1 and used simulated annealing, wire perimeter method according to SW2. We also used dynamic programming in our assignment which helped reduce the time taken in execution in this. We didn’t focus much on placement of gates for critical delay, as it will not much affect the time complexity. Overall, this assignment was really interesting and took a lot of time in thinking and implementing our code.