COL 215 Software Assignment -3

Himanshi Bhandari(2023CS10888)

Mannat(2023CS10138)

19 October 2024

Problem Statement

The primary goal is to arrange the gates in a way that minimizes the overall delay for the given connections and gate delay values . Gates must be placed without overlapping each other and also rotation of gates is not allowed.

Design Decisions

This report outlines the design, implementation, and validation of a greedy placement algorithm for positioning gates in a circuit layout. The algorithm aims to critical path delay while ensuring that gates do not overlap. The solution prioritizes gates with the maximum connections and attempts to find positions that minimize wire length in relation to previously placed gates. Additionally, the critical path delay is calculated based on gate delays and wire delays, and two different placement approaches are considered to optimize the layout further.

Input Processing

- Read the input file, creating Gate objects with their dimensions and pins
- Build the connection network between gates

Gate Sorting

- Sort gates by the number of connections in descending order
- This prioritizes placing more connected gates first, potentially leading to better overall wire length

Initial Placement

- Place the most connected gate at the origin (0, 0)
- This serves as an anchor point for the rest of the placement

Iterative Placement

For each remaining gate:

a) Select up to 4 candidate gates (most connected among the unplaced) b) For each candidate:

- Try placing it in 4 directions around each already placed gate
- Check for overlaps with existing gates
- If no overlap, calculate the resulting wire length
- Keep track of the placement with the minimum wire length .Place the gate at the position that resulted in the minimum wire length

Wire length calculation optimization:

- For n * connections > 250 * 3000: Minimize the wire length of the newly placed gate with previously placed gates
- Otherwise: Minimize the total wire length

- This optimization reduces execution time for larger circuits

Wire Length Calculation

The wire length calculation process involves:

1. Pin-based Iteration:
    - Iterate through all placed gates and their pins
    - Calculate wire length only for unprocessed pins to avoid redundancy
2. Connected Coordinates Collection:
    - For each pin, collect the absolute coordinates of itself and all connected pins
    - Skip already processed pins to avoid redundant calculations
3. Bounding Box Calculation:
    - Determine the bounding box encompassing all connected pins
    - Find minimum and maximum x and y coordinates among all connected points
4. Semiperimeter Addition:
    - Estimate wire length for each pin's connections as the semiperimeter of its bounding box
    - Calculate as (max_x - min_x) + (max_y - min_y)
    - Add this semiperimeter to the total wire length

This approach approximates the minimal wiring needed to connect all pins in the group.

Critical Path Delay Calculation

1. Identifying Primary Inputs and Outputs:
    - Primary inputs are identified as pins receiving external signals, while primary outputs send signals out of the circuit.
2. Iterating Over All Input-Output Pairs:
    - The algorithm checks all possible signal paths from each primary input to each primary output, aiming to explore all routes to find the one with the maximum delay.
3. Finding All Paths:
    - The get_total_paths function generates all possible signal paths between a primary input and output by iterating through gate connections and recursively exploring routes. In this I created a visited list to keep track of gates that I have visited because once a visited gate must not arrive again for a specified path.

4. Calculating Path Delay:

    - Once all paths are identified, the algorithm evaluates each path to determine its total delay:

        - Gate Delays: The inherent delay of each gate in the path is added.

        - Wire Delays: The wire length between connected gates is calculated and multiplied by a wire delay constant.

5. Finding the Maximum Path Delay:
   For each primary input-output pair, the function tracks the path with the maximum delay, determining the critical path.

Output Generation

- Calculate the final bounding box of the entire layout
- Shift all gate positions to ensure the layout starts at (0, 0)
- Write the results to the output file

Time Complexity Analysis

1. read_input function

- Time Complexity: O(n + m), where n is the number of lines in the input file and m is the total number of pins across all gates.
- Explanation:
  - Reading each line: O(n)
  - For each 'pins' line, we iterate over pin coordinates: O(m)
  - The 'wire' lines are processed in constant time each.

2. calculate_wire_length function

- Time Complexity: O(k), where k is the total number of connections, p is the total number of pins across all gates and c is the average number of connections per pin.
- Explanation:
  - Outer loop iterates over all gates and their pins: O(p)
  - For each pin, we process its connections: O(c)
  - Finding min and max coordinates: O(c)
  - The set operations (adding to processed_pins) are O(1) on average.
  - So time complexity is O(p*c) , p*c=k ,so O(k)

3. calculate_wire_length_gates function

- Time Complexity: O(f), where f is the total number of connections of the gate, p is the total number of pins across that gate and c is the average number of connections per pin.
- Explanation:
  - Outer loop iterates over all pins: O(p)
  - For each pin, we process its connections: O(c)
  - Finding min and max coordinates: O(c)
  - The set operations (adding to processed_pins) are O(1) on average.
  - So time complexity is O(p*c) , p*c=f ,so O(f)

4. calculate_bounding_box function

- Time Complexity: O(g), where g is the number of gates.
- Explanation:
  - Single pass over all gates to find min and max coordinates.

5. is_overlapping function

- Time Complexity: O(g), where g is the number of placed gates.
- Explanation:
  - Checks overlap with each placed gate.

6. greedy_placement function

- Time Complexity: O(g*k), where g is the number of gates, k is the total number of connections, p is the average number of pins per gate, and c is the average number of connections per pin.
- Explanation:
  - Sorting gates: O(g log g)
  - Main loop runs n times (for each gate):
    - For each gate, we try 4 positions around up to placed gates: O()
    - For each position, we call is_overlapping: O(g)
    - If not overlapping, we call calculate_wire_length: O(p * c)
  - Total: O(g * (g * (g + p * c))) = O(g^3 + g^2 * p * c)
  - The g^3 term is typically dominated by g^2 * p * c for realistic circuits, so we can simplify to O(g^2 * p * c)
  - g*p*c=total number of connections i.e. k. Therefore, time complexity is O(g*k).

7. get_total_paths Function

- The get_total_paths function explores all possible routes from a starting pin to an ending pin by checking every connection in the circuit.

Time Complexity:

Typical Case: O(b^d)

- b is the average number of connections a pin has.
- d is the number of steps it takes to go from the starting pin to the ending pin.
- In normal cases, the function grows quickly in complexity as more connections are explored at each pin, taking longer with more steps and connections.

Worst Case: O(p!)

- p is the total number of pins in the circuit.
- In very complex circuits with many pins and connections, the function may have to check every possible path, leading to a huge number of paths and much longer execution times.

8. **get_critical_path Function**

- **Time Complexity**: O(g * b^d), where g is the total number of gates and b^d is the time complexity of exploring paths as described above.

Explanation:

- This function calculates the critical path by iterating over all primary input-output combinations.
- For each combination, it calls get_total_paths to explore all possible paths and selects the one with the maximum delay.
- Since get_total_paths has a time complexity of O(b^d), and there are g gates to evaluate, the overall complexity is O(g * b^d).

8. . calculate_path_delay Function

- Time Complexity: O(p * c), where p is the number of pins in the path and c is the average number of connections per pin.

Explanation:

- This function calculates the delay for a given path by iterating through each pin in the path.
- For each pair of consecutive gates, it calculates the wire length between connected pins and adds the wire delay.
- The loop over the pins results in O(p), and for each pin, checking connections takes O(c).

9. calculate_wire_length_pins Function

- **Time Complexity**: O(k), where k is the total number of connections for the gate.
- **Explanation**:
  - The function iterates over all the connections for the gate: O(k).
  - The calculation of wire length between pins is done in constant time for each connection.
  - Therefore, the overall complexity is O(k).

10. main Function

- **Time Complexity**: O(g^2 * p), where g is the number of gates and p is the number of pins per gate.
- **Explanation**:
  - The complexity of the read_input function is O(n + m), where n is the number of lines in the input file and m is the number of pins.
  - The greedy placement is O(g^2 * p), as explained above.
  - Calculating wire length and bounding box dimensions takes O(g) and O(k), respectively.
  - Writing the final output takes O(g).
  - Therefore, the overall complexity is dominated by greedy_placement: O(g^2 * p).

Overall Time Complexity

The overall time complexity of the program is dominated by the get_critical_path and greedy_placement functions, which depend on the number of gates g and connections k.

- Time Complexity: O(g * b^d + g * k)
  - The term O(g * b^d) comes from exploring all possible paths between primary input and output pins in the get_critical_path function.
  - The term O(g * k) is from the greedy_placement function, where k is the total number of connections and g is the number of gates**.**

Implementation Validation

1)Input validation

We ensured that the input file was read and parsed correctly. This included verifying that:

- The number of gates, pins, connections, and the wire delay were accurately extracted.
- All gate dimensions, pin coordinates, and wire connections were properly parsed and stored.

So, in order to ensure this we added print statements for input and checked it by using small inputs with 4-5 connections.

3. Primary Input/Output Identification (get_primary)

This function determines which pins are primary inputs (left-side pins with no incoming connections) and which are primary outputs (right-side pins with no outgoing connections).

Verification:

We have correctly verified it since we are checking the coordinates of the pin if the x part is zero in the connections(i.e. a connected left pin)that results us with a left pin which behaves as input pin and we get all the left pins and finally subtract them to get the primary pins .similarly ,It goes for the right pins. That always states right .To verify this is running correctly, we ran a test case containing loop which raised an exception as expected.

4. **Pathfinding (get_total_paths):**

The main verification for this part is to check for loops and not to let gates be repeated in my path .For that I have made a visited array that keeps record of the gate visited and prevents a looped path to be a part of my paths.

5. Critical Path Calculation (get_critical_path)

The critical path calculation is a key component of the system, and we verified it by ensuring that the path with the maximum delay was correctly identified.

- Verification: we printed the critical path and its corresponding delay to confirm that the correct path was being identified as the critical one. I ensured that both gate delays and wire delays were considered in the path delay calculation.
- Test: we ran minor tests on circuits with multiple paths between inputs and outputs. I manually computed the delay for each path and compared it to the system's output, ensuring that the critical path had the longest delay.

6) Edge case handling

We tested the code against edge cases to ensure robustness:

- Circuits with only one gate and no connections.

Verification Steps:

- We created test cases for each edge case and ensured that the program handled them gracefully, raising errors when needed or producing correct outputs for minimal cases.

Public Test Cases

1)Test Case 1

Input	Output
g1 2 3 5 pins g1 0 1 2 2 g2 3 2 3 pins g2 0 0 3 1 g3 2 2 6 pins g3 0 1 0 2 2 1 wire_delay 4 wire g1.p2 g3.p1	bounding_box 5 5 Critical path found: g1.p1 -> g1.p2 -> g3.p1 -> g3.p3 critical_path_delay 23 g3 3 3 g2 0 3 g1 0 0

wire g2.p2 g3.p2	

2)Test Case 2

Input	Output
g1 2 3 5 pins g1 0 1 2 2 g2 3 2 3 pins g2 0 0 3 1 g3 2 2 6 pins g3 0 1 0 2 2 1 g4 3 1 4 pins g4 0 0 0 1 1 2 wire_delay 3 wire g1.p2 g3.p1 wire g2.p2 g3.p2 wire g2.p2 g4.p1 wire g3.p3 g4.p2	bounding_box 8 5 Critical path found: g1.p1 -> g1.p2 -> g3.p1 -> g3.p3 -> g4.p2 -> g4.p3 critical_path_delay 24 g3 3 3 g2 0 3 g4 5 3 g1 0 0

3)Test Case 3

Input	Output
g1 4 3 2 pins g1 0 2 4 1 g2 3 2 1 pins g2 0 0 3 2 g3 4 5 6 pins g3 0 5 0 3 4 5 4 3 4 1 g4 3 3 9 pins g4 0 2 3 1 g5 3 3 6 pins g5 0 0 3 3 0 3 g6 5 2 3 pins g6 0 1 5 0 wire_delay 2 wire g1.p2 g3.p2 wire g2.p2 g3.p1 wire g3.p5 g6.p1 wire g3.p3 g4.p1 wire g3.p3 g5.p3 wire g4.p2 g5.p1	bounding_box 16 8 Critical path found: g1.p1 -> g1.p2 -> g3.p2 -> g3.p3 -> g5.p3 -> g5.p2 critical_path_delay 46 g3 7 0 g2 4 0 g6 11 0 g1 0 0 g4 4 2 g5 7 5

4)Test Case 4

Input	Output
g1 4 5 2 pins g1 0 1 0 4 4 3 4 4 g2 4 3 3 pins g2 0 2 4 3 g3 4 2 10	bounding_box 13 7 Critical path found: g1.p1 -> g1.p4 -> g2.p1 -> g2.p2 -> g4.p1 -> g4.p3 -> g5.p1 -> g5.p2 critical_path_delay 95

pins g3 0 1 4 0 g4 5 5 4 pins g4 0 5 0 2 5 1 g5 2 3 6 pins g5 0 1 2 3 wire_delay 5 wire g1.p3 g3.p1 wire g1.p4 g2.p1 wire g2.p2 g4.p1 wire g3.p2 g4.p2 wire g4.p3 g5.p1	g4 4 0 g5 9 0 g2 9 3 g3 4 5 g1 0 0

Test cases

1)Test Case containing a loop: This test case tests the presence of a loop in the circuit which could have lead to infinite loop while I was measuring the criticak path delay.

Input	Output
g1 4 3 2 pins g1 0 2 4 1 g2 3 2 1 pins g2 0 0 3 2 g3 4 5 6 pins g3 0 5 4 5 g4 3 3 9 pins g4 0 2 3 1 g5 3 3 6 pins g5 0 0 3 3 g6 5 2 3 pins g6 0 1 5 0 wire_delay 2 wire g1.p2 g3.p1 wire g2.p2 g4.p1 wire g3.p2 g5.p1 wire g5.p2 g6.p1 wire g6.p2 g2.p1 wire g4.p2 g1.p1	bounding_box 18 8 raise Exception("No path found")

2)Single Gate :This test case ensured that my code ran correct while considering the boundary case where no connection was defined that we could only have considered the gate delay.

Input	Output
g1 3 3 5 pins g1 0 0 3 0 3 3 0 3 wire_delay 1	bounding_box 3 3 Critical path found: g1.p1 -> g1.p2 critical_path_delay 5 g1 0 0

3)Two gates and one connection :This test case being a minor test case helped me check various aspects in the code.

Input	Output
g1 10 10 3 pins g1 0 0 10 5 g2 20 30 4 pins g2 0 10 20 25 wire_delay 1 wire g2.p2 g1.p1	bounding_box 3 3 Critical path found: g1.p1 -> g1.p2 critical_path_delay 5 g1 0 0

4)No connections: This test case ensured me that my code was returning the maximum value of the delay since it has to give the maximum value of gate delay.

Input	Output
g1 35 40 4 pins g1 0 0 35 35 g2 40 45 2 pins g2 0 0 40 30 g3 20 25 3 pins g3 0 0 20 20 wire_delay 1	bounding_box 20 40 Critical path found: g2.p1 -> g2.p2 -> g1.p1 -> g1.p2 critical_path_delay 32 g1 0 30 g2 0 0

5) Square Layout(gates with same dimensions):This test case ensured that my code was running correct for the when I provided it with similar gates.

Input	Output
g1 3 3 3 pins g1 0 0 3 0 3 3 0 3 g2 3 3 4 pins g2 0 0 3 0 3 3 0 3 g3 3 3 2 pins g3 0 0 3 0 3 3 0 3 g4 3 3 5 pins g4 0 0 3 0 3 3 0 3 wire_delay 5 wire g2.p2 g1.p1 wire g4.p2 g2.p4 wire g3.p2 g1.p4	bounding_box 6 9 Critical path found: g4.p1 -> g4.p2 -> g2.p4 -> g2.p2 -> g1.p1 -> g1.p2 critical_path_delay 42 g1 3 3 g3 3 6 g2 3 0 g4 0 3

6)Maximum dimension gates: This test case ensured that my code ran correctly for limiting values of length and breadth.

Input	Output
g1 100 100 10 pins g1 0 0 0 50 0 100 100 0 100 50 100 100 g2 100 100 8 pins g2 0 0 0 50 0 100 100 0 100 50 100 100	bounding_box 200 100 Critical path found: g2.p1 -> g2.p6 -> g1.p3 -> g1.p4 critical_path_delay 18 g1 100 0 g2 0 0

wire_delay 5 wire g2.p4 g1.p1 wire g2.p5 g1.p2 wire g2.p6 g1.p3	

7)Multiple parallel connections: This test case ensured that my wire connections were appropriately measured on basis of semi perimeter method.

Input	Output
g1 4 3 2 pins g1 0 1 4 2 g2 3 3 3 pins g2 0 1 3 1 g3 4 4 4 pins g3 0 2 4 1 4 3 0 4 g4 3 3 2 pins g4 0 1 3 2 g5 4 3 3 pins g5 0 2 4 1 g6 3 3 1 pins g6 0 1 3 2 0 2 wire_delay 2 wire g1.p2 g3.p1 wire g2.p2 g3.p4 wire g3.p2 g4.p1 wire g3.p3 g5.p1 wire g4.p2 g6.p1 wire g5.p2 g6.p3	bounding_box 14 6 Critical path found: g2.p1 -> g2.p2 -> g3.p4 -> g3.p3 -> g5.p1 -> g5.p2 -> g6.p3 -> g6.p2 critical_path_delay 23 g3 4 0 g1 0 0 g2 0 3 g4 8 0 g6 11 0 g5 8 3

8)Single gate connected to all gates: This ensured that our code could handle the situation with single output .

Input	Output
g1 10 10 5 pins g1 0 1 0 3 0 5 0 7 10 1 10 3 10 5 10 7 g2 8 8 4 pins g2 0 2 0 6 8 2 8 6 g3 8 8 3 pins g3 0 1 0 5 8 1 8 5 g4 8 8 6 pins g4 0 2 0 4 8 2 8 4 g5 8 8 4 pins g5 0 3 0 7 8 3 8 7 wire_delay 3 wire g2.p3 g1.p1 wire g3.p3 g1.p2 wire g4.p3 g1.p3 wire g5.p3 g1.p4	bounding_box 26 24 Critical path found: g4.p1 -> g4.p3 -> g1.p3 -> g1.p5 critical_path_delay 44 g1 16 8 g5 8 8 g2 8 0 g3 8 16 g4 0 8

9) Larger test case: The last two tests cases ensured that my code ran correctly for larger number of gates .

Input	Output
g1 5 46 2 pins g1 0 29 0 35 0 13 5 12 5 44 5 19 g2 64 17 5 pins g2 0 15 0 13 0 10 64 14 64 9 g3 64 32 1 pins g3 0 16 0 20 0 18 64 22 64 11 64 19 g4 75 59 1 pins g4 0 50 0 37 0 46 75 6 75 48 75 4 g5 99 23 2 pins g5 0 17 0 6 0 10 99 12 99 3 99 17 g6 3 54 3 pins g6 0 22 0 50 0 20 3 5 3 17 3 24 g7 51 50 2 pins g7 0 48 0 22 0 11 51 45 51 39 51 47 g8 5 12 5 pins g8 0 8 0 5 0 10 5 1 5 8 5 12 g9 80 81 3 pins g9 0 35 0 1 0 42 80 8 80 57 80 64 g10 23 14 2 pins g10 0 11 0 7 0 9 23 6 23 5 23 14 g11 72 11 3 pins g11 0 3 0 5 0 8 72 7 g12 22 94 4 pins g12 0 91 0 3 0 29 22 64 22 61 22 33 g13 21 99 5 pins g13 0 49 0 56 0 9 21 22 21 83 21 96 g14 64 20 1 pins g14 0 3 0 19 0 20 64 12 64 8 64 20 g15 92 58 3 pins g15 0 27 0 23 0 33 92 28 92 11 92 21 g16 30 93 1 pins g16 0 11 0 58 0 71 30 67 30 36 30 88 g17 39 95 2 pins g17 0 19 0 50 0 64 39 4 39 43 39 95 g18 85 96 1 pins g18 0 9 0 69 0 22 85 35 85 71 85 83 g19 80 14 2 pins g19 0 8 0 3 0 12 80 5 80 4 80 14 g20 60 50 5 pins g20 0 45 0 1 0 33 60 33 60 27 60 26 wire_delay 1 wire g20.p6 g8.p1 wire g13.p5 g2.p1 wire g3.p6 g13.p3 wire g11.p4 g18.p2 wire g6.p6 g16.p2 wire g15.p6 g8.p2	bounding_box 342 294 Critical path found: g6.p1 -> g6.p6 -> g3.p3 -> g3.p6 -> g10.p3 -> g10.p6 -> g19.p1 -> g19.p6 -> g5.p2 -> g5.p6 -> g7.p2 -> g7.p6 -> g15.p2 -> g15.p6 -> g9.p1 -> g9.p6 -> g11.p3 -> g11.p4 -> g1.p3 -> g1.p6 -> g12.p3 -> g12.p6 -> g20.p1 -> g20.p6 -> g2.p2 -> g2.p5 -> g16.p3 -> g16.p5 -> g18.p1 -> g18.p6 -> g14.p2 -> g14.p6 -> g8.p3 -> g8.p5 -> g17.p1 -> g17.p4 critical_path_delay 3243 g10 116 110 g4 41 110 g15 116 124 g13 116 11 g11 116 182 g1 111 182 g3 41 78 g6 38 110 g7 60 182 g8 111 170 g9 116 193 g14 116 274 g12 196 193 g19 139 110 g2 139 93 g5 208 124 g20 0 182 g18 218 193 g17 303 193 g16 139 0

wire g10.p5 g1.p2 wire g7.p6 g11.p2 wire g2.p5 g16.p3 wire g11.p4 g16.p1 wire g13.p5 g15.p1 wire g6.p4 g1.p1 wire g1.p6 g12.p3 wire g7.p4 g9.p3 wire g11.p4 g1.p3 wire g20.p6 g18.p3 wire g20.p6 g2.p2 wire g10.p4 g5.p3 wire g13.p6 g19.p2 wire g16.p5 g18.p1 wire g4.p5 g15.p3 wire g5.p6 g7.p2 wire g10.p6 g17.p2 wire g19.p6 g5.p2 wire g9.p4 g12.p1 wire g6.p6 g3.p3 wire g12.p6 g20.p1 wire g18.p6 g17.p3 wire g15.p4 g12.p2 wire g4.p5 g19.p3 wire g7.p6 g15.p2 wire g4.p5 g10.p2 wire g12.p4 g14.p1 wire g9.p6 g11.p3 wire g10.p6 g19.p1 wire g14.p6 g8.p3 wire g4.p6 g3.p2 wire g10.p4 g13.p2 wire g6.p5 g4.p2 wire g20.p5 g14.p3 wire g8.p5 g17.p1 wire g3.p6 g10.p3 wire g18.p6 g14.p2 wire g15.p6 g9.p1	

10.)Test Case for larger values of gates :

We also checked out code for larger test cases .Two of them are attached which includes more number of gates .