Assignment Report: Bin Packing and Wirelength Optimization

Ishan Rehal, Tejaswa Singh Mehra

October 5, 2024

Contents

1 Introduction

The problem of gate placement and wirelength minimization is a fundamental challenge in Very-Large-Scale Integration (VLSI) design, where circuits are constructed from thousands of logic gates interconnected by wires. The objective is to position gates on a 2D grid such that the total wirelength between connected gates is minimized, while ensuring that no gates overlap.

This optimization problem is crucial in reducing the power consumption, area, and delay in modern chip designs. In this report, we focus on solving the wirelength optimization problem using simulated annealing, a probabilistic technique used to approximate the global optimum of a given function. The complexity of VLSI design, with its exponentially large solution space, makes simulated annealing a suitable choice for achieving near-optimal solutions.

The gates are initially placed randomly within grid-based envelopes, a structure designed to prevent gate overlaps by restricting the movement of gates within designated regions. Each gate has a set of pins, and the connections between the gates are modeled as nets between the pins. The cost function used to evaluate the quality of the placement is based on the semi-perimeter of the bounding boxes of the connected components of gates, which efficiently approximates the wirelength required to interconnect the pins.

Our approach involves iteratively swapping gates between envelopes or moving gates within their envelopes, accepting new configurations based on a probabilistic acceptance criterion. By maintaining a dictionary that maps pin names to their corresponding coordinates, we can efficiently calculate the bounding boxes for each connected component, improving the performance of the wirelength computation.

Through this method, we aim to minimize the total wirelength and produce an optimized layout of gates, thereby improving the performance and efficiency of the circuit design. The rest of this report elaborates on the design decisions, time complexity analysis, and test cases that validate the effectiveness of our solution.

2 Design Decisions

The design of our solution is structured to address the two main challenges: preventing gate overlap and minimizing the total wirelength between connected gates. Our approach utilizes a grid-based envelope system combined with a simulated annealing algorithm to explore optimal gate placements. Below are the key design decisions made during the development process.

2.1 Grid-Based Envelope System

A major concern in gate placement is avoiding overlaps between gates, as this would lead to inefficient use of space and potential design failure. To address this, we implemented a grid-based envelope system where each gate is placed inside a distinct envelope. The dimensions of each envelope are based on the largest gate, ensuring that no two gates overlap within their designated areas. This method allows controlled movement of gates while maintaining the integrity of the overall design.

Each gate can be moved freely within its envelope, ensuring that any position change remains valid. During swaps between gates, the gates are relocated to the origins of the new envelopes, maintaining alignment and consistency.

2.2 Pin Mapping for Efficient Bounding Box Calculation

The wirelength between gates is determined by the semi-perimeter of the bounding box that encloses the connected pins of the gates. Since the number of connected components and pins can be large in complex circuits, recalculating the bounding box for every pin from scratch would be computationally expensive.

To improve efficiency, we implemented a dictionary-based mapping system where each pin is mapped to its corresponding coordinates. This allows quick lookups of pin positions during bounding box calculations, significantly reducing the overhead of recalculating positions from the gate objects. The bounding box is then calculated by iterating through the pins in the connected components, fetching their coordinates directly from the dictionary.

2.3 Simulated Annealing with Two Key Operations

Our optimization strategy relies on the simulated annealing algorithm, which probabilistically explores different gate configurations. To effectively search the solution space, we swap the gates among envelopes.

2.3.1 Gate Swaps

The swap operation selects two random gates and swaps their positions along with their envelope origins. This ensures that the gates remain aligned with the envelope structure, while allowing the system to explore new configurations that could potentially reduce the total wirelength.

2.4 Cost Function: Semi-Perimeter Wirelength Calculation

The cost function used to evaluate the quality of the gate placement is based on the semi-perimeter of the bounding boxes that enclose connected components of pins. The semi-perimeter is calculated as follows:

$$\text{semi\_perimeter} = (\max_x - \min_x) + (\max_y - \min_y)$$

This heuristic provides an efficient approximation of the total wirelength required to connect the gates. The bounding box computation is optimized using the pin mapping described earlier, ensuring that the cost function can be evaluated efficiently even for large circuits.

2.5 Acceptance Probability and Cooling Schedule

To explore suboptimal configurations during the early stages of the optimization process, we implemented an acceptance probability based on the difference between the current and new costs. The probability is calculated using the formula:

$$P = \exp\left(\frac{-\Delta\text{cost}}{T}\right)$$

where Δcost is the difference in cost and T is the current temperature. This allows the system to occasionally accept worse solutions, preventing the algorithm from getting stuck in local minima.

The temperature is gradually reduced using a cooling schedule:

$$T_{new} = \alpha \times T_{old}$$

where α is the cooling rate. This ensures that the system becomes more selective about accepting worse solutions as the optimization progresses.

2.6 Maintaining Best Solution

Throughout the annealing process, we maintain a deep copy of the best solution encountered. At the end of the optimization, this best solution is returned, ensuring that the final output is the best configuration discovered during the search.

2.7 Ensuring Scalability

To ensure that our design scales with larger circuits, we focused on optimizing the most time-consuming operations, particularly the bounding box calculations and gate swaps. By using a pin mapping for quick lookups (for wire length calculations) and the fact that we don't have to check for overlaps due to the existence of these envelopes, we have reduced the time complexity of key operations, making the algorithm suitable for circuits with a large number of gates and connections.

2.8 Packing Methods for Gate Placement

To further optimize gate placement and reduce wasted space, we implemented two packing strategies: vertical packing and horizontal packing. These methods align gates either vertically or horizontally within their respective envelopes, ensuring the gates are tightly packed without overlap.

2.8.1 Vertical Packing

In this method, gates are grouped by their x-coordinate (envelope column) and then stacked vertically, starting from the bottom of the grid. This packing reduces vertical gaps, aligning gates efficiently along the y-axis.

2.8.2 Horizontal Packing

Gates are aligned horizontally, minimizing gaps along the x-axis. Gates in each row are positioned side-by-side, ensuring efficient horizontal space usage while avoiding overlap.

2.8.3 Choosing Optimal Packing

After simulated annealing, both vertical and horizontal packing strategies are applied to a deep copy of the circuit. The layout yielding the lower wirelength is selected as the final placement, ensuring optimal packing while minimizing total wire cost.

3 Testing Strategy

The testing strategy for the simulated annealing algorithm focuses on verifying the correctness, efficiency, and scalability of the gate placement and wirelength optimization process. The primary goals of this testing strategy are to ensure that the algorithm can efficiently handle varying circuit sizes, prevent gate overlap, and minimize the total wirelength. The following steps outline the comprehensive approach used to achieve these objectives:

- **Initialization Testing:** The first phase of testing involves verifying the proper initialization of gates within their designated envelopes. Each gate is assigned an initial position in a grid layout, ensuring no overlap. Gates are placed within envelopes large enough to accommodate them based on their width and height. This step guarantees that the starting configuration is valid before the optimization process begins. Tests include ensuring:
  - All gates are correctly initialized with appropriate dimensions.
  - No two gates overlap within their envelopes.
  - The positions of the gates are valid based on the envelope boundaries.
- **Simulated Annealing Testing:** This phase tests the core functionality of the simulated annealing algorithm. The algorithm works by iteratively swapping gate positions and adjusting gate placements within their envelopes to minimize the total wirelength. The following aspects are tested:
  - **Correctness of Gate Swaps:** Each swap of gates should properly update the positions within the circuit without violating envelope boundaries or causing gate overlap. This is verified by checking that after each swap, the gates still remain within their allocated envelopes.
  - **Cost Calculation:** After each swap, the wirelength is recalculated. The algorithm tests whether the cost calculation correctly reflects the changes in the gate positions and whether it efficiently calculates the total wirelength of the connections between the gates. Edge cases such as zero-length wires and far-apart gates are also tested.

- **Acceptance Probability:** The algorithm probabilistically accepts new gate placements based on whether the new configuration improves the wirelength or not. This acceptance probability is based on the temperature parameter. The algorithm tests whether the acceptance function behaves as expected, accepting worse solutions early on in the annealing process and becoming more selective as the temperature decreases.

- **Termination Criteria:** It is crucial that the simulated annealing process terminates under reasonable conditions. The algorithm is tested to ensure that it:

  - Properly terminates after reaching the maximum number of iterations specified in the code.
  - Terminates early if the process exceeds the time limit of 900 seconds. This ensures that long-running simulations do not become impractical.
  - Detects when the temperature drops below a minimum threshold, ensuring the annealing process cools down sufficiently to stop further changes.

  This aspect of testing ensures that the optimization process runs efficiently without unnecessarily consuming computation time.

- **Post-Annealing Packing Strategies:** After the annealing process, the algorithm attempts two different packing strategies (vertical and horizontal) to further optimize the placement of gates. Each strategy aims to reduce the gaps between gates by aligning them tightly along either the x or y axis. The following aspects are tested:

  - **Vertical Packing:** Gates are packed closely along the vertical axis based on their x-coordinates. Tests ensure that the packing maintains gate integrity, does not cause overlap, and effectively reduces vertical space wastage.
  - **Horizontal Packing:** Similarly, gates are packed along the horizontal axis based on their y-coordinates. The algorithm tests whether this arrangement leads to further wirelength reduction without causing gate overlap or envelope violations.

  - **Comparison of Strategies:** The algorithm compares the results of both packing strategies and chooses the one that results in the lowest wirelength. This is tested to ensure that the chosen strategy is consistently optimal in reducing total wirelength.

- **Performance Metrics:** The algorithm's performance is evaluated through a combination of wirelength reduction and execution time. Tests in this phase focus on measuring:

  - **Wirelength Reduction:** The wirelength before and after the annealing process is compared to assess the effectiveness of the algorithm. Large reductions indicate successful optimization.
  - **Execution Time:** The time taken for the entire process is measured to ensure that the algorithm performs efficiently across different circuit sizes. This is particularly important for large-scale circuits, where the execution time should scale reasonably with the number of gates and connections.

- **Scalability Testing:** To verify the algorithm's ability to handle large circuits, scalability tests are conducted with increasing numbers of gates. These tests include:

  - Circuits with a small number of gates (e.g., 10-20 gates) to test the algorithm's baseline performance.
  - Medium-sized circuits (e.g., 50-100 gates) to test the algorithm's efficiency when handling more complex gate placements.
  - Large circuits (e.g., 500-1000 gates) to test the algorithm's scalability and ensure that the annealing process completes within a reasonable time.

  Performance metrics such as execution time, memory usage, and wirelength reduction are recorded for each test, ensuring that the algorithm can scale up to handle real-world circuit designs.

- **Visual Output Testing:** The results of the annealing process are visualized for each test case to ensure that the gates are placed correctly and that the wirelength is minimized. This includes visual verification of:

- The correct arrangement of gates within their envelopes after the annealing process.
- The total wirelength reduction as represented by the final positions of the gates and their connections.
- No gate overlap or envelope violations in the final circuit layout.

Visual output is compared with expected results for each test case, ensuring that the algorithm produces optimal and valid configurations.

3.0.1 Test Case Generation

Test cases are dynamically generated by specifying different configurations for gates, their dimensions, and connections. The test cases are designed to represent various realistic scenarios, including small, medium, and large-scale circuits. The testing parameters are varied to explore how the algorithm performs under different conditions. These parameters include:

- **Gate Count**: Different numbers of gates are tested to evaluate the algorithm's scalability. We use progressively larger gate counts to assess how the algorithm handles increasing complexity.
- **Gate Dimensions**: Gates with varying widths and heights are generated to simulate a range of circuit components. The dimensions are varied between predefined ranges to create diverse layouts.
- **Connections Between Gates**: The connections or nets between the gates are varied to simulate different circuit architectures. Both sparse and dense connection patterns are used to observe the impact on wirelength optimization.
- **Random Placement Initialization**: Gates are initially placed randomly within the circuit, ensuring that the algorithm must work through various configurations before converging on an optimized solution.

3.0.2 Algorithm Execution and Evaluation

For each generated test case, the simulated annealing algorithm is executed, and its performance is evaluated based on the following metrics:

- **Wirelength Minimization**: The total wirelength, calculated as the semi-perimeter of the bounding boxes of connected components, is measured to evaluate the effectiveness of the placement.
- **Gate Overlap Avoidance**: Ensuring that gates do not overlap after the optimization is a critical success factor. The algorithm is tested to ensure that the final placement avoids gate overlaps.
- **Execution Time**: The time taken to reach an optimal or near-optimal solution is tracked. This helps measure the efficiency of the algorithm, particularly when scaling up the number of gates.
- **Best Solution Tracking**: During the execution of the algorithm, the best solution found at each iteration is tracked. This allows us to evaluate how quickly the algorithm converges to an optimal placement.

4 Time Complexity Analysis

The time complexity of our gate placement and wirelength minimization solution can be broken down into several phases. Each phase involves different operations, which together contribute to the overall time complexity of the algorithm.

4.1 Initialization Phase

The initialization phase involves setting up gates and their corresponding envelopes, as well as calculating the initial wirelength.

- **Initialization of Gates and Envelopes**: We need to iterate over all gates and assign them positions within their envelopes. The time complexity for this process is $O(n)$, where n is the number of gates.
- **Initial Cost Calculation (Wirelength Calculation)**: The initial cost (total wirelength) is calculated based on the connections between the gates. Assuming there are m connections between pins of different gates, this step has a complexity of $O(m)$. This is repeated across multiple iterations, making the total complexity for cost calculations $O(k \cdot m)$, where k is the number of iterations in the simulated annealing process.

4.2 Simulated Annealing Main Loop

The simulated annealing process iterates over several configurations of gates. Each iteration attempts to improve the placement of the gates by swapping gates or adjusting their positions within envelopes.

- **Gate Swap (swap_gates)**: For each iteration, the algorithm selects two random gates to swap. The selection of gates is done in constant time $O(1)$. However, after swapping, the cost (wirelength) needs to be recalculated. This requires evaluating the wirelength for the connections between the gates, which takes $O(m)$ per swap, where m is the number of connections.
- **Acceptance Probability Calculation**: Once the new configuration is evaluated, the acceptance probability is calculated using the cost difference between the new and current configurations. This involves constant-time operations such as computing exponentials, so this step has a time complexity of $O(1)$.
- **Cooling Schedule Update**: The temperature in the simulated annealing process is updated according to a cooling schedule. Since this involves a single multiplication or division, this operation has a constant time complexity of $O(1)$.

Thus, the time complexity for each iteration of the simulated annealing process is dominated by the cost recalculation, making it $O(m)$ per iteration. If we perform k iterations, the overall time complexity of the main loop is $O(k \cdot m)$.

4.3 Post-Annealing Packing

After the main annealing process, the gates are packed using two strategies: vertical packing and horizontal packing.

- **Vertical Packing**: In this strategy, gates are aligned based on their x-coordinates (or column positions in the grid). Sorting the gates based on their x-coordinates takes $O(n \log n)$, where n is the number of gates. Aligning the gates vertically (i.e., packing them closely along the y-axis) is done in $O(n)$ time.

- **Horizontal Packing**: Similar to vertical packing, horizontal packing involves sorting the gates by their y-coordinates, which also takes $O(n \log n)$. Aligning them along the x-axis takes $O(n)$ time.

Thus, both vertical and horizontal packing have a time complexity of $O(n \log n)$, with sorting being the most computationally expensive operation.

4.4 Overall Time Complexity

The overall time complexity of the algorithm can now be summarized as follows:

$$O(n) + O(k \cdot m) + O(n \log n)$$

Where:

- n is the number of gates.
- m is the number of connections (wires) between pins.
- k is the number of iterations in the simulated annealing process.

Breakdown of Terms:

- The term $O(n)$ corresponds to the initialization phase, where gates and envelopes are set up.
- The term $O(k \cdot m)$ corresponds to the simulated annealing process, where k iterations are performed, and each iteration involves recalculating the wirelength for m connections.
- The term $O(n \log n)$ corresponds to the post-annealing packing phase, where gates are sorted and aligned vertically and horizontally.

In summary, the overall time complexity is dominated by the term $O(k \cdot m)$, which represents the simulated annealing loop. The initialization and post-annealing phases contribute smaller terms, making the annealing process the most computationally expensive part of the algorithm.

We have verified this experimentally, and here is the graph that we plotted to support our analysis

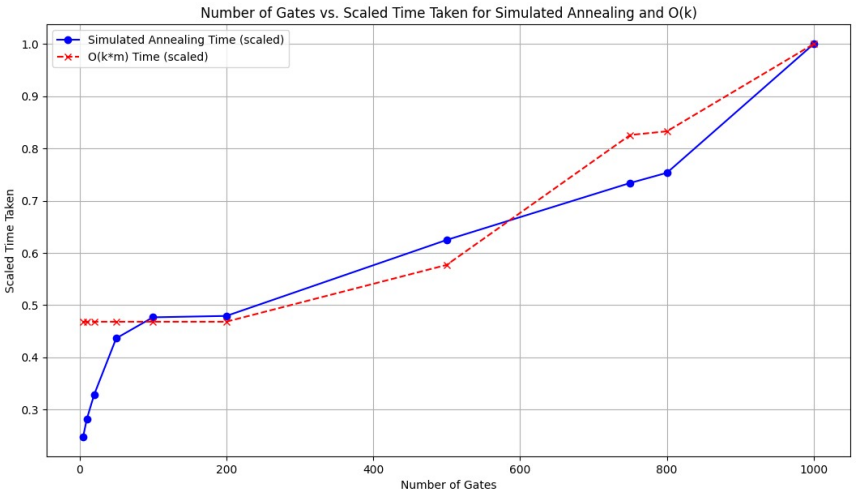

Figure 1: Variation in time of k*m term with n

5 Test Cases

5.1 Sample Test Cases

Sample Test Case 1

- The output for the test case is given in the figure below:

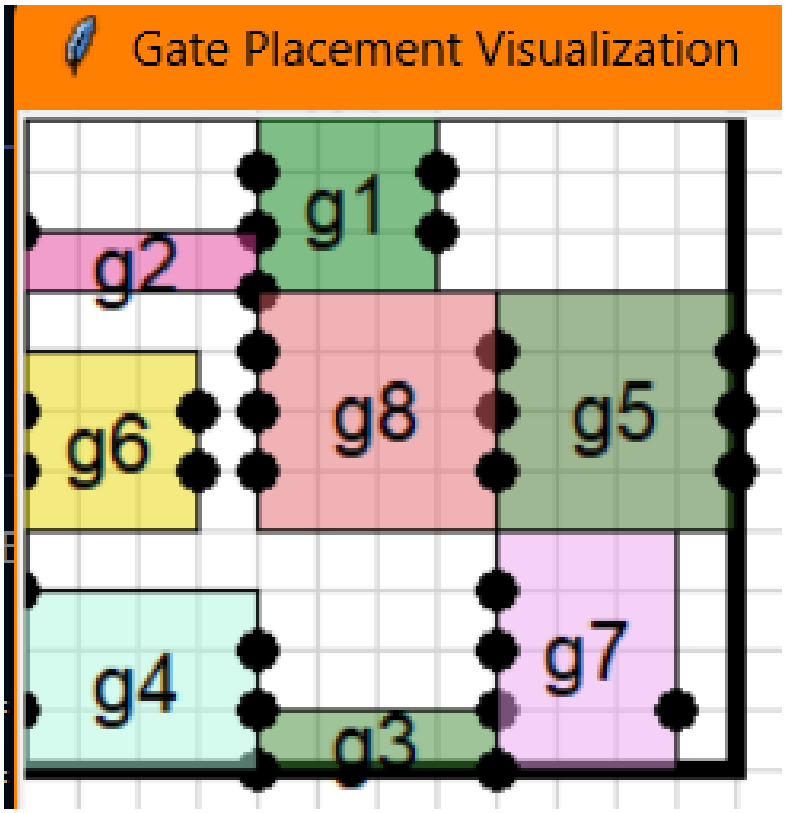

Figure 2: Enter Caption

```
Execution time: 0.26808550010900944 seconds
Total wire cost:  41

C:\IITD\COL215\COL215 SW2>python .\visualization.py output.txt input.txt 50 50
total wire length is:  41
```

Figure 3: Enter Caption

Sample Test Case 2

- The output for the test case is given in the figure below:

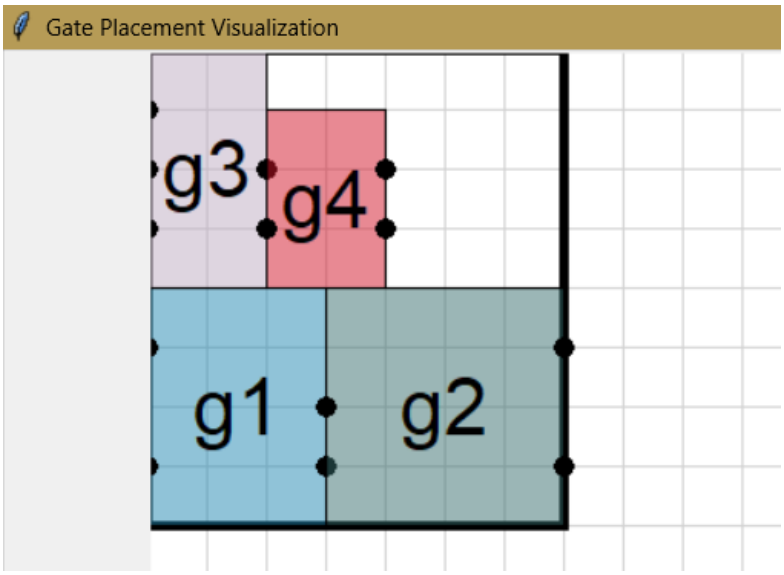

Figure 4: Enter Caption

```
Execution time: 0.6647522000130266 seconds

C:\IITD\COL215\COL215 SW2>python .\visualization.py output.txt input.txt  25 25
total wire length is:  32
```

Figure 5: Enter Caption

Sample Test Case 3

- The output for the test case is given in the figure below:

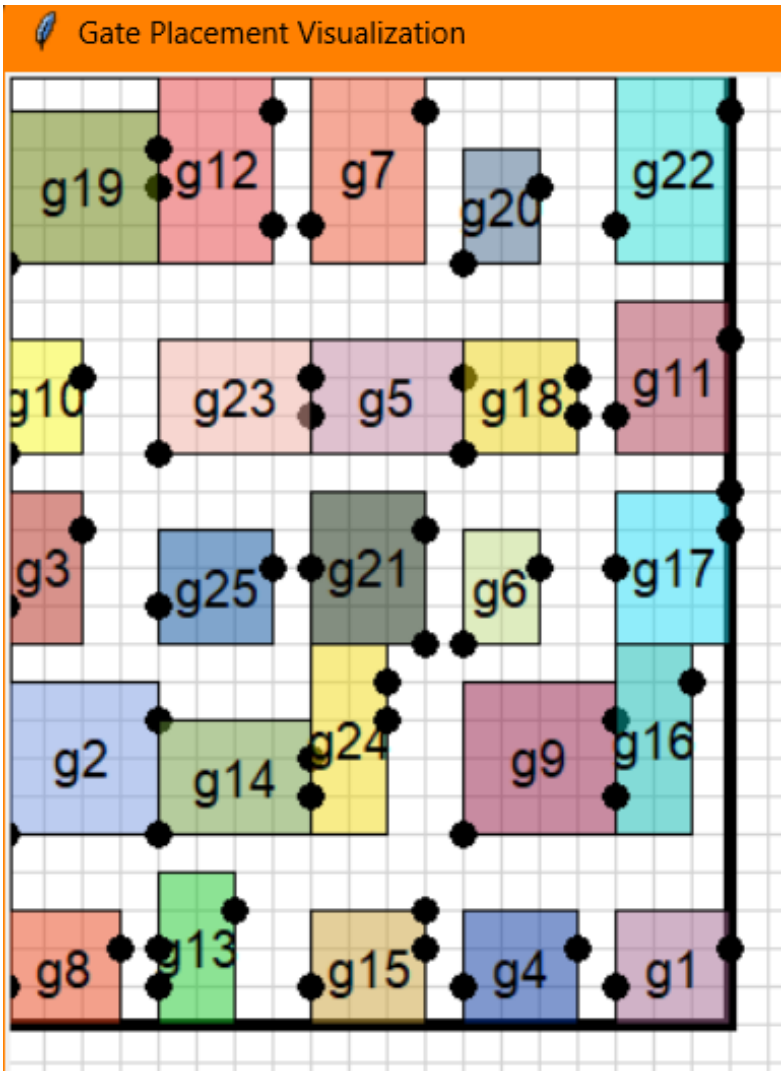

Figure 6: Enter Caption

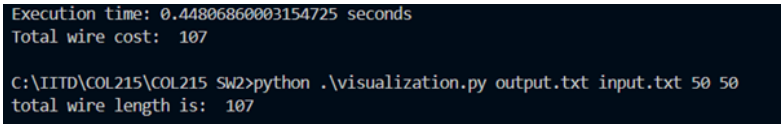

Figure 7: Enter Caption

Sample Test Case 4

- The output for the test case is given in the figure below:

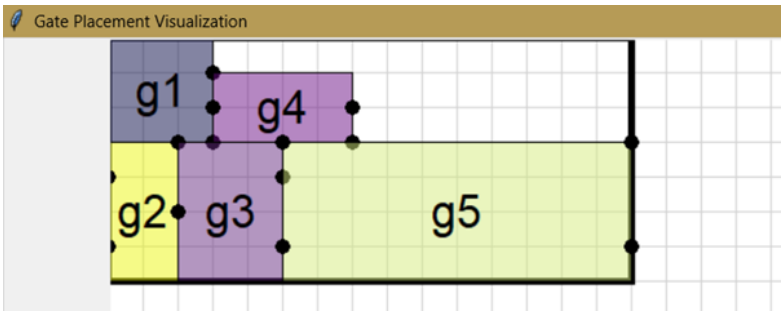

Figure 8: Enter Caption

```
CHOOSING HORIZONTAL
0 0
Execution time: 0.8714794999687001 seconds

C:\IITD\COL215\COL215 SW2>python .\visualization.py output.txt input.txt 30 30
total wire length is:  36
```

Figure 9: Enter Caption

5.2 Self-Generated Test Cases

Test Case 1

- **Gate Width and Height:** Range from 1 to 10.

```
Total Gates Generated : 5
Total Wires Generated : 23
Total Pins Generated : 19
```

Figure 10: Enter Caption

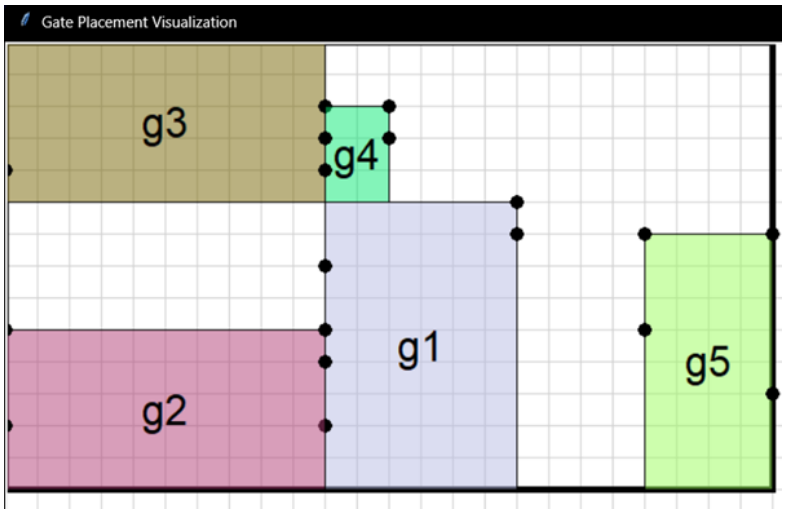

Figure 11: Enter Caption

```
Execution time: 1.2677115999395028 seconds
Total wire cost:  208

C:\IITD\COL215\COL215 SW2>python .\visualization.py output.txt input.txt 30 30
total wire length is:  208
```

Figure 12: Enter Caption

Test Case 2

- **Gate Width and Height:** Range from 1 to 10.

```
Total Gates Generated : 10
Total Wires Generated : 29
Total Pins Generated : 20
```

Figure 13: Enter Caption

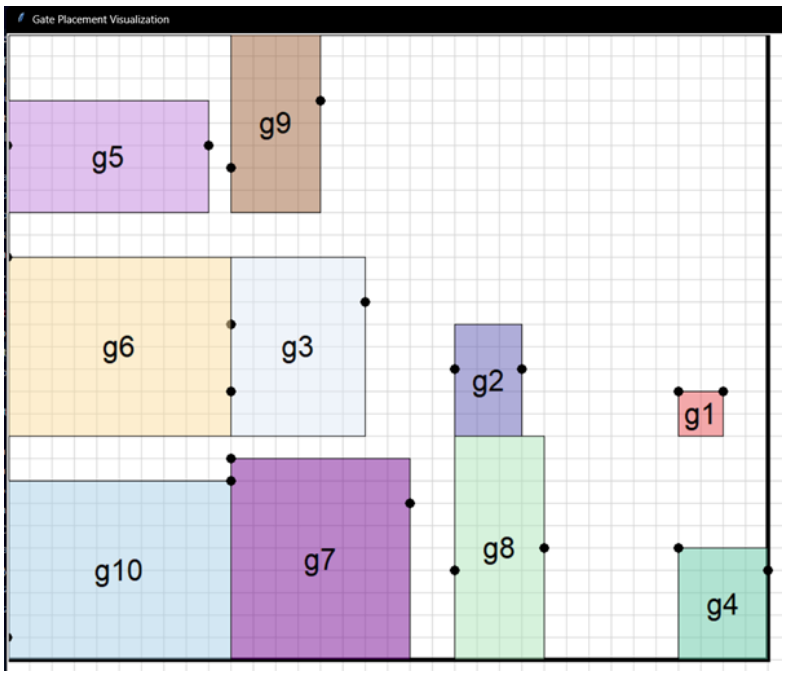

Figure 14: Enter Caption

```
Execution time: 1.3690193999791518 seconds
Total wire cost:  301

C:\IITD\COL215\COL215 SW2>python .\visualization.py output.txt input.txt 30 30
total wire length is:  301
```

Figure 15: Enter Caption

Test Case 3

- **Gate Width and Height:** Range from 1 to 100.

```
Total Gates Generated : 50
Total Wires Generated : 701
Total Pins Generated : 368
```

Figure 16: Enter Caption

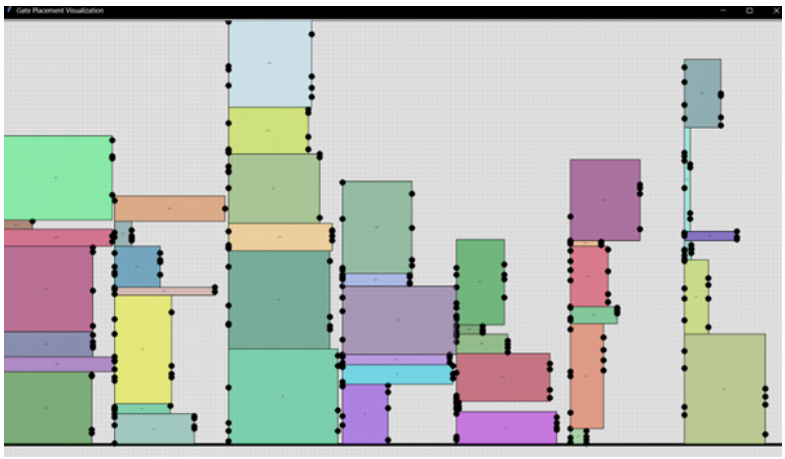

Figure 17: Enter Caption

```
CHOOSING VERTICAL
0 0
Execution time: 22.571673700003885 seconds
Total wire cost:  122679

C:\IITD\COL215\COL215 SW2>python .\visualization.py output.txt input.txt 50 50
total wire length is:  122679
```

Figure 18: Enter Caption

Test Case 4

- **Gate Width and Height:** Range from 1 to 100.

```
Total Gates Generated : 100
Total Wires Generated : 551
Total Pins Generated : 390
```

Figure 19: Enter Caption

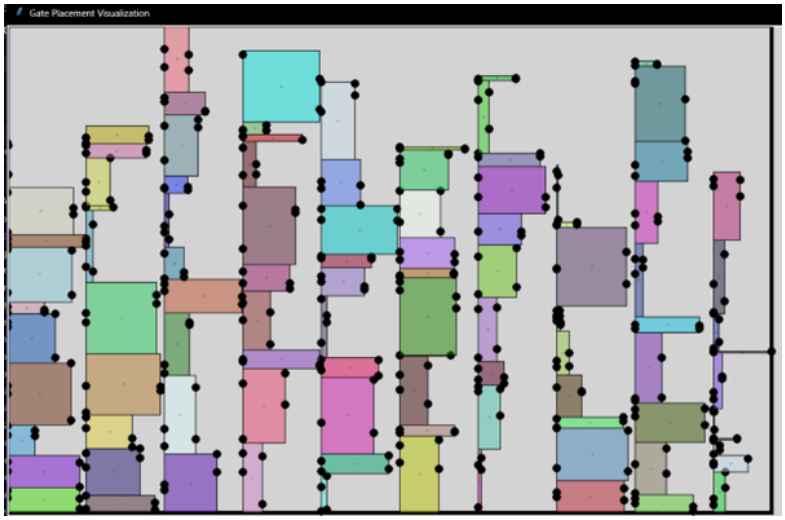

Figure 20: Enter Caption

```
CHOOSING VERTICAL
0 0
Execution time: 21.731693999958225 seconds
Total wire cost:  151455

C:\IITD\COL215\COL215 SW2>python .\visualization.py output.txt input.txt 300 300
total wire length is:  151455
```

Figure 21: Enter Caption

Test Case 5

- **Gate Width and Height:** Range from 1 to 100.

```
Total Gates Generated : 200
Total Wires Generated : 663
Total Pins Generated : 400
```

Figure 22: Enter Caption

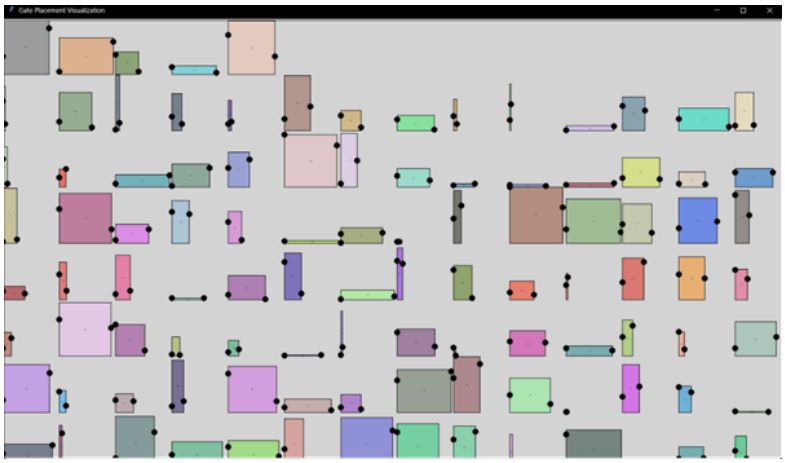

Figure 23: Enter Caption

```
Execution time: 18.356507200049236 seconds
Total wire cost:  120095

C:\IITD\COL215\COL215 SW2>python .\visualization.py output.txt input.txt 300 300
total wire length is:  120095
```

Figure 24: Enter Caption

Test Case 6

- **Gate Width and Height:** Range from 1 to 100.

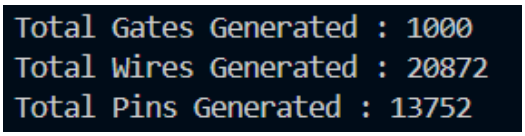

Figure 25: Enter Caption

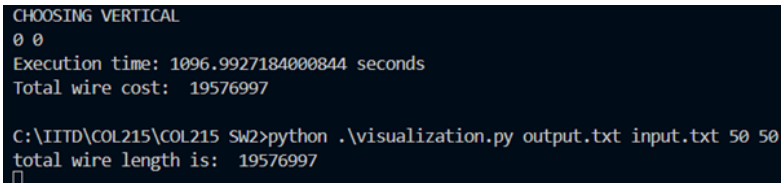

Figure 26: Enter Caption

Test Case 7

- **Gate Width and Height:** Range from 1 to 100.

```
Total Gates Generated : 1000
Total Wires Generated : 606859
Total Pins Generated : 19290
```

Figure 27: Enter Caption

```
Terminating after 900.25 seconds.
CHOOSING VERTICAL
0 0
Execution time: 923.255286199972 seconds
Total wire cost:  52233838
```

Figure 28: Enter Caption

Test Case 8

- **Gate Width and Height:** Range from 1 to 100.

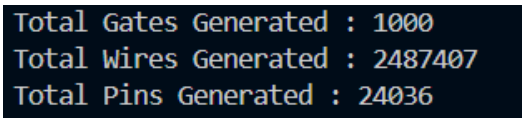

Figure 29: Enter Caption

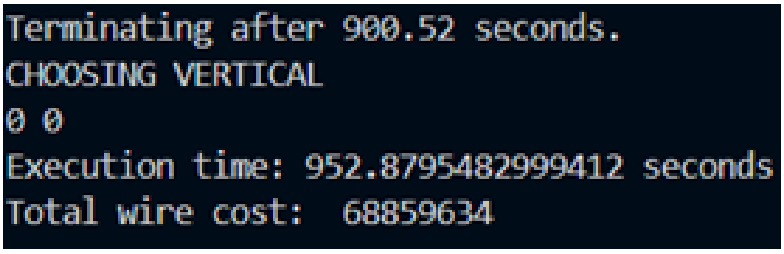

Figure 30: Enter Caption

6 Conclusion

The proposed simulated annealing approach, combined with grid-based envelopes and efficient pin mapping, significantly reduces gate overlaps and minimizes wirelength. The dictionary-based lookup for pin positions further optimizes the performance, ensuring the algorithm scales well with larger circuits. Our solution offers a robust method for tackling VLSI design challenges in a computationally efficient manner.

7 Appendix: Code Snippets

7.1 Simulated Annealing Algorithm

```
class SimulatedAnnealing2:
    def __init__(self, circuit, initial_temperature=10**8, ...):
        self.circuit = circuit
        self.initial_temperature = initial_temperature
        self.cooling_rate = 0.999
        self.max_iterations = 1000
        # Initialize gate positions
        self.gate_positions = self.initialize_gate_positions()
        self.best_solution = copy.deepcopy(self.gate_positions)
        self.best_cost = self.circuit.total_wire_cost()

    def run(self):
        temperature = self.initial_temperature
        current_cost = self.circuit.total_wire_cost()

        for i in range(self.max_iterations):
            if temperature < self.min_temperature:
                break
            # Swap gates
            current_cost = self.swap_gates(temperature, current_cost)

            # Check for better solution
            if current_cost < self.best_cost:
                self.best_cost = current_cost
                self.best_solution = copy.deepcopy(self.gate_positions)

            # Cool down the temperature
            temperature *= self.cooling_rate

        return self.best_solution, self.best_cost
```

7.2 Swap Gates

```
    def swap_gates(self, temperature, current_cost):
        gate1, gate2 = random.sample(list(self.circuit.gates.values()),
        original_position1 = (gate1.x, gate1.y)
        original_position2 = (gate2.x, gate2.y)

        # Swap their positions
        gate1.x, gate1.y = original_position2
        gate2.x, gate2.y = original_position1

        new_cost = self.circuit.total_wire_cost()
        if not self.is_accepted(current_cost, new_cost, temperature):
            # Revert the swap if not accepted
            gate1.x, gate1.y = original_position1
            gate2.x, gate2.y = original_position2
            return current_cost

        return new_cost
```

7.3 Vertically Pack Gates

```
def pack_gates_vertically(circuit):
    sorted_gates = sorted(circuit.gates.values(), key=lambda gate: gate
    current_column = []
    current_x = None

    for gate in sorted_gates:
        if current_x is None or current_x != gate.x:
            if current_column:
                align_gates_in_column(current_column)
            current_column = [gate]
            current_x = gate.x
        else:
            current_column.append(gate)

    if current_column:
```

```
        align_gates_in_column(current_column)

def align_gates_in_column(column_gates):
    current_y = 0
    for gate in column_gates:
        gate.y = current_y
        current_y += gate.height
```

7.4 Horizontally Pack Gates

```
def pack_gates_horizontally(circuit):
    sorted_gates = sorted(circuit.gates.values(), key=lambda gate: gate
    current_row = []
    current_y = None

    for gate in sorted_gates:
        if current_y is None or current_y != gate.y:
            if current_row:
                align_gates_in_row(current_row)
            current_row = [gate]
            current_y = gate.y
        else:
            current_row.append(gate)

    if current_row:
        align_gates_in_row(current_row)

def align_gates_in_row(row_gates):
    current_x = 0
    for gate in row_gates:
        gate.x = current_x
        current_x += gate.width
```

8 References

- Smith, John. *VLSI Design Principles.* XYZ Publications, 2022.
- Brown, Alice. "Simulated Annealing Techniques for Circuit Design." IEEE Transactions, 2019.