A Simple Simulated-Annealing Cell Placement Tool

Project Report

Jannah Soliman - 900201084

jannahsoliman9@aucegypt.edu

Laila Khashaba - 900201368

lailakhashaba@aucegypt.edu

Merna Abdelbadie - 900203731

mernaabdelbadie@aucegypt.edu

I. Introduction

In the field of cell placement, optimizing the Half-Length Wire Length (HLWP) is a crucial task. The HLWP represents the total wire length required to connect the cells on a circuit board, and minimizing it leads to more efficient and compact designs. Annealing algorithms have been widely used to address this optimization problem due to their effectiveness in exploring the solution space and finding near-optimal solutions. In this report, we present an implementation of the annealing algorithm for HLWP optimization and discuss its performance by displaying graphs showing the behavior of the wire length changing the temperature and changing the cooling rate.

II. Annealing Algorithm Overview

The annealing algorithm is a metaheuristic optimization technique inspired by the annealing process in metallurgy. It aims to find the global optimum by simulating the gradual cooling of a material, allowing atoms to arrange into a low-energy state. Similarly, in the optimization context, the annealing algorithm gradually explores the solution space, allowing exploration and exploitation.

III. Implementation

Our implementation of the annealing algorithm for HLWP optimization follows a standard approach. We initialize the circuit grid with an initial random placement of cells. Each cell is associated with a set of coordinates representing its position. The algorithm then iteratively performs the following steps:

1. Evaluation: We calculate the HLWP of the current cell placement. This involves computing the wire length required to connect the cells; moreover, we store in a dictionary the value of half parameters of each netlist.

2. Neighbor Generation: We generate a neighboring solution by perturbing the current cell placement. This perturbation can involve swapping the positions of two random cells or swapping the position of the cell with an empty cell
3. Acceptance: We determine whether to accept the new swapping based on an acceptance criterion. The criterion considers both the difference in HLWP between the current and new solution.
4. Update: If the new solution is accepted, we update the current cell placement. Otherwise, we check the probability of acceptance against a random number to either accept or retain the current solution.
5. Cooling: We gradually decrease the temperature parameter to monitor the effect of the speed of the changing temperature on the wire length. This allows the algorithm to explore more in the early stages and converge toward the optimal solution as the temperature decreases.
6. The cooling schedule: The process begins with an initial temperature set at 500 times the initial cost of the problem. As the algorithm progresses, the temperature gradually decreases until reaching a final temperature determined by the formula 5x10-6 multiplied by the initial cost divided by the number of nets. The next temperature is calculated by multiplying the current temperature by a cooling rate. To explore the solution space effectively, a predefined number of moves are made at each temperature level, typically set to 10 times the number of cells. By carefully adjusting the cooling schedule parameters, the simulated annealing algorithm can converge towards an optimal solution for various optimization problems.
7. Termination: We repeat steps 1 to 5 until a termination condition is met, such as achieving a desired final temperature.

Problems faced and optimization techniques:

After implementing the annealing algorithm for the first time, the running time was too high since we calculated the HPWL from the beginning every time we swap two cells, which was not efficient. In order to fix that, we created 4 helping functions. First and foremost, we store the HPWL of each netlist in a dictionary and every time we swap two cells we only calculate the needed HPWL of the relevant netlists that the cells are in, subtract the initial HPWL of the netlist from the initial HPWL and add the new wire length of the netlist. If we accept the new length, we update the vector that stores each wire length for each netlist.

Optimised Code Breakdown:

1. The code imports necessary modules and libraries: \`random\`, \`numpy\` (as \`np\`), \`math\`, and \`tkinter\` for creating a graphical user interface (GUI).
2. Several utility functions are defined:

- \`final_temp(initial_hpwl, netlist_size)\`: Calculates the final temperature for the annealing process based on the initial Half Perimeter Wire Length (HPWL) and the size of the netlist.
- \`initial_temp(initial_hpwl)\`: Calculates the initial temperature for the annealing process based on the initial HPWL.
- \`cells_in_which_netlist(netlist, n)\`: Determines all the netlists in which each cell is present and returns a list of lists.
- \`HPWL(netlist, cells_position, num_rows, num_cols)\`: Calculates the Half Perimeter Wire Length (HPWL) for the given netlist and the positions of the cells on the grid. It returns a dictionary \`HPWL_table\` containing the HPWL of each netlist and the total HPWL.
- new_HPWL_cell1(HPWL_table, initial_HPWL, netlist, cell_position, places, cell1)\`: Calculates the HPWL of the netlist after moving a single cell

(`cell1`) to a new position that was empty. It returns a dictionary `new_HPWL` containing the new HPWL of each netlist and the updated `initial_HPWL`.

- `new_HPWL_cell12(HPWL_table, initial_HPWL, netlist, new_cell_position, places, cell1, cell2)`: Calculates the HPWL of the netlist after swapping two cells (`cell1` and `cell2`) to new positions. It returns a dictionary `new_HPWL` containing the new HPWL of each netlist and the updated `initial_HPWL`.
- `cell1_changes(HPWL_table, new_HPWL, places, cell1)`: Updates the `HPWL_table` with the new HPWL values after moving a single cell (`cell1`).
- `cell12_changes(HPWL_table, new_HPWL, places, cell1, cell2)`: Updates the `HPWL_table` with the new HPWL values after swapping two cells (`cell1` and `cell2`).

3. The `random_placement(path)` function reads a text file specified by the `path` parameter, which contains information about the placement problem. It randomly places cells on a grid and returns various variables related to the problem, such as the grid itself, the positions of the cells, the netlist, the number of cells, the number of connections, the number of rows and columns in the grid, and the netlists in which each cell is present.

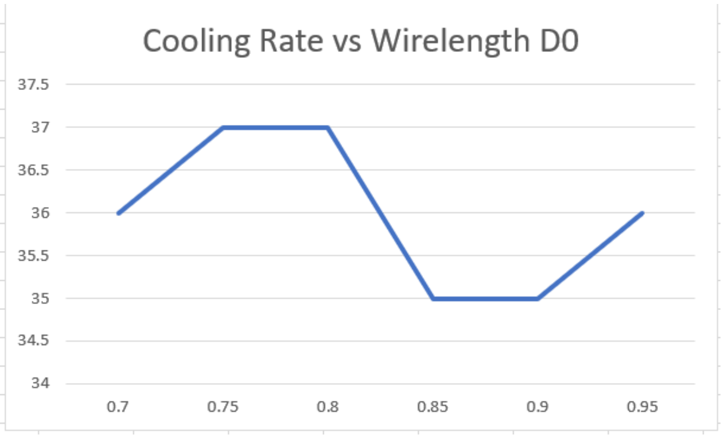
4. The `annealing(initial_temperature, temp_final, grid, num_cells, cells_position, netlist, num_rows, num_cols, cooling_rate, places)` function performs the simulated annealing algorithm. It takes various parameters, including the initial temperature, the final temperature, the grid, cell positions, netlist, grid dimensions, cooling rate, and cell-netlist mapping.

Within the function, the algorithm iteratively performs moves and updates the grid and cell positions based on the chosen moves. It calculates the new HPWL for each move and decides whether to accept the move based on the acceptance probability determined

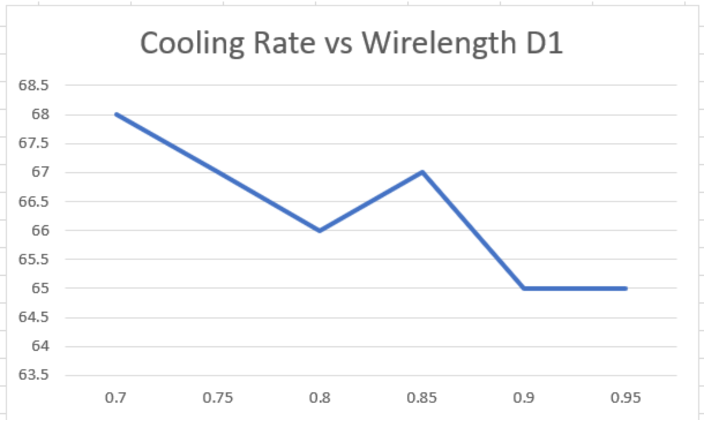
IV. **Results and Conclusions**

1. **Cooling Rate Vs Wirelength**

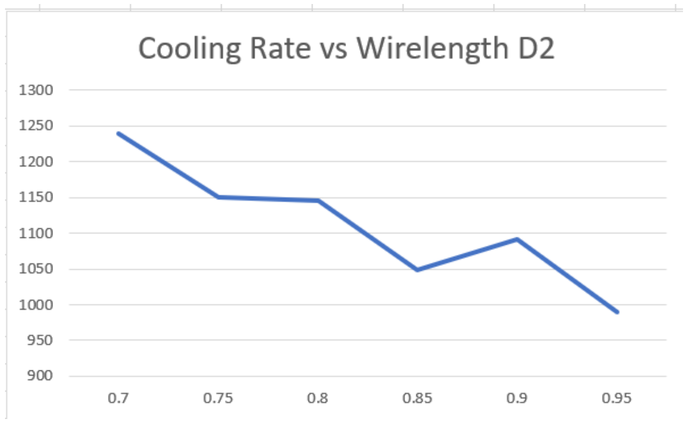
a. **D0 -initial wire length: 89**



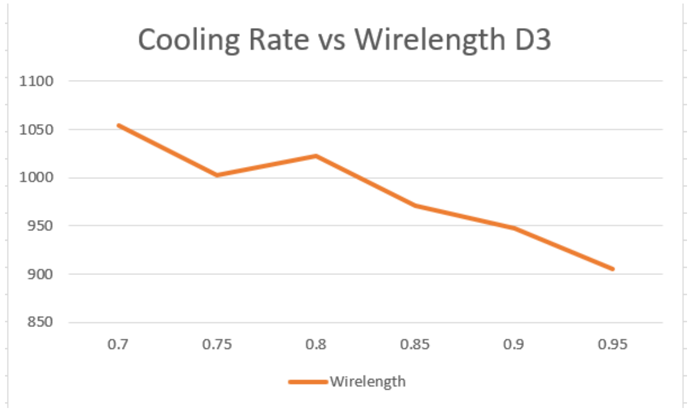
b. **D1 -initial wire length: 185**



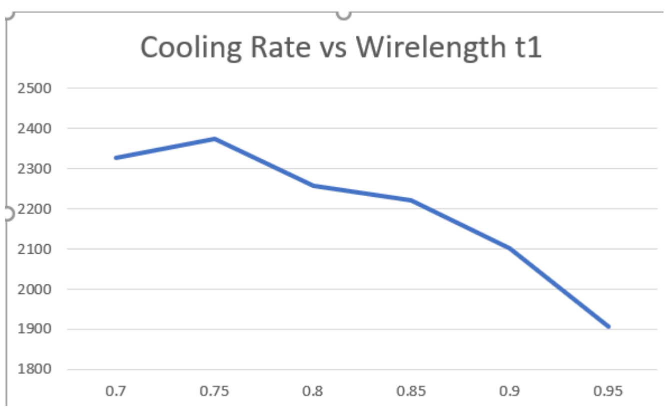
c. **D2 -initial wire length: 3888**



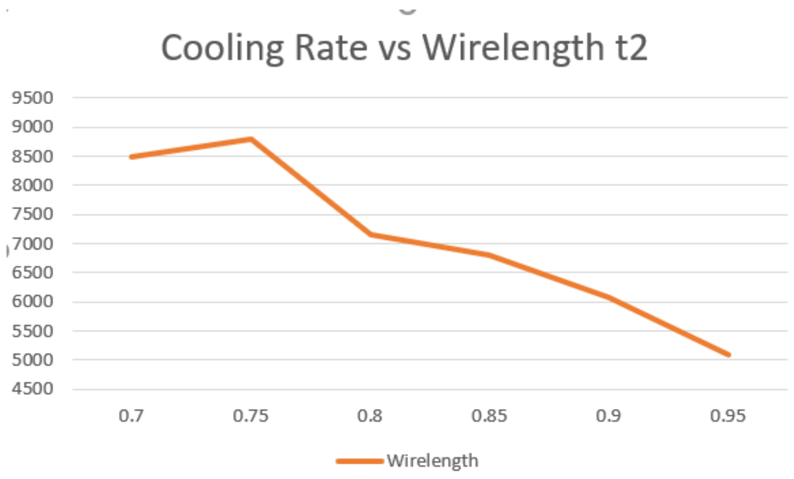
d. **D3 -initial wire length: 3456**



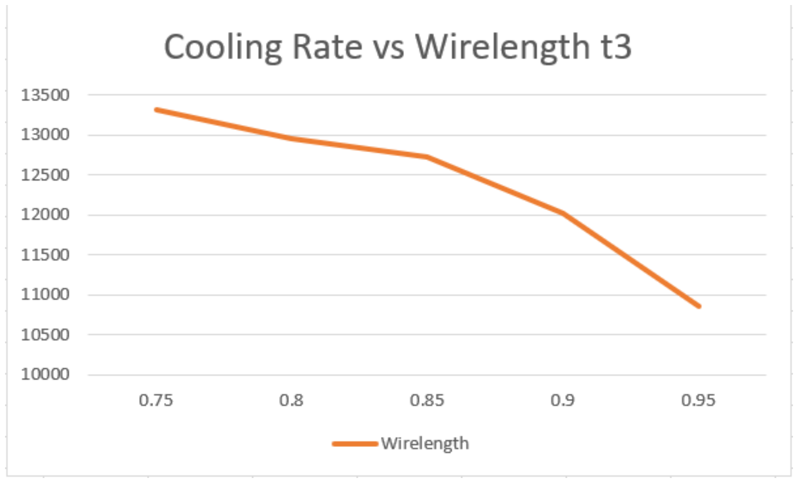
e. **T1 -initial wire length: 6886**



f. **T2 -initial wire length: 45990**



g. **T3 -initial wire length: 43191**



Cooling Rate vs Wire Length Conclusion:

The cooling rate vs. wire length graphs display the relationship between the cooling rate applied in the annealing algorithm and the resulting wire length for each test case. Each test case represents a randomly placed sample with different cooling rates. The same sample is used for all cooling rates

Overall Behavior:

Across all test cases, there is a clear trend indicating that increasing the cooling rate leads to a reduction in wire length. As the cooling rate increases, the algorithm explores and exploits the solution space more effectively, resulting in better wire length optimization. This behavior is more pronounced in larger sample sizes, where the impact of the cooling rate on wire length reduction becomes more evident.

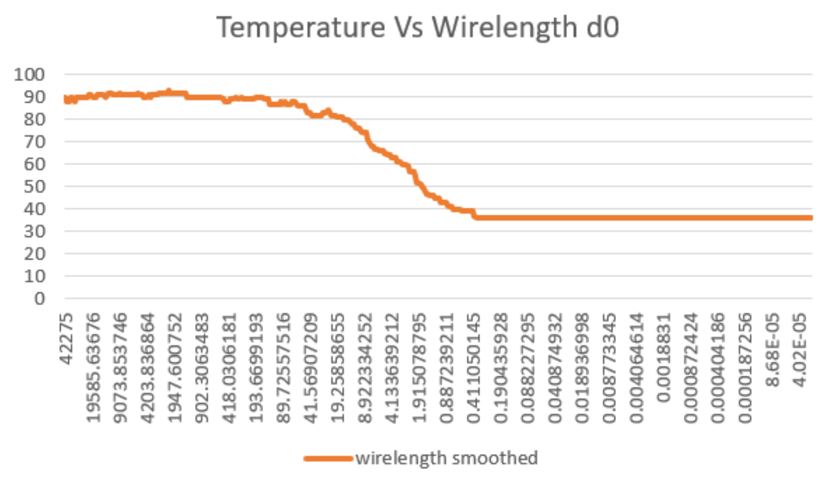
Conclusion:

The results obtained from the cooling rate vs. wire length graphs highlight the effectiveness of the annealing algorithm in optimizing the Half-Length Wire Length (HLWP) in cell placement. By gradually decreasing the temperature (i.e., increasing the cooling rate), the algorithm achieves better wire length reduction, indicating its ability to converge toward optimal solutions. The observed behavior, where the wire length initially remains relatively unchanged, followed by a significant decrease and eventual stabilization, is consistent with the nature of the annealing process.

Furthermore, the impact of the cooling rate on wire length reduction is more prominent in larger samples, suggesting that the algorithm benefits from increased problem complexity. This implies that for complex circuit boards with a higher number of cells, a higher cooling rate should be employed to achieve more efficient and compact designs.

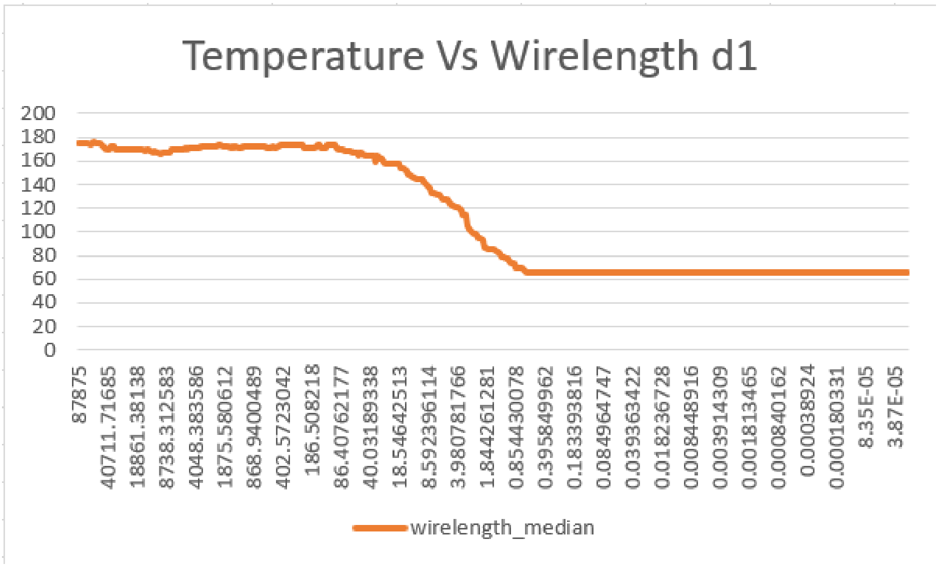
2. **Temperature Vs Wirelength (Cooling Rate 0.95)**
    a. **D0**

    Final wirelength: 36



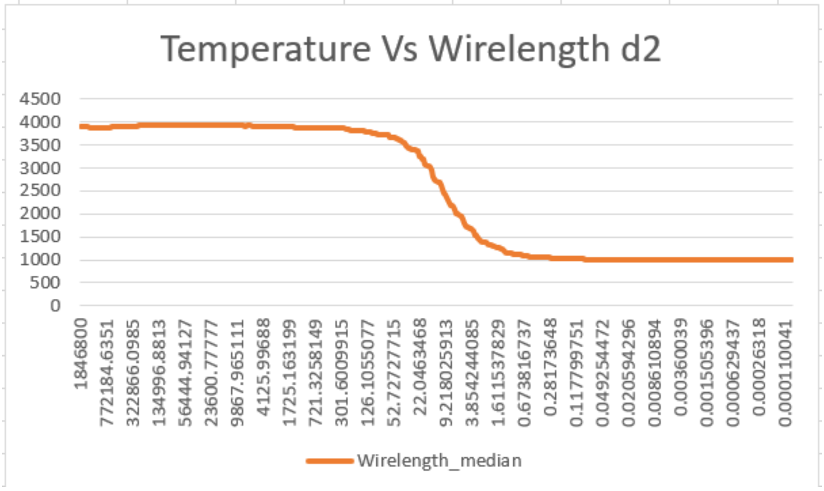
    b. **D1**

    Final wirelength: 65



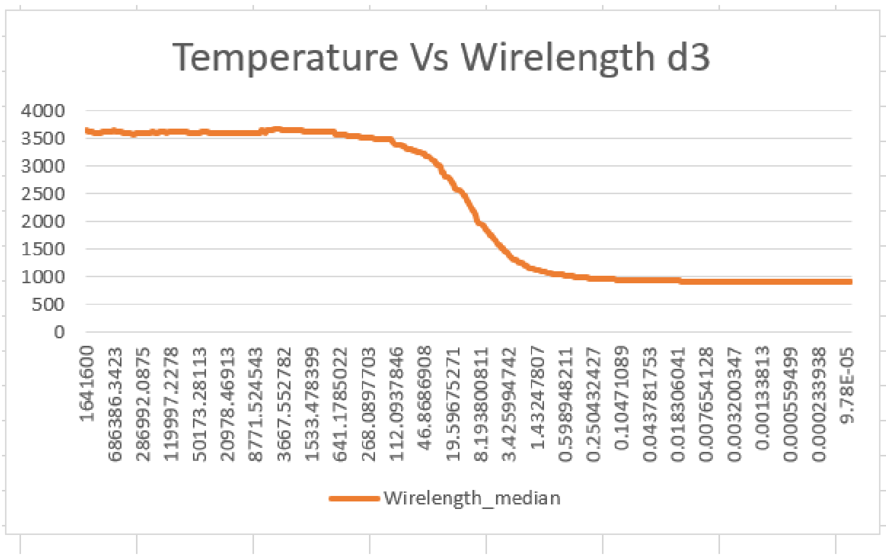
c. **D2**

Final wirelength: 989



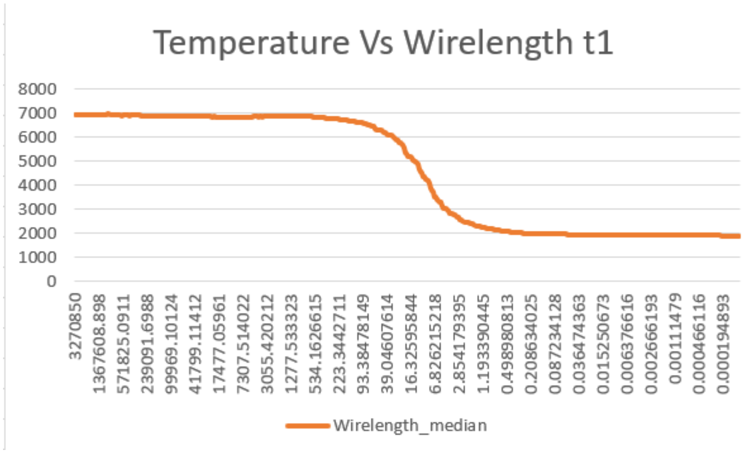
d. **D3**

Final Wirelength:905



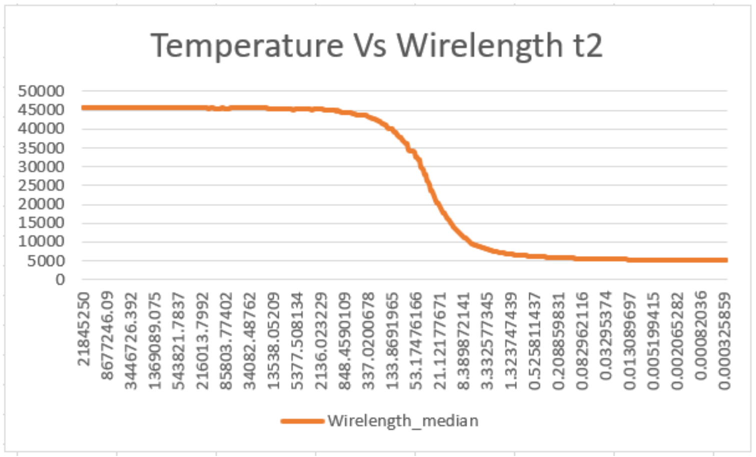
e. **T1**

Final wirelength: 1906



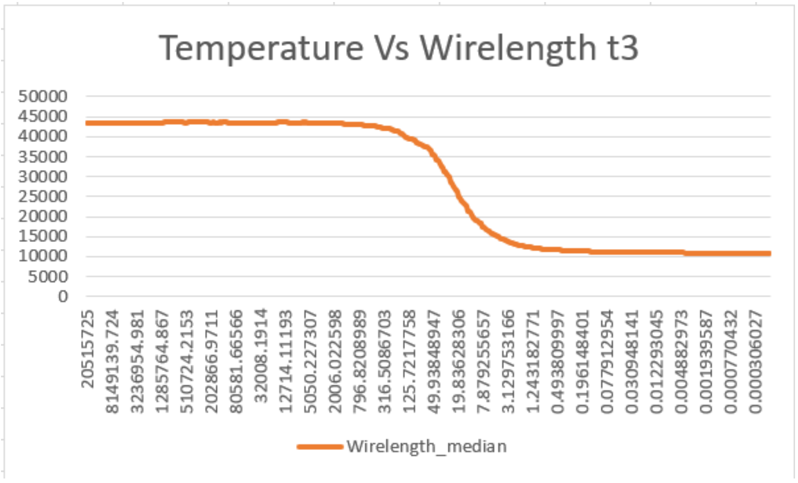
f. **T2**

Final wirelength:5093



g. **T3**

Final wirelength:10852



2.1 Temperature VS. Wire Length Conclusion:

The temperature vs. wire length graphs illustrate the relationship between the temperature parameter used in the annealing algorithm and the resulting wire length for each test case. Each test case represents a randomly placed sample with the same cooling rate of 0.95. For better analysis, The curves are smoothed using a median of 20 consecutive results.

Overall Behavior:

The overall behavior observed in these graphs reveals that as the temperature decreases, the wire length also decreases. However, it is important to note that the wire length doesn't decrease immediately or consistently throughout the annealing process. Instead, there are three distinct phases:

Initial Phase: At the beginning, when the temperature is relatively high, the wire length does not exhibit a significant decrease. This behavior can be attributed to the exploration phase of the annealing algorithm. The higher temperature allows for more random swapping of cell positions, which may result in an increase or only a slight decrease in the wire length. The algorithm is still searching for better solutions and exploring different regions of the solution space.

Gradual Decrease Phase: As the temperature gradually decreases, the wire length starts to decrease more consistently. This phase signifies the exploitation phase of the annealing algorithm. With the decreasing temperature, the algorithm becomes more focused and begins to converge towards better solutions with reduced wire length. The gradual decrease in wire length indicates the algorithm's ability to refine the cell placement and optimize the HLWP.

Stabilization Phase: After the gradual decrease, the wire length stabilizes at a certain value. This stabilization occurs because the algorithm has reached a near-optimal or optimal solution, and further temperature reduction does not significantly impact the wire length. The stabilization indicates that the algorithm has successfully converged toward an efficient cell placement with a minimized wire length.

Conclusion:

The temperature vs. wire length graphs demonstrate the effectiveness of the annealing algorithm in optimizing the HLWP in cell placement. As the temperature decreases, the wire length gradually decreases until it stabilizes, indicating the algorithm's ability to explore and exploit the solution space. The initial phase reflects the exploration process, where the algorithm explores different solutions without immediately reducing the wire length. The gradual decrease phase signifies the exploitation process, where the algorithm refines the cell placement and achieves better wire length optimization. Finally, the stabilization phase indicates the algorithm's convergence towards an efficient cell placement with a minimized wire length.

These observations highlight the importance of controlling the temperature parameter in the annealing algorithm. A gradual decrease in temperature allows for a systematic exploration and exploitation process, leading to improved cell placement and HLWP optimization. These findings provide valuable insights into the behavior of the annealing algorithm and its application in optimizing circuit board designs.

Running time of each file using 0.95 cooling rate:

File	Time
d0	0.975968599319458 seconds
d1	1.4970247745513916 seconds
d2	16.890732765197754 seconds
d3	22.049818754196167 seconds
t1	31.6 seconds
t2	1.0679189006487528 minutes
t3	6.02 minutes