CSCE330401 - Digital Design II

Simulated Annealing Project Report

Name: Yehia Elkasas

ID: 900202395

Name: Knzy El Masry

ID: 900202766

Introduction

Annealing is an algorithm that is used in the placement stage. It is used to find an approximate solution to an optimization problem, particularly when the search space is large and complex. It is inspired by thermal annealing. In other words, when the crystal is heated, its atoms move around quickly and it loses its shape, but as it cools down the atoms stop moving gradually creating different shapes. We use this concept in the greedy algorithm to make bigger steps at higher temperatures, and we decrease the number of steps as the temperature decreases. This report will review our code for a simple simulated annealing cell placement tool with results evaluation.

Algorithm

The cells are placed in the grid randomly at first. After that, we perform the simulated annealing process. We define an initial temperature, a final temperature and a cooling rate. The temperatures depend on the wire length and the number of nets. The loop ends when the current temperature becomes less than the final temperature. Furthermore, a number of moves (iterations) is defined in each temperature loop, and it depends on the number of cells to be placed.
In each iteration, 2 random cells are picked and swapped, and a new cost (wire length) is calculated. If the cost decreases, the change is accepted and we continue to the following loop. Otherwise, we calculate the probability of rejecting the change and if the probability exceeds 0.5 we reject the swap, and the swapping is reversed.
This process repeats until we reach the final temperature, and we output the final grid with the total wire length.

Implementation

The implementation will be discussed in the following order:

- Data structure
- Parsing the netlist file

- Random initial placement
- Calculating the cost/wire length
- Simulated annealing algorithm

Data Structure and Definitions

Several data structures are defined to handle various aspects of the problem, such as the netlist, grid, and cell coordinates. Below is a detailed description of each data structure and its role:

- `string test_case_file = "tests/t1.txt";`
    - Specify the path to the netlist file that contains the test case data.
- `double cooling_rate = 0.95;`
    - Defines the cooling rate used in the algorithm to reduce the temperature during the optimization process.
- `unordered_map<int, vector<int>> cellNets;`
    - Maps each cell to the nets it is a part of, helping to track the connections between cells and nets and optimization
- `vector<vector<int>> grid, nets;`
- `vector<pair<int, int>> cells;`
    - Holds the coordinates (x, y) of each cell in the grid, indicating their positions.
- `vector<int> nets_wire_lengths;`
    - Keeps track of the wire lengths for each net, used to calculate the cost associated with the net connections.
- `vector<int> new_wire_lengths;`
    - Purpose: Temporarily stores the new wire lengths during the optimization process to compare against the current wire lengths.
- `int numCells, numNets, numRows, numCols;`
- `vector<pair<int, int>> max_net_coordinates, min_net_coordinates`
    - These vectors track the maximum and minimum x and y net coordinates that are used to calculate the HPWL so that we do not have to recalculate

the max wire length for each net everytime we swap unless the cell swapped affected the HPWL.

Parsing the Netlist File

The readNetlist function is designed to read a netlist file containing information about cells, nets, and the grid dimensions of an electronic circuit. This data is essential for setting up the initial conditions for a simulated annealing algorithm.

The input to the placer is a netlist file with the following format

- The first line contains 4 values:
- The number of cells to be placed.
- The number of connections (nets) between the cells.
- The number of rows
- The number of columns/sites per row

Each of the following lines represents a net and it contains the following:

- The number of components attached to the net
- The list of components attached to the net

An example of a netlist file content is shown below:

3 3 2 2

3 0 1 2

2 2 0

2 1 2

- Line 1: The number of components is 3 and the number of nets is 3. The placement grid is 2x2 (2 rows; each of 2 sites).
- Line 2: The first net connects 3 components: 0, 1 and 2
- Line 3: The second net connects 2 components: 2 and 0
- Line 4: The third net connects 2 components: 1 and 2

Parameters of Parsing Function

- **const string &filename**: The name of the netlist file to be read.

- **int &numCells**: A reference to an integer where the function will store the number of cells.
- **int &numNets**: A reference to an integer where the function will store the number of nets.
- **int &numRows**: A reference to an integer where the function will store the number of rows in the grid.
- **int &numCols**: A reference to an integer where the function will store the number of columns in the grid.
- **vector<vector<int>> &nets**: A reference to a vector of vectors that will store the net-cell connections.
- **unordered_map<int, vector<int>> &cellNets**: A reference to an unordered map that will store the cells and the corresponding nets they are part of.

Detailed Explanation

1. Opening the File:
    - The function begins by attempting to open the file specified by the `filename` parameter.
    - If the file cannot be opened, an error message is displayed, and the function terminates.
2. Reading the First Line:
    - The first line of the file is read to extract the number of cells (`numCells`), the number of nets (`numNets`), and the grid dimensions (`numRows` and `numCols`).
    - The `nets` vector is resized to hold `numNets` vectors, preparing it to store the net-cell connections.
3. Reading the Nets:
    - The function iterates over the number of nets specified.
    - For each net, it reads the number of components (cells) connected to that net.

- It then reads the cell indices connected to the current net, adds these indices to the `nets` vector, and updates the `cellNets` map.
- This process ensures that for each cell, the corresponding nets it is part of are recorded accurately.

4. Closing the File

Random Initial Placement

The initial_placement function is responsible for placing cells randomly within the grid at the start of the optimization process. The function first creates a list of all possible grid indices (row, column pairs) based on the specified number of rows (numRows) and columns (numCols).
A random number generator is initialized using a random seed.
The list of grid indices is shuffled randomly to ensure that the cells are placed in random locations within the grid.
The function then assigns the shuffled grid indices to the cells. Each cell is placed at a unique position in the grid, corresponding to the shuffled index. The grid data structure is updated to reflect the position of each cell, where the cell's ID is stored in the appropriate grid location.

Calculating Cost/Wire Length

The function is used in simulated annealing to compute the wire lengths (or cost) of a given net after a cell swap. This calculation helps decide whether to update the cost and the grid after the random swapping or not.
If the update flag is true, the function checks if the cell being relocated affects the cost, which reduces the number of iterations for optimization.
By comparing the cell's position with the pre-existing bounds (maximum and minimum coordinates), the function determines whether recalculating the cost is necessary.
If the cell is within the bounds and does not affect the cost, the function returns the pre-existing wire length for the net.
Variables min_x, min_y, max_x, and max_y are initialized to extreme values to find the actual boundaries of the net.

The function iterates over all cells in the net to determine the minimum and maximum x and y coordinates.
These coordinates represent the bounding box of the net.
The maximum and minimum coordinates for the net are updated with the newly calculated values.
The wire length (cost) is calculated as the perimeter of the bounding box, given by the sum of the differences between the maximum and minimum x and y coordinates.
The computed wire length is stored in the wire_lengths vector for the net.
The function returns the calculated wire length.

Simulated Annealing Algorithm

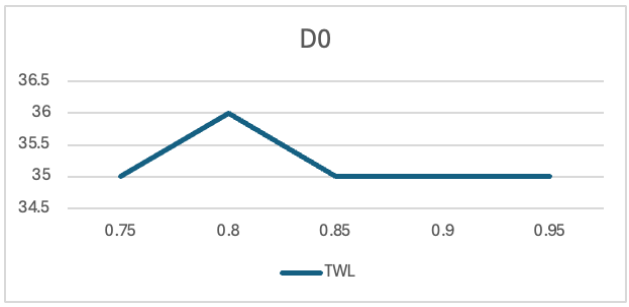
The algorithm is optimized to reduce the run time, especially when it comes to very large grids. Here's how it works:

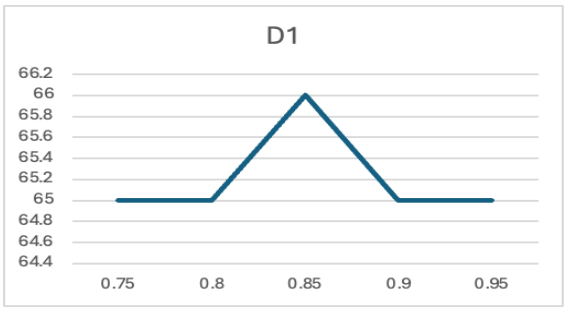
- Initialization: Start with an initial temperature and a current solution.
- Temperature Reduction: Gradually reduce the temperature according to a cooling schedule.
- Iteration: At each temperature, make a number of moves (swaps in this case) to explore the solution space.
- Evaluation: Calculate the change in cost for each move.
- Acceptance: Accept moves that decrease the cost or, with a certain probability, moves that increase the cost. This probability decreases as the temperature decreases, allowing the algorithm to escape local optima.
- Termination: Stop when the temperature falls below a certain threshold.

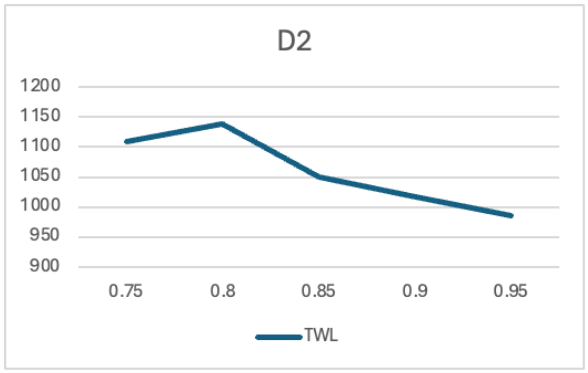
The algorithm iteratively swaps cells and accepts or rejects the swaps based on the change in cost and the current temperature, gradually reducing the temperature until a termination condition is met. For optimization, the cost is only calculated if the coordinates of the cells swapped affect the maximum and minimum x and y values, which are used to calculate the wire length.

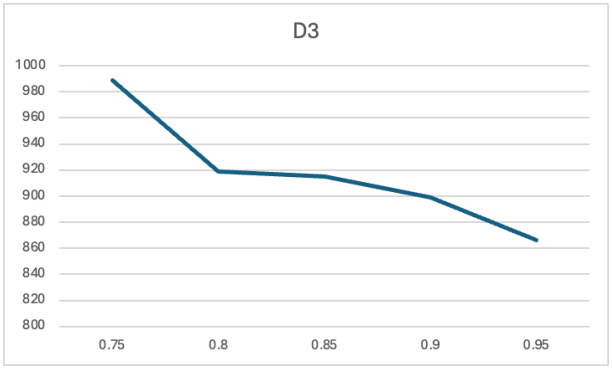
Results and Conclusions

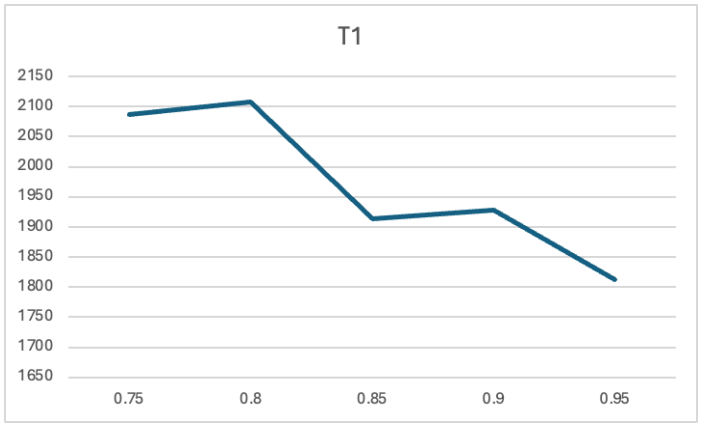
Cooling Rate VS TWL Graphs





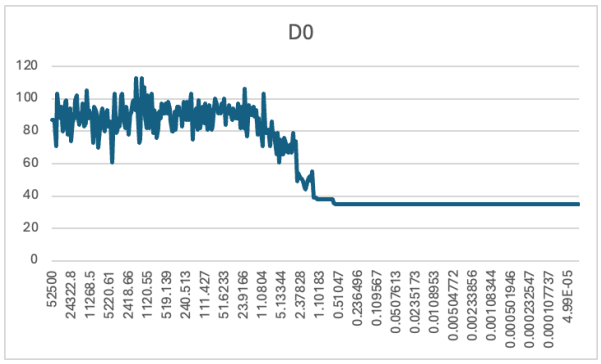


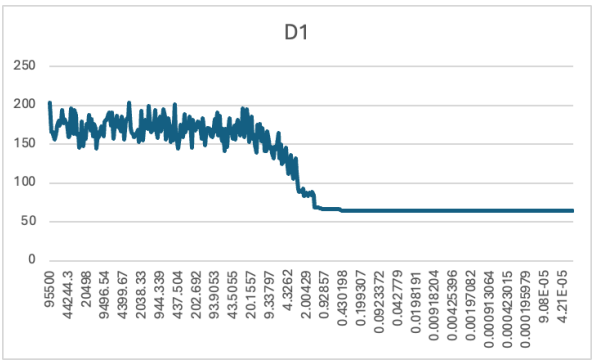


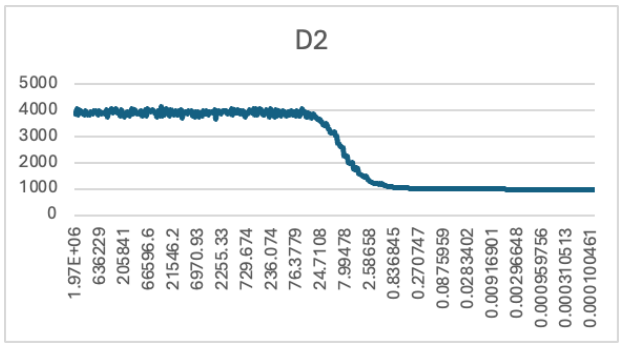


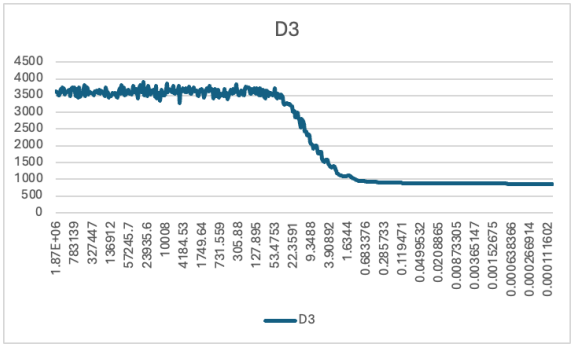
We can conclude from these graphs that the cooling rate affects the wire length as the bigger the rate the smaller the wire that we can derive because smaller steps taken in the algorithm allows more swapping to happen, which shortens the length of the wire.

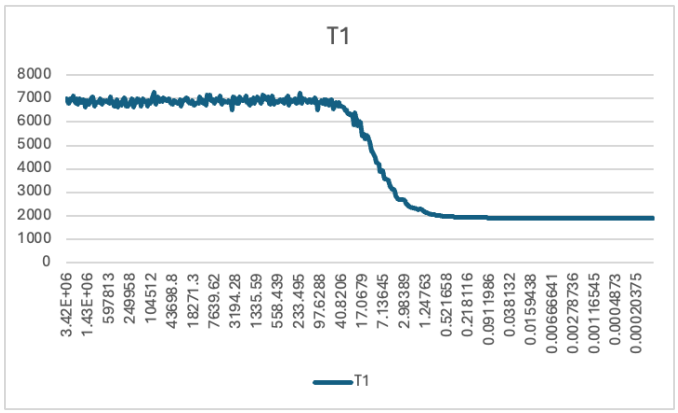
Temperature VS TWL











As we can see from the graphs, there is a pattern, where the TWL seems constant as we increase the temperature, and at some point, it gradually decreases until it becomes constant again, when it reaches its minimum length.