The American University in Cairo

# Simulated-Annealing Cell Placement Tool

CSCE 3304 – Digital Design II

Dr. Mohamed Shalan

Farah Kabesh - 900191706
Karim Sherif - 900191474
Youssef Fares -900192960

Monday 11th December 2023

**Project Summary:**

The aim of this project is to develop a simple simulated annealing-based placer that minimizes the total wirelength as well as study the affects of the cooling rate on the quality of the placement. The code reads an input text file, initializes the random placement of the components and calculates the initial wirelength. Simulated annealing is then performed to optimize the placement, and the final placement and wire length are outputted. The code was written using C++ programming language.

**Example of Input Netlist File:**

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: First net connects 3 components: 0, 1 and 2
• Line 3: Second net connects 2 components: 2 and 0
• Line 4: Third net connects 2 components: 1 and 2

**Simulated Annealing Algorithm:**

- An initial random placement is created.
- Initial temperature= 500 x Initial Wirelength.
- Final temperature = $5x10^{-6}$ x Initial Wirelength / Number of Nets.
- Moves per temperature = 10 x Number of Cells.
- The current wirelength is initialized by calculating the wirelength on the initial placement.
- The current temperature is set to the initial temperature.
- An iteration counter is initialized to 0.
- While the current temperature is greater than the final temperature:
  - For each move in the range of 0 to moves per temperature:
    - Randomly select two locations on the grid to swap places .
    - Perform swap.
    - Calculate new wirelength of the new placement after swap.
    - Calculate delta L which is the difference between the new wirelength and the current wirelength.
    - If dela L is less than 0, then accept swap by updating current wirelength to new wirelength.
    - If delta L is greater than or equal to 0, then calculate the acceptance probability which is equal to $e^{-\text{delta L/Current Temperature}}$
    - Generate a random probability value between 0 and 1. Random numbers are uniformly distributed.
    - If the random probability is less than the acceptance probability, accept the swap by updating the current wirelength to the new wirelength.
    - If random probability not less than acceptance probability, revert the swap and the current wirelength is not updated.
  - Decrease the current temperature using the cooling rate.
  - Increment the iteration counter.
- Return final wirelength.

## Implementation:

### readFile function

This function reads the input text file. It takes in the file name and references to the variables that will store the number of components, nets, rows, columns, as well as components which is a reference to a 2D vector of integers. The outer vector represents all the nets, and each net is represented by an inner vector that contains the indices of the components belonging to that net.

- The *ifstream* class is used to read input from the file.
- A string variable (*line*) is initialized to store each line in the file, and the *getline()* function reads the first line in the file.
- *istringstream* object (*Iss*) is created to treat the string as a stream and extract the values for the number of components, nets, rows and columns.
- The components vector is then resized to have a size equal to the number of nets extracted.
- A loop iterates *number of nets* times to process each line in the file after the first. For each iteration, *getline*() reads a line from the file and stores it in line and an *istringstream* object *Iss2* is initialized with line contents.
- The number of components of the current net is extracted and assigned to *netfileComponents*.
- The inner vector components[i] is resized to have a size of the number of components in the current net. This allocates space to store the component indices for each net.
- Each component in the current net is then iterated over and the component indices are extracted and assigned to *components*[i][j].

### initialPlacement function

This function initializes the placement of components on the grid. It takes in the number of rows, columns, the components vector, the grid vector, (2D vector representing the grid) and *savedPos* vector (2D vector that stores positions of the components in the grid).

- All elements in the grid are assigned a value of -1, which represents that these places are empty.

- Each net in the components vector is iterated over, and each component in each net is iterated over. The current component value being iterated over is assigned to value.

- An instance of the Mersenne Twister pseudo-random number generator *mt19937* is created. The number 2 is provided as a seed value. *mt19937* is used instead of *rand()* since it is a better random number generator and much faster. It also gives distributions such as *uniform_int_distribution* which provides perfectly uniform numbers.

- An if-condition is used to check if the current component (value) has not been saved yet. If the position has not been saved, then it means the component has not been placed yet.

- For each component, row and column values are randomly generated within the grid size until an empty place (-1) is found on the grid. Current component (value) is assigned to empty place found on grid.

- The position of the current component is then saved in the *savedpos* vector. The value of the current component is the index value. The first element of the position is assigned to row, and the second element of the position is assigned to column.

- The process is repeated for each component.

**printPlacement function**

This function prints the placement of the components on the grid. It takes in grid vector as input.

- Each element in the grid vector is iterated over.

- If the component index is equal to -1 (empty place), a dash (-) is printed. If not, then the component is printed.

**calculateWireLength function**

This function calculates the wirelength of the components placement on the grid. It takes the *savedpos* vector and *components* vector as input.

- The wirelength is initialized to 0.

- Each net in the components vector is iterated over. *minY, maxY, minx* and *maxX* are initialized with the maximum possible integer value and minimum possible integer value.

- An inner loop is used to iterate over each component in each net. The value of the current component is used as an index to access the position of the component in the saved pos vector.
- If-statements are used to compare the position of the current component with the minimum and maximum values found and are updated accordingly.
- *minY, maxY, minx* and *maxX* represent the bounding box of each net. The wirelength is obtained by summing the absolute difference between the maximum and minimum x-coordinates as well as the absolute difference between the maximum and minimum y-coordinates.

**swap function**

This function takes in the grid vector, the *savedpos* vector, the row and column indices of the two locations to be swapped. If-statements are used to check the elements being swapped. If they are not empty places (-1), then the new positions of the component values are updated in the *savedpos* vector.

**simulatedAnnealing function**

This function implements the simulated annealing algorithm to optimize the placement of components on the grid. It takes the grid vector, the initial temperature, the final temperature, and the number of moves per temperature as input.

- The current wirelength is initialized. The current temperature is initialized to the initial temperature. An iteration counter is initialized and set to 0.
- A loop is entered till the current temperature reaches the final temperature. Within each iteration, the function performs a number of moves determined by *movesPerTemp*.
- In each move, two places on the grid are randomly selected for their components to be swapped.
- The new wirelength is calculated after the swap. The difference between the current wirelength and new wirelength is calculated (delta L). If difference is less than zero, then the swap is accepted and the current wirelength is updated. .

- A random probability value (between 0 and 1) is randomly generated, and the acceptance probability is initialized to $e^{-\text{delta L/Current Temperature}}$

- If the difference is not less than 0 but the random probability value is lower than the acceptance probability, then the swap is accepted and the current wirelength is updated.

- If random probability value is not lower than acceptance probability, then swap is rejected and components are swapped back.

- The current temperature is reduced by a cooling rate of 0.95 and the iteration counter is incremented.

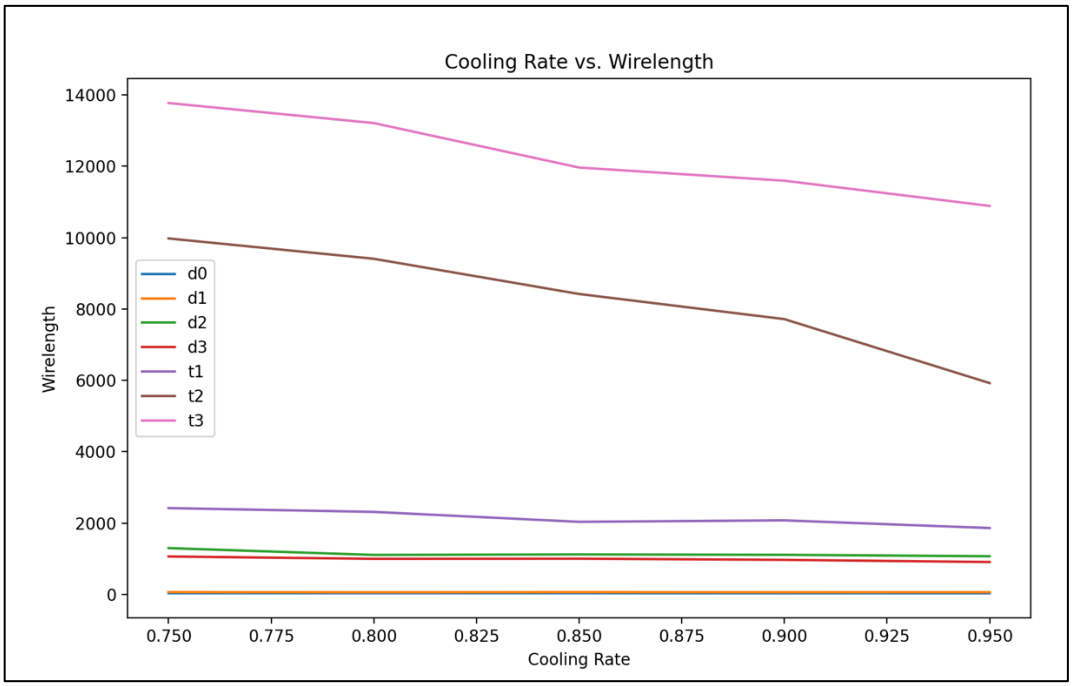
**printBinaryFormat function**

This function prints the placement of the components on the grid in binary format. It takes in grid vector as input.

- Each element in the grid vector is iterated over. If the component index is equal to -1 (empty place), a '1' is printed. If not, then '0' is printed to represent an occupied cell.

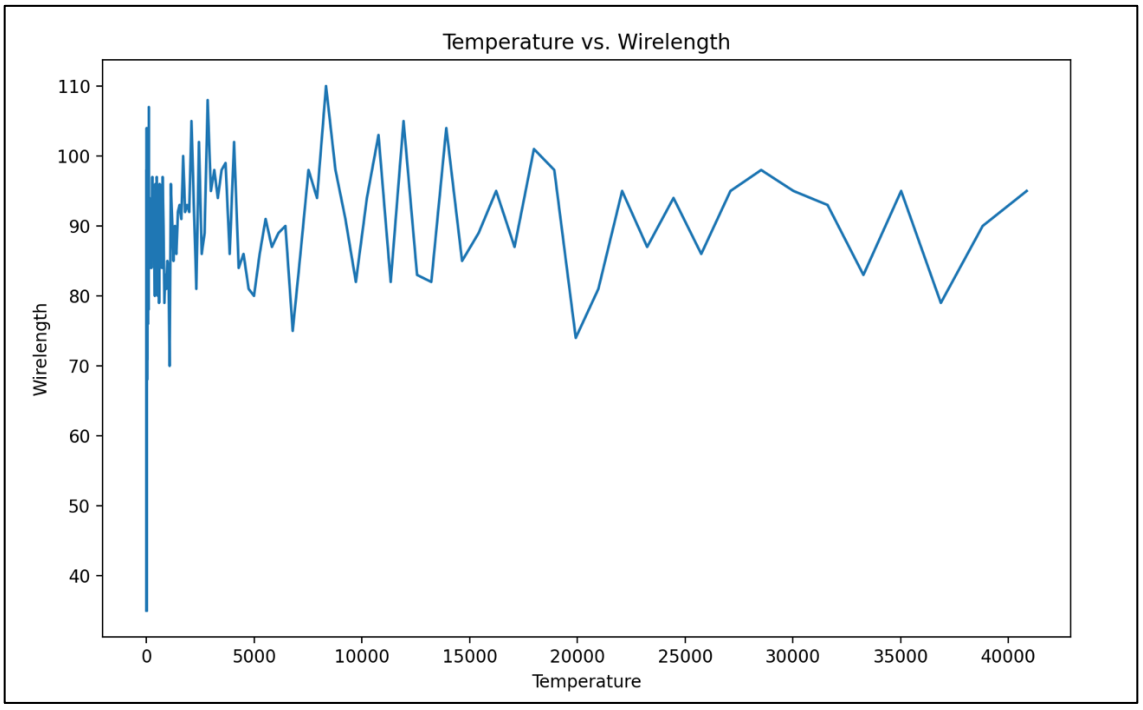
## Cooling Rates vs Total Wirelength (TWL) Graphs

The Cooling Rates used are: 0.75, 0.8, 0.85, 0.9, and 0.95.

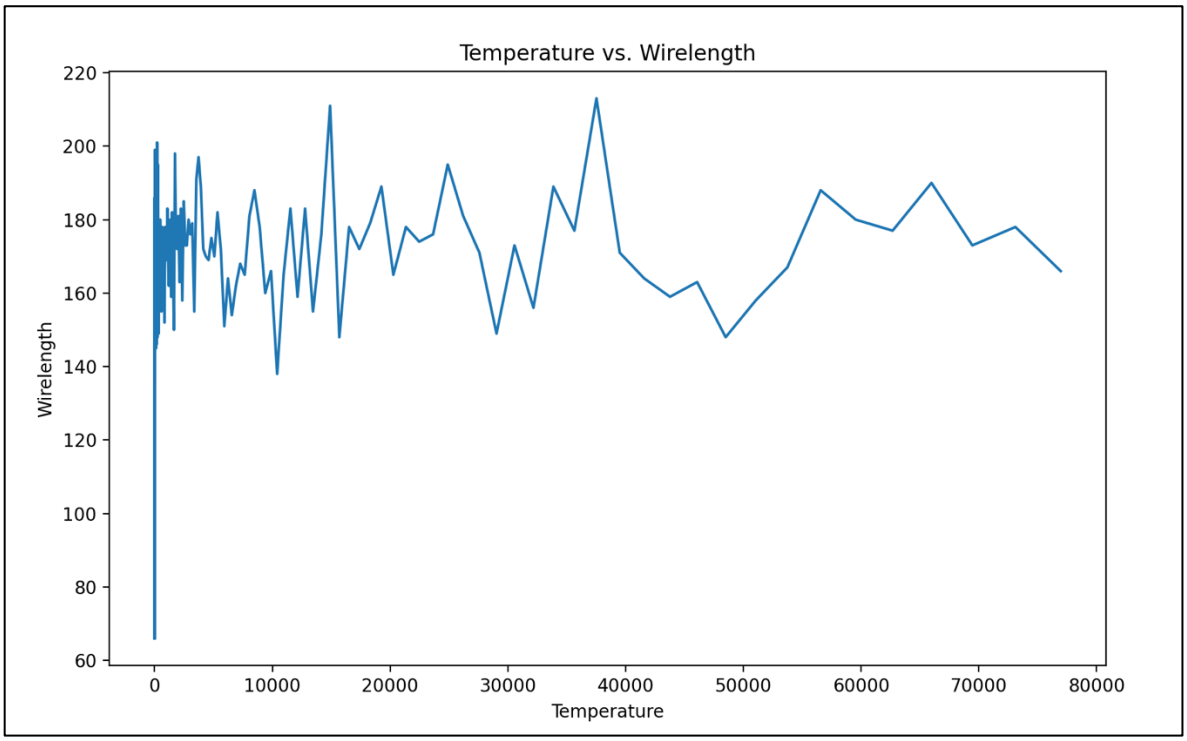



## Temperatures vs Total Wirelength (TWL) Graphs

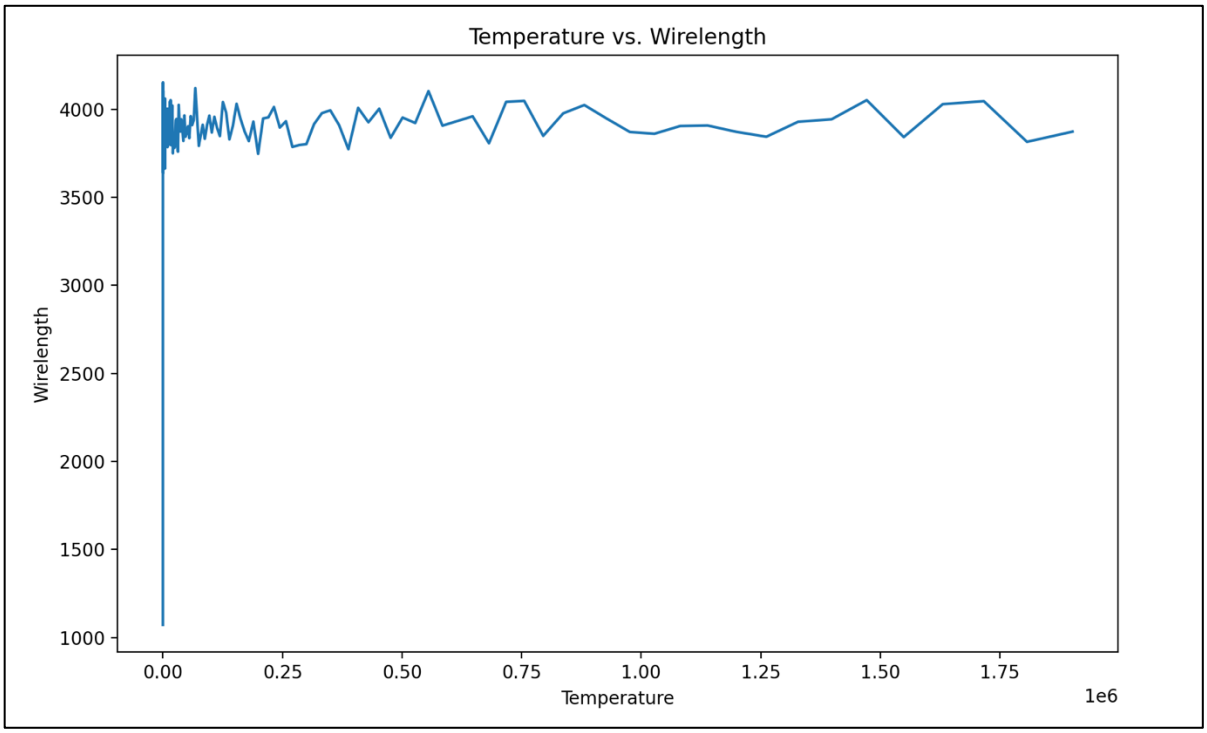
### Design d0



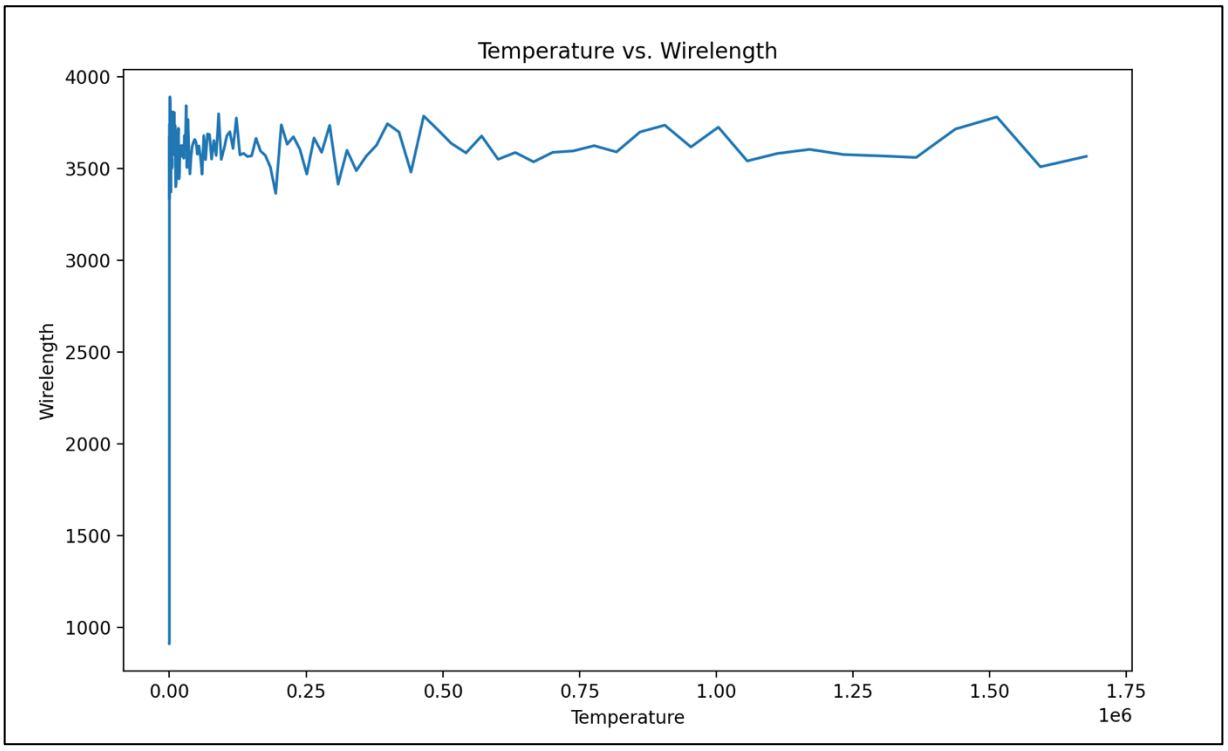
**Design d1**



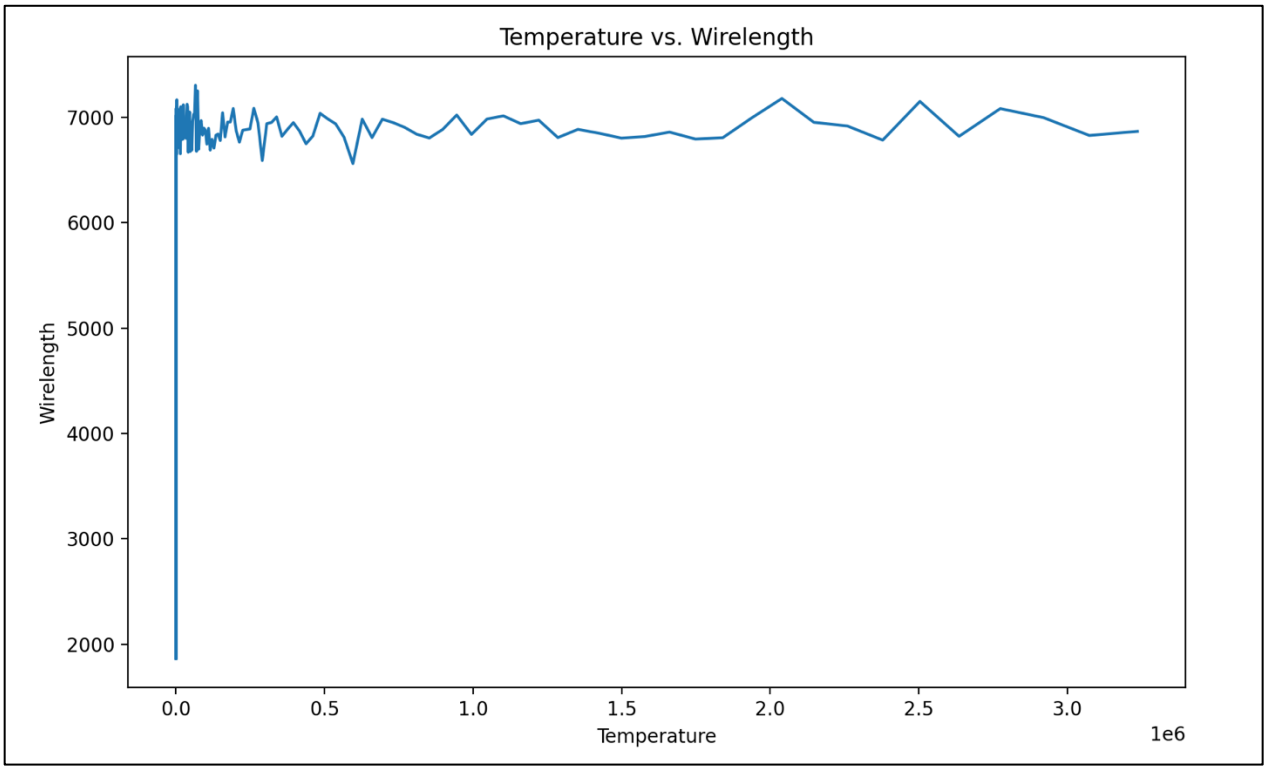

**Design d2**



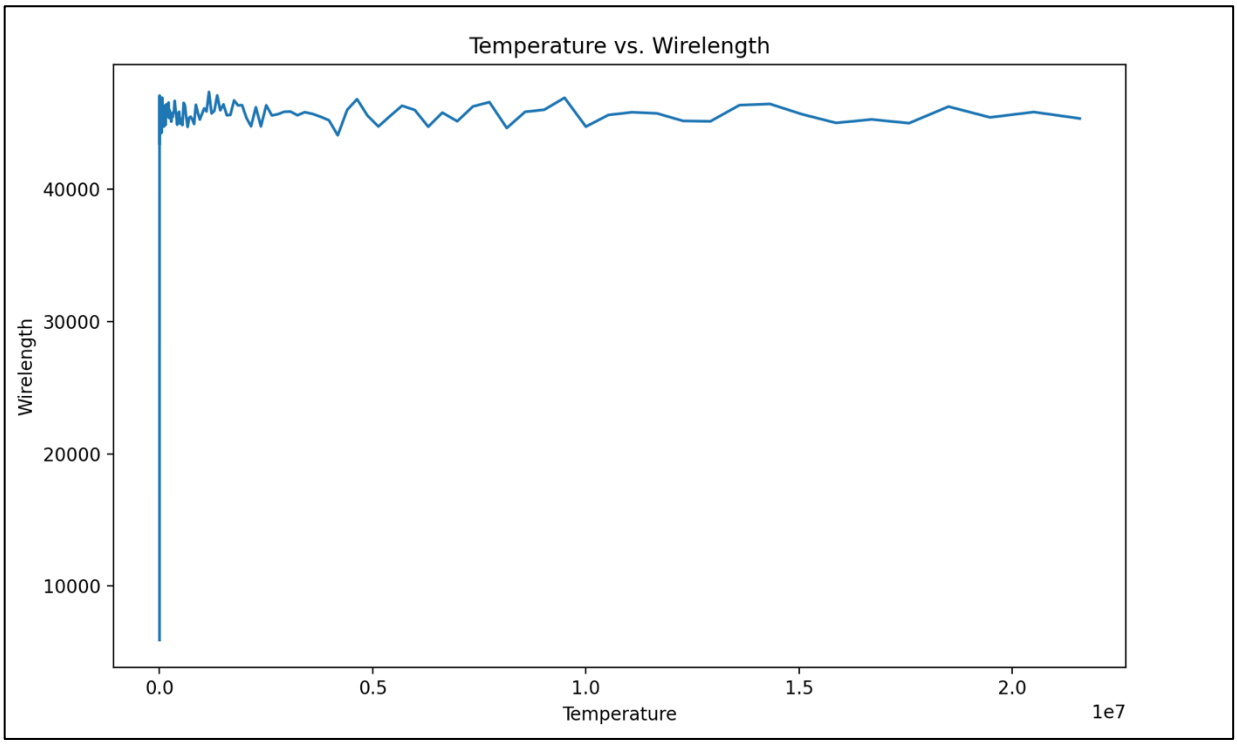
**Design d3**



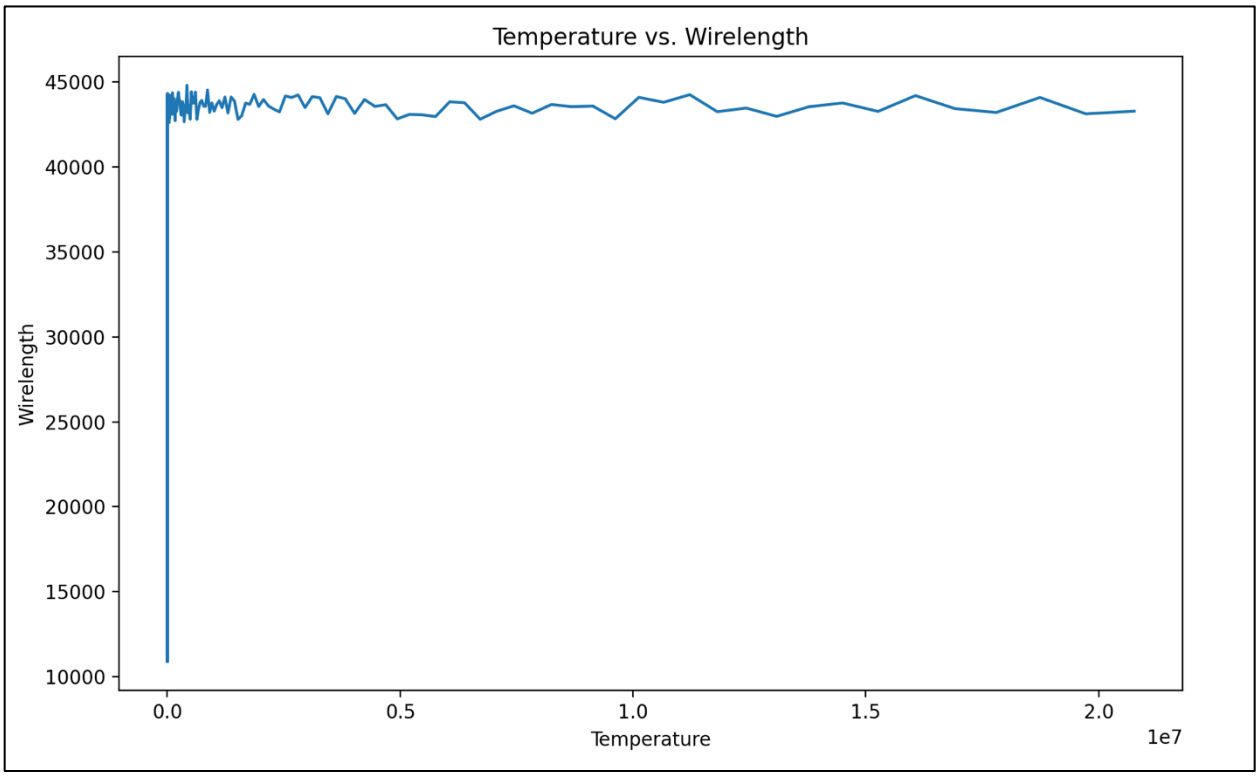

**Design t1**



**Design t2**




**Design t3**



## Conclusions

The cooling rate affects the speed of convergence and how quickly the temperature decreases during the simulated annealing process. A lower cooling rate means that the temperature decreases more quickly and the number of iterations is less (faster convergence). However, this might result in suboptimal final wirelength values, since less exploration takes place.

On the other hand, a higher cooling rate means that the temperature decreases more slowly and the number of iterations is more (slower convergence). A higher cooling rate can lead to more optimal final wirelength values to be obtained, since this allows more exploration of the optimal placements of the components in the grid.

The graph below (design t3) shows the number of iterations is plotted against the temperature. As observed, the Simulated Annealing Algorithm is correctly represented. Initially, a high temperature is used in order to allow the exploration of the possible moves that may not improve the final cost (wirelength). However as the number of iterations increases, the temperature decreases and gradually converges towards an optimal cost (wirelength) while avoiding local minima.