THE AMERICAN UNIVERSITY IN CAIRO

الجامعة الأمريكية بالقاهرة

Simple Simulated-Annealing Placement Tool

CSCE3304 - Digital Design 2 (2022 Fall)

Mohamed Nassr
Omar Miniesy

Table of Contents

Introduction

The goal of this project is to minimize the total wire length when placing nets on a circuit. Due to the fact that the placement process needs routing, we used half perimeter wire length(HPWL) to try and estimate the wire length. The (HPWL) is a rectangle drawn that encloses all the cells, it's calculated by taking the sum of the difference between the minimum and the maximum of x and y coordinates (delta x + delta y). Initially, the cells of the circuit are randomly placed and the total wire length is calculated based on that randomly initialized placement.

A proposed algorithm can iterate however amount of times and swap between components, however only accepting swaps that would lead the total wire length to decrease. This proposed algorithm is described as a greedy algorithm, which causes us to reach a local minimum very quickly.

Simulated annealing provides a solution to this problem by allowing the algorithm to initially accept swaps which would increase the total wire length, and then after a certain amount of iterations only accept swaps that would decrease the total wire length. This results in an algorithm that can help discover other local minimums in this optimization problem, increasing the probability of finding the actual global minimum.

Algorithm

Simulated Annealing implements a temperature that follows a cooling schedule, such that the temperature is a function of the probability to accept bad placements. As the temperature decreases the probability of accepting bad swaps decreases.

```
Create an initial random placement
T = T_init   // Very high temp
while(T > T_final)
    Pick 2 random cells and swap them
    calculate the change in WL (ΔL) due to the swap
    if (ΔL < 0) then accept
    else reject with probability (1 - e^(-ΔL/T))
    T = schedule_temp()
```

(The above pseudo-code was taken from Dr. Shalan's "Placement" lecture slides)

This is the pseudocode of the algorithm that was implemented in this project.

Initially, we start with a temperature which is based on the initial total wire length. The final temperature is also calculated based on the initial total wire length and the number of nets.

We then randomly swap 2 cells, if the total wire length decreases we accept this swap. However if the total wire length increases we reject this swap with a probability of (1-e^(-(New HPWL - Old HPWL))/T) where T= current temperature. After a certain number of swaps, we cool down the temperature by a certain factor.

This process is repeated until the current temperature is less than or equal to the final temperature.

Implementation

We first parse through the provided text file, obtaining the grid size in which the cells are to be placed, the cells and the nets.

The nets provided in the text files are then stored in a list of lists, where each element in the list is a net. In our implementation we called that data structure new_lines.

Four other important data structures;

- Cells, which contains all the cells to be placed on the grid
- Dict, which contains the x and y coordinates of all the cells
- Dict_indx, which contains the indices of where each cell lies in the netlist
- Hpl_list, contains the HPL length of each netlist
- Grid, the grid that contains where all cells are placed

We then randomly generate a unique (x,y) for each cell, storing these values in Dict.

We implemented a function called **hpl_list_init,** which initials the hpl_list with the length of netlist.

To randomly pick a cell or an empty space to swap with we simply generate two x's and two y's. Where Grid[x1][y1] or Grid[x2][y2] gives us a random cell or empty space. Note: The program never swaps with two empty cells.

We implemented a function called **hpl_list_init,** which initializes the Hpl_list with the length of netlist.

We implemented a function called **calculate_new_length_one_cell,** which takes the Dict_indx, Hpl_list, and Dict. The function only re-calculates the wire length of the nets that contain the swapped cells; in other words only a subset of Hpl_list is updated when cells are swapped. If the new HPWL is accepted, Dict and Hpl_list are both updated, if not they are kept the same.

Probability implementation:

After calculating the probability of rejecting a bad swap, we then generate a number between 1-100, if the generated number/100 is greater than the probability of rejection we accept the bad swap.

Cooling schedule implemented:

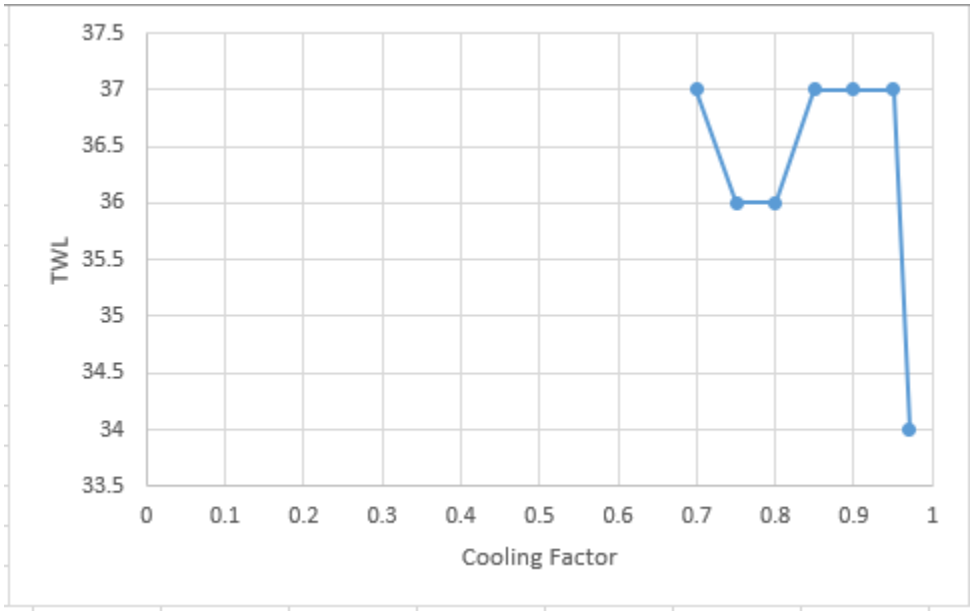
Initial Temperature = 500 x Initial Cost
Final Temperature = 5x10-6 x (Initial Cost) / (Number of Nets)
Next Temperature = 0.95 x Current Temperature
Moves/Temperature = 10 x (Number of cells)

Given the above data structures, functions and cooling schedule implementation, probability implementation we were able to implement the pseudocode of the discussed algorithm.

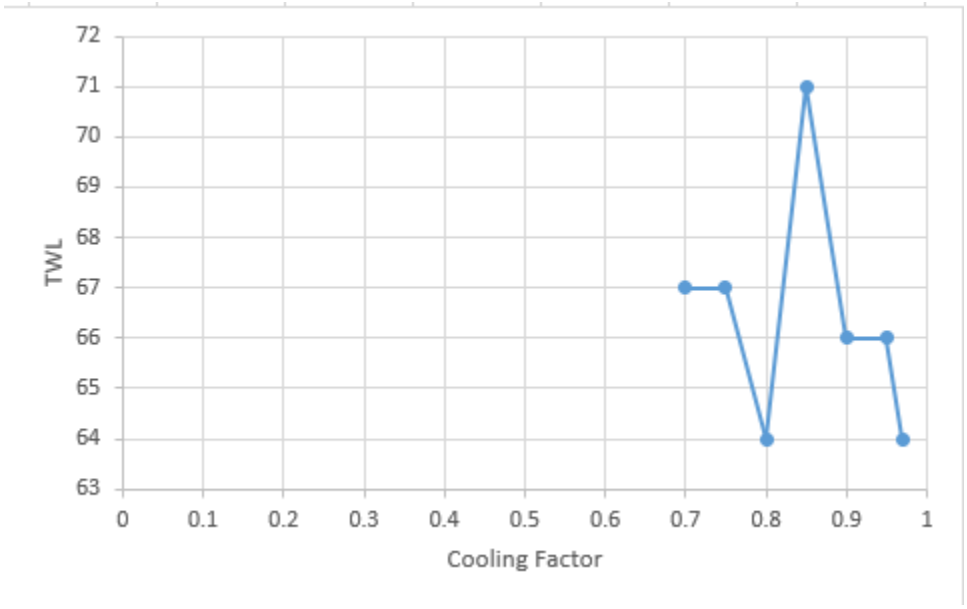
Experimental results

Cooling Rate vs TWL:

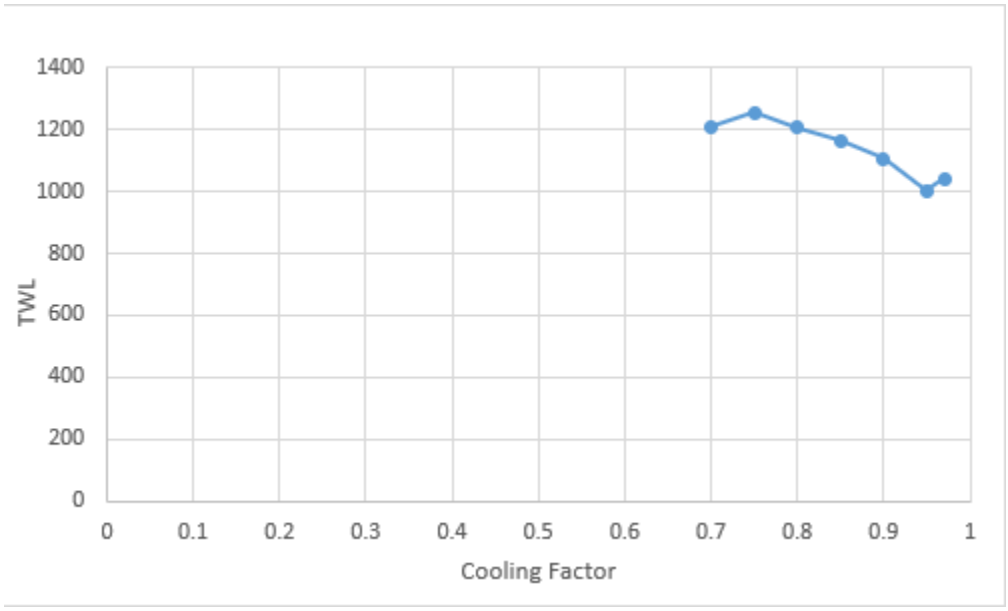
D0.txt



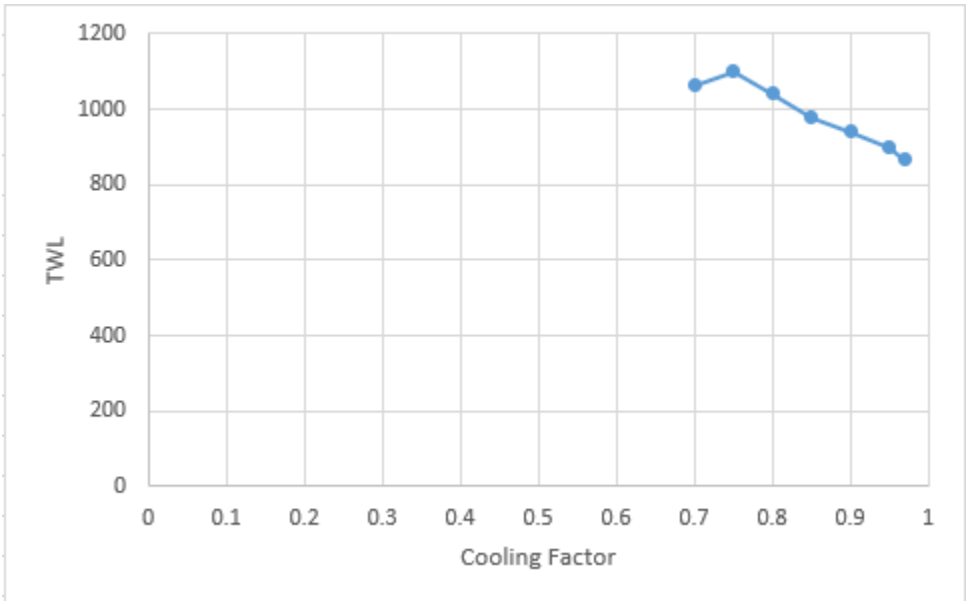
D1.txt


72
71
70
69
68
67
66
65
64
63
TWL
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Cooling Factor

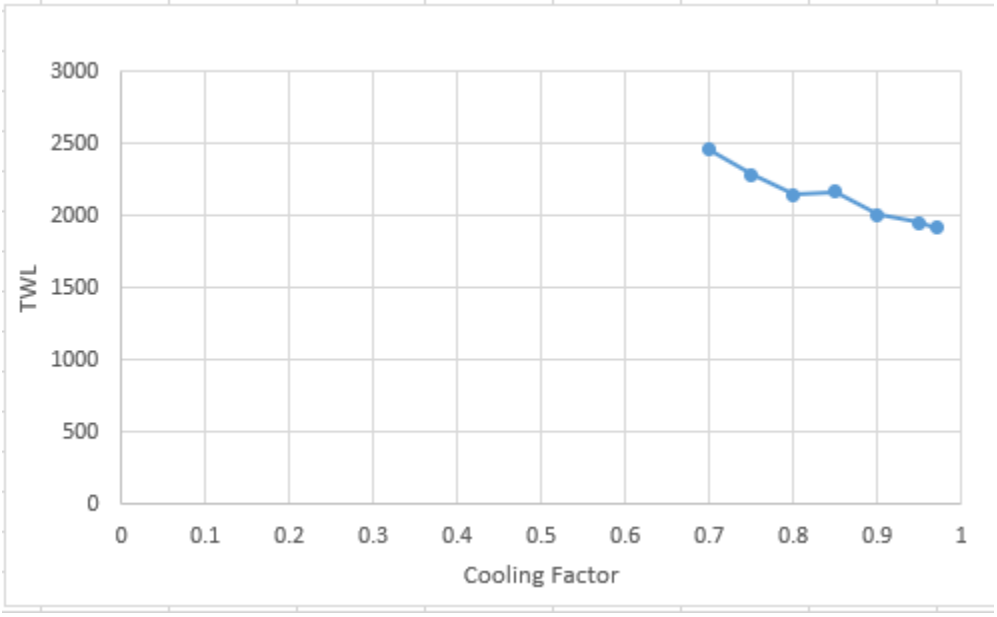
D2.txt


1400
1200
1000
800
600
400
200
0
TWL
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Cooling Factor

D3.txt

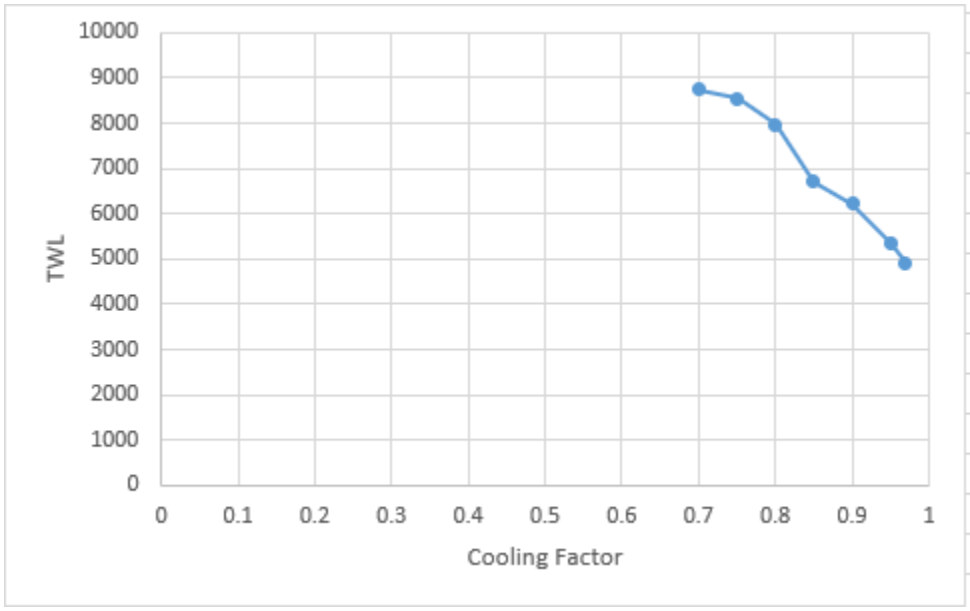

1200
1000
800
600
400
200
0
TWL
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Cooling Factor

T1.txt

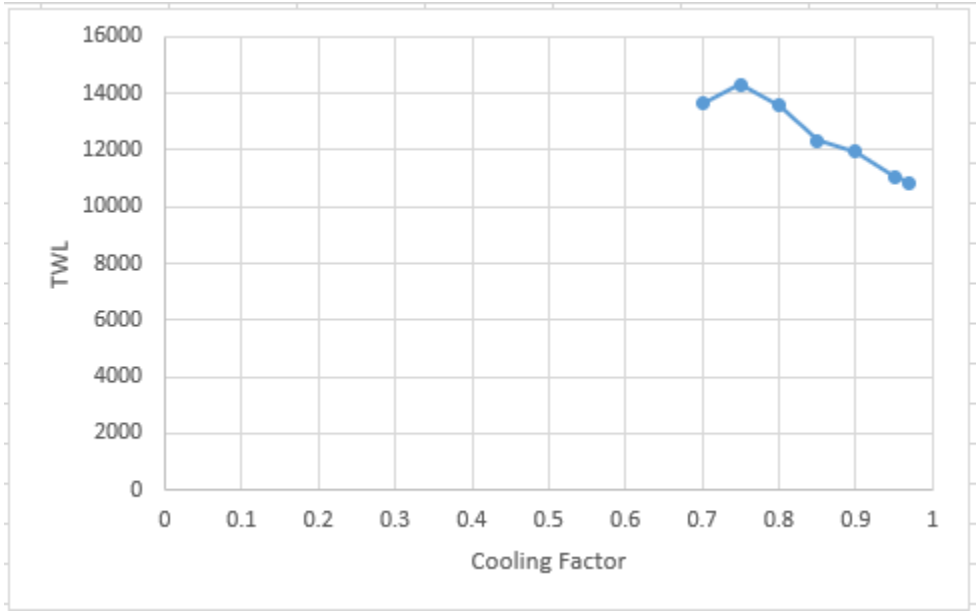

3000
2500
2000
1500
1000
500
0
TWL
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Cooling Factor

‘

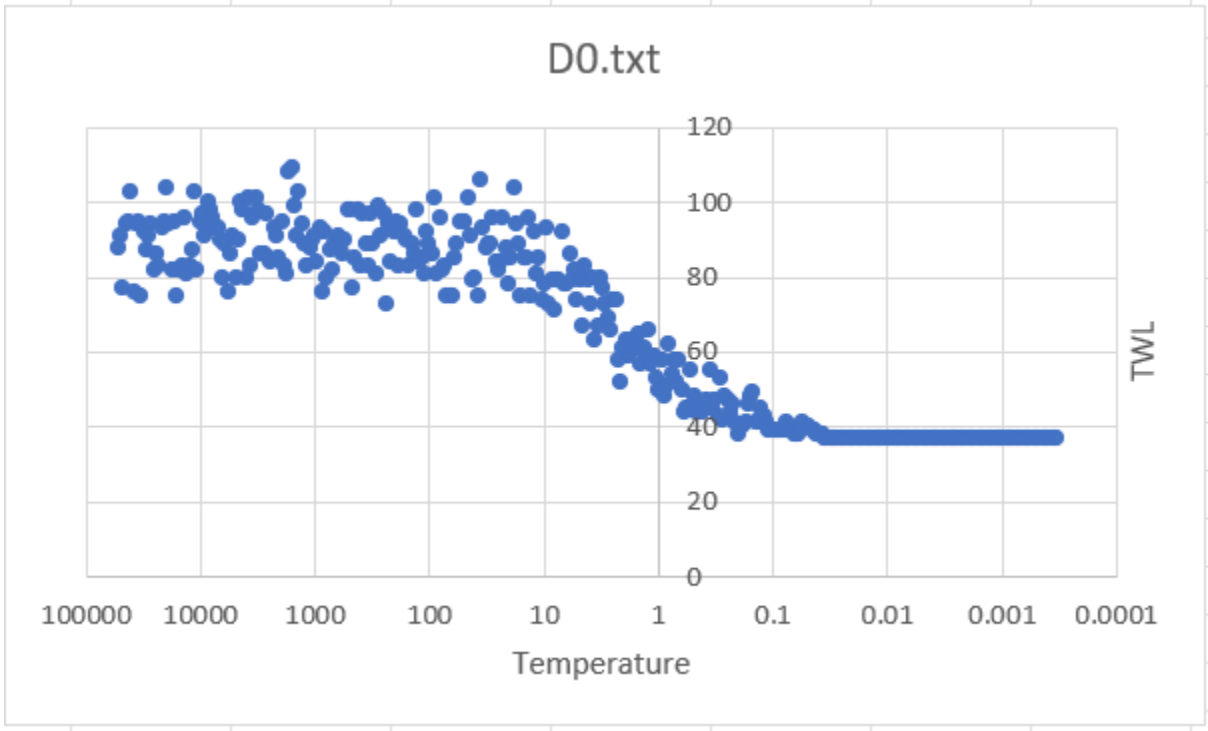
T2.txt

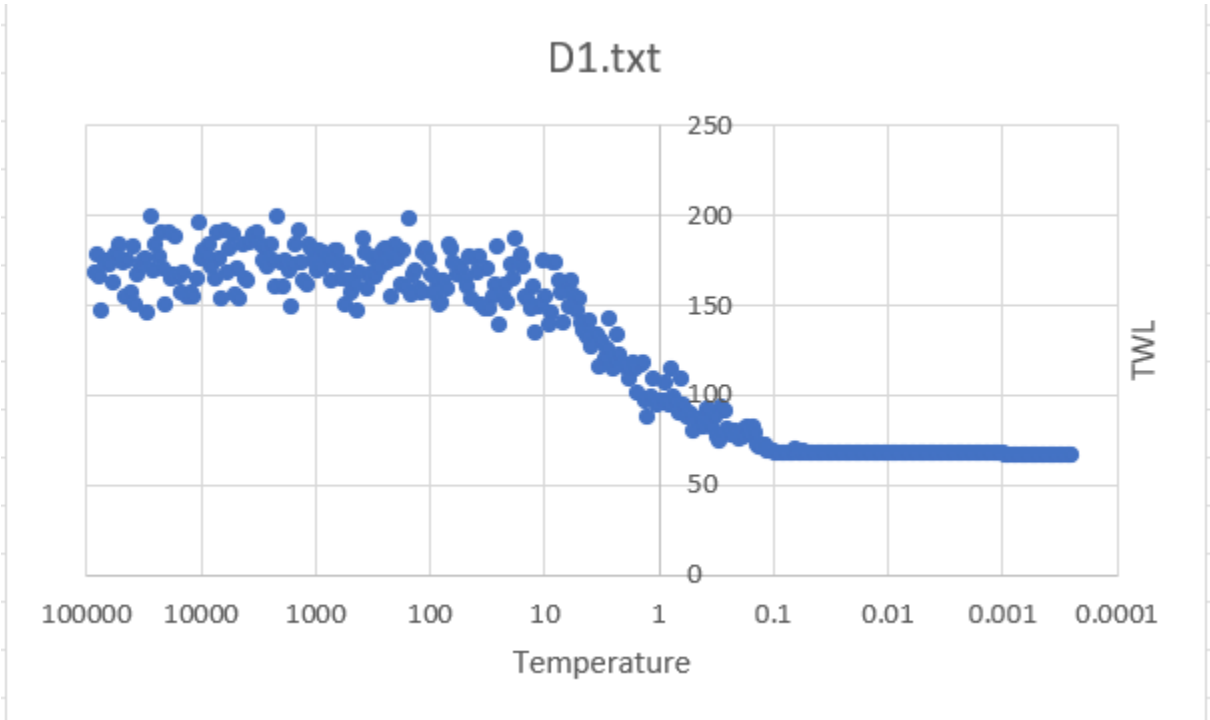

10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
TWL
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Cooling Factor

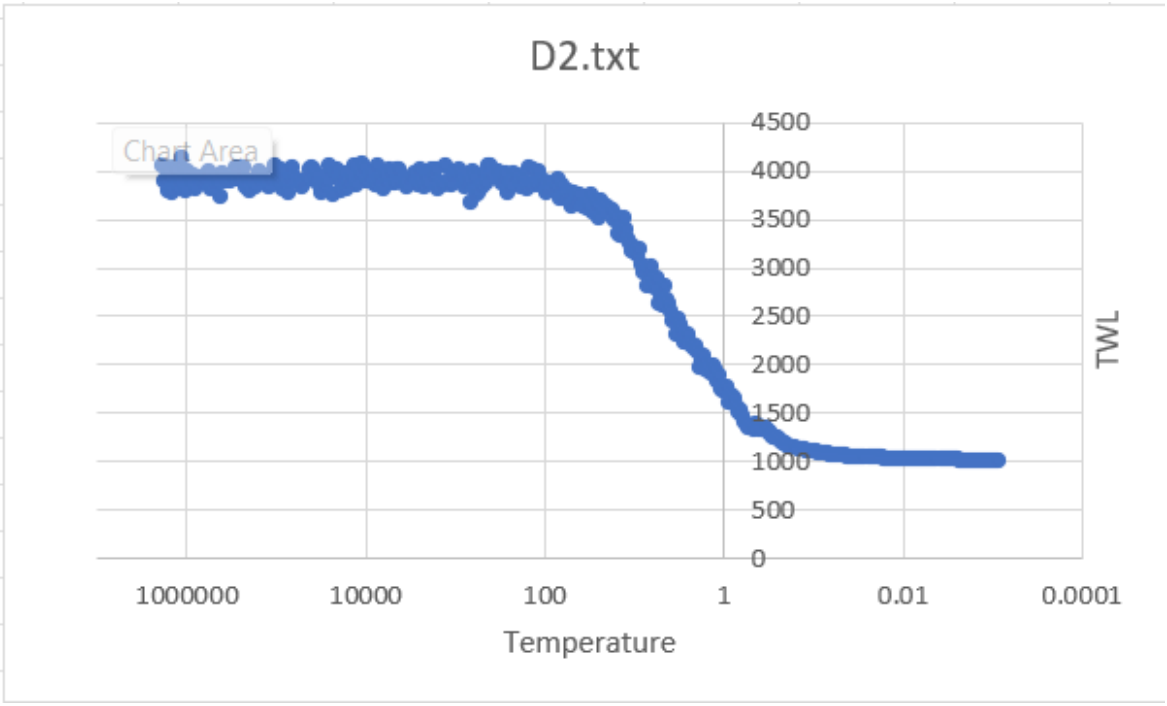
T3.txt

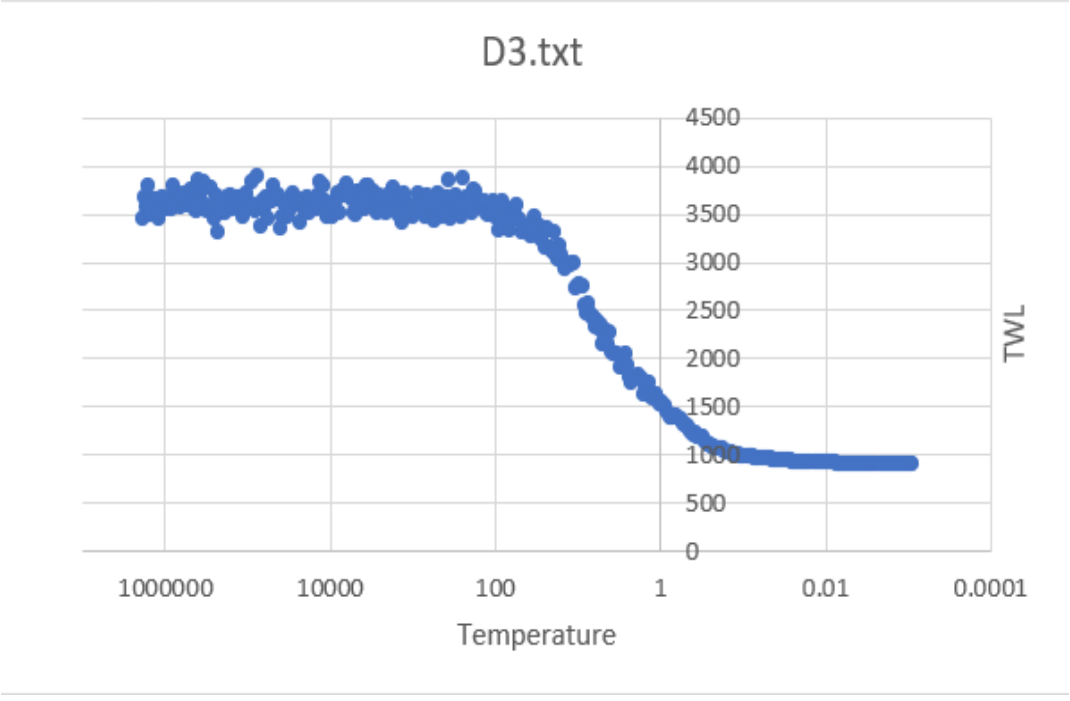

16000
14000
12000
10000
8000
6000
4000
2000
0
TWL
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Cooling Factor

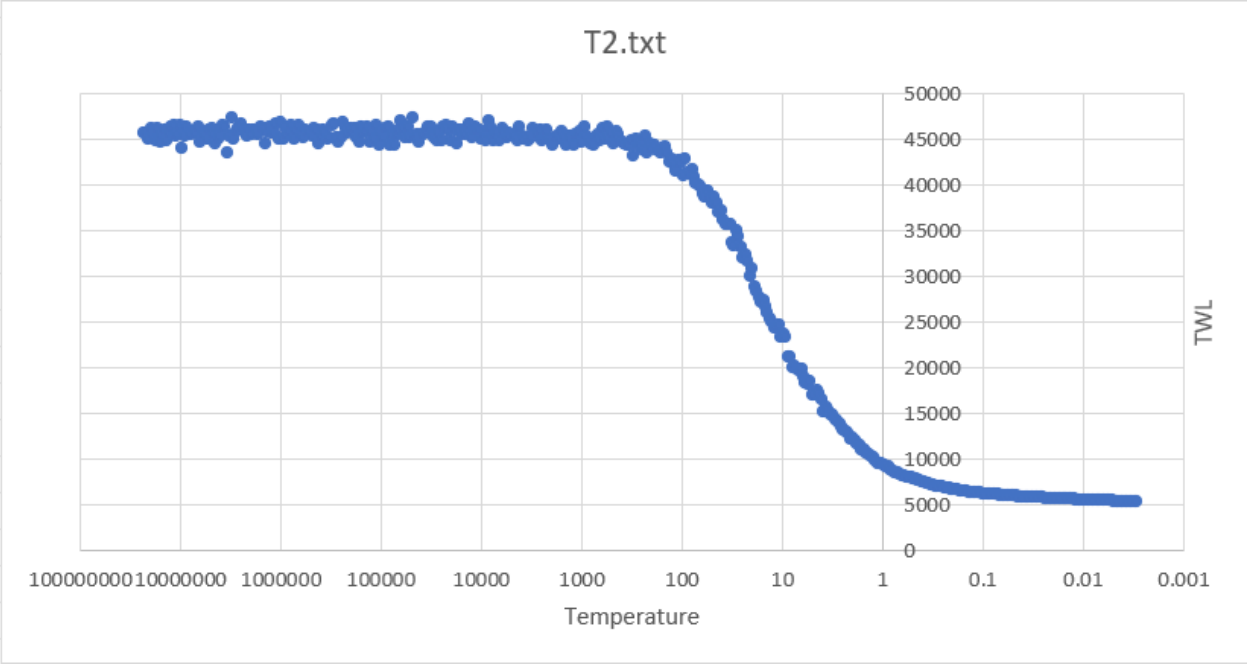
Temperature VS TWL:

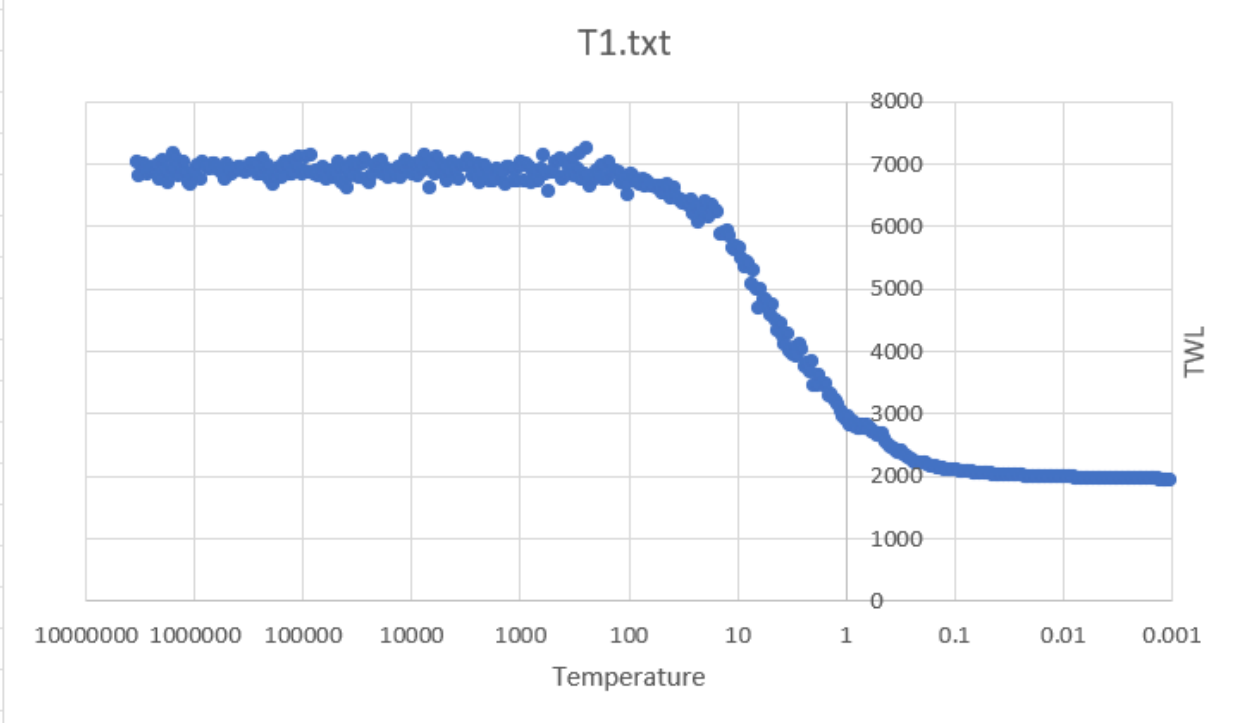


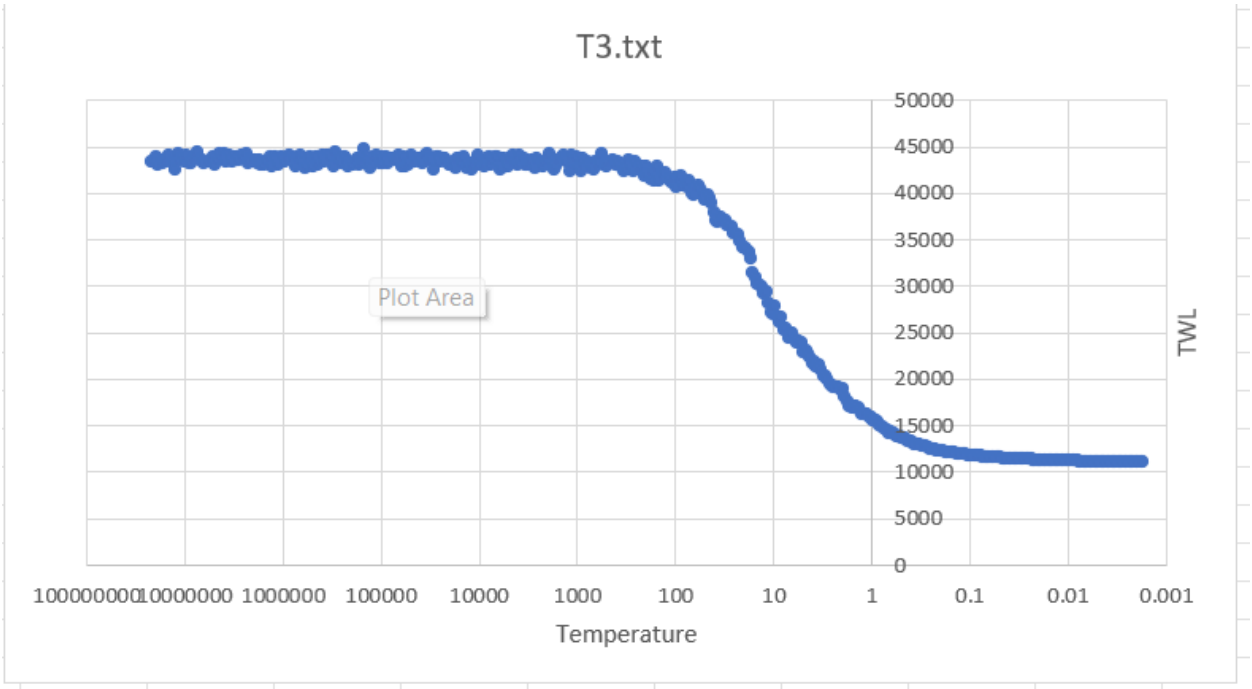



D2.txt
Chart Area
4500
4000
3500
3000
2500
2000
1500
1000
500
0
TWL
1000000
10000
100
1
0.01
0.0001
Temperature


D3.txt
4500
4000
3500
3000
2500
2000
1500
1000
500
0
TWL
1000000
10000
100
1
0.01
0.0001
Temperature


T2.txt
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
TWL
100000000 10000000 1000000 100000 10000 1000 100 10 1 0.1 0.01 0.001
Temperature


T1.txt
8000
7000
6000
5000
4000
3000
2000
1000
0
TWL
10000000 1000000 100000 10000 1000 100 10 1 0.1 0.01 0.001
Temperature



Based on these results, we see that increasing the cooling rate decreases the final wire length. Also the TWL decreases as the temperature decreases.