



**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
- Algorithm
- Implementation
- Experimental results

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 = Tinit // Very high temp
while(T > Tfinal)
    Pick 2 random cells and swap them
    calculate the change in WL ( $\Delta L$ ) due to the swap
    if ( $\Delta L < 0$ ) then accept
    else reject with probability  $(1 - e^{-\Delta L/T})$ 
    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 $(1 - e^{-(\text{New HPWL} - \text{Old HPWL})/T})$ 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](#).

Two 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

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

We implemented a function **calculate_total_length** which calculates the total wire length (HPWL) using [Dict](#) and [new_lines](#). We also implemented a function **swap** which randomly swaps between two cells and returns whether that swap would lead to a smaller or larger total wire length, if a swap is accepted the x and y coordinates in the [Dict](#) are updated, if not the [Dict](#) is 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 = $5 \times 10^{-6} \times (\text{Initial Cost}) / (\text{Number of Nets})$

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

Due to “d2.txt” large number of cells, and nets. We decided to use it when experimenting with the cooling rate. This gives a good estimation towards correctness and completeness of results.

On average, the initial placement of the cells leads to total wire length between 3800-4100. Therefore the 3950 will be used as the average initial total wire length. We ran each cooling rate 3 times and calculated the average final wire length. Moreover, the difference between the average initial wire length and the average final wire length is calculated.

Cooling rate	0.95	0.65	0.35	0.1
Final average wire length	$(1195 + 1144 + 1164) / 3 = 1168$	$(1335 + 1345 + 1299) / 3 = 1326$	$(1428 + 1523 + 1521) / 3 = 1491$	$(1672 + 1652 + 1712) / 3 = 1679$
Average difference between initial and final wire length	$3950 - 1168 = 2782$	$3950 - 1326 = 2624$	$3950 - 1491 = 2459$	$3950 - 1679 = 2271$

Based on these results, we see that increasing the cooling rate decreases the final average wire length, but increases the difference between the initial wire length and the final wire length. This is due to the fact that increasing the cooling rate means iterating through the algorithm more times, which leads to performing more swaps. Hence, increasing the probability of finding the global minimum, which results in shorter wire length solving the optimization problem.