Simple Simulated-Annealing Cell Placement Tool

Digital Design II

Omar Fayed 900191831

Youssef Abouelenin 900192509

Table of Content

Objectives

- Develop a simple simulated annealing-based placer that minimizes the total wirelength.
- Understand the effect of cooling rate on the quality of the placement.
- Enhance the half-perimeter wire length by obtaining the smallest bounding box for every net.

Introduction

Cell placement is a very important step in the production of integrated circuits. It is the third step in the physical design procedure, following floor-planning and power planning. It is very significant because it impacts the quality of routing. Improving the placement of cells minimizes the total wire length, decreases the routing congestion, and decreases the overall delay. In this project, we will improve the placement by using a simulated annealing placer tool. The tool does not provide an optimal solution, but it is good at avoiding local minima.

Input

The program takes an input .txt file with the first line consisting of 4 integers (number of cells, number of connections or nets, number of rows on the grid, number of columns on the grid). Then each line in the .txt file represents a net or a connection with the first integer in the line represents the number of cells in the net followed by the cells connected in the net.

Example:

```
3 3 2 2
3 0 1 2
 2 2 0
 2 1 2
```

Number of cells = 3

Number of nets = 3

Number of rows = 2

Number of columns = 2

Algorithm

In the simulated annealing algorithm, we first perform an initial random placement of the cells. Then we set the temperature at a very high value, so we can avoid being trapped in a local minima. The temperature is responsible for the rejection probability of swapping two random cells. After setting the temperature, we start looping until the temperature goes lower than the final temperature. In the loop, we pick two random cells and swap them in each iteration. Then, we calculate the change in the total wire length. In our case, we used the half-perimeter wire length (HPWL) estimation to obtain the total wire length in our grid. If the wire length decreases, we accept this swap and proceed; otherwise, we reject the swap with probability (1 - e^(delta(L)/T)) and we proceed in looping.

The cooling schedule is the method which reduces the temperature parameter. If the temperature parameter drops quickly, this will have a big impact on the quality of the solution. Also, dropping the temperature slowly would result in a longer computing time, thus the cooling schedule is vital for the quality of the solution.

The following is the cooling schedule used in this project:

- 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)

Functions

1. Hpwl(): used to obtain the half-perimeter wire length of the connection by using minimum and maximum values of rows and columns. The wire length is calculated with a regular hpwl calculation rule.
2. Swap(): used to swap the values of two cells. The function takes two locations of cells and swaps them then updates their locations in the grid.
3. Anneal(): initially the program sets the placement of each cell on the grid randomly. Using the algorithm of the simple annealing it uses the swap function and the rest of the functions to alter the placement of the cells to reduce the total wire length. then finds the wire length with the least cost.
4. Updatexyval(): The function updates the new placement of each cell in the grid. The 2d vector xyval stores each cell with its x and y coordinates on the grid to calculate the HPWL of the net. After each swap done this function is called to update the coordinates of the cells according to the new placements.
5. MaxHpwl(): It gets the maximum and minimum x and y cell placement of each net and then calls the function HPWL to calculate the HPWL of each net then sums them.
6. Print(): prints the grid with the corresponding total wire length.

Results Observation

The output changes slightly in each run as the code runs for a specific number of times until it reaches the final temperature using the cooling effect and also as we swap randomly so the slight change is predicted. The running time increases as we increase the number of cells as the moves per temperature is = 10 * number of cells so execution time is directly proportional to the number of cells.