Foundations of VLSI CAD - Course Project

Optimal 2D placement using Simulated Annealing

Report by Khushhall Chandra Mahajan (13D070064)

Objective - We want to place k circuit elements on a board. These k elements are connected to each other on a circuit board of grid N*N. Longer wirelength introduce delays in the system. Hence, we want to place the elements such that total wire used for interconnection is minimum.

Neighbour States:
In the algorithm for Simulated Annealing, at each iteration, we have to find a neighbouring state to our current state, in order to move towards the optimum as the system cools down. The neighbours of a state are new states of the problem that are produced after altering a given state in some well-defined way. These moves usually result in minimal alterations of the last state, as the previous example depicts, in order to help the algorithm keep the better parts of the solution and change only the worse parts. In the traveling salesman problem, the parts of the solution are the city connections. Our aim is to reach a state that has no better neighbour states. Metaheuristics use the neighbours of a solution as a way to explore the solutions space, and although they prefer better neighbours, they also accept worse neighbours in order to avoid getting stuck in local optima. If the algorithm were run for an infinite amount of time, the global optimum would be found.
In our code, we find the neighbour states probabilistically. That is, with 50% likelihood, we swap the positions of two existing positions, otherwise we shift one element to a new unoccupied position on the grid.

In the annealing function-
We implement the fundamental iterative function that tries to reduce its cost function every time and also, updates the temperature, to implement slow cooling. If the cost of the neighbouring state is better (lower), we move to it, but if it isn't, there still exists a chance that we will move to it, but with reducing likelihood as the temperature decreases. This is how simulated annealing ensures that the global optimum will be achieved given infinite time, and that the algorithm will not get stuck in a global minimum.

Future Work -
Testing the code out for bigger graphs and with more element nodes and plotting the results.