The American University in Cairo

A Simple Simulated-Annealing Cell Placement Tool

Fall 2022

Digital Design II

Dr. Mohamed Shalan

Renad ElKady 900191378

Alaa AlKady 900191122

Table of Contents:

1. *Introduction*

Simulated Annealing is a stochastic global search optimization algorithm, used for resolving both bound, and unconstrained optimization issues. The technique mimics the physical procedure of raising a material's temperature and then gradually decreasing it to reduce flaws while conserving as little energy as possible in the process. The simulated annealing process generates a new point at random points, after each repetition. Based on a probability distribution scale, proportional to the temperature, the new point's separation from the present point, or the

scope of the search is determined. All new points that reduce the target are accepted by the algorithm, but the points that raise the objective are also accepted, but with a fixed probability. Accepting points that raise the objective allows the algorithm to search globally for a higher potential of finding solutions that do not keep the algorithm stuck in a local minima. To gradually lower the temperature while the algorithm runs, an annealing schedule is used. The said algorithm is used to narrow down the scope of its search as the temperature drops. The main aim of doing this is in order to converge towards the minimum.

2. *Problem Definition*

The aim of the optimization is to reduce the half-perimeter of the smallest bounding box that contains all of the pins for each net, added together for all nets. Each cell in a circuit is given a specific position by the device. Create a straightforward placer based on simulated annealing that reduces the overall wirelength. There was also a need to research the effect of different cooling rates. This is a version of a tool for placing standard cells during simulated annealing.

3. *Algorithm*

The coming lines are to explain the overall algorithm used, and our logic behind illustrated implementations, which performs a number of steps. Starting with, the algorithm begins to generate a random point of trial. A probability distribution with a scale dependent on the current temperature is used by the algorithm to select the trial point's separation from the present point. The algorithm generates the trial point, and if necessary, moves it to stay inside boundaries. Each infeasible element of the trial point is shifted by the algorithm to a value that is uniformly selected at random between the bound that was broken and the value that was - feasible - in the previous iteration. Whether the new point is superior to or inferior to the existing point is determined by the algorithm. Becomes the next point if the new point is superior to the one that comes before it. Even though the new point is inferior to the existing point, the algorithm may nevertheless choose it as the next point. Based on an acceptance function, the algorithm accepts a poorer point. The probability of acceptance algorithm is then used, where delta, and T are kept positive; between the values 0, and half, the smaller temperature results in a lower likelihood of acceptance, while greater temperature results in a lower acceptance probability. The algorithm, then, gradually reduces the temperature while storing the current optimal point, and hence, updating the temperature.

4. *Code*

Our code performs a number of functions; parsing, random placement, wire length calculation, swapping, and updating, based on calculating, and comparing. We also have a numer of assumptions;

1) HPWL (half-perimeter of the smallest bounding box containing all pins for a net) is used to estimate the wirelength of any net
2) The core area is an 2D array of empty squares (sites)
3) Each cell is a square and matches the site size.
4) The site size is 1x1.
5) No site is assigned more than one cell.
6) The distance between two cells is measured from the center of one cell to the center of the other.

Used functions are a) initial_random_placement, b) swap_cells, c) simulated_annealing_placer, and d) allocating class, and e) HPWL.

a) initial_random_placement:

This function is used to apply the random placement needed, using the greedy algorithm, to assign cells random grid locations. It picks a random cell, and applies random iterative improvement. First, picks two random cells, swaps their locations and evaluate the change in total wirelength. If total wirelength got smaller, then swap is accepted, else, swap is undone. This is repeated until the wirelength stops improving.

b) swap_cells:

The swap_cells performs the actual swapping function that was mentioned, and explained above in initial_random_placement.

c) simulated_annealing_placer:

The main idea behind this function is to create an initial random placement of the initial temperature; very high temp, where while T is more than Tfinal, the code picks 2 random cells, and swaps them, and calculates the change in the wire length (ΔL). Due to the swap, if the change in the wire length is less than 0, then this is accepted, else, reject. Used probability is (1 - e-ΔL/T). Used initial temperature is 500 * initial cost, and final temperature is 5 * 10^-6. The rate of cooling is 0.95 * temperature.

d) allocating class:

This class initializers the cells, the connected wires, the rows, and columns on our tracks, or the grid. It also contains the vector ‘cell’, which took the datatype int, and its mapping.

e) HPWL:

This function is used using HPWL as an estimating factor, put a bounding box then add, representing the width and the height. The code then loops among the wires, and from within, loops onto the size of all wires in order to finally return the half-perimeter wire length. This is done using the minimum, and maximum values calculated of x, and y.

5. Implementation, and Results

Some errors caused by last-minute changes caused our code to not display results that were previously shown. The error is not working at this very moment, however, if the code works anytime soon, we will be resubmitting.