CPEN513 Assignment 2: Placement

Martin Chua
35713411
February 20, 2023

Summary

This report details the implementation of the simulated annealing algorithm and improvements. The simulated annealing algorithm is run on benchmark circuits given on Canvas. The algorithm is implemented in Python. The graphics library is implemented using matplotlib. A copy of source code and documentation can be found in Appendix A. This report is divided into the following sections: base simulated annealing algorithm implementation, initiatives on top of the base simulated annealing algorithm to improve its performance, how the code was tested, and graphics.

Implementation of the simulated annealing algorithm

The simulated annealing algorithm is implemented as follows in the pseudocode below:

```
Algorithm 1: simulated_annealing(circuit, start temperature, end temperature, # iterations, beta)
    1. Start with initial placement and set current temperature (T) equal to start temperature
    2. While T < end temperature:
        a. for a static # of iterations:
            i. randomly choose two cells to swap
            ii. calculate the cost difference (c) needed to swap
            iii. swap only if random(0,1) < exp(-c/T))
            iv. update cost
        b. lower T according to temperature schedule
```

Starting from step 1, the implemented algorithm starts with an initial placement. In step 2, one cell is randomly chosen from the list of placed cells and the other cell is chosen randomly from all other available cells, including open cells. This is done to prevent unnecessary computations where two empty cells are chosen. Further, the cost difference needed to swap is not computed on all nets but only on the affected nets. This is done via an array that keeps track of the nets that are affected by each of the placed cells. This reduces the cost difference update runtime from an O(n*m) computation to O(m).

The cost is calculated by the equation below:

$$cost = \sum_{net}^{nets} (X_{max} - X_{min}) + 2(Y_{max} - Y_{min})$$

Eq. 1: Total circuit cost equation

where the cost for each circuit is the sum of the half-perimeter wire length of all nets. X_{max}, X_{min}, Y_{max}, and Y_{min} are the corner bounding boxes of each net. The "2" multiplication constant when calculating the Y subterm is to take into account the routing channel. As we assume that the distance between two cells is measured from the center of one cell to the center of the other, horizontally adjacent cells cost 1 instead of 2, while vertically adjacent cells still cost 2 due to the routing channel consideration in between each row.

Initial temperature is static and experimentally set to 100. The threshold end temperature is set to 0.001. Temperature update is according to the equation below:

$$T_{new} = \beta T_{old}$$

Eq. 2: Temperature update equation

where β is a constant parameter. This value should not be set too low as it may cause quenching. Experimentally, this is set to 0.99.

Table I: Comparison of the base simulated annealing algorithm on provided benchmark circuits. Unless otherwise specified in each, the values are: threshold (end temperature) = .001, start_temp=100, num_iters=100, beta=0.99.

Circuit	Cost (base)	Cost (num_iters=1000)	Cost (num_iters=1000, threshold=0.00001)
apex1	13554	9572	9486
apex4	26590	18117	17629
cm162a	106	105	107
cps	11642	8380	8088
cm138a	60	52	52
paira	12468	6842	6100
pairb	15294	7833	8925
alu2	1454	1252	1271
cm150a	100	98	98
e64	3797	2882	2847
C880	1736	1492	1493
cm151	48	45	47
average cost	7237.4	4722.5	**4678.6**
time elapsed, normalized to base	1x	10x	13.7x

Table I above compares the performance of the base simulated annealing algorithm with changes in parameters. As the threshold parameter decreases and number of iterations increase, the runtime significantly increases and the average decreases. The runtime significantly increases with a lower rate of cost improvement because as temperature decreases, there is an increased number of swaps that are not taken. As a result, it is clear that some improvements should be made to decrease runtime, which in turn could mean a faster convergence to a lower cost. This will be discussed in the initiatives section.

Initiatives

Several ideas were brainstormed to improve the runtime and performance of the base simulated annealing algorithm. The details of their implementation are below:

1. Dynamic moves, by modifying a static number of moves per iteration to a dynamic number of moves according to the equation: $moves = kN^{4/3}$, where k is a constant and N is the number of placed cells. A "max iterations" variable is defined to prevent annealing from taking too long for large circuits (i.e. apex4). This is set to a max limit of 20000.
2. Early exiting when cost has not varied in the past X iterations, regardless of temperature. Experimentally, X is chosen to be 50. This is done by computing variance of the last X costs and early exiting when the variance is 0.
3. Initial clustering. As the initial state is non-optimal, we can try to cluster larger nets closer together in the middle of the circuit. Because of this initial heuristic, the initial temperature must be greatly lowered otherwise the structure created will be destroyed by the randomness of a starting high temperature anneal.
4. Range windows. Limiting the distance to which a cell can swap to could improve swap acceptance and reduce cost further. This distance is limited using the "range_window_divisor" variable, which is

a fraction of the size of the circuit being annealed. A “range_window_min” is also available to set the minimum window size. Experimentally, range_window_divisor is set to 3. As a result, the early exit variance must be set to a threshold rather than 0, as it is possible that the entire range window consists of placed cells and zero empty cells, which will change the cost.

5. For the sake of time, bounded range windows (as seen in RLPlace reading #3), multiple cell swaps, and parallel computations (i.e. partitioning) were not attempted.

Without changing parameters, the runtime is significantly reduced by 45% for smaller circuits compared to without, although total runtime is still dominated by larger circuits. In an attempt to lower average cost, several parameters are changed: β is changed to 0.995, k is set to 1, and initial temperature is set to 17. As seen in table II, average cost is reduced compared to the best case in the base simulated annealing algorithm results seen in table I. Run-to-run variations exist and the results reported may not be the absolute best case results seen with the adjusted parameters. Base + initiative cost per circuit results are in Appendix B, which also includes results with a different cost assumption. There is a tradeoff between longer runtime and achieving a better cost result. A shorter base + initiative runtime with a lower β = 0.9 gave an average cost of around 4300.

Table II: Comparison of the best run from Table I’s base simulated annealing results versus one run with the base + initiative.

	Base (best)	**Base + initiative**
Average cost	4678.6	3818.1

Verification
Testing was done in two parts: manually using the debugger + breakpoints and automatically using assertions/if-else statements for unit testing individual functions (rather than explicit tests using Pytest as done in assignment 1). Small checks are also within the algorithm code to ensure correct functionality, and the code is well-documented. Blackbox testing was done only via visual inspection of the cost and temperature graphs as it would take significantly more time than writing the actual code. Examples of the animation are in the next section.

Graphics
The asn2.py script can be run via command line or an IDE such as Visual Studio Code + Python extension. A video example of the preliminary base simulated annealing can be found here: simulated anneal of apex1.

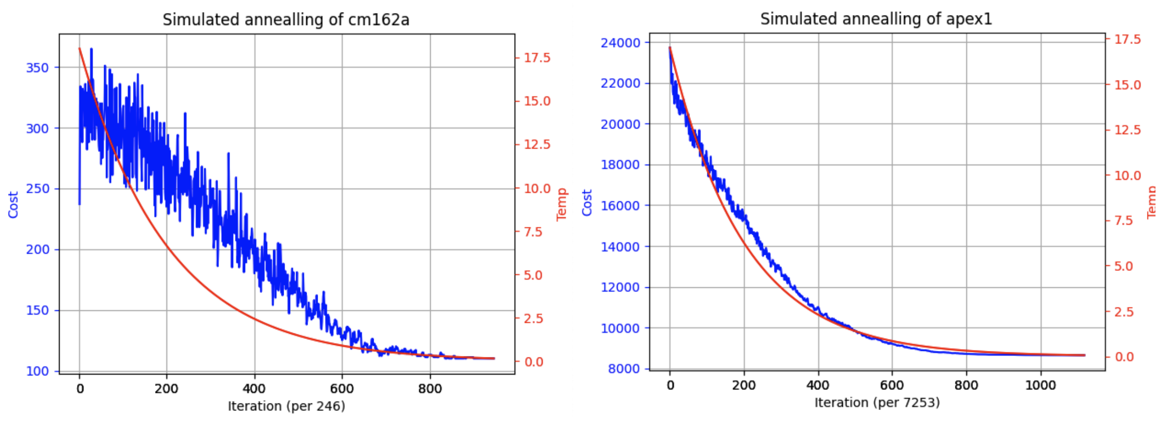


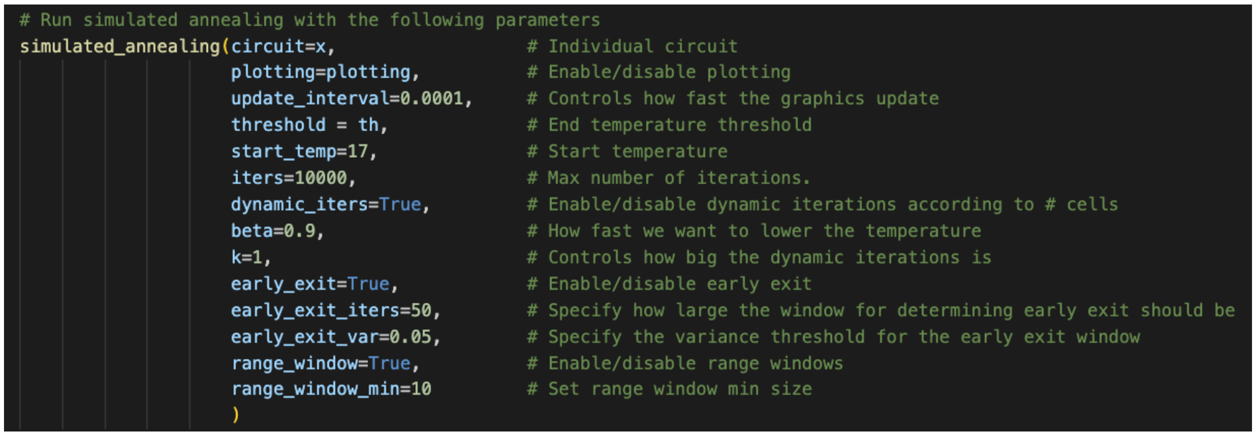
Fig. 1: Visualized simulated annealing intermediate progress of the cm162 circuit (left) and apex1 (right) with the base + initiative implementation. The red curve is temperature, and blue is cost.

The script is designed to be configurable (see Appendix A). Similar to assignment 1, the speed of the algorithm’s progress can be sped up or down using update_interval.

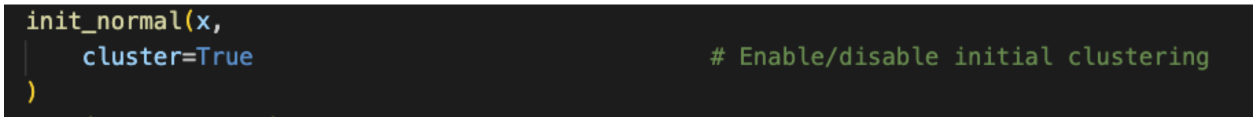
Appendix A: Source code

Source code and documentation can be found here:
https://github.com/mchuahua/CPEN513/tree/master/asn2

Configurable variables:



```
# Run simulated annealing with the following parameters
simulated_annealing(circuit=x,                 # Individual circuit
                    plotting=plotting,         # Enable/disable plotting
                    update_interval=0.0001,    # Controls how fast the graphics update
                    threshold = th,            # End temperature threshold
                    start_temp=17,             # Start temperature
                    iters=10000,               # Max number of iterations.
                    dynamic_iters=True,        # Enable/disable dynamic iterations according to # cells
                    beta=0.9,                  # How fast we want to lower the temperature
                    k=1,                       # Controls how big the dynamic iterations is
                    early_exit=True,           # Enable/disable early exit
                    early_exit_iters=50,       # Specify how large the window for determining early exit should be
                    early_exit_var=0.05,       # Specify the variance threshold for the early exit window
                    range_window=True,         # Enable/disable range windows
                    range_window_min=10        # Set range window min size
                    )
```



```
init_normal(x,
    cluster=True                            # Enable/disable initial clustering
)
```

Appendix B: Initiative results

Circuit	Best cost from Table I	Initiative Cost (with routing channel, β=0.995)	Initiative Cost (with routing channel, β=0.995, max_iters=20000)	Initiative cost (without routing channel, β=0.9)
apex1	9486	8647	8650	6254
apex4	17629	15375	15247	11289
cm162a	107	105	110	84
cps	8088	6458	6336	5252
cm138a	52	56	52	35
paira	6100	5036	4984	4595
pairb	8925	5139	5026	5602
alu2	1271	1195	1175	967
cm150a	98	106	97	70
e64	2847	2689	2673	2069
C880	1493	1427	1422	1074
cm151	47	47	45	34
average cost	4678.6	3856.6	3818.1	**3110**

Without the routing channel consideration (ie. assumption that the routing channel is included in the HPWL cost function calculation, hence the "2" multiplication constant), the best initiative cost drops significantly.