CPEN513 Assignment 2 - Placement

Brandon Freiberger
86799228

I. Description

The goal of this assignment was to write a simulated annealing tool that targets FPGAs. This tool takes in a text file containing the netlist and grid dimensions, and proceeds to place the constituent cells on the circuit of the provided grid dimensions.

II. Implementation

A. General Structure

The 2 main structures in the code are the netlist and the block locations. Both structures are stored as dictionaries, where the key is the net number in the netlist and the block number in block locations.

In the netlist, the first entry corresponds to a list containing the constituent cells as provided in the text input file. The second value at each key is the total wirelength of the net. In block locations, the first entry is the location (x, y) of each block, while the second value is a list that represents all nets that the block is a part of. Both of these structures are used to reuse as much data as possible: with this, the cost does not need to be calculated from scratch each iteration.

B. Efficiency Improvements

There were several optimizations made to make the algorithm run more efficiently:

1) The length of each net was stored in the netlist structure to avoid having to recalculate all of the netlist lengths each time 2 cells were swapped. The addition of the list of netlists lengths is a very short portion of the overall run time: the number of additions is linear with respect to the number of nets.
2) A sublist was created for each block containing the nets it was associated with, which meant that each time a block was swapped, only those nets' lengths would have to be updated.
3) The netlist is only updated if there is an accepted move: there are no temporary updates to the netlist.

C. Cost Improvements

In order to lower the cost of the placement, the following features were implemented:

1) 50 random initial placements are done, with the best one being taken to perform placement on.
2) If the temperature or accepted move percentage drops beneath a certain threshold (.7 or .001, respectively), there would be only a single further iteration. In this iteration, the temperature was set to 0 for a greedy approach to only accept good moves (as suggested in the course slides).

The initial temperature was set at 22, while the final temperature was set to .7. Additionally, the temperature was decreased by 5% each iteration until the terminal thresholds were reached.

Finally, as suggested by the slides, the number of swaps performed at each temperature was set to $Nk^{1.33}$, where N was empirically set to 20, and k represents the number of nets in the circuit.

III. Results

The final results can be seen in Table I. All results were generated with decreasing the temperature by 5% each iteration with an N value of 20.

TABLE I
Cost of Placement

Benchmark	Standard Annealing	Range Window
cm138a	58	54
cm150a	91	81
cm151a	51	51
cm162a	112	106
alu2	1906	1891
e64	3479	3461
paira	7573	7463
pairb	7636	7463
apex1	12909	12994
cps	12286	12271
apex4	24336	23802

IV. Testing

V. Extra

In an attempt to achieve better placements, I implemented a version of range windows. The window is made larger or smaller using the formula $(max(num\_rows, num\_columns))^{.33}$, which was found to give favorable results. The size of the window was increased if the acceptance rate was above .5, and decreased if it was below .4. The maximum size of the window was capped at $max(number\_cols, number\_rows)$, while the minimum was capped at 3. The window was square in shape.

However, using range windows did not have a significant improvement with respect to cost (6330 with range windows 6441 without) as can be seen in Table I. Varying the rate at which the window size was changed also seemed to have little or no effect.

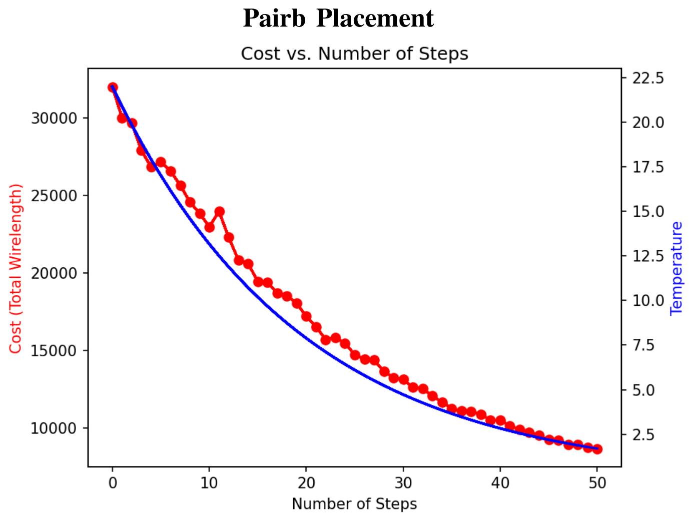

Fig. 1. Example of the output at the end of simulated annealing. The red line represents the cost of the placement cost with respect to the total wire length, while the blue line represents the temperature.

VI. Discussion

VII. Running the Code

There is a README file in the github (https://github.com/Undeadpapaya/CPEN513_A2) that provides a guide on how to run the program. To run placements, type ”python3 annealing.py” into the terminal. The input file can be modified on line 331 of annealing.py. By default, it is set up to use range windows.

After each temperature update, the plot will be updated to show the current cost as well as the current temperature. Note that with larger arrays, each iteration will take quite some time to process. If running on a larger example (e.g. apex4), it may take up to 1 minute before the graph appears. Running on a Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz with 16GB of ram, Paira takes around 10 minutes to run. Alu2 might be a bit better to do a quick run, as it takes around 3 minutes.

After the last update, the graph will pause for 10 seconds before disappearing.