Assignment Report:

A2: simulated-annealing based placement tool

EECE 583: CAD Algorithms for Integrated Circuits

Prof. Steven Wilton

M.Eng. Peixu Ren

1. Basic Implementation of Simulated Annealing Algorithm

1.1. Data structure

There are three levels of data structure in FPGA block placement.
(1) Blocks. Every block records (a) its own position, marked as a pair of coordinates, (b) its block number. Because all the cells have their individual number. When a grid does not have a cell, its block number is set as -1.
(2) Nets. The member data includes (a) the number of this net. This number does not mean anything, but only an index for data storing searching. (b) All the cells in this net, including the number of cells. (c) whether the net is "involved" in a swap. This will be used in the progress of cost updating (Section 3).
(3) The whole placement. Contains all the nets and all the cells. And the algorithm actually operates on this level of data structure.
Note: there are nx×ny grids, which is different from cells (no more than grids). Grids are all the possible placement block. Each may have or have no cell placed on it.

1.2. Basic operations

The program includes TWO parts of important functions: drawing the UI and application of simulated annealing. The UI will be introduced in Section 5. The simulated annealing is implemented as a basic version, except that some schedules are changed.

1.3. Basic version of annealing schedule

Four schedules are applied:
(1) Initial temperature: fixed as 10000
(2) Update of T: $T_{new}=0.99\times T_{old}$.
(3) Number of moves for each temperature: $k=10\times N^{4/3}$. Where N is the number of cells for the placement.
(4) Exit criteria: if the cost has decreased to 30% of the original cost.

1.4. Cost function

The cost function is implemented as the minimum sum of half perimeter for bounding box of all the nets. Take X direction for an example. Try to find the cell with least X-position value, and the cell with largest X-position value for a net. Do subtraction with the two X values, and it gets the result of the X half perimeter for the bounding box. Do the same calculation for Y direction.

2. Schedule Modification

My schedule modification is mainly based on VPR method which is proposed in the paper from Betz and Rose [1]. In the paper, all of my four annealing schedules aforementioned are modified, except the number of moves for each temperature, which remains as k=10×N4/3. And an important shrinking range window is introduced.

A critical value is involved - acceptance ratio. It is the ratio of accepted moves to the total moves for each T. The number of accepted step is initialized as the number of total moves, i.e., $k=10\times N^{4/3}$, and it decreases by 1 if a rip-up occurs.

2.1. Initial temperature

My program performs 50 random swaps, pairwise for cells blocks or one-way move for a cell block to an empty grid, and computes the standard deviation of the 50 costs.
Then the initial temperature is set as $T=20\times$std(50 costs).

2.2. Update of T

A new temperature is computed as $T_{new}=\alpha T_{old}$, where α depends on acceptance ratio (R_{acc}). The decision scheme is shown in Table 1.

Table 1: decision scheme for α

R_{acc}	α
$R_{acc}>96\%$	0.5
$80\%<R_{acc}<96\%$	0.9
$15\%<R_{acc}<80\%$	0.95
$R_{acc}<15\%$	0.8

2.3. Exit criteria

When $T<0.005\times cost/N_{nets}$, the iteration is finished. Actually, when this criterion is achieved, the cost reaches its convergence naturally.

2.4. Range window

According to Lam and Delosmet [2], the temperature decreases most rapidly when the acceptance ratio is 44%. To maintain this value, a range limiter is introduced, which is actually a range window: only interchanges of blocks that are within the window are attempted.
The window is confined as a function of R_{acc}, which is $\mathrm{D}_{\mathrm{limit}}^{\mathrm{old}} = \mathrm{D}_{\mathrm{limit}}^{\mathrm{new}}\times(1-0.44+\mathrm{R}_{\mathrm{acc}})$. And D_{limit} corresponds to both x and y direction.

3. Other Attempts

3.1. Calculate the involved nets' cost

At the beginning, the cost function is calculated for all the nets when a swap happens. Obviously, however, calculation for only changed nets' cost reduces the time complexity.
The addition work is to set a flag bit for each net - involved. When a swap happens or even as early as the swapped grids' coordinates are chosen, it should traverse all the nets' cells. If a net has the cell number for the particular swap cell, the involved flag for this net should be set as true. Before and after the swap, only the involved nets' costs are calculated.

3.2. Swap between two empty grids

In my implementation, once the program chooses two empty grids for swapping, this move is skipped. When a skip occurs, this move does not count, and the move is neither accepted nor not accepted. Assume the number of skipped move is N_{void}, the acceptance ratio is calculated as

$$R_{acc} = (N_{acc} - N_{void})/(N_{total} - N_{void})$$

This skip will impact the program in two cases:
(1) Less likely to choose two empty grids. That means N_{void} is a small number, then R_{acc} does not change too much compared to "full" moves.
(2) Quite likely to choose two empty grids. For example, the netlist "pairb.txt" has 900 cells with 3500 grids in total. The chance to choose two empty grids is

$$P(\text{empties}) = \frac{2600}{3500} \times \frac{2600}{3500} = 55.18\%$$

Even this makes total moves for one temperature drop a half, but the basic calculation method for acceptance ratio does not change, which is still the accepted moves divided by valid moves. In addition, the skips accelerate the program largely, with have of the iterations being ignored.

4. Results

The results include two measurements. One is the execution time, which is measures with a clock in C++ library < ctime>, and the other one is the the cost, including the ratio of final cost to initial cost. And versions of my program are
(1) Basic Implementation. It has basic annealing schedule.
(2) VPR version. It computes the total cost before and after the swap.
(3) VPR version. It computes the changed cost before and after the swap.
However, only the version (3) is measured, which actually reveals my final results.
The "average" values are calculated with 5 consequent runs.
The "minimum" values are the observation of all the trials that I have tried.
Table 2 shows the results for benchmark c880, e64, paira, pairb, and apex4

Table 2: results for banchmarks

	Average Runtime (s)	Average Final Cost	Average Cost Ratio	Min Final Cost
c880	1.886	995	25.75%	976
e64	4.218	1930	27.77%	1851
paira	24.494	3929	14.74%	3836
pairb	11.452	4319	9.58%	4176
apex4	60.474	10512	24.10%	10387

5. UI and User Instructions

As I use EasyGL for UI drawing, users cannot open a file through graphical window. But users can type the name of the benchmark in command line. However, users need to modify the path in the main cpp file (583A2.cpp). Use command "make" to make execution file. Library and header link need to be modified for Linux and Mac.

6. Reference

[1] Lam, J. and J. M. Delosme, "Performance of a New Annealing Schedule." Proc. 25th Design Automation Conf., 1988, pp. 306-11.
[2] Betz, Vaughn and Jonathon Rose. "VPR: A New Packing, Placement and Routing Tool for FPGA Research." International Workshop on Field Programmable Logic and Applications, 1997: 213-22.