

University of Toronto, Faculty of Applied Science and Engineering
Department of Electrical and Computer Engineering

ECE 1387 - CAD for Digital Circuit Synthesis and Layout
Assignment #2 – AP Fitting and Greedy Placement Optimization in VPR

Feb 13, 2009

J. Anderson

Assignment Date: February 13, 2009

Due Date: March 6, 2009, at the beginning of class.

Late Penalty: -2 marks per day late, with total marks available = 20

Note: This assignment must be done individually.

The objective of this assignment is for our ECE 1387 class to create a working analytical placer for FPGAs, within the VPR framework. While there are publicly available analytical placers available for custom ICs, there has yet to be one for FPGAs. We are aiming to create one in this course. You may choose to do **either** part of this assignment; you do not have to do both parts. This document describes the second part of the assignment. The first part is analytical placement and spreading.

You are to write an implementation of an analytical placer “fitter” and implement a greedy placement optimizer within the VPR framework. As described in class, a fitter accepts the floating point placement produced by an analytical placer as input, and it snaps the cells onto the discrete placement grid. Your greedy optimizer will execute after fitting to improve the placer “score”. You will use the existing scoring mechanism within VPR.

The VPR source code, test circuits, architecture file, and the input files for this assignment are given on the course webpage. There is one input file that corresponds to each of the test circuits, as indicated by the input file name.

The first line of the input file has the following format:

X Y

This line gives the extent of the placement area in the X and Y dimensions, respectively. The placement slots for the core logic cells are on a grid at the integer locations: $([1:X],[1:Y])$ – that is, there are $X*Y$ available placement slots for core logic cells. While core logic positions are from $([1:X],[1:Y])$, I/O cells are placed in a ring around the core cells. Meaning, I/O cells are placed in the 0^{th} and $(X+1)^{st}$ column, or in the 0^{th} and $(Y+1)^{st}$ row.

The remaining lines of the input file have the following format. There is one line in the input file for every cell being placed.

blockname Xpos Ypos SubBlock core #comment

First, note that core field has a value of either Y or N. Y means that the cell is a core logic cell; N means that the cell is not a core cell – rather, it is an I/O cell. As described in class, I/O cells are pre-placed in analytical placement and their placement is fixed.

blockname is the name of the cell being placed. Xpos is the position of the cell in the x-dimension. Ypos is the position of the cell in the y-dimension. SubBlock is the sub-block (sub-slot) position. For core logic cells, the SubBlock is always 0. For I/Os, however, the SubBlock may be either 0, 1, or 2. The reason for this is that in VPR, three I/O cells can fit together at one I/O placement slot. So, for I/Os, the SubBlock field distinguishes the correct sub-block slot for the I/O. The #comment field can be ignored.

Observe that for I/O cells, Xpos and Ypos are integer values. I/Os are fixed blocks and your fitter must place them at precisely the position given in the input file. Core logic cells, on the other hand, have floating point values for their

Xpos and Ypos, as computed by an analytical placer. You must “fit” the core logic cells into discrete placement slots that are as close as possible to the Xpos and Ypos values given in the input file.

Part A – Fitting

Develop a fitter that maps the core logic cells into the discrete placement grid. Each core cell should be placed as close as possible to the Xpos, Ypos given for the cell in the input file. For each input file, report the fitting error:

$$Error = \sum_{i \in CoreLogicCells} |X_i - X_{pos_i}| + |Y_i - Y_{pos_i}|$$

where X_i is the position on the discrete grid where your fitter locates cell i and X_{pos_i} is the position of cell i given in the input file. The parameters are defined similarly in the y-dimension. Naturally, your fitter must ensure that no two core cells are located at the same grid location. Also, your fitter must place I/O cells at exactly the same position as specified in the input file.

A simple fitting algorithm can be used, as follows: First, fit a core cell into the placement slot that is closest to its X_{pos} and Y_{pos} . For example, if the cell’s X_{pos} , Y_{pos} is (2.1,1.9), then fit it into the slot at (2,2). Then, move onto a second core cell and find the placement slot closest to its X_{pos} and Y_{pos} . If that spot is occupied, use breadth-first search to find the closed vacant position that can accommodate the cell. Repeat the process iteratively for all cells.

Part B – Greedy Placement Optimization

Having produced a legal placement in Part A, greedily optimize the placement within VPR to improve the score, which is comprised of performance and wirelength. You should endeavour to improve the score as much as possible, with as little run-time as possible. In class, we will discuss the results achieved by everyone and the approaches taken.

One possible approach to greedy optimization is called “windowed optimization”. Consider the case where the placement area spans from (1,1) to (10,10). One can *slide* a 2x2 window over the placement area, and consider all possible cell swaps in that window. A cell swap is “accepted” if placement score improves, otherwise it is rejected. After considering all possible swaps in the window, the window is shifted. Window shifts can be horizontally or vertically. Windowed optimization on a placement can be repeated multiple times, for further placement improvement. 2x2 is just an example; other window sizes may yield improved results.

Note: Your development should be done in the `place.c` file of VPR. You must disable the simulated annealing-based placement in the `try_place(...)` function of `place.c` and insert your code that reads the input file, calls your fitter and greedy optimizer. Your greedy optimizer should use the scoring functions that are already built into VPR. You will need to understand a bit about VPR’s data structures and how VPR proposes and scores moves in its annealing placer. After running your fitter, update VPR’s data structures with your fitting placement results. Invoke your greedy optimization algorithm. Use the existing routines in VPR to write out the placement results.

What to hand-in:

- A listing of your code changes in VPR. Send me a pointer to the directory containing your code.
- A brief description of your fitting algorithm. Report the fitting error for each of the input files. Report any innovations you implemented.
- A brief description of your greedy optimization algorithm. Report any novelties in your algorithm. Report the improvement in placer score for each of the input files. Report the final placer score for each of the input files. Report the run-time of your algorithm for each of the input files.
- A pointer to a directory containing the placement results, as produced by VPR. Please change the permissions on directories so I can access them.