EE 5301 - FALL 2024

PROJECT 3: ANALYTIC PLACEMENT WITH GENERATIVE AI

Due date: Dec 3, by midnight
Note: this document talks about Phase 1 and Phase 2, but you will be submitting the whole project by the deadline. Phase 1 is to help you pace yourself well.

In this programming assignment, you are asked to:

- Implement an analytic placement algorithm to (partially) solve the placement problem based on the FastPlace paper.
- The input parser for the ibmxx benchmarks is provided as "suraj_parser.cpp" and "suraj_parser.h".
- **NOTE**: a potential source of error is how the array `pinLocations` is indexed. Make sure you read the comments in the .h file.
- You will use ChatGPT / Co-Pilot or other generative AI to implement any part of the code that you like. Here are a few suggestions:
    - I especially recommend having it write the code for the matrix solver (no Cholesky decomposition needed, just the gradient descent).
    - Generate Python code to visualize your placement with zoom in/out capabilities.
    - Calculate the total wire length from cell coordinates (see the "Program Output" section for more details).
    - Calculate the chip width / height based on all cell and I/O pad coordinates.
- You need to use the formulation in the paper to convert edges and hyperedges to cliques or stars.
- You need to generate the Q matrix and the d_x and d_y vectors, solve for x and y vectors, and then apply **only one** iteration of cell spacing using the grid boundary method explained in the paper. Set the grid to be 5x5 over the whole chip area.
- Consider all cells to have a dimension of 1x1. For the movement of the cells, use a fixed value of 0.8 for α_x and α_y.
- Your code needs to be modular, i.e., divided into multiple files that are compiled together using a Makefile. Avoid writing very long functions.

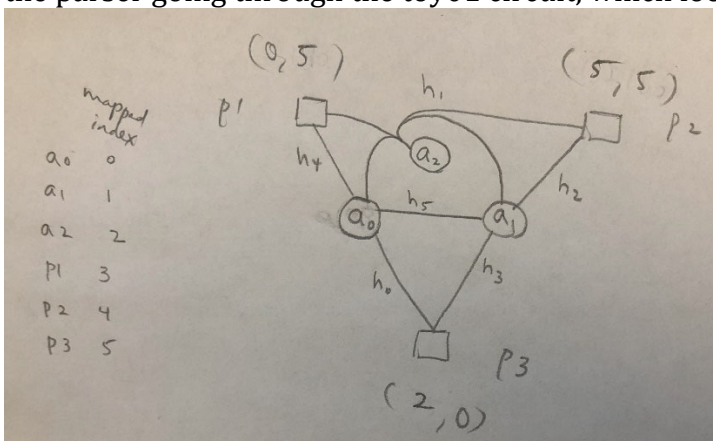
Input Files

The toy circuits are named toy0x, with extensions .net, .are, .kiaPad. Larger files are called ibmXX.net, .are, .kiaPad.

The .net has the circuit size numbers (number of cells, I/O pads, nets) and the netlist. The .are has the area number (which you can ignore), and the .kiaPad file has the I/O pad coordinates.

You don't need to know the format of the files, but you do need to understand the format of the data structures that Suraj's parser creates very well. The .cpp file contains detailed explanations of a small circuit.

If you want to better understand the data structures, you may want to single-step the parser going through the toy01 circuit, which looks like this:



Command Line Parameters

There is only one command line argument, and that is the circuit file name WITHOUT EXTENSIONS. The three .net, .are, and .kiaPad files should be present.

Program Output

Your program should output the square root of the sum of squared wirelengths of the circuit. Note that you calculate the squared wirelength of individual nets (possibly derived from hyperedges), using the weight scheme presented in the FastPlace paper (γW and $k \gamma W$ with $\gamma=1/(k-1)$ for clique and star hyperedges. Simple edge weights remain W). Report the wire length both before and after cell spreading.

Your program should also create a file in which you write the x,y coordinates of all cells (I/O pads and moveable cells). The format should be similar to the .kiaPad file.

It should also create a graphical representation of the placement both before and after spreading cells, which means that you need to output the coordinates both before and after spreading cells in two separate files.

Phase 1

In Phase 1, you should build the backbone of the program: reading the input file, populating the Q matrix and d vectors. You can test your code using the toy example(s) and verify your matrix entries manually.

You should also test with the ibm files and make sure your program does not crash.

You can assume all moveable cells are at (ChipWidth/2, ChipHeight/2), and calculate wirelength. Feel free to report your numbers on the forum so that you can compare against others' numbers and debug your code if necessary.

You will also implement the graphical user interface (offline viewing of the results that you have written in a file).

Phase 2

In Phase 2, you would implement solving the matrix equation using the Conjugate Gradient method, and then spreading the cells.

Submission Process

The two phases are for you to pace yourself. You don't need to submit anything in Phase 1, except for the results that you will post on the forum.

Before the assignment deadline, please submit your code as one .zip containing all source and header files (including Python code, and also the Makefile), the output coordinate files, and a text file showing your wirelength values for each benchmark.

Also include .jpg files showing your placement before and after spreading.

Grading

- Submission and compilation of the program: 25%
- Graphical user interface: 10%
- Correct WL numbers (after spreading) for the toy example(s): 10%
- Correct WL numbers (after spreading) for ibm circuits: 55%

Appendix: Benchmark Statistics

Ckt	#Cells	#Pads	#Nets	#Pins	#Rows
ibm01	12506	246	14111	50566	96
ibm02	19342	259	19584	81199	109
ibm03	22853	283	27401	93573	121
ibm04	27220	287	31970	105859	136
ibm05	28146	1201	28446	126308	139
ibm06	32332	166	34826	128182	126
ibm07	45639	287	48117	175639	166
ibm08	51023	286	50513	204890	170
ibm09	53110	285	60902	222088	183
ibm10	68685	744	75196	297567	234
ibm11	70152	406	81454	280786	208
ibm12	70439	637	77240	317760	242
ibm13	83709	490	99666	357075	224
ibm14	147088	517	152772	546816	305
ibm15	161187	383	186608	715823	303
ibm16	182980	504	190048	778823	347
ibm17	184752	743	189581	860036	379
ibm18	210341	272	201920	819697	361

Source for the table:
http://vlsicad.eecs.umich.edu/BK/Slots/cache/www.public.iastate.edu/~nataraj/ISPD04_Bench.html