Simulated Annealing: TimberWolf-like Placement Tool

I. Compilation Instructions

The floorplanning tool (part a.) and the placement tool (part b.) use the same makefile. Invoking the `make` command will compile both tools. Alternatively, `make timberwolf` will compile only the placement tool. The name of the placement binary is `timberwolf`.

II. Execution Instructons

`timberwolf` takes up to four arguments—invoke `timberwolf` without parameters for a description of those arguments. In general, the following arguments are useful for demonstration:

```
timberwolf <input file> 10000.0 10 1.02
```

III. Data Structures

The placement is realized as a matrix of cells, called a "lattice." This matrix is large enough to represent any possible floorplan—one spot in the matrix is occupied by one cell at a given time. Therefore, the number of columns in the matrix is equal to the maximum row length divided by four (that is, equal to the maximum number of cells in a row, since all cells have length four). The number of rows is equal to the number of cells, as the worst-case height of a placement results from having all rows of length one. The actual implementation of the matrix stores pointers to a fixed set of cells—this significantly conserves memory.

IV. Structure and Organization

`timberwolf` is structured similarly to the floorplaning tool from part a. The code describing the data structures are found in Cell.h, Cell.cpp, CellLattice.h, and CellLattice.cpp. However, all modification to the CellLattice objects is done within the simulated annealing algorithm (found in SA_twolf.cpp).

Execution begins in main_twolf.cpp, where arguments are parsed. Parameters are then passed to code in SA_twolf.cpp, which contains the SimulatedAnnealing() and Metropolis() functions, similar to the text's outline of the simulated annealing approach to heuristic-based optimization. A number of small-subfunctions are also called (from SA_twolf.cpp), chief among these are CostLattice() which calculates the cost of the current placement, and Rand01() which produces a pseudo-random (`double`) value in the range of [0,1] with three significant digits.

Cost function: The timberwolf cost function considers wire length and number of rows used—the number of rows is weighted by 2500 to scale the value to the same range as the wire length.

V. Improvements

`timberwolf` in its current form has a tendency to prefer placements with heights greater than widths—I believe this can be solved by introducing further heuristics when calculating cost, either by separately weighting the cost of vertical interconnect, or by penalizing a given placement based on the total number of rows used in the placement. It would also be useful for `timberwolf` to consider proximity of input/output nets (and

cells) to the “edges” of the placement—this is entirely ignored (not even specified) in this implementation, but the actual TimberWolf algorithm does consider the placement of I/O pads.