Simulated Annealing: Floorplanning 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 floorplan` will compile only the floorplanning tool. The name of the floorplanning binary is `floorplan`.

II. Execution Instructons

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

```
floorplan <input file> 0.75 10000.0 10.0 1 1 1
```

III. Data Structures

The floorplan is realized as both a linked-list data structure and a binary tree data structure. Aspects of both of these data structures are useful to the floorplanning algorithm (list behavior is useful for swaps, tree behavior is useful for evaluating cost). This polymorphic data structure is built of two node types: “Modules” represent the modules as given in the input file, “Operators” represent cuts (type V or H).

IV. Structure and Organization

The structure of the floorplanner consists of two realms: data structure maintenance/modification and the simulated annealing algorithm itself. The code necessary to maintain and change the polymorphic data structure (from III) is found in files Floorplan.h, Floorplan.cpp, Module.cpp, and Operator.cpp. The remainder of the code implements the simulated-annealing functionality, and is found primarily in SA_floor.cpp, with some argument processing occurring in SA_main.cpp.
Execution begins in SA_main.cpp, which first reads its input parameters (one call into Floorplan.cpp), and then begins the annealing process in SA_floor.cpp. SA_floor.cpp makes calls into Floorplan.cpp whenever a move is performed. Floorplan.cpp in turn calls functions in Module.cpp and/or Operator.cpp as needed.

V. Improvements

This floorplanning tool could be improved if, instead of replicating the current solution and then performing some perturbation, it operated directly on the current solution and then reversed the move if it is not accepted. The majority of the running time (roughly 60%) of this algorithm in two out of three of its development platforms is devoted to dynamic memory management (in the OS)—to reduce calls to `new` and `delete` (or `malloc`() and `free`()) would be to greatly reduce the overall running time.