

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 #3 - Branch and Bound Optimization
Applied to the Bi-Partitioning Problem**

March 2009

J. Anderson
(Orig: J. Rose)

Assignment Date: March 13
Due Date: March 27 (before lecture begins)
Late Penalty: -1 marks per day late, with total marks available = 20

You are to write an implementation of the branch-and-bound partitioning algorithm described in class. It is to take netlist of (equal-sized) blocks and divide it **in half** with the goal to minimize the number of **nets** that connect blocks in the two partitions; that is, you are to minimize the “cut set” (crossing count). You should write *two* implementations of the branch-and-bound algorithm:

1. An exact one that is guaranteed to find the optimal solution.
2. An approximate one that uses an approximate bound of your own devising. Your approximate bounding function should aim to find a “good” (though not optimal) solution, with a drastically reduced search space.

Aim to design clever bounding functions to reduce the size of the explored search space.

As in the previous assignments, your program should display its progress and results using X11-based graphics. Illustrate, as best you can, the present state of the decision tree, and the pruning as it progresses.

The netlist format specifies the blocks (cells) and the connectivity between them. Each line has the following form:

blocknum netnum₁ netnum₂ netnum₃ ... netnum_n -1

Where blocknum is a positive integer giving the number of the block, and the netnum_i are the numbers of the nets that are attached to that block. Every block that has the same netnum_i on its description line is attached. Note that each block may have a different number of nets attached to it. Each line is terminated by a -1. A -1 appearing by itself on a line terminates the netlist. Example input file:

```
1 2 3 4 -1
2 5 4 -1
3 5 6 2 -1
4 6 3 -1
-1
```

In this example, block 1 is connected to nets 2, 3 and 4. Note that each net may be connected to more than two blocks (that is, there are multi-fanout nets). Also note that net numbers are not related to block numbers.

Test your program on the four test circuits provided on the course web page. Do not allow unequal partition sizes. Your program may or may not be able to find a solution for the fourth circuit in “reasonable” run-time.

Note: you are to minimize the number of **nets** that connect blocks in the two partitions. This means that each net (even a multi-fanout net) contributes either 0 or 1 to the total crossing count, and it never contributes more than this. A net contributes 1 to the crossing count if it connects cells that fall in both partitions.

What to hand in:

1. The location of the executable and code on ECF or EEGG. **Please change permissions so that I can access and run it and peruse your code.**
2. A short 2-3 page description of the flow of your program. You should describe:
 - The branching structure (that is, how you designed your decision tree).
 - How you determine the initial “best” solution.
 - The bounding function(s) that you use.
 - How you traverse the tree and prune it with the bound.
3. For each version of the program (exact and approximate) provide the following:
 - A paper plot of the results from the test circuits.
 - A count of the number of tree nodes that are visited. There will be a **prize** for the smallest number of nodes traversed on cct2. NOTE: Tuning to the algorithm to the exact problem is not allowed. Regarding the definition of “visited”, if you are calling your bounding function for a node, that should count as “visiting” the node. Likewise, if a node is disqualified owing to balance constraints, the node should still count as “visited”.
 - The run-time for each of the test circuits.
 - The value of the net crossing count that you achieved on each test circuit.
4. Summarize and discuss your results. How close are the approximate answers to the exact ones? What run-time improvement was offered by the approximate algorithm?