UNIVERSITY OF BRITISH COLUMBIA
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CPEN 513: CAD Algorithms for Integrated Circuits
2022/2023 Term 2

Assignment 3: Branch and Bound Partitioning

Due: March 13th, 11:59pm

In this assignment, you are to implement a branch-and-bound based bi-partitioning algorithm. Remember that branch-and-bound is guaranteed to give the optimal partitioning solution, however, the worst-case run-time is exponential. If you prune your solution space properly, you will be able to run your program on medium-sized benchmark circuits.

The input format is exactly the same as in Assignment 2 (except the two numbers representing the chip x and y dimensions are not included in the first line of each file). This means you can re-use your input parser and possibly your internal data structures. This will save you a lot of time. All of the benchmark circuits from Assignment 2 are possible, however, because some of them are pretty large, I will give you a new set of benchmark circuits that are smaller. You should report your final partitioning results on these new benchmark circuits (even some of these smaller circuits may be too big for your program). The files are on the Canvas site.

Note: you are to minimize the number of nets that connect blocks in the two partitions, while keeping the sizes of the two partitions exactly equal (or differing by at most 1 if there are an odd number of blocks). This means that each net (even a multi-fanout net) contributes either 0 or 1 to the total crossing count, never more than this. A net contributes 1 to the crossing count if it connects cells that fall in both partitions.

You may or may not decide to start by choosing an "intelligent" partition before running the branch and bound algorithm. The final result will be the same, but your run-time might be significantly reduced if you do.

Since this program is guaranteed to give the optimum solution, everyone should be getting the same answers. If you and a friend are getting different results on the same benchmark circuit, at least one of the two assignments (probably your friend's assignment) is not correct.

As in previous assignments, you should include some graphics. For this assignment, it is ok to draw your final partitioning solution only (you don't need to show the progress of the algorithm as it proceeds). However, I can imagine that it would be possible to somehow draw the decision tree and graphically show branches being "pruned"… if you want to do this, go ahead.

You should hand in:
You must handin your files using the Canvas web site. You should hand in a zip file containing the following:

- A text of PDF file (report.pdf or report.txt) containing:
  - A description of how your program works. Be sure to explicitly address each of the criteria in the attached marking scheme. If you are hoping for “initiative” marks, you need to justify them in your report.
  - A table of results for the example files. For each benchmark circuit, your table should contain the minimum cut-size you found (the number of nets that cross the partition; see the comment above), and the approximate run-time of your program on that benchmark circuit. It is very possible that some benchmarks will be too big for your program; in that case, write N/A in your table.
  - One or more figures showing the graphics
- The source code. If you are using an on-line repository, you can include a link in your report. Otherwise, you can submit the source code in your zip file.

Hints:

1. Recursion will help here. You can write this algorithm without it, but it is much easier if you do employ recursion. One way you could structure your code is something like this (where "current assignments" is the set of vertices assigned so far, "next node to assign" is the next vertex to assign to a partition, and "best solution so far" somehow encodes the best solution that has been encountered):

```
    steves_routine( current assignments, next node to assign, best solution so far)
     {
         if there is no next node to assign {  // we are at a leaf
             if this is the best solution so far, record it
         } else {
             calculate label x
             if (x < best solution so far) {
                 temporary current assignment = current assignment + {next node to assign in left
                                                                      position}
                 temporary next node to assign = node after (next node to assign)
                 steves_routine ( temporary current assignments, temporary next node to assign,
                                  best solution so far)

                 temporary current assignment = current assignment + {next node to assign in right
                                                                      position}
                 temporary next node to assign = node after (next node to assign)
                 steves_routine ( temporary current assignments, temporary next node to assign,
                                  best solution so far)

             }
         }
 }
```

the first time you call this routine, current_assignments would be null, next_node_to_assign would be (arbitrarily) vertex 0, and the best solution so far would be null. The above doesn't check for the legality of the generated (partial) partition; you will have to add code for that (remember that each of the sizes of each of the two sets can differ by at most 1).

Of course, your implementation may look completely different.

2. The decision tree does not need to be a physical data structure inside your program. It *could* just be a representation of the recursive calls needed to search for the optimum solution.

Marking Scheme for Assignment 3: Maximum Score = 21

Does the algorithm give the correct results?

Score	0	2	4
Description	No.	Yes, except off by a constant factor.	Always.

Does your algorithm correctly prune the search space?:

Score	0	2	4
Description	The search space is not pruned correctly.	The search space is pruned correctly most of the time, but there are some errors or omissions (times when you could prune the search space but you do not).	The search space is pruned correctly.

Efficiency:

Score	0	2	4
Description	The implementation is not efficient; run-time is much longer than it needs to be.	The implementation is somewhat efficient, but small changes could make it faster.	The implementation is efficient; there is clear evidence of care creating an algorithm that runs fast.

Graphics:

Score	0	1	2
Description	No or very little graphics.	The graphics are adequate in showing the final solution, or parts of the final solution are not graphically displayed.	The graphics are very good and clearly show the final solution.

Code Quality:

Score	0	1	2	3
Description	Code is lacking in structure and comments.	Code is of sufficient "academic code" quality, including some comments.	Code is of good "academic code" quality including extensive comments. Code is well structured.	Code is of industrial quality, including evidence of unit tests and/ or extensive system tests.

Report (maximum 3 pages):

Score	0	1	2
Description	Report is unclear or difficult to read and/or understand..	Report describes most aspects of marking scheme clearly. The English has grammar/clarity errors that would not be acceptable in a major IEEE or ACM journal or conference.	Report describes all aspects of marking scheme clearly. The English is of a professional standard that would be acceptable in a major IEEE or ACM journal or conference.

Initiative: (be sure to identify any extensions in your report):

Score	0	1	2
Description	Assignment implemented as in handout.	The implementation contains one or more straightforward extensions.	The implementation goes beyond what is described in the handout in a non-trivial way.