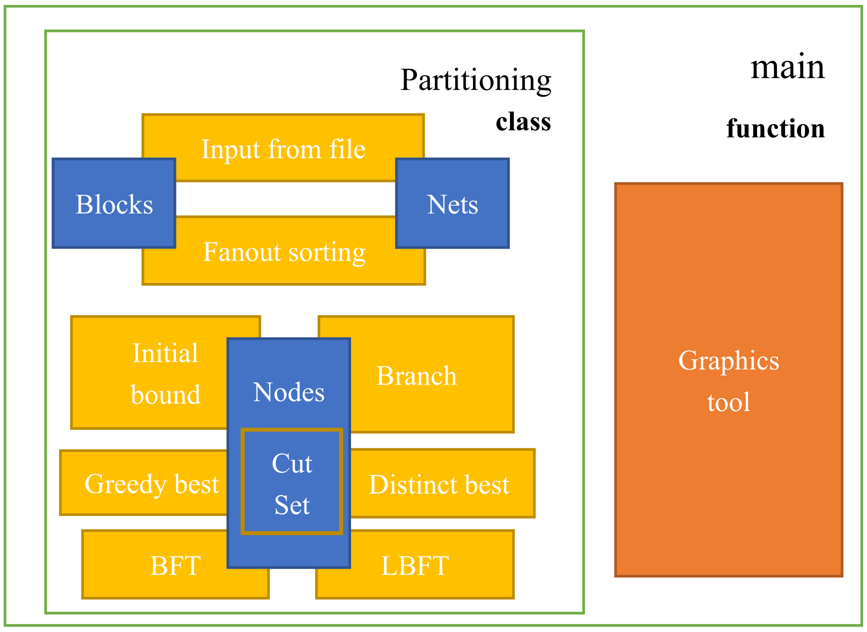
ECE1387 Assignment3 Report -B&B Partitioning Zhiyuan Gu (1004920400)

1. Software description

1) Overview



Above is the basic structure of the program. The position of the rectangles indicates the general relationships of them. Main function instantiates an object of class named "Partitioning" to solve the partitioning problem and calls functions in order to use the graphics tool. Inside class "Partitioning", "Blocks" and "Nets" are two elements related to the input netlist giving information on blocks and nets that connect them. "Nodes" represents the nodes on the B&B decision tree. Included in one node is the partial partitioning solution, cut set, lower bound of cut size, etc. Among them, cut set is the most important member in that it keeps a record of all nets regarding whether or not the net is creating a cut. Before contributing a cut to the total cut size, the one chunk that nets sit in is also recorded to efficiently update the status of each net after branching.

2) Main methods

a) Input from file

Read in the information of connections between blocks and nets.

b) Fanout sorting

This method is mentioned in handout and results have shown that sorting the blocks in fanout descending order decreases the complexity of the problem a lot. One explanation would be it helps create cuts faster and leads to an earlier pruning of branches.

c) Initial bound

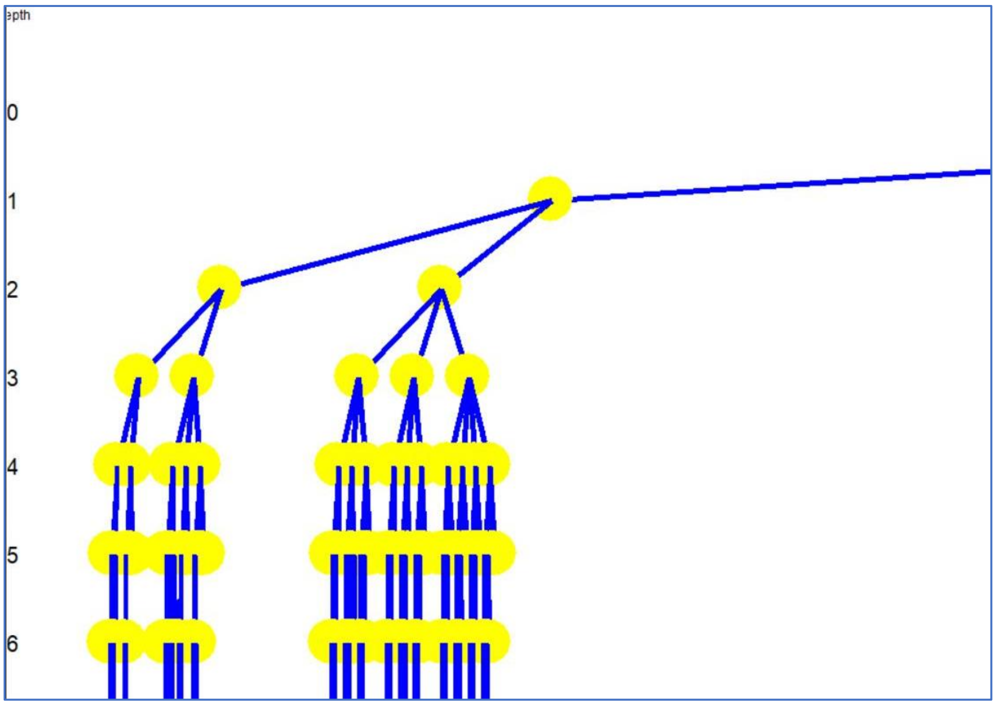
Since one net can only contribute one cut to the overall cut size, one can tell easily that a net is bound to create a cut if the size of the net is larger than a chunk. So initial bounding deals with such nets and sets them as "not-cared". Other smaller nets that are possible to avoid creating cuts are set as "cared" nets. Blocks in them are called "cared" blocks and pushed to the front to get partitioned first. No matter where the blocks other than "cared" blocks are partitioned, cut size will not be affected in the end. By doing so, the number of nodes need to be visited decreases a lot, especially for circuit 1 and circuit 2 in which most of the blocks are “not-cared”.

d) Branch

Braching is the basic operation performed to each node on the decision tree. Depth is incremented, a new block is put into chunk as part of current partial solution and the cut set is updated indicating how many nets are now creating a cut.

e) Branching structure

A 4-way partitioning seems to be much more complicated than 2-way partitioning. But considering the 4 chunks are not unique, a lot of branches can be saved due to symmetry. Symmetry means when there are more than one empty chunks existing, it doesn’t matter which one is chosen to put the current block into. All of the choices lead to the same subtree because chunk content rather than chunk position affects the overall cut size.



The decision tree used to be creating four branches from each node into the next level or depth. However, thanks to symmetry, a lot of duplicated branches are eliminated. As is shown in the top-left tree graph above, the first block only takes the left chunk. From that, the second block now have two choices, going left or going centre-left. It doesn’t matter which empty chunk is chosen but non-empty chunks matter because interaction with other blocks could lead to different cut sizes. Same rule goes on and continues to cut branches in the tree. If the first 5 blocks are all put in the left chunk, the number of required child nodes of that partial solution is still just two to represent staying with other 5 blocks or staying in a different chunk. This branching structure also harshly reduces the number of nodes required to be visited. For circuit 3, visited node count dropped from 566,565 to 24,164. The improvement is estimated to be over 20x for large circuits.

f) BFT vs. LBFT

The two traversal orders are no different except that nodes are pushed into priority queues in LBFT rather than an ordinary queue used in BFT. Nodes with the smallest lower bound function value are picked first to branch in the hope of updating the best solution faster and pruning more. The effect of it will be discussed later.

g) Initial “best” solution

Initial solution is very critical to this assignment, especially for circuit 3. The reason why initial solution is so important is because most of the branches tend to reach a quite large cut size quickly thanks to fanout sorting. Then, this growth in cut size during the process of branching deeper slows down when the cut size of the node is close to total net count. This basically tells that lower the initial solution cut size even a bit can lead to a huge gain. This is especially true for circuit 3 in that the circuit is very complex and cannot be solved in reasonable time until the initial best solution is lowered from 40 to 38. One observation is that circuit 1, 2 and 3 seem very inflexible in that the final best solution is very close to the total net count. Therefore, it is hard to find an easy way of obtaining a good initial solution. Here are some of the trials:

i) Greedy best

This method is simple by looping through blocks of a certain order to partition and pick the best cut size from all the branches as partial solution to branch deeper. The order of blocks is first net order which means blocks are ordered by their appearing sequence in all nets. Then fanout descending order is tried. However, the greedy best initial solution seems to be just first filling one chunk, then another, and so on. The local best does not represent global best at all.

ii) Distinct best

Even if the block order is carefully determined, greedy best initial solution can only lower cut size from 40 to 39 for circuit 3 which still cannot help solve it in reasonable time. The target of distinct best initial solution is to find whether cut size 38 can be achieved. The idea is to find two distinct “cared nets”, having no shared blocks, and put them into different chunks in the beginning. This would make sure 2 cuts from 2 nets can be avoided. The problem become more complicated if 3 cuts rather than 2 cuts need to be avoided. This method might seem to be targeting the specific circuit. But since the circuit is very inflexible as mentioned above and the fact that each net has almost the same size as the chunk, only this method could reasonably lead to a good initial solution to possibly solve circuit3.

h) Bounding functions

Some of the bounding methods are introduced above in initial bound and branching structure section, the rest of them are listed below:

i. Balance bounding

If any of chunk is full, no more block can be placed into that partition. Initial bound which excludes nets with more blocks that a single chunk can take can also be viewed as an early insight of possible balance bounding. In this software, balance bounding condition is checked before creating the actual new node. However, nodes that are bounded due to balance reasons are also marked as visited because the balance bounding function is called.

ii. Lower bound function

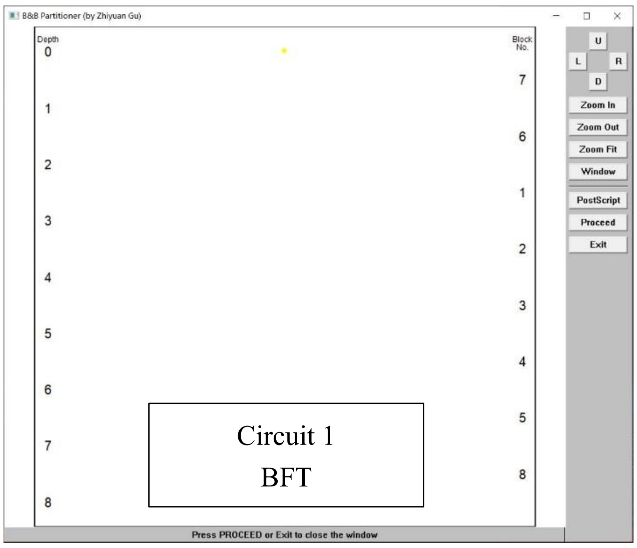
The original lower bound function was just returning the current cut size of the node. This gives no insight into future possible cuts. The later version takes into consideration a special case that guarantees cut size increase in the subtree of the node. Find if there are two "cared" nets that are not yet creating cuts while they share a block that is not yet partitioned. Given another requirement that the unique blocks in the two nets are more than a single chunk can hold, then a cut must be created in the future by the shared block in the two nets. In other words, the two nets not only cannot be completely separated into two chunks on their own but also cannot be put together in one chunk. This early cut estimate can increase the lower bound and thus the probability of getting pruned. Result from circuit 3 has shown a total visited node count dropping from 461,337 to 24,164.

iii. Upper bound function

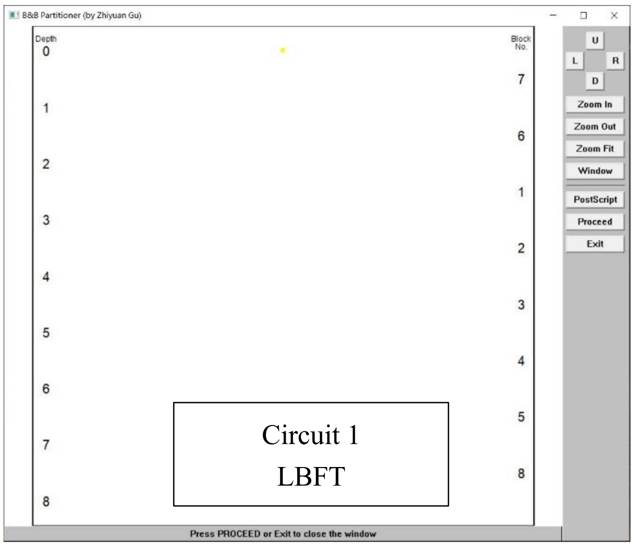
While introduced in class that an upper bound function could be applied to speed up termination of block partitioning close to the end and update of best solution, the method is not included in this work considering the possible improvement. As is introduced above, the test circuits are very inflexible, meaning there are little change that the partitioner could make regarding the final cut size. As a matter of fact, the initial best solution is the final best solution for the first three circuits. Upper bound function is not going to work as expected because there is no room to find a better solution. The first three circuits actually terminate before even reaching the end of "cared blocks" because of sorting according to block fanout. The cut size of nodes degrades quickly giving no chance for an upper bound which is even worse than the worst to work.

2. Result

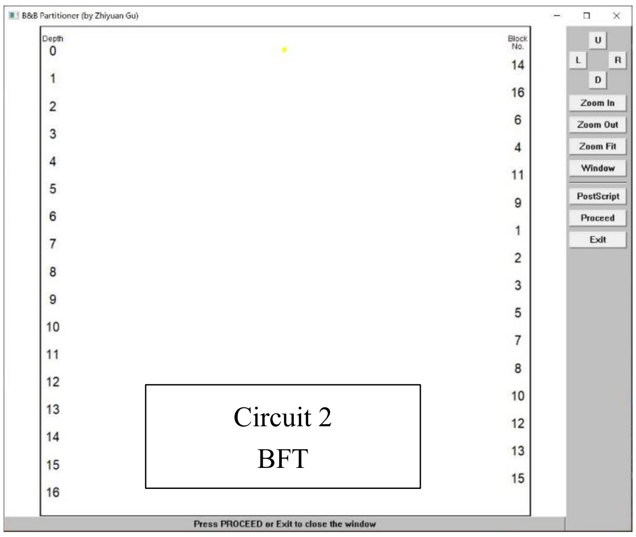
a) Paper plot



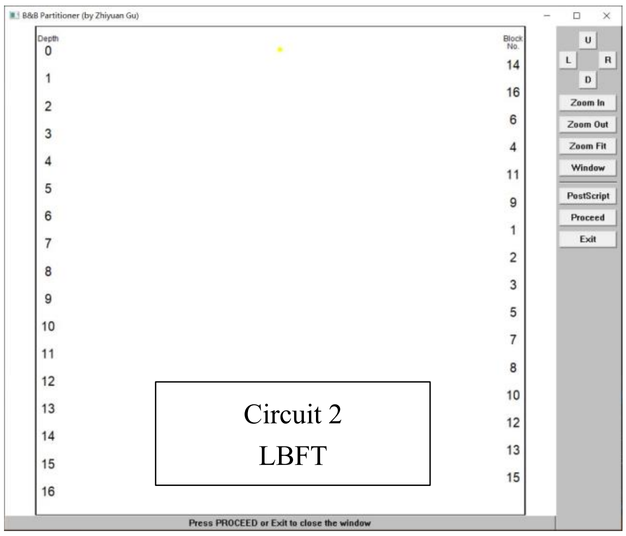
Press PROCEED or Exit to close the window



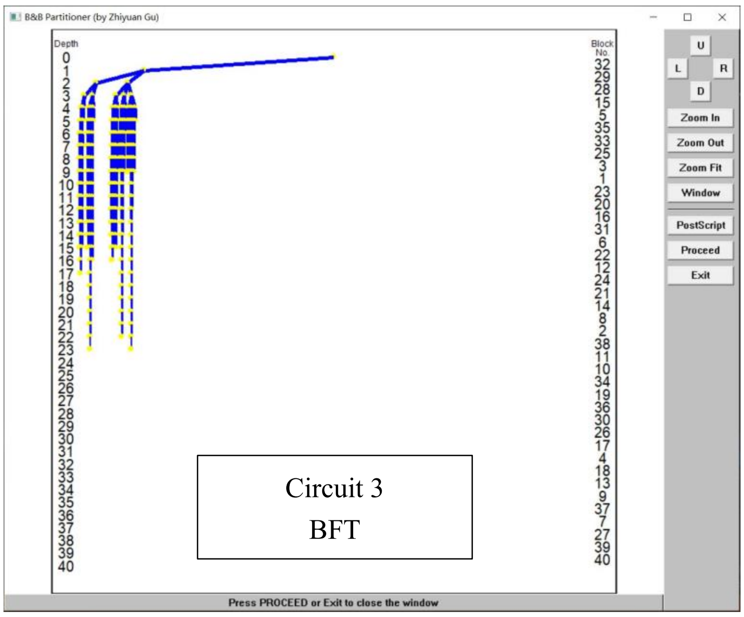
Press PROCEED or Exit to close the window



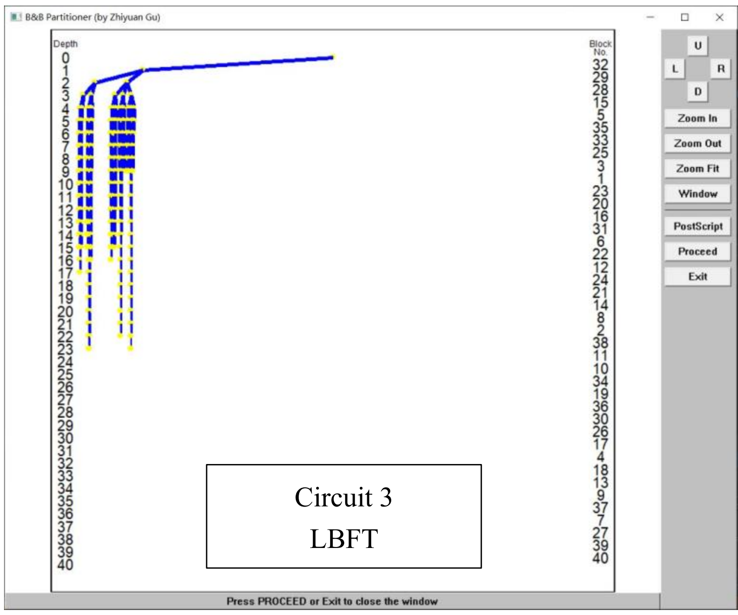
Press PROCEED or Exit to close the window



Press PROCEED or Exit to close the window



Press PROCEED or Exit to close the window



Press PROCEED or Exit to close the window

b) Run-time, visited tree node count & cut size

i. Breadth First Traversal

	cct1	cct2	cct3	cct4
cut size	9	19	38	N/A
visited nodes	1	1	24,164	N/A
run-time	0ms	0ms	1510ms	N/A

ii. Lower Bound First Traversal

	cct1	cct2	cct3	cct4
cut size	9	19	38	N/A
visited nodes	1	1	24,164	N/A
run-time	0ms	0ms	1820ms	N/A

3. Summary and discussion

1) Circuit 4 seems to be too complicated to be solved in a reasonable amount of time. The conclusion is that B&B partitioning is not suitable for big circuits.
2) Both the partitioning for circuit 1 and circuit 2 can be finished by visiting just one node, the initial root node. For circuit 1, 9 nets are bound to create cuts because they could not be put in a single chunk. And that lower bound is equal to the initial solution, leading to pruning in the beginning. For circuit 2, the initial bound cut size was 18 but further estimate into future shows the two "cared" nets left are definitely going to create a cut. As a result, the lower bound of the root node becomes 19 which is equal to that number of initial solution.
3) The three circuits all stop before reaching the bottom of tree. This is because there is no better solution than the initial best solution. All branches are pruned to avoid visiting useless nodes.
4) BFT would branch deeper level by level and the tree is getting wider first and then narrower. This is because pruning is more likely to take place when nodes go deeper. On the other side, LBFT reaches the possible deepest nodes very quickly but spends more time to explore branches from nodes above later when running out of hopeful nodes. Run-time of LBFT is slightly more than that of BFT. This is reasonable as insertion and erasing of priority queue take more time than ordinary queue. However, the run-time difference is not much because the total number of nodes visited are the same.
5) Tree traversal order does not impact the results of the three test circuits. The main cause is the fact that initial best solution equals to final best solution. None of the circuits can find a better solution, update best solution and reduce the number of visited nodes. The test cases seem to be extreme circuits which prefer certain optimizations while remain too hard to solve without those optimizations. To better show the influence of LBFT, I would suggest more flexible circuits with nets of smaller fanout or less complicated circuit than circuit 3 and circuit 4 in the problem of 4-way partitioning.