ECE1387

Project 3

Branch and Bound Optimization Applied to the 3-Way Partitioning Problem

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Stu:Dongxu Ren

1. Branching tree and data structure

For the branching tree, I used the standard trinomial tree and each tree node has at most three branchs which looks approximately like the graph in fig one.

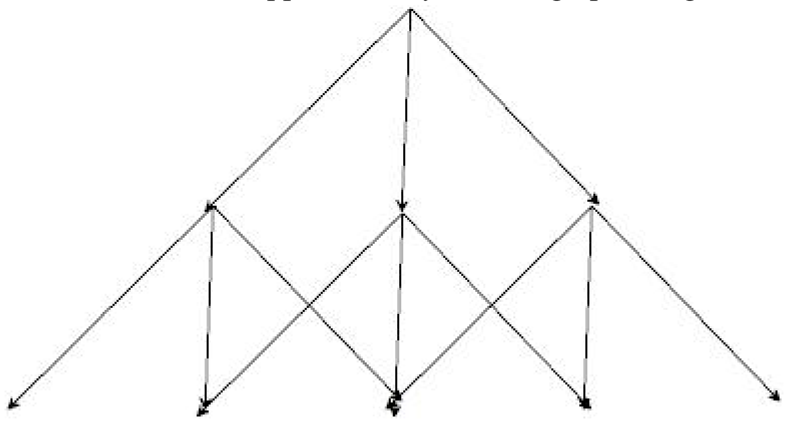


Fig one. branching tree

As we need to seperate the blocks into three partitions, which means that we need a symbol for each one of the three positions(left, middle and right). From my point of view, the most straight forward way is to give each position a different number and I use 1 to represent a block has a postion in the left and 2 for a block in the middle and 3 for a block in the right, that part is “block data” in the following graph, and that part of data should has the same length of block number such as 12 for circuit 1 and 18 for circuit 2.

In addition, apparently we need some other data to tell if a net has already been cut for the reason each net cut can only be count as 1 for once(at first I mistakenly double count the cut size and built a partitioner based on a different algorithm in that way the best for circuit 1 is 62 and circuit 2 is 104), thus we need to mark the net if it has already been cut, and that part is “net data” in the graph, if a net has already been cut, then it is marked as “1” otherwise it is marked as “0”.

After that another thing that we need take into consideration is the number of blcoks in each position, because to get an even partition each position must has one third of the total blocks. Then we need three datas to count the number in each position and eliminatet the illeagal partition.

Finally, we need two datas to count the cut size and a prediction of total cut size(algorithm will be disgussed later).

block data	net data	number of left middle and right(3bits)	cut size and prediction cut size(2bits in total)

Fig two.data structure

2. Initial Best Solution

The goal of the initial partition is to find a partition which is smallest as possible while do not require much calculations, and I came up with two solutions to get the initial best.

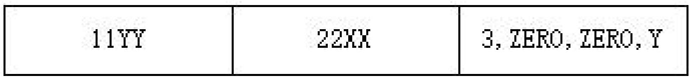
Solution One:

One possible way to find the initial solution is to put the block which tends to have the most connection between each other in the same position. Thus we need to find the connection of each block between other blocks. To achieve this, I bulit a data

matrix which contains the data of connection among each other and find the two blocks which has the biggest connection (for example: block 1 and block 5 are connected through 4 different nets and that 4 number is the biggest connection). However, another problem is that the circuit may has several pairs of blocks that has the biggest connection(4 in the example) and we cannot tell which block to locate first. My solution to this problem is to count the number of nets that each block is connected to and put the pair which has the biggest net connections in the first place, for example: block 4 and block 5, block 7 and block 9 both connected by 4 different nets, block 4 and 5 has a fanout of 10 and 8 seperately and block 9 and 4 has a fanout of 6 and 8 then we should choose block 4 and 5 to be placed first because after decrease the number of 4 nets that they are connected to, they has more nets uncutted. In addition, I tend to put the net which has the most fanouts in the same position in order to minimize cut size. Thus after defining the locating order we simplly put them into the same position until that position is full. I achieved an initial cost of 18 for circuit 1 using this method.
Solution Two:
Another way to do the initial partition is to queue up the blocks by fanouts and put the blocks that has the biggest fanouts in the same position and so on until each block is full, surprisingly this method achieved a better initial cost than the first solution, for circuit 1 the initial best is 16 and I used this algorithm to calculate the initial best in the following calculation.

3. Bounding Functions

The simplest bounding function is the method disgussed in class that the lower bound is calculated by the cut size of the partial solution, but that is not enough when dealing with a big circuit. Thus, firstly I implemented the algorithm above and then came up with another way to make the lower bound as big as possible and I call this method as "cut size prediction"
From my point of view, one efficient way to pull up the lower bound is to estimate the total cut size based on the partial solution that we already obtained. For example:we have a partial solution of "11232" which means we have already determine the position of five blocks in the circuit, and let's say that there are 12 blocks in total which means that there are 4 blocks in each position and 7 blocks has not determine their position yet and let's imagine that block in the fifth position has no connection to 2 blocks in the circuit then I may say the rest cut size for this partial solution is no smaller than 12-5-2-2=3*weight;Weight is the estimation of cut size between block in the fifth place and the rest circuit.



11YY	22XX	3, ZERO, ZERO, Y

Fig.ZERO is the block which has no connection with fifith block, Y is the block that must has a cut number with fifth block, X is the block that are assigned the same location with fifith block

At first, 12-5 is the number of rest undecided blocks and we can see that in position 2 there are two spaces left and let say the two blocks which have the most connection with fifth block are put in position 2 as well so there are no connection between them

and let the two blocks which has no connection with fifith block in position 1 or 3 and thus there are three blocks left that definately has a cut size of non-zero and we give them a weight bigger than 1(but not too big, I did not calculate the exact number because that would take much calculation).
I tested this bounding function of circuit 1 and circuit 2 when the weight is no bigger than 2, the branch and bound algorithm can find the same best solution as the cut size bounded one.

4. Plot Results

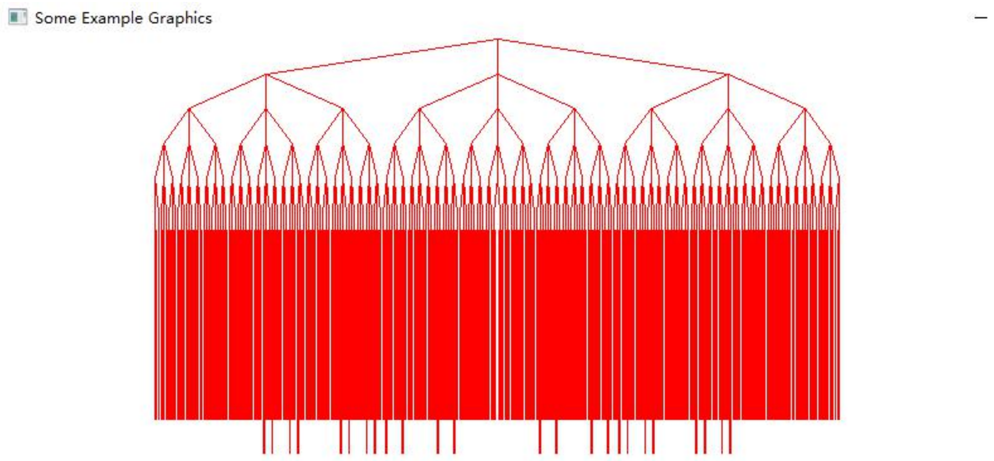


Fig.cct1 depth first using cut size as lower bound

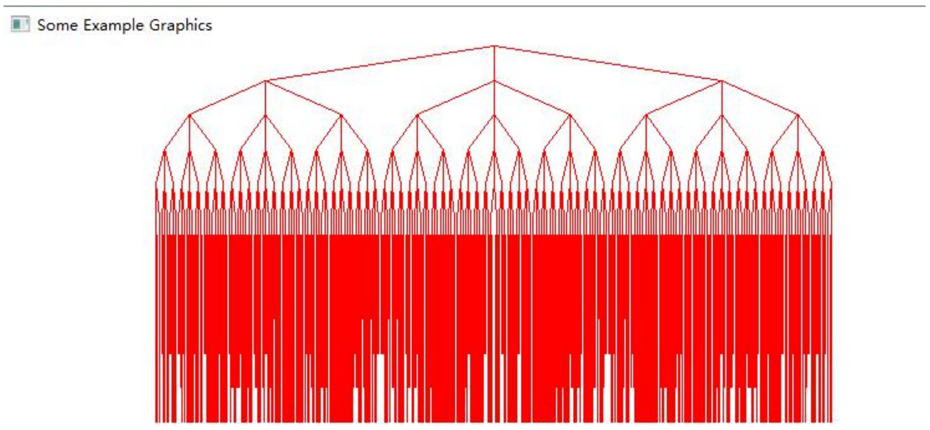


Fig.cct1 lower bound first using cut size as lower bound

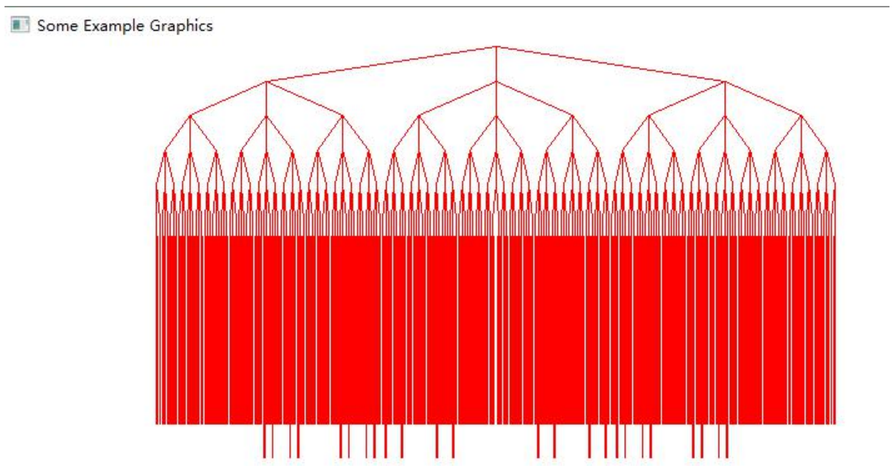


Fig.cct1 depth first using prediction bounding function

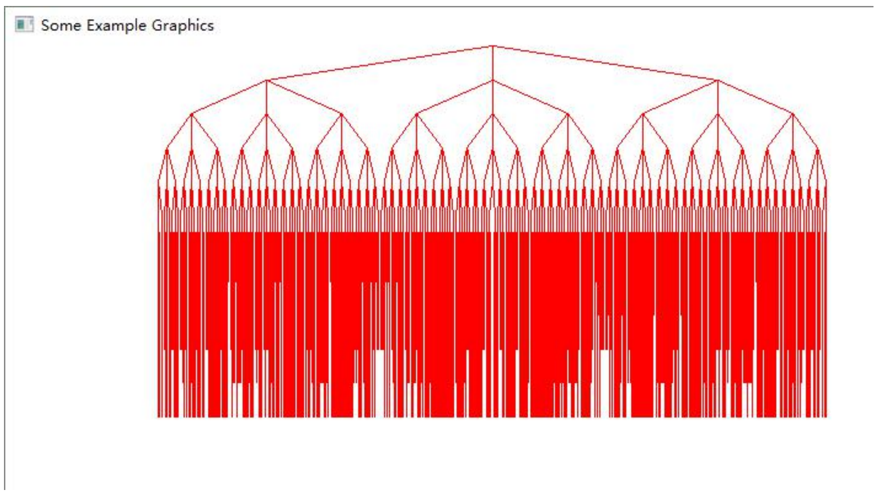


Fig.cct1 lower bound first using prediction bounding function

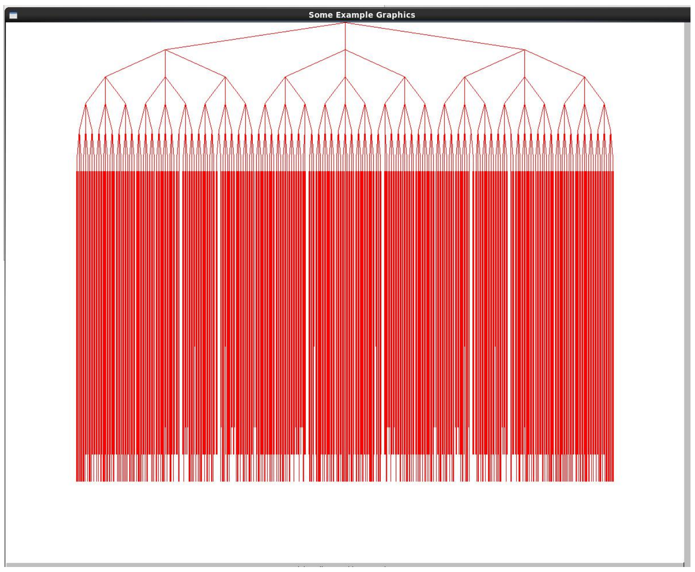


Fig.CCT2 Lower Bound First prediction weight=1

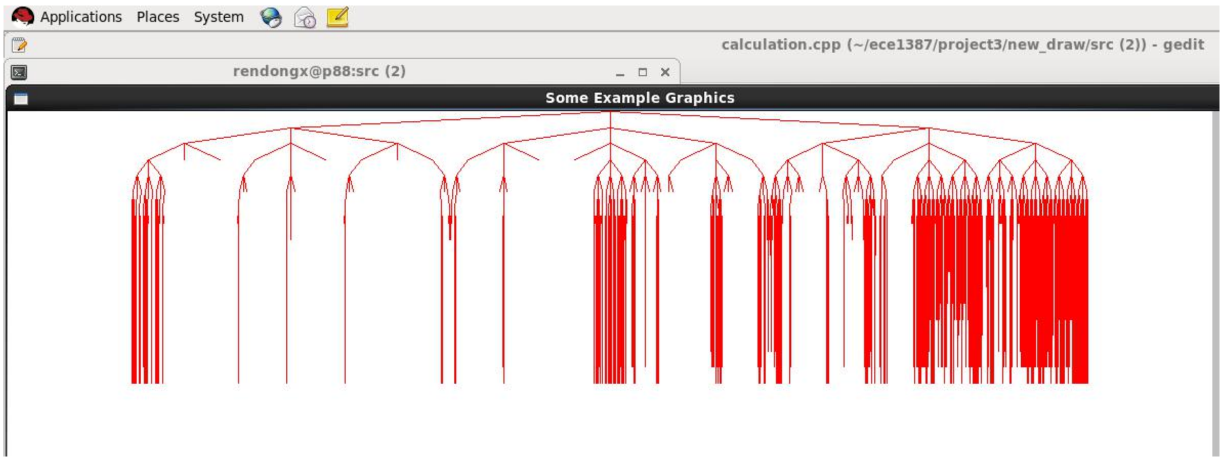


Fig.cct2 with a weight of 2

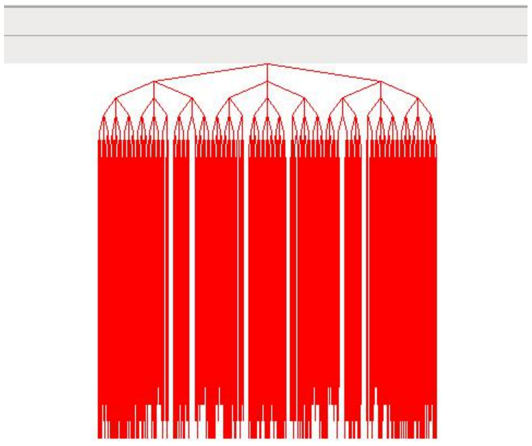


Fig.cct3b Lower bound first weight=2

5. Run Time

I measured the run time by using the chrono system_clock of linux and got the following results(these values are absolute values, calculated by the subtraction of start and end of system clock).

	Cct1	Cct2	Cct3b
depth	6192155		
Low bound weight=1	697730	240446672	
Low bound weight=2	293405	126260281	1930011406

6. Nodes Visited

Lowboundfirst	Cct1	Cct2	Cct3	Cct4
Cut size bound	84282(weight=1)			
Prediction bound	19590(weight=1)	1858566(weight=1)	10264568(weight=2)	
	8337(weight=2)	364463(weight=2)		

7. Cut Size

	Cct1	Cct2	Cct3b	Cct4
Cut size	13	19	57	

When I tried to run cct3b with a weight of 2 for prediction, I found the running was extremely slow at the beginning, I suppose that is because the first taken nodes tends have a lower cost and thus unlikely to be deleted from the decision tree and I started running the program with the best cut size of 71 from my initial best algorithm and it kept running for two hours before it reached the cputime limit and the best solution I could have is a cut size of 57 and then to make the program stop quicker I modified the initial best to be 50 and the program terminated within an hour an it did not provide any successful ppartition with a cut size of 50 which means the best cut size must be bigger than 50 and after that I modified the best to be 53 and it did not provide any solution with that cut size before 2 hours' temination arrives and after that I also tested 56 for the initial best, still it failed to give a partitiion within a reasonable time, thus I suppose 57 must be or very close the the best cut size for cct3b and that is the solution I came up with to make the program terminate.
Solution for cct3b when cut size is 57:
Initial solution:
24,10,18,17,11,25,4,16,8,15,6,13,20,2,22,1,12,5,3,21,23,0,14,9,19,26,27
111112133223212322232123333(1 represent the block is in left, 2 means the block in middle, 3 means the block in right partition, for example: 24,10,18,17,11,4,2,0 are in the left partition)
Solution for cct2 when cut size is 19:
Initial solution:
4,13,15,10,2,7,6,5,1,14,9,11,0,3,12,8,16,17
321123312213221133 each one of the three parts has 6 blocks
Solution for cct1 when cut size is 13:
Initial solution:
9,10,8,0,4,6,3,1,5,7,2,11
331212113322 each section has 4 blocks

8. Comments

From my point of view, obviously the traversal order deeply influence the speed, as we can see that depth first order takes much more nodes than lower bound first order and is significantly slower than the later. I suppose the difference is caused by the timing of best cut size's update, for depth first order the best cut size can only be updated at the last visited nodes while lower bound first can reach to the outcomes earlier and tends to find the low cost partitions at the early stage of the partition process and thus may be able to update the best cut size earlier and delete more illeagal nodes during iteration.