Assignment 3 - Partitioning
EECE 583
Ken Mansfield
March 7, 2016

Info:
Developed in Linux.
Usage: ./Assignment3.exe [filename] [display mode - optional]
Display mode
0 = normal
1 = don't draw the lines that cross the partitions
2 = no graphics (terminal only)

Ex. ./Assignment3.exe apex4.txt 0
Makefile and test script (testAllFiles.sh) provided.

1. Partitioning

Re-using the data structures from Assignment 2 made the development process much faster, however, it also means I inherited a critical bug from that assignment. The professor noted that my final results were not as good as the other classmates. I found out that I was not removing the first item of each line in the netfile (the number of cells in each net). Each net had 1 more cell than it should have. '2 34 55' means that there are 2 cells in the net, but instead I was including 2 as a cell. Because many nets have 2 cells, in apex4.txt for example, I had cell #2 placed in around 850 nets (3/4 of all nets). This was an easy mistake to make because that number is redundant information since we can already determine the number of cells in a line by counting. With a 3 line fix, I fixed this in Assignment 2 and found the final costs were all around 2/3's smaller afterwards.

After reading the netlist the first thing to do is pick a side for each of the cells. Three different ways of doing this were evaluated. The first method I chose was simply to place the first half of the cells on one side, and the second half on the right side. I then tried to see if randomizing the initial placements had an effect on the initial (and final) cut size, which it did. Predictably, changing the seed changed the cut size each time. One possible method for performing K&L might be to run the algorithm with multiple initial random placements and choosing the best result (however this would of course increase the run time). The final method for initial placement I tried was to place all the cells that were connected to many nets on one side, and place the cells connected to fewer nets on the other side. The reasoning was that perhaps cells that were connected to many nets were connected to the same nets. The resulting initial cuts were slightly less for most netlists, but not all.

2. Cost and Gain Functions

KLCostFunction(..) returns the cut size of the entire circuit. As we determined in class, for each net we simply increment the cut if there are any cells on opposing sides of the circuit.

The gain function is what influences the final result of the partition more than any other part of the algorithm. I tried three different approaches to calculating the gain. The first, simplest approach is to calculate the gain much like the cut size is being calculated. For a given cell, for each net it belongs to, increment the gain if there are any other cells in that net that are on the opposite side (and decrement the gain if all the cells in a given net are on the same side). What this does is to influence that cell to switch over to the other side if many of the nets it belongs to are not contained within a single partition.

The second approach I tried was to increment the gain by the number of cells that were on the opposite side (rather than just incrementing by 1) minus the number of cells on the current side. The results of this method were not as good as the first method, so this approach was not investigated further.

The third approach built upon the first approach by identifying its weakness. By simply incrementing the gain if there is a cell on the opposing side we encourage the cell to move to the other side. But it may be better if the cell on the other side moved instead in the case that it was the only cell in that net that is on the other side. We can identify cells that are the only cell on the opposite side by adding a bias weight to influence it make the jump instead of other cells in the netlist. Another problem with the first approach was that many of cells have exactly the same gain value since our gains were very coarse grained and resulted in the algorithm converging in only a couple iterations. Adding a bias adds more granularity, allowing more items to change each iteration and allowing the solution to continue improving all the way until the 6th iteration. I initially set this bias weight to 0.1 to simply break the tie between cells with equal gains. This had a drastic effect on the final cut size, improving results by about 20%. Testing several different values on the results for apex4.txt, I settled on a key cell bias of 1.855 (below). The result is a final but that is about 30% less than without the bias.

Key Cell Bias	**Final Cut Size**
0 (no bias)	472
0.1	403
1.34	356
1.77	346
1.855	**346**
1.9	346
2	385
2.33	349
2.64	361
3.34	370
3.87	382

The last optimization I attempted was to update the gains of nets that had been updated on every swap, however this ultimately made the results worse, and greatly increased the execution time, so this was left out.

3. Efficiency

The algorithm runs extremely fast. The gains are calculated only once before each K&L iteration. The algorithm simply finds the cell with the highest gain and performs the swap. This cell is then removed from the list of unlocked cells so that the list shrinks. The cut is then recalculated and then we then find the next cell with the highest gain. Since there are only 6 K&L iterations, each netlist takes milliseconds to compute.

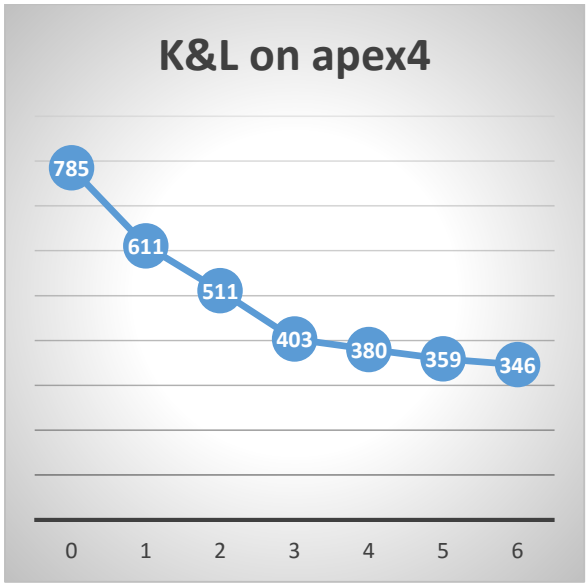
4. Results

Here is a full table of final results. Initial Placement #2 is the placement with the cells connected to many nets on one side, and connected to few nets on the other side.

	Initial Random Placement	**Initial Placement #2**	**K&L 1**	**K&L 2**	**K&L 3**	**K&L 4**	**K&L 5**	**K&L 6**
alu2	126	121	88	65	58	56	53	53
apex1	497	500	377	327	292	266	261	255

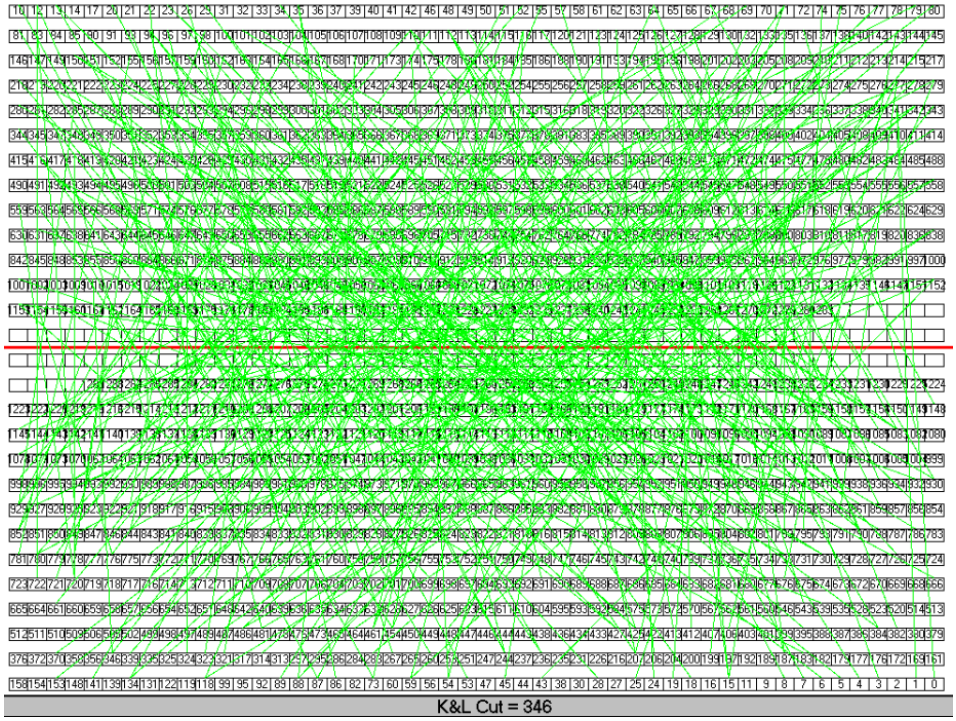
apex4	800	785	611	511	403	380	359	346
c880	153	163	117	88	81	80	79	76
cm138a	12	12	7	4	4	4	4	4
cm150a	26	16	12	12	12	12	12	12
cm151a	14	13	10	8	7	7	7	7
cm162a	22	23	17	16	16	16	16	16
cps	468	484	362	266	238	224	221	217
e64	242	229	174	151	140	133	129	128
paira	521	514	395	339	309	300	297	292
pairb	521	514	395	339	309	300	297	292

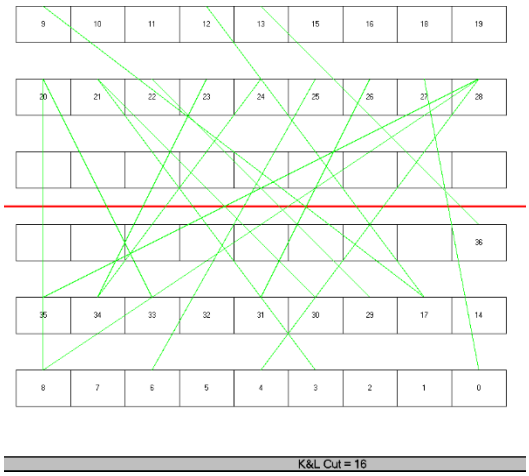
Plotting the results for apex4:

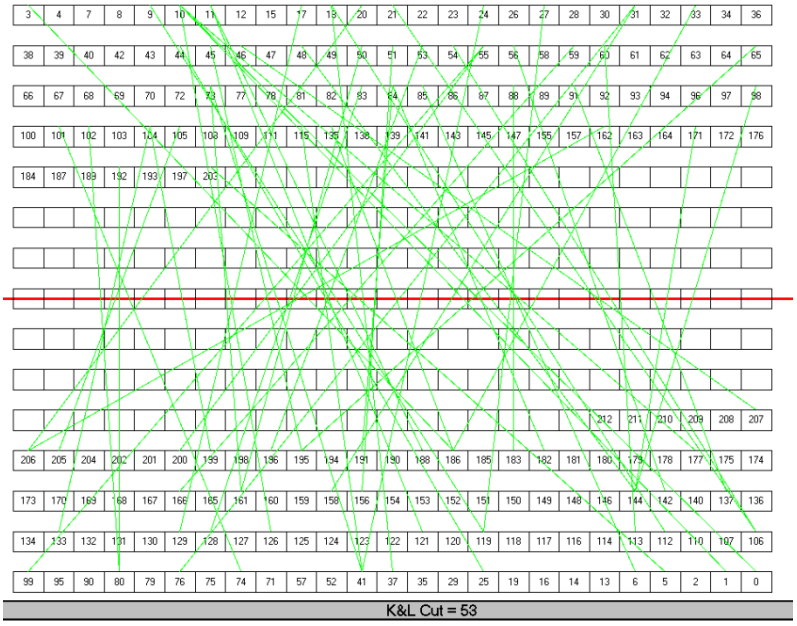


Graphics

Counterclockwise: apex4, cm162a, alu2. The final results are shown. I've also drawn lines that cross the partitions. I do not show any lines that connect cells that are on the same side. The user also has the option to disable the lines, as well as show no graphics at all. For placement, I have just placed the cells in order on their respective sides, they are not placed to minimize net lengths as in simulated annealing. A red line is drawn to show the separation between partitions.







5. Initiative

As mentioned in the previous sections, I tried multiple methods of initializing each partition, and tried multiple ways of calculating the gains. Adding the different methods together have resulted in much smaller final cuts. The algorithm has also been made to be as fast as possible.