Dan Edmond
March 5, 2017

Assignment 3 : Kerrighan-Lin bi-partitioning algorithm for hypergraphs

Algorithm

The bi-partitioning algorithm developed is an enhanced version of the Kerrighan-Lin (KL) bi-partition algorithm designed to handle multi-terminal nets. The based KL algorithm is as follows:

```
Start with random placement
For 6 passes {
    Unlock all nodes
    While some nodes unlocked {
        Calculate gains
        Choose node with the highest gain  whose move does not cause imbalance
        Swap node and lock
    }

}
```

The gain equation specified in the KL algorithm is calculated based on the edge crossings in partitioned graph as follows:

Gain = # incident edges that cross partition - # incident edges that do not

We can consider the contribution that each net has on the gain of a particular node. A net has a +1 contribution if the node crosses the partition and moving the node removes the crossing. A net has a -1 contribution if the node doesn’t cross the partition and moving the node adds a crossing.

When considering the gain contribution of a multi-terminal net to a particular node, this equation only contributes +1 to the gain of a node if it is the only node crossing the partition in the multi-terminal net, and it will contribute -1 if all nodes are contained the in same partition. Outside of these boundary cases, the multi-terminal net contributes 0 to the gain of a node, making no effort to group these terminal together.

To adapt the KL algorithm to work with multi-terminal nets, I have modified the gain contribution of a net to a node for cases other than the -1, or +1 boundary cases described above (cases that would otherwise contribute 0 according to original KL equation):

net_gain = (# cells in opposite partition) / (num cells - 1) -
(# cells in same partition - 1) / (num cells - 1)

The net gain is calculated based on the proportion of cells in each partition, and can be interpreted as the sum of swap attraction force from terminals in the opposite partition and the swap repulsion forces from terminals in the same partition. This modified equation yielded the best results of different modifications attempted and was very simple to implement.

Results

Table 1: Cutset results on provided benchmarks

Benchmark	**Minimum cutset over 100 runs**	**Average cuset over 100 runs**
alu2	28	38.16
apex1	122	165.52
apex4	195	252.21
C880	29	38.83
cm138a	4	4
cm150a	6	6.8
cm151a	4	4.79
cm162	6	6.69
cps	109	146.55
e64	52	65.79
paira/pairb	2	29.20

The algorithm was successful in quickly reducing the cost from initial random placement in all cases. However, the results indicate a substantial spread between the average and minimum cut-set results in several of the benchmarks, which highlights some definite room for improvement. It may not be practical in some applications to start with several different placements. The modified gain equation was intended to be as simple as possible, but could probably be modified to further motivate grouping together nodes in the hypergraph.

Figures 1 and 2 respectively show the alu2 benchmark after initial placement and after algorithm completion.

Figure 1: alu2 benchmark after random partition

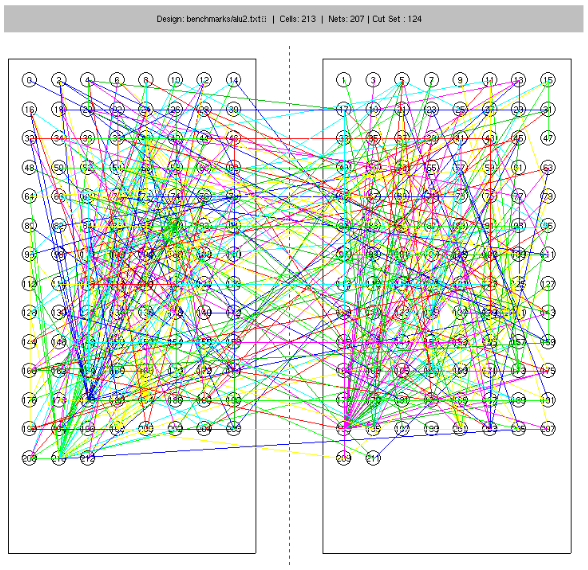
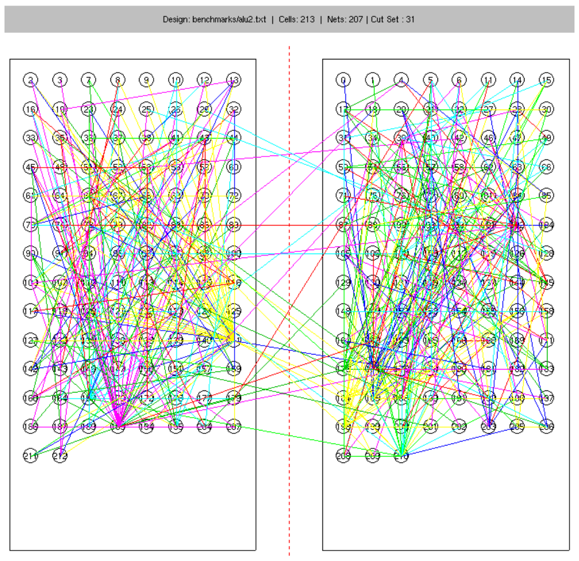

Design: benchmarks/alu2.txt | Cells: 213 | Nets: 207 | Cut Set : 124

Figure 2: alu2 benchmark after algorithm completion


Design: benchmarks/alu2.txt | Cells: 213 | Nets: 207 | Cut Set : 31

Figures 3 and 4 respectively show the apex1 benchmark after initial placement and after algorithm completion.

Figure 3: apex1 benchmark after random partition

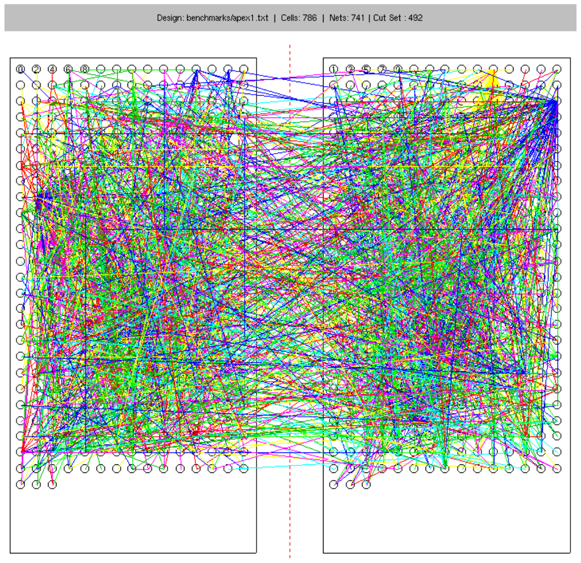
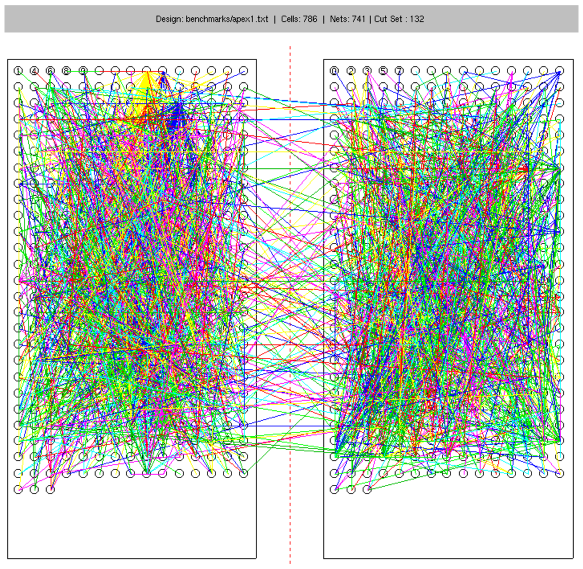

Design: benchmarks/apex1.txt | Cells: 786 | Nets: 741 | Cut Set : 492

Figure 4: apex1 benchmark after algorithm completion


Design: benchmarks/apex1.txt | Cells: 786 | Nets: 741 | Cut Set : 132

User Guide

Clone the following repository:
https://github.com/edmondda/eece583_a3.git

```
> ./assignment3 -h

Kernighan-Lin based bi-partitioning algorithm for hypergraphs
Usage:
  assignment3 INFILE [OPTIONS]
Options:
  -help       (or -h): Print this message
  -gui        (or -g): Enable graphical user interface (GUI)
                       Automatically enabled if postscript is enabled
  -verbose    (or -v): Enable verbose logging,
                       Automatically enabled if GUI is disabled
  -runmode    (or -r): Run mode, followed by one of the following
                       'step'  (or 's'): display results every step
                       'pass'  (or 'p'): display results every full pass
                       'final' (or 'f'): display print final result only
                       Default mode is 'step'
Examples:
  assignment3 cps.txt (using default options)
  assignment3 cps.txt -gui
  assignment3 cps.txt -g -r f
```

In INFILE input to the script must be located in the ./benchmark directory.

In the GUI, press the “PROCEED” button to advance the algorithm. The display will update depending on the specified ‘-runmode’.