- AIM: To develop a Hypergraph partitioning tool based on kernighan-Lin algorithm. The tool is responsible for partitioning the initial hypergraph into a final minimal cutsize hypergraph and also to output relevant statistics such as Nodes in final partitions, initial and final cutsize, d values, e values , i values etc. The tool is developed using C++ language.

- THEORY:
  - Kernighan-Lin Algorithm: KL algorith is a heuristic algorithm used for partitioning of graphs. This algorithm is used to divide the components on a printed circuit board or substrates into partitions, such that the number of connections between the partitions is minimum.
  - The algorithm takes into account few parameters to achieve the final output.
    - I values
    - E values
    - D values
    - G values
    - Initial cutsize
  - I value refer to the internal connections of a node(i.e connections between that node and other nodes of the same partition)
  - E value refers to external copnnection of a particular node (i.e connection between that node and the nodes of other partition)
  - D value refers to the difference between e value and I value of a particular node. The D value for a node x is Dx = Ex - Ix.
  - Initial cutsize refers to number of connection between two partitions.
  - G value refers to gain achieved by swapping two nodes, one from each partition.
    - Consider node x from partition1 and node y from partition 2.
    - If we swap these two nodes, the gain value g is calculated as

      $$Gxy = Dx + Dy - 2(Cxy)$$

      Here, C value is 1 if both x and y are connected and 0 otherwise.

- INPUTS:
  - The inputs to the tool are a nets file and a nodes file.
  - The nodes file contains the nodes and total number of nodes.
  - The nets file contains the information regarding the connections between nodes.(Edges and Hyperedges)

- PROCEDURE:
  - Initially, the nodes are divided into two random partitions.
  - Respective E value, I value and D value of each node are calculated.
  - Then, g values are calculated by swapping every possible pair (one from partition 1 and other from partition 2).
  - The highest g value is taken into account and corresponding g value pair is swaped and locked. These locked pair nodes are no more considered in the further computation of G values. The corresponding g value is marked as g1.
  - Then the d values of all other nodes connected to the locked pair nodes are updated. Updated d value = Dn + 2Dx - 2Dy (here x,y are locked nodes and x,d belongs to same partition before swap)
  - Computing of G values and updation of d values is continued until every node becomes a part of the locked pairs. (g2,g3,g4 and so on)
  - We will sum up all the highest gains. The point where the sum becomes maximum , which then decreases and never becomes greater than the summed up gain at current point, is considered as the end point. The nodes are swapped until this point and the final cutsize is obtained.

    Final cutsize = Initial cutsize - Max sum of gains obtained.

- Sample Nets file and Nodes file :
  - Nodes file:

    ```
    ************************************************************

    UCLA nodes 1.0

    # Created        : Wed Sep 22 14:45:00 1999

    # User           : caldwell@nexus11.cs.ucla.edu (Andrew Caldwell)

    # Platform       : SunOS 5.7 sparc SUNW,Ultra-5_10


    NumNodes :                10

    NumTerminals :             2

    p1 terminal

    p2 terminal

    a3

    a4

    a5

    a6
    ```

a7

a8

a9

a10

****

This indicates that there are 10 nodes in total and the notations of each node are presented.

- Nets file:

****

UCLA nets 1.0

Created : Wed Sep 22 14:45:00 1999

User : caldwell@nexus11.cs.ucla.edu (Andrew Caldwell)

Platform : SunOS 5.7 sparc SUNW,Ultra-5_10

NumPins : 100

NetDegree : 2

p1 B

a4 B

NetDegree : 2

p1 B

a8 B

NetDegree : 2

p1 B

a10 B

NetDegree : 2

p2 B

a7 B

NetDegree : 2

p2 B

a8 B

NetDegree : 2

a3 B

a6 B

****

Here, node p1 and a4 are connected,p1 and a8 are connected and so on.

- DATA STRUCTURES AND FUNCTIONS USED:
  - MAPS
  - VECTORS
  - CLASS: A class named nets is uded which stores information regarding each net, I.e the net degree(number of nodes connected together) and index of nodes. Also, assosciates a set degree function, setnode, get degree and getnode.
  - A nets vector is also used which only stores the c values and corresponding nodes connected.
  - FUNCTIONS:
    - void show_e_i_d_values: The input parameters to this function are a string vector and a double vector. This function is called to display the e values or I values or d values of partitions.
    - vector<double> compute_d_values: The input parameters to this function are two double vectors, which are e values and I values respectively, to compute d values of nodes of a partition.
    - bool find_node_t : The input parameters to this function are two strings. The second string is actually a node and this functions return a true if this node(second string) is present in first string.
    - vector<double> compute_e_values : The input parameters to the function are a string vector, two integers and nets class variable names net_ind.This function is called to compute e values of nodes of a partition.
    - vector<double> compute_ivalues: The input parameters to the function are a string vector, two integers and nets class variable names net_ind.This function is called to compute ivalues of nodes of a partition.
    - void show_nodes: the input parameters to this function are a string vector. This function is usually called to displaty the nodes present in a partition.
    - string Nodes_list : The input parameter to this function is a string. This function returns the nodes present.
    - int find_int : The input parameter to this function is a string. This function is called to find the integer value present inside the string.
    - void openinputfile : This function is used to extract the name of the file from the used and then to open the file.

- PSEUDOCODE:

```
OpeninputFile(nodes,txt);
function read_nodes_file;
OpeninputFile(nets.txt);
function read_nets_file;
Vector<string> partition1;        //initial random partition1
Vector<string>partition2;        //initial random partition 2
function compute_e_values(partition,length_p,total_connections,net_ind); //
both partition 1 and partiton 2
function compute_i_values(partition,length_p,total_connections,net_ind); //
both partition 1 and partiton 2
function compute_d_values(e_values_p,i_values_p); //both partition 1 and
partition 2
Initial_cutsize= Σe_values_p;
for(int i=0;I<length_p1;i++)
{
        function compute g_values;
                string max_g_pair=node_pair; int max_g_value;
                        g_values.insert(make_pair(node_pair,max_g_value));
        lockedpairs_1.push_back();
        lockedpairs_2.push_back();
        //deleting the locked nodes
        function update_d_values;
}
Σg_values;
int max_gain;
final_cutsize= intitial_cutsize - max_gain;
shownodes(Final_partition1,Final_partition2);
```

- RESULTS:

BENCHMARK CIRCUIT	INITIAL CUTSIZE	FINAL CUTSIZE
spp_N151_E192_R8_232	47	44
spp_N179_E225_R11_158	71	40
spp_N189_E227_R6_229	54	49
spp_N193_E227_R11_153	44	33
spp_N199_E232_R11_154	54	33

- CONCLUSION:
  - The initial cutsize, final cutsize, e values,l values, d values, final and initial partitions etc have been successfully computed.
  - Ample usage of vectors,maps and their assosciated loops in this tool, leads to a moderate increase in time complexity.
  - Presence of graphics would have made the tool more user friendly.