Given a Graph, Find the Minimal Cutset

Ajay Saini (IRM2014003)
Dept. of Information Technology,
Indian Institute of Information Technology, Allahabad

Ayonya Prabhakaran(IHM2014502)
Dept. of Information Technology,
Indian Institute of Information Technology, Allahabad

Ankit Mund (IIT2013169)
Dept. of Information Technology,
Indian Institute of Information Technology, Allahabad

Abstract—***The paper details the algorithm, to find minimal cutset in a given graph. A cut is a partition of the vertices of a graph into two disjoint subsets. Our proposed algorithm will give minimum cut weight in given graph, so that by removing edges having sum as minimum cut weight, graph can be divided into two disjoint subsets.***

Index Terms—graphs, cuts, cut-sets,edges.

1. Introduction

A cut is a partition of the vertices of a graph into two disjoint subsets. Any cut determines a cut-set, the set of edges that have one endpoint in each subset of the partition. These edges are said to cross the cut. In a connected graph, each cut-set determines a unique cut, and in some cases cuts are identified with their cut-sets rather than with their vertex partitions.

Min-cut: Min-Cut of a weighted graph is defined as the minimum sum of weights of (at least one) edges that when removed from the graph divides the graph into two groups. A minimum cut is a cut for which the size or weight of the cut is not larger than the size of any other cut. For an unweighted graph, the minimum cut would simply be the cut with the least edges. For a weighted graph, the sum of all edges' weight on the cut determines whether it is a minimum cut. Figure 1 shows an example of a min-cut of a weighted graph having min-cut weight 4.

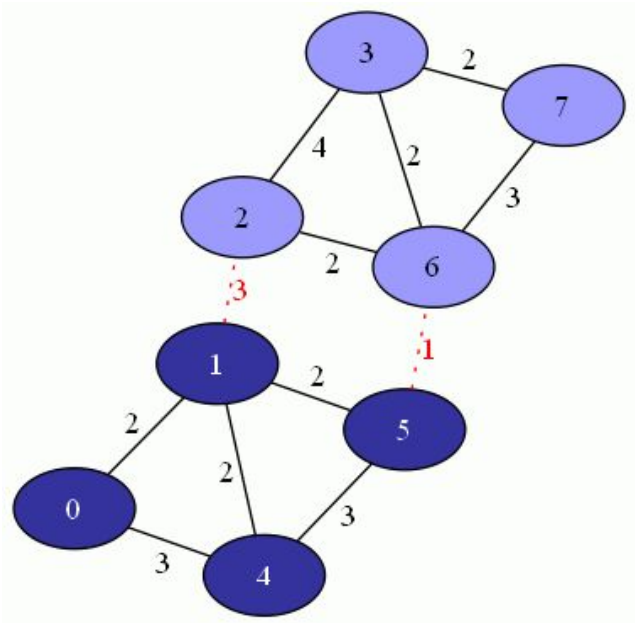

Fig 1. Example of a min-cut of a weighted graph having min-cut weight 4

2. Methods/Algorithms

A. Input

The user inputs the required graph for which minimal cutset is to be found

B. Algorithm

We are using two algorithms here to find minimal cutset in a graph.

1. Karger's algorithm for Minimum Cut
2. Stoer–Wagner algorithm

1. Karger's algorithm for Minimum Cut:

The idea of the algorithm is based on the concept of contraction of an edge *(u,v)* in an undirected

graph *G=(V,E)* the contraction of an edge merges the nodes *u* and *v* into one, reducing the total number of nodes of the graph by one. All other edges connecting either *u* or *v* are "reattached" to the merged node, effectively producing a multigraph. Karger's basic algorithm iteratively contracts randomly chosen edges until only two nodes remain; those nodes represent a cut in the original graph. By iterating this basic algorithm a sufficient number of times, a minimum cut can be found with high probability. Karger's algorithm is a Monte Carlo algorithm and cut produced by it may not be minimum.

Algorithm 1 Karger's algorithm

---

1) Initialize contracted graph CG as copy of original graph.
2) While there are more than two vertices.
   a) Pick a random edge (u, v) in the contracted graph.
   b) Merge (or contract) u and v into a single vertex (update the contracted graph).
   c) Remove self-loops
3) Return cut represented by two vertices.

Pseudocode for the algorithm is given below:

Pseudocode 1 Karger's algorithm

```
Procedure Main
        Input graph G
        Print( kargerMinCut(G) )

Function kargerMinCut (graph)
        V = graph->V, E = graph->E
        Initialize 2d array of size V: subsets
        for v=0 to V
                subsets[v].parent = v
                subsets[v].rank = 0
        vertices = V
        while vertices>2 do
                i = rand() % E
        subset1 = find(subsets, edge[i].src)
        subset2 = find(subsets, edge[i].dest)
        if (subset1 == subset2)
                continue
        else
                vertices--
                Union(subsets, subset1, subset2)
cutedges = 0
for  i=0 toE
        subset1 = find(subsets, edge[i].src)
        subset2 = find(subsets, edge[i].dest)
        if (subset1 != subset2)
                cutedges++
 return cutedges

function find(subset, i)
        if (subsets[i].parent != i)
                Subsets[i].parent=find(subsets,
                subsets[i].parent)
  return subsets[i].parent

Function Union(subsets, x,  y)
   x = find(subsets, x)
   y = find(subsets, y)
   if (subsets[x].rank < subsets[y].rank)
        subsets[x].parent = y
   else if (subsets[x].rank) >    subsets[y].rank)
        subsets[y].parent = x
    else
         subsets[y].parent = x
         Subsets[x].rank++
```

Time complexity: Karger's algorithm can be implemented in $O(E) = O(V^2)$ time.

2. Stoer–Wagner algorithm:

The Stoer-Wagner algorithm is a recursive algorithm to solve the minimum cut problem in undirected weighted graphs with nonnegative weights.

The algorithm works on a method of shrinking the graph by merging the most tightly

connected vertex until only one node is left in the graph and for each step performed, the weight of the merged cut is stored in a list. Minimum value in the list would be the minimum cut value of the graph.

Pseudocode for the algorithm is given below:

Algorithm 2 Stoer–Wagner algorithm

```
Function: MinCutPhase(Graph G, Weights W, Vertex a):
   A ← {a}
   while A != V:
      add tightly connected vertex to A
  store cut_of_the_phase and shrink G by merging
   the two vertices added last

minimum = INF
Function: MinCut (Graph G, Weights W, Vertex a):
   while |V| > 1:
      MinCutPhase(G,W,a)
      if cut_of_the_phase < minimum:
         minimum = cut_of_the_phase
   return minimum
```

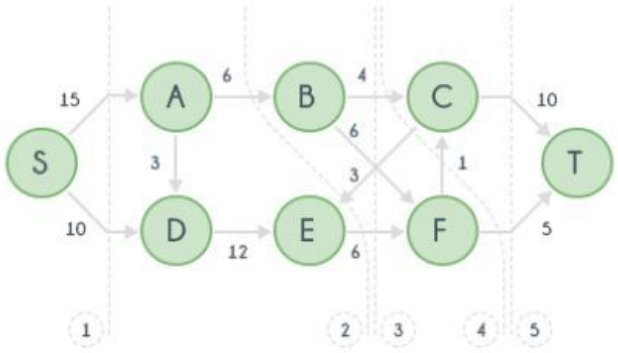

Fig 2. Example of many possible cuts in a graph.

Few possible cuts in the graph of Fig 2. are shown in the graph and weights of each cut are as follows: *Cut1: 25, Cut2: 12, Cut3: 16, Cut4: 10, Cut5: 15.* These are only few possible cuts but considering any valid cut would not have weight less than *Cut4.* Here *Cut4* is minimum cut in given graph.

Time complexity:

The running time of the algorithm MinimumCut is equal to the added running time of the |V|-1 runs of MinimumCutPhase, which is called on graphs with decreasing number of vertices and edges. For the MinimumCutPhase, a single run of it needs at most $O(|E|+|V|log\ |V|)$ time.

Therefore, the overall running time should be the product of two phase complexity, which is $O(|V||E|+|V|^2\ log\ |V|)$.

3. Conclusion

Karger's algorithm and Wagner algorithm are two algorithms that can find minimum cutset efficiently. Karger's algorithm doesn't guarantee to give minimum cutset always because it is based on random vertex selection, while Wagner algorithm is the improved version of Karger's algorithm which chooses the tightly connected vertices and gives minimum cutset value every time.

4. References

1. *https://en.wikipedia.org/wiki/Stoer%E2%80%93Wagner_algorithm*
2. *https://en.wikipedia.org/wiki/Minimum_cut*
3. *https://en.wikipedia.org/wiki/Karger%27s_algorithm*