Notes 11 for CS 170

1 Min Cut

Given an undirected graph $G = (V, E)$, a *cut* is a partition of the set of vertices into two non-empty subsets S and $V - S$. The *cost* of a cut $(S, V - S)$ is the number of edges of E that are incident on one vertex in S and one vertex in $V - S$. The *global min-cut* problem (or, in short, the min-cut problem) is the problem of finding a cut $(S, V - S)$ of minimum cost.

Another way to look at the problem is that we are looking at the smallest set $C \subseteq E$ of edges such that removing those edges from the graph disconnects it. If the cost of a min-cut in a graph is k, then removing any $k-1$ edges from the graph leaves it connected, but there is a way of removing k edges that disconnects it. A graph whose min-cut has cost k is also said to be k-connected. In a k connected graph, there are always at least k different simple paths between any two edges. Computing the min-cut of the graph of a network is then, for example, a way to estimate the worst-case reliability of the network.

More generally, we may be also given a weight function that associates a positive value weight(u, v) to each edge $(u, v) \in E$. In this case, the cost of a cut $(S, V - S)$ is the sum of the weights of the edges that are incident on a vertex in S and a vertex in $V - S$.

Algorithms to compute the min-cut of a graph can be derived from network flow algorithms that we will see later in the course. Such flow-based algorithms are not very efficient, and they are complicated to implement and to analyze. Today, we will see a randomized algorithms that seems just too simple to possibly work, and then we will see that it also has a fairly simple analysis. A straightforward implementation of the algorithm would be quite slow, but it is possible to optimize the implementation somewhat, and come up with a very efficient, and still simple, randomized algorithm.

2 The Contract Algorithms

We introduce the following graph operation (that was implicit in one the minimum spanning tree algorithms that we have seen): given an undirected graph $G = (V, E)$, a weight function weight() and an edge $(u, v) \in E$, `contract ((u,v),G)` returns a graph G' that is identical to G except that the vertices u and v have been replaced by a new vertex uv, and that all neighbors of u and v in G are now neighbors of uv in G'; the weight of the edge (uv, z) in G' is the sum of the weights of the edges (u, z) and (v, z) in G (if any).

2.1 Algorithm

The algorithm is as follows.

```
algorithm Contract(G=(V,E): graph)
    while |V| > 2 do
       pick at random an edge (u,v) ∈ E
          // with probability proportional to weight(u,v)
       (G,w) = contract(G,w,(u,v))
    return names of two remaining vertices in G
```

Basically, the algorithm keeps contracting random edges, so that, after a few rounds, the graph just contains a few vertices that corresponds to sets of vertices in the original graph. When only two "macro-vertices" are left, they represent a cut, and such a cut is returned by the algorithm.

2.2 Analysis

Our strategy will be the following. We fix a min-cut $(S, V-S)$ in the input graph G, and we argue that there is a reasonable probability that the algorithm converges to that particular min-cut.

In order for the algorithm to converge to $(S, V-S)$, it must always contract edges that do not cross the final cut: in fact, this is a necessary and sufficient condition.

Suppose the cost of a min-cut in G is k. Then it follows that, for every vertex v in G, the sum of the weight of the edges incident on v is at least k. (Otherwise, the cut $(\{v,\}, V-\{v\})$ would be better than the optimum.) Consider now the expression $\sum_{v\in V}\sum_{(v,z)\in E}\text{weight}(v,z)$. On the one hand, by the above argument, we have

$$\sum_{v\in V}\sum_{(v,z)\in E}\text{weight}(v,z) \geq \sum_{v\in V} k = n\cdot k$$

where $n = |V|$; on the other hand, the expression is counting twice the weight of every edge, so we have

$$\sum_{v\in V}\sum_{(v,z)\in E}\text{weight}(v,z) = 2\sum_{(v,z)\in E}\text{weight}(v,z)$$

Now, the probability of picking an edge is proportional to its weight, the total weight of the cut $(S, V-S)$ is k, the total weight of all edges is at least $kn/2$, it follows that there is at least a probability $1-2/n$ of doing well in the first step of the algorithm.

If we think about it, we quickly realize that, if the algorithm does well in the first pick, at the second stage it has a graph with $n-1$ nodes where the cut $(S, V-S)$ still is optimum and has cost k. A reasoning similar to the one above will give us that there is a probability at least $1-2/(n-1)$ that we do well at the second step.

Overall, we can see that the probability of converging to the cut $(S, V-S)$ is at least

$$\left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\left(1-\frac{2}{n-2}\right)\cdots\left(1-\frac{2}{4}\right)\left(1-\frac{2}{3}\right) \tag{1}$$

We can write Expression 1 slightly differently as

$$\frac{(n-2)}{n}\cdot\frac{(n-3)}{n-1}\cdot\frac{(n-4)}{n-2}\cdot\frac{(n-5)}{n-3}\cdots\frac{3}{5}\cdot\frac{2}{4}\cdot\frac{1}{3}$$

which is clearly equal to $2/n(n-1) > 2/n^2$.

Now, if we repeat the algorithm $n^2/2$ times, the probability of not finding the cut $(S, V-S)$ at least once is at most

$$\left(1 - \frac{2}{n^2}\right)^{n^2/2} \approx 1/e$$

and if we repeat the algorithm $100n^2$ times, the probability of finding the right cut at least once becomes very close to 1. What we will do, then, is to repeat the algorithm $100n^2$ times, and then take the solution of minimum cost. At least one solution will be optimal, and so we are done.

Notice that, in one run of the algorithm, we have a probability at least $2/n^2$ to converge to each fixed optimum cut: this means that there are never more than $n^2/2$ distinct optimum cuts. (This is in contrast to the case of the minimum spanning tree problem, where, if the weights are not all different, there can be an exponential number of minimal trees.)

The `contract` operator can be implemented in $O(n)$ time, the operator is invoked n times in each iteration of the basic algorithm, and we have $O(n^2)$ iterations of the basic algorithm. In total, the running time is $O(n^4)$. The best known min-cut algorithm based on the `contract` operator and on the above ideas runs in $O(n^2(\log n)^{O(1)})$ time.