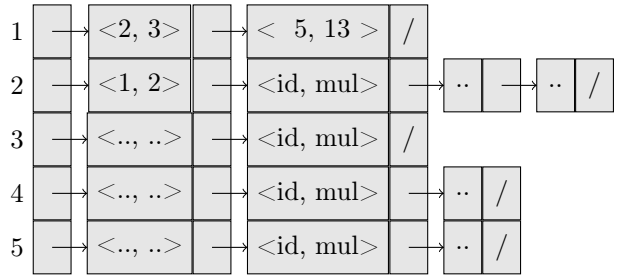
Randomized min-cut algorithm

Consider the randomized min-cut algorithm discussed in class. We have seen that its probability of success is at least $1/\binom{n}{2}$, where n is the number of its vertices.

- Describe how to implement the algorithm when the graph is represented by adjacency lists, and analyze its running time. In particular, a contraction step can be done in $O(n)$ time.
- A weighted graph has a weight $w(e)$ on each edge e, which is a positive real number. The min-cut in this case is meant to be min-weighted cut, where the sum of the weights in the cut edges is minimum. Describe how to extend the algorithm to weighted graphs, and show that the probability of success is still $\geq 1/\binom{n}{2}$. [hint: define the weighted degree of a node]
- Show that running the algorithm multiple times independently at random, and taking the minimum among the min-cuts thus produced, the probability of success can be made at least $1 - 1/n^c$ or a constant $c > 0$ (hence, with high probability).

SOLUTION

First Point
In order to solve this point, we use two data structure, pre-processed, to have linear access on the data. The graph is saved as adjacency list but, instead saving in a list one entry for each edge, we will save a cell that contains the edge ID and its multiplicity.



Another data structure that we will use is a *Virtual Table* V_T. V_T is an array of size n, that contains for each entry the node that includes the i^{th} node. E.g. The item 4 contains the value 2 ($V_T[4] = 2$): this means that the node 4 is included in the node 2 because the nodes are contracted. Moreover, we assume that the adjacency lists are ordered (if not, then order them!).

Random choice of edge To choose a edge we will use another data structure, an array Δ that are initialized as $\Delta[i] = \delta(i)$ (where $\delta(i)$ = the grade of node i). Generate, at random, an integer $r \in [0, 2m]$ and use r to choose the edge:

```
i = 0
while r > 0 do
    if Δ[i] < r then
        r = r − Δ[i]
    else
        choose Δ[i]
    end if
    i++
end while
```

Another, equivalent, approach is to imagine the adjacency lists all concatenated and scroll them until $r > 0$, subtracting from r the multiplicity of the item considered:

```
i = 0
L = all the adjacency lists concatenated
while r > 0 do
    if L[i].multiplicity < r then
        r = r − L[i]
```

else
choose $L[i]$
end if
$i++$
end while

Merging two adjacency lists To merge two nodes we need to scroll the lists of the merged nodes and, assuming the ordered data, we can merge into one list in $O(2n) = O(n)$ (we use an approach like the merge sort where the resulting multiplicity is the sum of the multiplicity, if exists, of the original lists).

Second Point

In the algorithm we change the way we choose an edge to contract: edges with high weight will have a greater probability to be chosen with respect to others with lower weight. We define $weight : E \longrightarrow \mathbb{R}$ as the function that associate the weight to a given edge. We will use the $weight$ function also on subsets of E, intending for $weight(X)$ with $X \subseteq E$ the sum of the weight of all the edges in X.

$$weight(X) = \sum_{e \in X} weight(e) \qquad\qquad where\ X \subseteq E$$

As seen in class, the *min cut* is not unique, nevertheless the sum of weights in all *min_ cuts* is the same, than with a notation abuse we will call it $weight(min\_cut)$. For each edge $x \in E$ the probability to be choose for a contraction is given by its weights normalized with the total sum of the weights of all the edges of the graph.

$$Pr[extract\ x] = \frac{weight(x)}{weight(E)}$$

Now the probability of making an error, when extracting an edge at random, is the probability of extracting one of the "bad" edges, i.e. edges such that their contraction cause a variation in $weight(min\_cut)$. We name BAD the set of "bad" edges.

$$Pr[error] = \sum_{e \in BAD} Pr[extract\ e] = \sum_{e \in BAD} \frac{weight(e)}{weight(E)} = \frac{weight(BAD)}{weight(E)} \tag{1}$$

As usual bad edges are those belonging to every *min cut*, so the total weight of bad edges is less or equal to the weight of *min cut*.

$$weight(BAD) \leq weight(min\_cut) \tag{2}$$

Given a node v, we define $star(v)$ as the set of all the edges touching v. For the same reasons explained in class, for each node v the weight of *min cut* must be less or equal to the sum of the weights of the edges touching v. This is because otherwise we would have a cut, $star(n)$, with weight less than *min cut*, which is absurd.

$$\forall\ v \in V\ .\ weight(min\_cut) \leq weight(star(v)) \tag{3}$$

We can reformulate the *handshaking lemma* for weighted graph in the following way, in which $weight(star(v))$ is something like the "weighted degree" of the node v.

$$weight(E) = \frac{\sum_{v \in V} weight(star(v))}{2} \tag{4}$$

Now we can derive:

$$\begin{aligned} weight(E) &= \frac{\sum_{v \in V} weight(star(v))}{2} && \text{(from 4)} \\ &\geq \frac{|V| weight(min\_cut)}{2} && \text{(from 3)} \end{aligned}$$

$$weight(min\_cut) \leq \frac{2\ weight(E)}{|V|} \tag{5}$$

Finally we have all the ingredients for the proof:

Theorem 1. *Selecting the edges for the contraction according to the probability described, each time we choose an edge the probability of choosing a "bad" edge is less or equal than* $2/|V|$.

$$Pr[error] \leq \frac{2}{|V|}$$

Proof.

$$\begin{aligned} Pr[error] &= \frac{weight(BAD)}{weight(E)} && \text{(from 1)} \\ &\leq \frac{weight(min\_cut)}{weight(E)} && \text{(from 2)} \\ &\leq \frac{\frac{2\ weight(E)}{|V|}}{weight(E)} && \text{(from 5)} \\ &= \frac{2\ weight(E)}{|V| weight(E)} = \frac{2}{|V|} \end{aligned}$$

□

The probability of making an error when selecting an edge is the same as for the case of a normal graph seen in class, so the probability of success for the algorithm (i.e. the probability of choosing well all the times) is still $\geq 1/\binom{n}{2}$.

Third Point

We know that $P(n) \geq 1/\binom{n}{2}$ is the probability of choose well for N times. We have seen also that the probability of error is $\leq (1 - 1/\binom{n}{2})$. Then the probability of having an error each time in N repetitions of the algorithm is

$$leq(1 - \frac{1}{\binom{n}{2}})^N \approx e^{-\frac{2N}{n(n-1)}}$$

Given a c we want the probability of success to be $\geq 1 - 1/n^c$. This is equivalent to ask that the probability of error is $\leq 1/n^c$. Making some calculations:

$$\begin{aligned} \frac{1}{e^{\frac{2N}{n(n-1)}}} = e^{-\frac{2N}{n(n-1)}} &\leq \frac{1}{n^c} \\ n^c &\leq e^{\frac{2N}{n(n-1)}} \\ ln(n^c) &\leq ln(e^{\frac{2N}{n(n-1)}}) = \frac{2N}{n(n-1)} \\ N &\geq \frac{n(n-1)ln(n^c)}{2} \end{aligned}$$

So, for each given c, to make the probability of success at least $1 - 1/n^c$ is sufficient to take N grater or equal than $\frac{1}{2}n(n-1)ln(n^c)$.