Hands-On 4: Algorithm Design A.Y. 2024

Emanuele Buonaccorsi Jhonatan Azevedo
Giacomo Trapani

12 April 2024

1 Exercise 1

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.

One way we can approach this problem, is to use the algorithm presented in class and re-define its steps, considering the use of adjacency lists.

```
While there are more than 2 nodes:
        Choose randomly uniformly an edge (u, v) in E(G)
        Contract the edge G = G - (u, v)
Return |E(u1, u2)|
```

Knowing that the while loop is an iteration over the nodes, it runs at most n times. So, in order to analyze the new algorithm complexity, we must calculate the complexity of its inner steps.

The first step is to randomly chose one of the edges. Since we are using an adjacency list to store the graph, we know that for each node we store at most n elements. And, the random choice of an element from a linked list of length n takes $O(n)$ time. It is also important to consider the case of multigraphs. For those, a simple solution is to store pairs $(node, multiplicity)$ in each adjacency list, so that the number of elements in the list stays the same.

The second step is the edge contraction. To contract an edge (u, v), we need to traverse the two linked lists of the nodes u and v, and remove the related edges. Furthermore, we also need to store the related edges in the new node. Both of these operations take $O(n)$ time.

In consequence, the loop is composed of two steps that take $O(n)$ time. If we're running a loop with an inner complexity of $O(n)$ at most n times, we have a total complexity of $O(n^2)$. □

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 the 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 $\geq 2/n$.

Once again, we are able to use basically the same algorithm. By altering the second step of the algorithm presented before, we are able to find the min-cut of a weighted graph.

$$D(u) = \sum_{(v \in G)} w(u, v)$$

With $D(u)$ as the weighted degree of a node, we may now choose a node u with a probability inversely proportional to $D(u)$. Then, for the second node v, we choose it with a probability inversely proportional to $w(u, v)$, the weight of the edge. Seeing that, each of these choices require $O(n)$ time we keep the same algorithm complexity as discussed before.

To discover the new probability of success, we follow the very same reasoning used in class for the original min-cut algorithm. We call: k the weight of the min-cut, t the sum of all weights, $BadEdges$ the set of bad edges, then, we calculate the probability that the edge (u, v) we have chosen is bad. As such:

$$\Pr[(u, v) \text{ is bad}] = \frac{\sum_{\forall (x,y) \in BadEdges} w(x, y)}{t} \quad \leq \frac{k}{t}$$

Now, observing that $t \geq \frac{nk}{2}$ (since $\sum_{\forall v \in V} D(v) = 2t \geq nk$). We can upper bound our probability by:

$$\frac{k}{t} \leq \frac{2k}{n \times k} = \frac{2}{n}$$

□

Show that running the algorithm N times independently at random, and taking the minimum among the min-cuts thus produced, the probability of success can be made at least $1-(1/n)^c$ for a constant $c > 0$.

We know that the probability of success for the Karger's algorithm, running it N times, is

$$1 - \left(1 - \frac{2}{n \times (n-1)}\right)^N.$$

Using the following relation

$$1 - x \leq e^{-x}$$

We can rewrite the rightmost term as:

$$\left(1 - \frac{2}{n(n-1)}\right)^N \leq \exp\left(\frac{-2N}{n(n-1)}\right)$$

Consequently, for $N = O(cn^2 \log n)$ we get

$$\Pr[\text{success}] \geq 1 - \frac{1}{n^c}$$

□