Principles of Distributed Computing 04/12, 2017

Lecture 8

Lecturer: Mohsen Ghaffari *Scribe:*

In this lecture, we discuss *sublinear-time centralized algorithms* for graph problems. As we shall see soon, the core part of these algorithms is a local procedure, quite similar to the LOCAL distributed algorithms that we have been discussing over the past few lectures.

Sublinear-time (centralized) algorithms are gaining importance with the constant increase in the size of the graph problems that need to be solved[1]. In particular, for a range of problems of interest, the graphs have become so large that spending a linear-time to read the whole graph to solve the problem is well-beyond the time that we can afford. We instead want centralized algorithms that, on an n-node graph, spend $\Theta(1)$ time or at most $\text{poly}(\log n)$ time and provide some meaningful answer (e.g. approximation) about the problem at hand. As we will see, the prototypical way of achieving such results is via *local algorithms*. In a very rough sense, we will poke the graph at a few random places, and determine the solution at each of those places by performing a local procedure, which checks only a small neighborhood around there. At the end, we try to infer something about the overall solution by putting together the solutions at these few randomly sampled places.

Model We consider a graph $G = (V, E)$ with $n = |V|$ nodes and maximum degree Δ. Moreover, we will assume that the graph has no isolated vertex. For the graphs of interest, which are thought to be extremely large and complex, it is typical to assume that degrees are much smaller than the network size. Thus, we will think of Δ as a constant compared to n. The graph is represented by a query-access model, where each query is as follows:

- Query $Q(v, i)$ asks "*who is the i^{th} neighbor of node v*?".

We can also access a random node v, chosen uniformly at random from V. The main performance measure is the number of queries, which we will refer to as the algorithm's *query complexity*. The goal would be to have a query complexity which depends only on Δ (aside from some precision and certainty parameters) and not on n. Moreover, we clearly prefer smaller dependencies on Δ, e.g., $\text{poly}(\Delta)$ is preferred to 2^{Δ}.

1 Approximating Maximum Matching

Consider the problem of computing an approximation of the size of maximum matching. Recall that a matching is a set of edges $M \subseteq E$ such that no two of the edges in M share an end-point. The size of a matching is simplify the number of its edges. Computing a maximum matching, or approximating it, are classic optimization problems which have been studied for decades, since the 1965 work of Edmonds [Edm65], which presented a polynomial-time algorithm for computing a maximum matching.

We next discuss an algorithm that computes a $(2+\epsilon)$-approximation of the size of maximum matching, for a desirably small constant $\epsilon > 0$ — say $\epsilon = 0.01$ — with query-complexity $2^{O(\Delta)}/\epsilon^2$, independent of how large the graph size n is. We note that there are some more involved sublinear-time algorithms that achieve approximations approaching 1. We do not cover those algorithms in this class, for the sake of simplicity.

[1] Does *"Big Data"* ring a bell?

The Algorithm's Outline We will present a simple way of constructing a *maximal matching* M, namely a random greedy maximal matching procedure, which we will be able to simulate in a local manner. This allows us to pick a set S of randomly sampled nodes and infer whether each sampled node $s \in S$ is matched in the maximal matching M or not, using a small number of queries around s. Then, the average fraction of the sampled nodes that are matched gives us an estimation of the size of the maximal matching M. Since any maximal matching is a 2-approximaton of the maximum matching, we get a 2-approximation of the maximum matching. Using a large enough number $k = |S|$ of sampled nodes S, we can adjust the accuracy and certainty of this estimator. We next explain the details of this outline.

1.1 A Local Procedure for Generating a Maximal Matching

Maximal Matching A *maximal matching* is a matching $M \subseteq E$ such that we cannot add any edge of $E \setminus M$ to M, without violating the property that we have a matching. To put it more positively, in a maximal matching M, we have the property that for each edge $e \in E$, at least one endpoint $v \in e$ is incident on a matching edge $e' \in M$. Notice that a maximal matching is not necessarily a maximum matching. However, as we prove next, any maximal matching has a size close to that of a maximum matching.

Lemma 1. *In any graph, any maximal matching has size at least* $1/2$ *of the maximum matching.*

Proof. Consider a maximum matching M^* and an arbitrary maximal matching M. We prove that $2|M| \geq |M^*|$. For that, we perform a simple charging/blaming argument. We start with M^* dollars, where we give one dollar to each M^*-edge. Then, we ask each M^*-edge e to pass its dollar to one of its incident edges in M. Each edge has such an incident M-edge due to the maximality of M. This way, each M-edge receives at most two dollars, at most one through each of its endpoints (why?). Hence, we have redistributed $|M^*|$ dollars on the $|M|$ edges of M in a way that each receives at most two dollars. That means $2|M| \geq |M^*|$. □

Next, we explain a simple algorithm for computing a maximal matching. We will not run this algorithm in its entirety, but rather, we will sample a few nodes and try to locally infer what would be the output of these nodes in the algorithm.

The Random Greedy Maximal Matching Algorithm Suppose that for each edge $e \in E$, we pick a random number $r_e \in [0, 1]$. Then, we process the edges in non-decreasing order of their random numbers r_e, and we greedily add them to the matching M. More concretely, as we go through the (non-decreasingly) sorted order of the edges, each time we add the edge e to the matching M if and only if no edge e' incident to e with a lower random number $r_{e'} < r_e$ has been added to the matching M before. This process always keeps M a matching, and eventually, once we are done with the process, M is a maximal matching.

Approximating the Size of Maximal Matching To estimate the size of the maximal matching that the above algorithm generates, we pick a set S of k randomly chosen nodes (with replacement). The fraction of these nodes that are matched in M is an unbiased estimator of the fraction of vertices that are matched in M. More precisely,

$$\mathbb{E}[\sum_{s \in S} 1_{(\textit{vertex } s \textit{ matched in } M)}/|S|] = 2|M|/n.$$

Thus, we can output

$$\frac{n}{2|S|} \cdot \sum_{s \in S} 1_{(\textit{vertex } s \textit{ matched in } M)}$$

as an unbiased estimator of $|M|$, which by Lemma 1, we know is within a 2-factor of the size of the maximum matching.

The key thing that remains to be discussed is how do we deduct whether a given randomly sampled node s is matched in M or not, without running the entire random greedy maximal matching algorithm. Ideally, we should just check a few things around s and be able to infer whether s is matched in M or not. In particular, we will check each of the edges e incident on s separately, to see whether $e \in M$.

1.2 Checking If $e \in M$ for One Edge $e \in E$, and its Query Complexity

We would like to see if a given edge e is in the matching that the random greedy maximal matching computes. Instead of running the entire random greedy maximal matching procedure to figure out whether $e \in M$ or not, we do something simpler, which should get the same answer: We determine the random value r_e and also all the values $r_{e'}$ for edges e' incident on e. We then *recursively* check whether any of these edges e' for which $r_{e'} < r_e$ is in the matching M or not. If none of them is in the matching, then e is in the matching.

If we use the same random numbers $r_{e''}$, for each edge $e'' \in E$, as in the random greedy maximal matching, we can deduce whether $e \in M$ or not, when we run the entire algorithm.

Analysis

The only remaining aspect is to determine the query complexity of this recursive procedure. A priori, we might end up opening another recursion branch on each neighboring edge and thus we may conceivably end up with a query complexity up to $O(m)$. However, that is not likely. We next prove that for a given edge e, the expected query complexity of the recursive procedure is upper bounded to $2^{O(\Delta)}$, which is independent of n.

Lemma 2. *The expected query complexity of the algorithm for an arbitrary given edge e, which is selected independent of the random values r'_e for $e' \in E$, is at most $2^{O(\Delta)}$.*

Proof. Let $Q(x)$ be the maximum expected number of queries for any edge e, conditioned on $r_e = x$. Notice that $x \leq y$ implies $Q(x) \leq Q(y)$. Let $e_1, \ldots, e_\ell$ be the edges incident on e, where clearly we have $\ell \leq 2\Delta - 2$. We claim that

$$Q(x) \leq (2\Delta - 2) + \sum_{i=1}^{\ell} Pr[r_{e_i} < x] \cdot \mathbb{E}[Q(r_{e_i}) | r_{e_i} < x].$$

This is because we first read all the up to $2\Delta - 2$ incident edges and then, for each neighboring edge e_i with $r_{e_i} < r_e$, we start a new recursion at e_i, which we know in expectation will take at most $Q(r_{e_i})$ queries. What remain is to obtain an upper bound on this recursive relation $Q(x)$.

To simplify the task of upper bounding this recursive inequality, we *discretize* it in some sense, that is, we upper bound it in only a bounded number of (well-spread) places, for $x = \frac{j}{4\Delta}$ for each $j \in \{1, \ldots, 4\Delta\}$. For any $j \in \{1, \ldots, 4\Delta\}$, by setting $x = \frac{j}{4\Delta}$ in the above inequality, we have

$$\begin{aligned}
Q(\frac{j}{4\Delta}) &< 2\Delta + \sum_{i=1}^{2\Delta} Pr[r_{e_i} < \frac{j}{4\Delta}] \cdot \mathbb{E}[Q(r_{e_i}) | r_{e_i} < \frac{j}{4\Delta}] \\
&\leq 2\Delta + \sum_{i=1}^{2\Delta} \sum_{k=1}^{j} \mathbb{E}[Q(r_{e_i}) | r_{e_i} \in [\frac{k-1}{4\Delta}, \frac{k}{4\Delta})] \cdot Pr[r_{e_i} \in [\frac{k-1}{4\Delta}, \frac{k}{4\Delta})] \\
&\leq 2\Delta + \sum_{i=1}^{2\Delta} \sum_{k=1}^{j} Q(\frac{k}{4\Delta}) \cdot \frac{1}{4\Delta} \leq 2\Delta + \frac{1}{2} \sum_{k=1}^{j} Q(\frac{k}{4\Delta}).
\end{aligned}$$

By rearranging the terms around this inequality, we arrive at the following simpler recursion:

$$Q(\frac{j}{4\Delta}) \leq 4\Delta + \sum_{k=1}^{j-1} Q(\frac{k}{4\Delta}).$$

Given this recursion, by a simple induction on j, we can prove that $Q(\frac{j}{4\Delta}) \leq 4\Delta(2^j - 1)$. Hence, $Q(r_e) \leq Q(1) = Q(\frac{4\Delta}{4\Delta}) \leq 4\Delta(2^{4\Delta} - 1) = 2^{O(\Delta)}$. □

Observation 3. *The overall expected query complexity for checking a set S of nodes, to see whether they are matched in M or not, is at most $|S| \cdot \Delta 2^{O(\Delta)} = |S| \cdot 2^{O(\Delta)}$.*

1.3 Adjusting the Sample Set For the Desired Accuracy and Certainty

We usually would like to say that the estimator has a good probability to be a good approximation of its target value. Next, we discuss how by picking a large enough sample set size $k = |S|$, we can satisfy these desires.

Lemma 4. *For any certainty parameter $\delta \in [0, 0.25]$ and any precision parameter $\epsilon > 0$, suppose that we choose a set S of $k = \frac{5\Delta \log 1/\delta}{\epsilon^2}$ nodes at random, with replacement, and check whether each $s \in S$ is matched in the maximal matching M or not. Then, the function*

$$\frac{n}{2|S|} \cdot \sum_{s \in S} 1_{(\textit{vertex } s \textit{ matched in } M)}$$

provides a $(1+\epsilon)$ approximation of the size of maximal matching, with probability at least $1-\delta$. Hence, this is a $2(1+\epsilon)$ approximation of the size of maximum matching.

Notice that by the above observation, the expected query complexity of our algorithm becomes $k2^{O(\Delta)} = 2^{O(\Delta)} \cdot \frac{\log 1/\delta}{\epsilon^2}$.

To prove Lemma 4, we use the Chernoff bound, which is a basic concentration of measure tool that allows us to show that the summation of certain independent random variables is distributed with concentration around its expected value, i.e., it is not likely to deviate from its expectation. Before explaining the proof of Lemma 4, we present the Chernoff bound. The proof of Chernoff bound can be found in most textbooks on randomized algorithms and probabilistic processes. See, e.g., [MR10].

Theorem 5. *(Chernoff Bound) Suppose X_1, X_2, ..., X_ℓ are independent random variables taking values in $[0, 1]$. Let $X = \sum_{i=1}^{\ell} X_i$ denote their sum and let $\mu = \mathbb{E}[X]$ denote the sum's expected value. For any $\delta > 0$, we have*

$$Pr[X \notin [\mu(1-\epsilon), \mu(1+\epsilon)]] \leq 2e^{-\epsilon^2 \mu/3}.$$

We next use this theorem to prove Lemma 4.

Proof of Lemma 4. Define X_i to be the random variable that is equal to 1 if the i^{th} node in our sample set S is matched in M, and is equal to 0 otherwise. Notice that $\Pr[X_i = 1] = \frac{2|M|}{n}$. Hence, $\mu = \mathbb{E}[\sum_{i=1}^{k} X_i] = \sum_{i=1}^{k} \mathbb{E}[X_i] = \sum_{i=1}^{k} \Pr[X_i = 1] \cdot 1 = k\frac{2|M|}{n}$. Therefore, by Chernoff bound, the probability that $X = \sum_{i=1}^{k} X_i$ deviates by more than a $(1+\epsilon)$ factor from its expectation μ is at most

$$2e^{-\epsilon^2 \mu/3} = 2e^{-\epsilon^2 k \cdot 2|M|/(3n)} = 2e^{-\epsilon^2 \frac{5\Delta \log 1/\delta}{\epsilon^2} 2|M|/(3n)} = 2e^{-10\frac{\Delta|M|}{3n} \cdot \log 1/\delta} \leq \delta.$$

The last inequality uses $|M| \geq \frac{n}{4\Delta}$. This fact holds because we have assumed that the graph has no isolated vertex, and thus it has at least $n/2$ edges, and moreover, each edge in the maximal matching M can hit at most 2Δ edges (including itself) and all edges must be hit. □

Remark The algorithm presented here is (a streamlined variant of) a result of Nguyen and Onak [NO08]. See their paper for how this technique can be used for a range of other approximation problems. In the exercises of this lecture, we will see an alternative method for computing an approximation of maximum matching, which has a better query complexity as a function of Δ, but with a slightly worse dependency on the precision parameter ϵ.

References

[Edm65] Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of Research of the National Bureau of Standards B*, 69(125-130):55–56, 1965.

[MR10] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Chapman & Hall/CRC, 2010.

[NO08] Huy N Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *Foundations of Computer Science. FOCS'08. IEEE 49th Annual IEEE Symposium on*, pages 327–336. IEEE, 2008.