COMS E6998-9: Algorithms for Massive Data (Spring'19) Mar 28, 2019

Lecture 18: Size of Minimum Vertex Cover, Linearity Testing

Instructor: *Alex Andoni* Scribes: *Tao Tao, Yifan Shen*

1 Review

1. In last lecture, we give an algorithm *[Gavril, Yannakakis]* that approximates the minimum vertex cover in polynomial time. The output $\widehat{MVC}$ is a vertex cover and $MVC \leq |\widehat{MVC}| \leq 2 \cdot MVC$ where MVC is the size of the true minimum vertex cover. The idea is to relate minimum vertex cover to maximal matching.

2. The result of *[Nguyen, Onak' 2008]* shows that, suppose a given graph $G(V, E)$, where $|V| = n$, is represented as adjacency-list and has maximum degree $\leq d$. Let ϵ be the error parameter, $\widehat{VC}$ be the estimated size of the minimum vertex cover. There exists algorithm computes $\widehat{VC}$ such that:

   1) $MVC - \epsilon n \leq \widehat{VC} \leq 2 \cdot MVC + \epsilon n$.
   2) The expected time complexity is $O(d \cdot 2^d)$.

   In this algorithm, d can be replaced by $d = \frac{m}{n}$. The expected time complexity implies it is a sub-linear algorithm for graphs whose $d \ll n$.

3. The idea of the sub-linear algorithm is to reduce estimating the maximal matching size in G to estimating the size of maximal independent set (MIS) in G', where G' is constructed by G in the following way:

   > The vertices in G' is the edges in G. Suppose e, e' are two arbitrary vertices in G', (e, e') is an edge in G' if and only if e and e' share a vertex in G.

4. We give an algorithm for finding MIS in $G(V, E)$:

Algorithm 1: A Simple Greedy Algorithm for Finding MIS

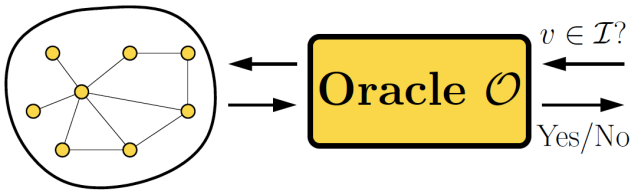
Input : Graph $G(V, E)$
Output: A maximal independent set I

1 Initialize $I = \emptyset$.
2 Consider all $v \in V$ in order, add v to I if and only if all neighbours of v are not in I.
3 Output I as MIS.

2 Estimating the MVC Size in Sub-linear Time

Note that, the goal is to estimate $|I|$ in sub-linear time, so we cannot go through the whole graph to find MIS. The high-level plan is to sample a portion of vertices, calculate the size of vertices that are sampled

to be in some consistent I, and scale the result to the whole graph. To be able to do this, we need to construct a local oracle $\mathcal{O}(v)$ to determine whether a specific vertex $v \in I$ in sub-linear time.
More specifically, the oracle takes queries "Is $v \in I$?", outputs answer in sub-linear time, answers "Yes" if and only if $v \in I$ with respect to some consistent I, and "No" otherwise. This consistent I is a random maximal independent set of the graph, and is independent of the queries.
The oracle can be illustrated as below.



Let $N(v)$ denote the set of all neighbours of a vertex v. The main challenge in designing the oracle $\mathcal{O}(v)$ is the "long chain dependence". That is, to determine whether $v \in I$, we need to determine whether $w \in I$ for all $w \in N(v)$; and in order to determine whether $w \in I$ we need to determine whether $u \in I$ for all $u \in N(w)$; etc. The solution in *[Nguyen, Onak' 2008]* is to assign each vertex v a random value $r_v \in [0, 1]$ when $\mathcal{O}$ first explores v, and store r_v for later use (so I is consistent).

Algorithm 2: Algorithm for Computing Oracle $\mathcal{O}(v)$:

Input : Graph $G(V, E)$, a vertex $v \in V$
Output: "Yes" if $v \in I$ where I is some MIS the oracle implicitly keeps; "No" otherwise

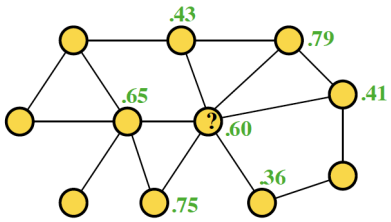
```
if r_v is less than all r_w where w ∈ N(v) then
    Return v ∈ I (i.e. O(v) = "Yes").
for all vertices w ∈ N(v) do
    if r_w < r_v then
        Check recursively if w ∈ I.
if None of w ∈ I then
    Return v ∈ I (i.e. O(v) = "Yes").
else
    Return v ∉ I (i.e. O(v) "No").
```

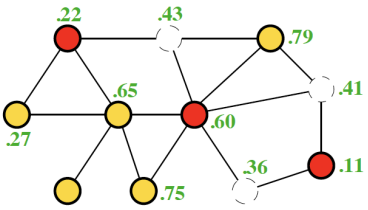
Example: Suppose we want to determine whether the node (with "?") in the figure is in a MIS, the first step of the algorithm (i.e. assign $r \in [0, 1]$ to v and $w \in N(v)$) is shown as below.
The final result is shown as below (The nodes in red are determined to be in MIS).
Analysis of Algorithm 2:

Claim 1. $\mathbb{E}\left[\textit{Number of visited vertices}\right] = O(2^d)$.

Proof:

1) Since r_w is assigned at random, for a path of length k, the probability that the i-th vertex in the path is explored equals to the probability that r_{w_i} is the smallest among the first i vertices, which





is $\frac{1}{i}$. That is, in the path, the probability that the 1st vertex is explored is 1, ..., the probability that the $(k+1)$-th vertex is explored is $\frac{1}{k+1}$. Thus we have

$$\Pr[\text{Fixed path of length k is explored}] = \frac{1}{(k+1)!}$$

2) Since the maximum degree is d, there are at most d^k paths of length k.

3) To explore a vertex v, the expected number of vertices other than v explored is

$$\begin{aligned}
\mathbb{E}[\text{Number of explored vertices}] &= \sum_{k=1}^{\infty} [\text{Number of paths of length } k] \cdot \Pr[\text{This path is explored}] \\
&\leq \sum_{k=1}^{\infty} d^k \cdot \frac{1}{(k+1)!} = \frac{1}{d} \sum_{k=1}^{\infty} \frac{d^{k+1}}{(k+1)!} \\
&\leq \frac{1}{d} \cdot e^d
\end{aligned}$$

So the total expected time is $O(d \cdot \frac{1}{d} \cdot e^d) = O(e^d)$.

□

We can improve the algorithm with the heuristic: When considering neighbours $w_i \in N(v)$, we examine w_i in increasing order of r_{w_i}. That is, we sort all $w_i \in N(v)$ with respect to r_{w_i}, and examine w_i in this order. Once any w_i where $r_{w_i} < r_v$ is in I, the algorithm stops and output $v \notin I$.
With this heuristic, an algorithm given by *[Yoshida, Yamamoto, Ito' 2009]* has the result:

$$\mathbb{E}_{\text{permutations, start vertex}}[\text{Number of recursive calls}] \leq 1 + \frac{m}{n}$$

where $\frac{m}{n} \leq d$.
This gives the expected query complexity for random vertex $= O(d^2)$.
By Chernoff/Hoeffding bound, to get a $\pm\epsilon n$ MIS size estimation, check $O(\frac{1}{\epsilon^2})$ vertices is enough. So we have the following algorithm for $\pm\epsilon n$ MIS size estimation.

Algorithm 3: Algorithm for $\pm\epsilon n$ MIS Size Estimation

Input : Graph $G(V, E)$
Output: An estimation of MIS size
1 Pick $k = O(\frac{1}{\epsilon^2})$ vertices v at random. For all the vertices picked, ask $\mathcal{O}(v)$ "Is $v \in I$?"
2 Estimate MIS size $= \frac{n}{k} \cdot$ [Number of 'Yes']

Note that, in this algorithm MIS is implicitly defined by the oracle.
At last, we return to the high-level plan. The goal is to estimate the size of minimum vertex cover in G by constructing G' and estimating the size of maximal independent set in G'. To determine a vertex v in G is matched or not, we need to examine all its $O(d)$ adjacent edges (i.e. $O(d)$ vertices in G'). Thus, to estimate the size of MVC in G, we need $O(\frac{d}{\epsilon^2})$ oracle queries to G'. The total time complexity is $O(\frac{d}{\epsilon^2} \cdot 2^d)$.
More Results:
[Yoshida, Yamamoto, Ito' 2009][2]: There exists algorithm that can compute size of maximum matchings up to $\pm\epsilon n$ factor in time $O(\frac{d^4}{\epsilon^2})$. The method is based on *[Nguyen-Onak' 2008]* and analysis of the heuristic.
[Onak, Ron, Rosen, Rubinfeld' 2012][3]: There exists algorithm that can compute size of minimum vertex cover up to $\pm\epsilon n$ factor in time $O(\frac{d}{\epsilon^3} \cdot \log^3(\frac{d}{\epsilon}))$ and makes $O(\frac{d}{\epsilon^3} \cdot \log^2(\frac{d}{\epsilon}))$ queries. The method is based on further refinements of *[Nguyen, Onak' 2008]* and *[Yoshida, Yamamoto, Ito' 2009]* and sampling from the neighbor sets.
Lower Bound:
$\Omega(d)$ queries is necessary for estimating MVC such that

$$MVC - \epsilon n \leq \text{Output} \leq 2 \cdot MVC + \epsilon n$$

due to *[Parnas, Ron' 2007]*[4].

3 Linearity Testing

Until above is all our discussion on sub-linear algorithm for graphs. Now we turn to another topic of linearity testing. It is in some sense quite similar to monotonicity testing.

3.1 Introduction

Definition. A function $f : \{0,1\}^n \to \mathbb{R}$ is *linear* if $f(x+y) = f(x) + f(y)$.
To admit the special group structure, i.e. $f : \mathbb{Z}_2^n \to (\mathbb{R}, +)$, we may write linearity condition as $f(x \bigoplus_2 y) = f(x) + f(y)$, which is testing if the function is a homomorphism. In our context, linear and group homomorphism are really interchangeable. Or more conveniently, through group isomorphism, $f : (\{-1,+1\}^n, \cdot) \to (\mathbb{R}, +)$, linearity condition is $f(xy) = f(x) + f(y)$.
To know if a function is linear surely takes linear time. Again, we relax to distinguish between

- f is linaer
- f is ϵ-far from linear

Note that the ϵ-far is the same familiar notion from monotinicity testing. More precisely, $\forall g$ linear, f is different from g on at least ϵ fraction of the entries.

3.2 Motivation

In '90, Blum-Luby-Rubinfeld (RBL) proposed a testing scheme of linearity. A special thing about their method is that it corrects a function in the "blurred" region, i.e. a function that is not linear but ϵ-near, to a linear function. Before we dive into the theorem, we take a look at an important motivation for linearity testing.
Probabilistically Checkable Proof. PCP is a special type of proofs that admits a special property. Informally, we can think of this as:

- φ is a CNF formula.
- We can transform it into φ'.
- φ is satisfiable $\iff$ φ' is satisfiable.
- φ' is not much larger
- φ' satisfiability can be checked by reading $\mathcal{O}(1)$ entries.

It is of important use in proving inapproximability. For example, we can show it is NP-hard to get an approximation upto $\sqrt{n}$ factor of the clique problem.

3.3 Problem

This somehow motivates us to ask, if linearity testing is possible in $\mathcal{O}(1)$ time? A naive canonical algorithm is easily as:

- Pick k random point. Check if $f(xy) = f(x) + f(y)$
- If any of the test fails, we say f is ϵ-far from linear.
- If none fails, we say f is linear.

Note that each time we check the linearity at that point we need 3 queries to f.
Remark.

- If f is linear, it obviously passes the test.
- If f is ϵ-far, $\Pr_{x,y}[f(xy) \neq f(x) + f(y)] > \epsilon$.

Counterexample [Coppersmith].
It is crucial to note that the above paragraphs are not true in general for any mapping between groups. Consider $f : \mathbb{Z}_{3^k} \to \mathbb{Z}_{3^{k-1}}$ and $f(3h+d) = h$, where $h \in \mathbb{Z}_{3^{k-1}}$ and $d = \{-1, 0, +1\}$. Easily, $3h + d$ covers all elements in the domain, i.e. we can split the domain into $\mathbb{Z}_{3^k} = S_{-1} \cup S_0 \cup S_{+1}$. Of course, restriction of f on each of the set is linear, but f itself is 2/3-far from linear, as d really ruins all the linear structure.

On the other hand, Pr[above algo fails on this f] $= 2/9$. To see why, say we picked $x = 3h_1 + d_1$ and $y = 3h_2 + d_2$. Hence $f(x) + f(y) = h_1 + h_2$. But $f(xy) = f\big(3(h_1 + h_2) + d_1 + d_2\big)$. This equals $h_1 + h_2$ if $d_1 + d_2 \in \{-1, 0, +1\}$. Among 9 possibilities, $(-1) + (-1) = -2$ and $1 + 1 = 2$ makes it fails to be linear. The thing to note is the discrepency between ϵ-far and the probability of failure. Indeed we need some special structure and technique to two things consistent. In the next lecture, we shall see how Fourier Analysis can be applied to our scheme.

References

[1] The figures are from:
http://www.mit.edu/ andoni/F15_AlgoTechMassiveData/files/l20_KrzysztofOnak.pdf

[2] Yoshida, Yamamoto, Ito. An Improved Constant-Time Approximation Algorithm for Maximum Matchings

[3] Onak, Ron, Rosen, Rubinfeld. A Near-Optimal Sublinear-Time Algorithm for Approximating the Minimum Vertex Cover Size

[4] Parnas, Ron. Approximating the Minimum Vertex Cover in Sublinear Time and a Connection to Distributed Algorithms