6.889 Sublinear Time Algorithms April 10, 2019

Lecture 17

Lecturer: Ronitt Rubinfeld *Scribe: Yinzhan Xu*

1 Outline

Today we covered the following topics:

- Learning and testing Poisson Binomial Distributions.
- Introduction to local computation algorithms.
- An outline for a local computation algorithm of maximal independent set.

2 Poisson Binomial Distributions (PBDs)

The Poisson binomial distribution is the distribution of a sum of independent Bernoulli variables that are not necessarily identically distributed. More specifically, given a sequence $p_1, p_2, \dots, p_n$ of real numbers in $[0, 1]$, we define

$$\mathrm{PBD}(p_1, p_2, \dots, p_n) : X = \sum_{i=1}^{n} X_i,$$

where X_is are independent random Bernoulli variables, such that $\mathbb{E}[X_i] = p_i$. Note that since p_i can be different for each i, these X_is are not necessarily identically distributed.

Example 1 *When $p_i = \frac{1}{2}$ for every $1 \le i \le n$, $\mathrm{PBD}(p)$ is a binomial distribution.*

Example 2 *Suppose $p_1 = \frac{1}{2}, p_2 = 1$, and $p_i = 0$ for all $i \ge 3$. Then*

$$\Pr[\mathrm{PBD}(p) = k] = \begin{cases} \frac{1}{2} & : k = 1 \text{ or } k = 2 \\ 0 & : \text{otherwise} \end{cases}$$

In the remainder of this section, we will compare PBDs with some other families of distributions, and show two ways to learn PBDs.

2.1 PBDs vs Poisson

We first review the definition of Poisson distributions:

$$\mathrm{Pois}(\lambda) : \Pr[X = k] = e^{-\lambda} \cdot \frac{\lambda^k}{k!}.$$

Many useful and interesting facts about Poisson distributions are mentioned during the class, but they are not necessarily related to this lecture.

Fact 3 $\mathbb{E}[\mathrm{Pois}(\lambda)] = \mathrm{Var}[\mathrm{Pois}(\lambda)] = \lambda$.

Fact 4 *If $X_i \sim \mathrm{Pois}(\lambda_i)$ for $i = 1, \dots, n$ are independent, then*

$$Y = \left(\sum_{i=1}^{n} X_i\right) \sim \mathrm{Pois}\left(\sum_{i=1}^{n} \lambda_i\right)$$

.

Surprisingly, the converse of Fact 4 is also true:

Theorem 5 (Raikov's theorem, [2]) *If the sum of two independent random variables is Poisson-distributed, then so are each of those two independent random variables.*

Fact 6 *If n is large and p is small enough, then $\mathrm{Pois}(np)$ is close to $B(n,p)$.*

The following theorem shows that the PBDs are actually pretty close to some Poisson distributions.

Theorem 7 (Le Cam's theorem, [1]) *For any sequence $p_1, p_2, \ldots, p_n$, where each p_i is a real number in $[0,1]$,*

$$\left|\mathrm{PBD}(p) - \mathrm{Pois}\left(\sum_{i=1}^{n} p_i\right)\right|_1 \leq 4\sum_{i=1}^{n} p_i^2.$$

We omit the proof of Theorem 7. Theorem 7 indicates that Poisson approximation of PBDs is pretty good, but not arbitrarily good: the ℓ_1 distance is determined by p, and we cannot find a Poisson distribution that is ϵ-close to $\mathrm{PBD}(p)$, for an arbitrary ϵ.

2.2 PBDs vs Translated Poisson Distribution

Using translated Poisson distribution, we can obtain a better approximation to PBDs. We first define the translated Poisson distribution $\mathrm{TP}(\mu, \delta^2)$: $Y = \lfloor \mu - \delta^2 \rfloor + Z$, where $Z \sim \mathrm{Pois}(\delta^2 + \{\mu - \delta^2\})$. Here, the notation $\{x\}$ means the fractional part of x, which formally is $x - \lfloor x \rfloor$.

The following theorem states that we can approximate PBDs with translated Poisson distribution.

Theorem 8 ([3]) *Given a sequence $p_1, p_2, \ldots, p_n$, define $\mu = \sum_i p_i$, and $\delta^2 = \sum_i p_i(1-p_i)$. Then*

$$\left|\mathrm{PBD}(p) - \mathrm{TP}(\mu, \delta^2)\right|_1 \leq 2 \cdot \frac{\sqrt{\sum_{i=1}^{n} p_i^3(1-p_i)} + 2}{\sum_{i=1}^{n} p_i(1-p_i)}.$$

When each p_i is within $[c, 1-c]$ for some constant c, the right hand side of Theorem 8 gives $O(n^{-1/2})$; whereas Theorem 7 only gives an $O(n)$ bound.

2.3 Learning and Testing PBDs

In this section, we will describe two methods to learn the PBDs. The first method uses the fact that PBDs are unimodal, and then reduces the problem to learning monotone distributions. The second method uses the structural theorem, which classifies all PBDs to two categories. The structural theorem also implies a way to test PBDs.

In both of these methods, we will use the cover method discussed in the last lecture. For a quick reminder, if we have an ϵ-cover $\mathcal{C}$ of a set of distributions $\mathcal{D}$, then we can learn $p \in \mathcal{D}$ using $O(\frac{1}{\epsilon^2}\log|\mathcal{C}|)$ samples of p.

2.3.1 The Unimodal Method

Theorem 9 *Every Poisson binomial distribution is unimodal over $[n]$. More specifically, there exists $k \in [n]$, such that the distribution is monotonically increasing in $[1,k]$, and monotonically decreasing in $[k,n]$.*

With Theorem 9, we can construct a cover with reasonable size. The number of possible modal k is $O(n)$. For each fixed k, the distribution is a combination of two monotone distributions. Since we can construct an ϵ-cover for monotone distributions of size $O(\frac{1}{\epsilon^{\log n/\epsilon}})$, we can construct an ϵ-cover $\mathcal{C}$ for Poisson binomial distributions of size $O\left(n \cdot (\frac{1}{\epsilon^{\log n/\epsilon}})^2\right)$. Thus, there exists an algorithm that takes

$$O\left(\frac{1}{\epsilon^2}\log|\mathcal{C}|\right) = O\left(\frac{1}{\epsilon^3}\log n \cdot \log\frac{1}{\epsilon}\right)$$

samples.

2.3.2 Structure Theorem

Even though the unimodal method already gives a sublinear time algorithm, the dependency on $\log n$ is not ideal. Also, it does not provide a way to test whether a distribution is PBD. The following theorem resolves these two issues.

Theorem 10 (Structure Theorem) *Every Poisson binomial distribution is ϵ-close in ℓ_1 distance to a distribution Y in one of the following two families:*

- *($\frac{1}{\epsilon}$-sparse) The support of Y is on an interval of length $O(1/\epsilon^3)$.*
- *($\frac{1}{\epsilon}$-heavy Binomial) Y is a binomial distribution on $\Omega(\text{poly}(1/\epsilon))$ i.i.d. variables.*

Learning using Structure Theorem Still we construct a small ϵ-cover. In the $\frac{1}{\epsilon}$-sparse case, there are n possible places for the support. Inside the support, we can enumerate all possible probabilities up to granularity ϵ^4. Thus, the size of the cover would be $O(n \cdot (1/\epsilon^4)^{1/\epsilon^3})$. For the $\frac{1}{\epsilon}$-heavy Binomial case, there are up to n possible choices for the number of i.i.d. variables, and we can enumerate possible probabilities of each variables up to granularity $O(\text{poly}(\epsilon))$. Thus, the size of the cover for this case is at most $O(n \cdot \text{poly}(1/\epsilon))$. The logarithm of the size of the cover would be $O(\log n \cdot \text{poly}(1/\epsilon))$, so number of samples required for learning is $O(\log n \cdot \text{poly}(1/\epsilon))$.

High Level Idea for Removing the $\log n$ dependency For the $\frac{1}{\epsilon}$-sparse case, we can learn the offset of the interval by a few samples since almost all probability mass is in the interval. After we know the interval, the size of the ϵ-cover will not have a factor of n. The $\log n$ dependency for the $\frac{1}{\epsilon}$-heavy Binomial case can also be removed, but we did not cover this in lecture.

Testing using Structure Theorem For the $\frac{1}{\epsilon}$-sparse case, the effective support size is small, so we can test the distribution in time polynomial in the support size. For the $\frac{1}{\epsilon}$-heavy Binomial, we know that almost the entire probability mass of this distribution is centered on the middle $O(\sqrt{n})$ elements, by concentration bound like Chernoff. Therefore, we can test the distribution in $O(n^{1/4})$ time.

3 Local Computation Algorithm

In this section, we switch to a completely different topic. In previous lectures, we care about problems that have a large input size, but the output size is usually small. In reality, many problems also have large output size. In many cases, we don't need to compute the whole output. This section will introduce some example problems that fall into this category, describe the model for local computation algorithms, and outline a locally computable maximal independent set algorithm.

3.1 Examples

Locally Decodable Codes Given a string of bits w, we use some code to encode it to get $\text{ENC}(w)$. The decoding problem is to compute w from $\text{ENC}(w)$. In many applications, we don't need the whole string w. Instead, we only want to compute a single bit w_i. Can we achieve this by only looking at a small number of encoded bits? The answer is yes by using locally decodable codes.

Local Property Reconstruction Suppose the data has some property $\mathcal{P}$, but it was corrupted. Can we change a small portion of the data so that it satisfies the property again? For instance, suppose the data was monotone, but ϵn positions of the data have been modified. We need to change at most $O(\epsilon n)$ places in the data to change it back to monotone. Local Property Reconstruction imposes another constraint to the problem: to output the new value of a certain data point, we can only look at a small number of old data values.

Estimating Graph Parameters Many problems fall into this category, like page rank, communities, dominating set, and maximal independent set. Take dominating set for example. For each node, we need to output whether it is in the dominating set, by only querying a few other nodes.

3.2 Local Computation Algorithm: a Model

A local computation algorithm (LCA) outputs a bit y_i of the output if an index i is queries, and it probes several bits x_j of the input. The queries $i_1, i_2, \ldots$ can happen in either sequential or parallel.

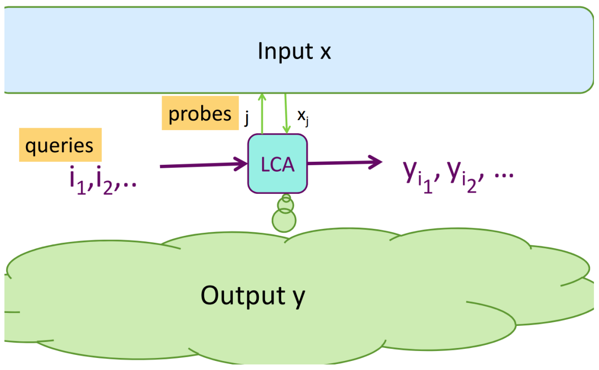

Figure 1: A graphical illustration of the model.

Usually, there can be multiple LCAs that process queries at the same time. One difficulty in this model is that there can be more than one valid output. For instance, if we query whether some vertex is in a maximal independent set, we can always answer "yes" to each individual queries, since every vertex is in some maximal independent set. Therefore, we need some way for all LCAs to have a consistent view.

One common method to overcome this issue is to share some (short) random strings among all instances of LCAs at the beginning. Afterwards, each instance needs to compute independently.

3.3 Maximal Independent Set

In this section, we discuss local computation algorithms for maximal independent set of an undirected graph $G = (V, E)$. The graph is sparse: each vertex has degree at most d. The maximal independent set I is a subset of V, such that any two vertices in I are not connected, and all vertices in $V \setminus I$ are connected to at least one vertex in I.

Unlike *maximum* independent set, which is NP-complete, it is easy to compute a *maximal* independent set. For instance, we can try to add vertices to I one by one. If a vertex v_i is not connected to any vertex in I, then we add it to I; otherwise, we do not add it to I. This algorithm actually produces a lexicographically-first maximal independent set. However, we do not expect to have very fast local computation algorithm to compute the lexicographically-first maximal independent set because of the following theorem.

Theorem 11 *Lexicographically-first-MIS is P-complete. That is, if there exists a parallel algorithm of* polylog(n)*-depth that computes it, then every problem in P has a parallel algorithm of* polylog(n)*-depth.*

A common way to design a local computation algorithm is to simulate a distributed algorithm. If there is a k round distributed algorithm for maximal independent set, then the output of a particular vertex v depends only on inputs and computations of the k-radius ball around v. Since there are at most $O(d^k)$ vertices in the k-radius ball, we can simulate this algorithm in $O(d^k)$ probes. Unfortunately, the best algorithm runs in $O(\log n)$ rounds, which yields an $O(d^{c \log n}) = 2^{O(\log d \log n)}$ LCA.

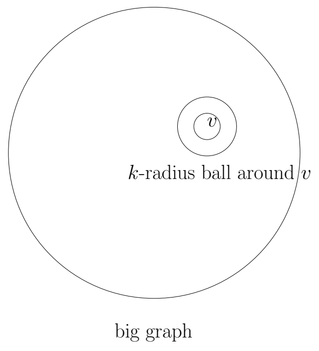

Figure 2: The k-radius ball is small compared to the whole graph.

The main idea for the sublinear time LCA is to use distributed algorithm to "shatter" the graph. Basically, we want to run a distributed algorithm for a small number of rounds, so that we only need to consider connected components of polylog$n$ size.

Many distributed MIS algorithms have the following outline:

Algorithm 1: Distributed MIS

```
1 All nodes start out "live"
2 for each round do
3     Live nodes toss coins
4     Live nodes use coins and interaction with neighbors to decide whether to join MIS
5     If node or its neighbor joined MIS, node "dies"
```

Usually, Algorithm 1 will repeat until all nodes die, which would require $O(\log n)$ rounds. Here, we will repeat until a constant fraction of nodes die. It turns out we only need $O(d)$ rounds in this case. In fact, after we run $O(d)$ rounds of a variant of Algorithm 1, we can prove that with high probability, the remaining "live" connected components are logarithmically small. The proof is delayed to the next lecture and will use a Beck-like analysis for algorithmic Lovasz Local Lemma.

Now again we can use a sequential execution to simulate the above distributed algorithm. Let ALIVE(v) be the sequential simulation of v's local view of distributed algorithm in $d^{O(d)}$ probes. Using ALIVE(v), we can define the following LCA algorithm.

Algorithm 2: LCA(v)

```
1 if v's output is determined by ALIVE(v) then
2     output it
3 else
4     Find v's small "live" connected component via BFS, by calling ALIVE(*)
5     Output "yes" if v is in the lexicographically first MIS in this live component
```

The number of times Algorithm 2 calls LCA is at most d times the size of the connected component of v. Therefore, Algorithm 2 runs in $O(d^{O(d)}\text{polylog}n)$ time.

Next lecture, we will prove the size of the "live" connected components are $O(\text{polylog}n))$.

References

[1] Lucien Le Cam et al. An approximation theorem for the poisson binomial distribution. *Pacific Journal of Mathematics*, 10(4):1181–1197, 1960.

[2] D Raikov. On the decomposition of poisson laws. In *Dokl. Akad. Nauk SSSR*, volume 14, pages 9–12, 1937.

[3] Adrian Röllin et al. Translated poisson approximation using exchangeable pair couplings. *The Annals of Applied Probability*, 17(5/6):1596–1614, 2007.