6.889 Sublinear Time Algorithms February 13, 2019

Lecture 3

Lecturer: Ronitt Rubinfeld *Scribe: Matthew Thomas Sturm*

1 Outline

Today we will continue discussing **estimating the average degree of a graph**:

- Setup and Review from Last Time
- Estimate Average Degree: First Attempt (using only degree queries)
- Estimate Average Degree: Second Attempt (now using neighbor queries as well)

2 Setup and Review from Last Time

The goal of this problem is to find an algorithm that will estimate the average degree of a graph in sublinear time. What we are given is a vector of the degrees of each node as well as the graph in adjacency list representation. Thus, our algorithm must sample the vector of degrees and the adjacency list to form its approximation. As a note about our adjacency list representation, we have the ability to not only query a node for a linked list of its neighbors, but we can query a node for an array of its neighbors, and thus are able to query the j^{th} neighbor of a node in $O(1)$ time. As a reminder from last time, we provide the following definitions and assumptions:

Definition 1 *The average degree of a graph $G = (V, E)$ is defined as*

$$\overline{d} = \frac{\sum_{u \in V} d(u)}{n}.$$

Assumption 2 *We assume that G is simple, i.e., that it has no parallel edges or self-loops. In addition, we will assume that G has no nodes of degree zero/ no isolated nodes*[1]*. Next, we assume that G is not "ultra-sparse", i.e., we assume that the number of edges in G is $\Omega(n)$. Finally, our representation for G will allow the two following types of queries:*

- ***Degree Queries:*** *We can query the degree of any node in $O(1)$ time.*
- ***Neighbor Queries:*** *We can query the j^{th} neighbor of any node in O(1) time.*

In order to approximate the degree, the first idea that might come to mind is simply Naive Sampling: sample S nodes from G and output $\frac{1}{S}\sum_{v \in S} d(v)$. The problem with this is that when using straightforward Chernoff/ Hoeffding analysis, we would need $\Omega(n)$ samples to get the estimates we want.

2.1 Estimating the Average Value of a Vector

It is worth noting that in the general case, it is not possible to reasonably estimate the average value of a vector in linear time. For example, with a vector such as

$$\langle 0, 0, \cdots, n-1, \cdots, 0, 0 \rangle,$$

while the average value is really approximately one, any estimate that runs in sublinear time will very likely conclude the average value is zero.

[1]The assumption that G has no nodes of degree zero is not required for the analysis to follow but is included here for simplicity. The reader may remove this assumption and check that the analysis still goes through.

However, the problem of estimating the average value of a graph *is* possible because graphs must have additional structure that arbitrary vector may not. For example, while the vector of degrees might be

$$\langle 1, 1, \cdots, n-1, \cdots, 1, 1 \rangle,$$

it is just not possible for the vector of degrees to be

$$\langle 0, 0, \cdots, n-1, \cdots, 0, 0 \rangle;$$

if a node has high degree, then those edges must be accounted for on other nodes as well. Thus, this motivates the intuition that due to this extra structure that must exist in a graph, we might be able to achieve an estimate with something like a multiplicative error of two in sublinear time.

2.2 Lower Bounds

2.2.1 The "Ultra-Sparse" Case

Suppose we needed to distinguish between a graph with no edges at all, and thus average degree zero, and a graph with a single edge, and thus average degree $2/n$.

Figure 1: We compare an "ultra-spare" graph with no edges to one with a single edge.

In the second case, we would clearly need $\Omega(n)$ time to find the edge, so any sublinear time algorithm would very likely approximate the degree to be zero, which not within any multiplicative factor of $2/n$.

Thus, in the "Ultra-Sparse" case we would need $\Omega(n)$ time, which is why we require that G is not "ultra-sparse" in Assumption 2.

2.2.2 Finding a $\sqrt{n}$-clique

Suppose we need to distinguish between a graph that is just an n-cycle, and one that consists of a $n - c\sqrt{n}$ cycle and a $c\sqrt{n}$-clique.

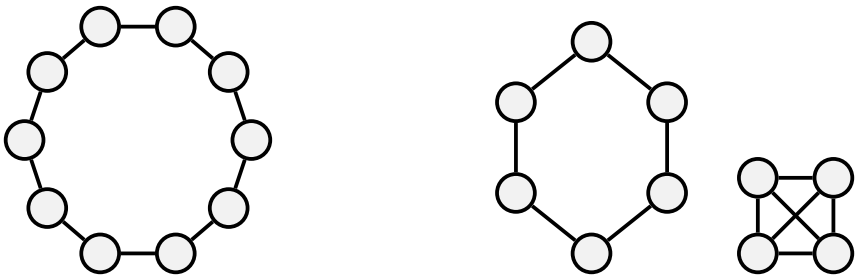

Figure 2: We compare an n-cycle to a $n - c\sqrt{n}$ cycle and a $c\sqrt{n}$-clique

While the n-cycle will have average degree of two, the $n - c\sqrt{n}$ cycle and $c\sqrt{n}$-clique will have average degree

$$\overline{d} = \frac{(n - c\sqrt{n})(2) + (c\sqrt{n})(c\sqrt{n} - 1)}{n} = 2 + c^2 - 3\frac{c}{\sqrt{n}} \approx 2 + c^2.$$

Thus, the two graphs differ in average degree by about c^2, and it will take $\Omega(\sqrt{n})$ time before we expect to sample nodes from the $c\sqrt{n}$-clique. Thus, any algorithm we find that approximating the average degree within a small multiplicative factor must run in $\Omega(\sqrt{n})$ time.

2.3 Bucketing

Because simply sampling random nodes will not work, we need another strategy. One of the main ideas of the algorithm to come is the technique of *bucketing*; we create buckets that contain nodes of similar degree. Then, the goal of our algorithm will be to estimate the average degree by getting an estimate of the size of each bucket. One advantage here is that each group of nodes will now have bounded variance.

More rigorously, we will define $t = O\left(\frac{\log n}{\epsilon}\right)$ buckets and let $\beta = \epsilon/c$ for some constant c. We then let bucket B_i be defined as

$$B_i = \{v | (1+\beta)^{i-1} < d(v) \leq (1+\beta)^i\} \text{ for } i \in \{0, \cdots, t-1\}.$$

We note that $B_1, B_2, \cdots$ may be empty for small i, but this is not a problem, and we will just be concerned with large enough values of i such that B_i is nonempty.

Next, because we can say that the total degree of the nodes in B_i, call this d_{B_i}, satisfies

$$(1+\beta)^{i-1}|B_i| \leq d_{B_i} \leq (1+\beta)^i |B_i|,$$

it follows that

$$\sum_{i=0}^{t-1} (1+\beta)^{i-1}|B_i| \leq \sum_{u \in V} d(u) \leq \sum_{i=0}^{t-1} (1+\beta)^i |B_i|,$$

and further that

$$\frac{\sum_{i=0}^{t-1}(1+\beta)^{i-1}|B_i|}{n} \leq \overline{d} \leq \frac{\sum_{i=0}^{t-1}(1+\beta)^i|B_i|}{n}.$$

Thus, if we can estimate $|B_i|/n$, we can estimate $\overline{d}$.

3 Estimate Average Degree: First Attempt (using only degree queries)

Now that we have set up the problem, we present our first attempt at an algorithm to solve the problem:

Algorithm 1: Average Degree Approximator

Input : Vector of degrees d

Output: $\hat{\overline{d}}$

1 Take sample S from d ;
2 $S_i \leftarrow S \cap B_i$;
3 $\rho_i \leftarrow |S_i|/|S|$ (estimate degree contribution from B_i) ;
4 Return $\sum_{i=0}^{t} \rho_i (1+\beta)^{i-1}$

Here we use the notation $\hat{\overline{d}}$ as our estimate for $\overline{d}$. As some notes about the algorithm, first note that

$$E[\rho_i] = E\left[\frac{|S_i|}{|S|}\right] = \frac{1}{|S|}E\left[\sum_{j=1}^{|S|} X_j^{(i)}\right] = \frac{1}{|S|}\left(|S|\frac{|B_i|}{n}\right) = \frac{|B_i|}{n},$$

as we would hope (where $X_j^{(i)}$ is the Bernoulli random variable that equals one if the j^{th} sample in S is in bucket B_i). Note also that in choosing to output $\sum_{i=0}^{t} \rho_i(1+\beta)^{i-1}$, we will be underestimating $\overline{d}$.

3.1 The Problem with Small Buckets

One problem that becomes apparent with this algorithm is that the size of very small buckets will likely be poorly estimated by $|S_i|$. To illustrate this, consider the following example.

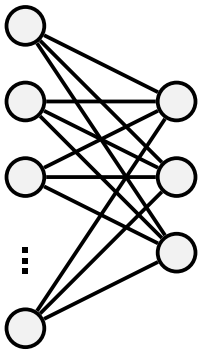

Figure 3: We consider a complete bipartite graph with $n-3$ nodes on the left and three nodes on the right.

In the case of the complete bipartite graph with one group of $n-3$ nodes and the other with three, the average degree will be

$$\overline{d} = \frac{(n-3)(3)+(3)(n-3)}{n} = 6 - \frac{18}{n} \approx 6.$$

However, our algorithm is very likely not going to sample the three nodes on the right and will estimate the average degree to three. In particular, let us choose a to be the integer such that

$$(1+\beta)^{a-1} < 3 \leq (1+\beta)^{a+1}$$

and b such that

$$(1+\beta)^{b-1} < n-3 \leq (1+\beta)^{b+1}.$$

This will give us that $|B_a| = n-3$ and that $|B_b| = 3$ (and that $|B_i| = 0$ for all other i), and thus with high probability (with a sublinear sample size) our algorithm will set $\rho_a = 1$ and return $\hat{\overline{d}} = 3$. The problem here is that B_b contributes $3(n-3)$ to the total degree of the graph, which is half of all edges, but won't be sampled at all!

From this, we can get a sense that if we only use degree queries, we will not be able to do much better than a 2-approximation. If we wish to do better, we will have to also use neighbor queries, which will be covered in the next section.

3.2 Addressing the Problem of Small Buckets

We now present an updated version of algorithm 1 where we address the problem of small buckets. In particular, we simply set $\rho_i = 0$ if $|S_i|$ is "small" and argue that if some buckets are small enough, it is

okay to ignore them (you can think of the intuition for setting $\rho_i = 0$ for "small" $|S_i|$ as a way to make the analysis easier, because we don't need to consider the very small buckets anyway). In addition, we now address how large we need to make $|S|$ with explanations and analysis to follow.

Algorithm 2: Average Degree Approximator (updated)

Input : Vector of degrees d
Output: $\hat{\overline{d}}$

```
Take sample S from d, where |S| = Θ(√n · poly(log n) · poly(1/ε));
S_i ← S ∩ B_i ;
for all i ∈ {0, ··· , t − 1} do
    if |S_i| ≥ √(ε/n)|S|/(c · t) (|S_i| is not "small") then
        ρ_i ← |S_i|/|S|
    else
        ρ_i ← 0
Return Σ_{i=0}^{t} ρ_i(1 + β)^{i−1}
```

We now justify why we chose the sample S to be the size we did. In particular, we want the result that for all of the "big" buckets B_i, $|S_i|$ is a good approximation for $|B_i|$. More specifically, for $\gamma = \Theta(\epsilon)$, we would like

$$(1-\gamma)\frac{|B_i|}{n} \leq \rho_i \leq (1+\gamma)\frac{|B_i|}{n}$$

with probability greater than $1 - 1/(\text{const} \cdot \log n)$ for i such that $|S_i|$ is "large". In order to get this by union and Chernoff bound, it would suffice to have

$$|S_i| \geq \Omega(poly(\log n) \cdot poly(1/\epsilon)).$$

Thus, if we need the cutoff between "big" and "small" to be

$$|S_i| \geq \sqrt{\epsilon/n}|S|/(c \cdot t)$$

as we set in the algorithm, we need

$$|S| > t\sqrt{\frac{n}{\epsilon}},$$

and specifically may choose

$$|S| = \Theta(\sqrt{n} \cdot poly(\log n) \cdot poly(1/\epsilon))$$

as we did. (Why we choose the cutoff between "big" and "small" for $|S_i|$ where we did will be seen in the analysis).

3.3 Analysis

3.3.1 Output Not Too Large

We will first argue that, with high probability, the output will not be too large. First, consider the unrealistic case where for all i, ρ_i is a perfect estimate and $\rho_i = |B_i|/n$. Here, we will have that

$$\sum_{i=0}^{n-1} \rho_i(1+\beta)^{i-1} = \sum_{i=0}^{n-1} \frac{|B_i|}{n}(1+\beta)^{i-1} \leq \overline{d}.$$

However, recall that we chose the size of $|S|$ such that, with high probability,

$$(1-\gamma)\frac{|B_i|}{n} \leq \rho_i \leq (1+\gamma)\frac{|B_i|}{n}.$$

Thus, we may suppose that in a realistic case we will have $\rho_i \leq (1+\gamma)\frac{|B_i|}{n}$ for all i, and thus will have that

$$\sum_{i=0}^{n-1} \rho_i (1+\beta)^{i-1} \leq \sum_{i=0}^{n-1} (1+\gamma)\frac{|B_i|}{n}(1+\beta)^{i-1} \leq (1+\gamma)\overline{d},$$

i.e., that with high probability, our output is not more than a factor of $(1+\gamma)$ larger than the true average degree of the graph.

3.3.2 Output Not Too Small

We next argue that, with high probability, the output will not be too small. First, we again consider the unrealistic case where for all i, ρ_i is a perfect estimate and $\rho_i = |B_i|/n$. Here, we will have that

$$\sum_{i=1}^{t-1} \rho_i (1+\beta)^{i-1} = \sum_{i=1}^{t-1} \frac{|B_i|}{n}(1+\beta)^{i-1} \geq (1-\beta)(1+\beta)\sum_{i=1}^{t-1} \frac{|B_i|}{n}(1+\beta)^{i-1} \geq (1-\beta)\sum_{i=1}^{t-1} \frac{|B_i|}{n}(1+\beta)^{i} \geq (1-\beta)\overline{d}.$$

Now considering a realistic case, we first consider the buckets such that $|B_i|$ is "large". We again have that with high probability, due to our choice of the size of $|S|$, $\rho_i \geq (1-\gamma)|B_i|/n$ for all i such that $|B_i|$ is large[2]. Thus, if all nonempty buckets were "large", we would have that

$$\sum_{i=1}^{t-1} \rho_i (1+\beta)^{i-1} \geq \sum_{i=1}^{t-1} (1-\gamma)\frac{|B_i|}{n}(1+\beta)^{i-1} \geq (1-\beta)\sum_{i=1}^{t-1} (1-\gamma)\frac{|B_i|}{n}(1+\beta)^{i} \geq (1-\beta)(1-\gamma)\overline{d},$$

and we would be done. However, we must now consider the buckets that are "small". The danger is that by ignoring the "small" buckets, we may be undercounting too much and thus get a result that is too small.

3.3.3 How Much are We Undercounting?

We will split our edges up into the following three groups[3]:

- **big-big:** Edges with both endpoints in "big" buckets
- **big-small** Edges with one endpoint in a "big" bucket and one is a "small"
- **small-small** Edges with both endpoints in "small" buckets

Thus, we can see that our algorithm will count the big-big edges twice, count the big-small edges once, and miss the small-small edges entirely. Thus, if we are aiming for a factor of two estimate, the big-big and big-small edges are not a problem, but the small-small edges could cause trouble.

To illustrate what these three categories of edges might look like, consider the following example.

[2]If $|B_i|$ is "large", then we expect both that $|S_i|$ is large enough such that $|S_i| \geq \sqrt{\epsilon/n}|S|/(c \cdot t)$, setting $\rho_i \leftarrow |S_i|/|S|$ rather than $\rho \leftarrow 0$, and that $\rho_i \geq (1-\gamma)|B_i|/n$ specifically.

[3]Here we mean that a bucket is "big" if the algorithm classified it as big by finding $|S_i| \geq \sqrt{\epsilon/n}|S|/(c \cdot t)$ for that particular sample, and small otherwise. Thus, the classification of an edge into one of these three groups could vary depending on the run of the algorithm.

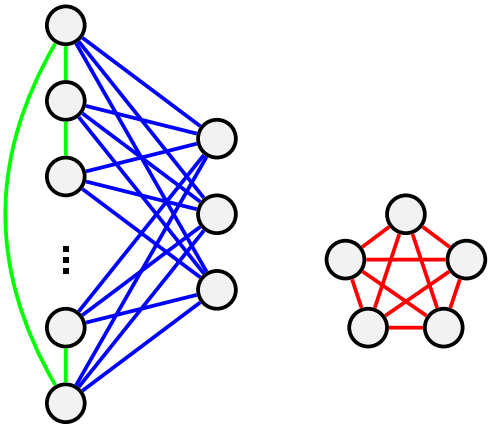

Figure 4: We consider a graph that consists of a 5-clique and a nearly bipartite graph with $n-8$ nodes on the left and three nodes on the right, where the $n-8$ nodes on the graph are also a cycle.

To put names to the buckets in the above example, let us choose a to be the integer such that

$$(1+\beta)^{a-1} < 5 \leq (1+\beta)^{a+1},$$

b to be an integer such that

$$(1+\beta)^{b-1} < n-8 \leq (1+\beta)^{b+1},$$

and c to be an integer such that

$$(1+\beta)^{c-1} < 4 \leq (1+\beta)^{c+1}.$$

Thus, $|B_a|$ will consist of the $n-8$ nodes on the left, $|B_b|$ will consist of the three nodes in the main graph on the right, and $|B_c|$ will consist of the five nodes in the 5-clique. Of these, it is clear that only $|B_a|$ will be considered "big". This will then give us that the green edges between nodes in $|B_a|$ will be big-big, the blue edges between nodes in $|B_a|$ and the nodes in $|B_b|$ will be big-small, an the red edges between the nodes in $|B_c|$ will be small-small.

Thus, we will correctly count the green edges twice, we will only count the blue edges once, and we won't count the red edges at all. To get a sense of how much these small-small edges hurt our algorithm, we may calculate that the average degree of our graph is

$$\overline{d} = \frac{(n-8)(5)+(3)(n-8)+(5)(4)}{n} = 8 - \frac{44}{n} \approx 8.$$

However, our algorithm will likely not sample any nodes from either B_b or B_c, will set $\rho_a = 1$, $\rho_b = \rho_c = 0$, and will calculate the average number of nodes to be approximately five. Because five is within a factor of two of the true average degree, eight, we might hope that this means the small-small edges don't contribute much to the average degree.

3.3.4 Bounding How Much We Lose from the Small-Small Edges

The good news is that the small buckets can't have too many nodes in them, and thus can't have too many small-small edges between them. To be more precise, we have that if $|B_i| > 2\sqrt{\epsilon \cdot n}/(c \cdot t)$, then

$$E[|S_i|] = |S|\frac{|B_i|}{n} \geq |S| \cdot 2\sqrt{\frac{\epsilon}{n}}\frac{1}{c \cdot t}.$$

Because this is twice the threshold required for our algorithm to call S_i "big", by Chernoff we can conclude that the algorithm will declare as such with high probability (it is this analysis that informed

our choice of the threshold in the algorithm). Thus, we can assume that for any "small" S_i that cause $\rho_i \leftarrow 0$, that $|B_i| \leq 2\sqrt{\epsilon \cdot n}/(c \cdot t)$. Thus, the total number of small-small edges, T_S, is upper bounded by

$$T_S \leq \left(\frac{2\sqrt{\epsilon \cdot n}}{c \cdot t} \cdot t\right)^2 = O\left(\frac{\epsilon \cdot n}{c^2}\right) = O(\epsilon \cdot n).$$

Thus, we may finally conclude that if we ignore these small-small edges, they may affect our estimation of average degree by at most a multiplicative factor[4] of $(1+\epsilon)$ or an additive factor of ϵn.

3.3.5 Bringing It All Together

Finally, we may conclude that our algorithm gives us a $(2+\epsilon)$-multiplicative approximation, where the factor of two comes from us only counting the big-small edges once and the factor of ϵ comes from us not counting the small-small edges at all.

Additionally, we observe that our algorithm runs in

$$\widetilde{O}\left(\frac{\sqrt{n}}{poly(\epsilon)}\right)$$

time, requiring that many degree queries (where the notation $\widetilde{O}$ indicates that we are dropping log terms).

4 Estimate Average Degree: Second Attempt (now using neighbor queries as well)

Now that we have seen an algorithm to estimate the average degree that only uses the degree queries, we want to see if we can do better if we use neighbor queries as well. The basic idea of this next approach is that in the first algorithm, most of our error came from only counting the big-small edges once, so we are going to now estimate the fraction of big-small edges and then add to our estimate to account for them. Specifically, our final algorithm is given below.

Algorithm 3: Average Degree Approximator (Final)

Input : Random access adjacency list E and vector of degrees d

Output: $\hat{\bar{d}}$

```
Take sample S from d, where |S| = Θ(√n/ε · t);
S_i ← S ∩ B_i ;
for all i ∈ {0, ··· , t − 1} do
    if |S_i| ≥ √(ε/n)|S|/(c · t) (|S_i| is not "small") then
        ρ_i ← |S_i|/|S| ;
        for all v ∈ S_i do
            Choose a random neighbor, u, of v ;
            X(v) ← {1 if u is in a "small" bucket, 0 otherwise}
        α_i ← |{v ∈ S_i | X(v) = 1}|/|S_i|
    else
        ρ_i ← 0
Return Σ_{i=0}^{t} ρ_i(1 + α_i)(1 + β)^{i−1}
```

[4] We must assume that the average degree of the graph is greater than or equal to one.

The main difference of this algorithm is that we estimate the factor α_i, which is the fraction of edges in B_i that are big-small, and then add the total $\sum_{i=0}^{t} \rho_i \alpha_i (1+\beta)^{i-1}$ to our estimate to account for the big-small edges that we only counted once.

Before we continue, we will explain a new query that appears in algorithm 3. In particular, we make use of a *choose random neighbor* query. The way we implement this is that given a node v and asked to choose a random neighbor, we

- Make a degree query for v, $d(v)$
- Choose a random number $i \in [1, \cdots, d(v)]$
- Make a neighbor query (v, i)

4.1 Analysis

The key piece of analysis that we must do is confirm that we are well estimating α_i using this sampling method and thus that we are well estimating $\overline{d}$. One worry is that we can not uniformly sample random edges from B_i.[5] Rather, we can sample random nodes from B_i (by sampling nodes until we get one that is in B_i) and then sample a random edge from each of those nodes (using the choose random neighbor query defined above). If all nodes in B_i had the same degree, then this would be a random sample of the edges in B_i, but they don't. However, all nodes in B_i have similar degree, so we will argue that them being within a factor of $(1+\beta)$ is good enough.

4.1.1 The Easy Case

In algorithm 3, we attempt to estimate the number of big-small edges in B_i by taking a random sample of nodes from B_i (given by S_i) and then choosing a random neighbor, and thus a random edge, e, from each of those. In particular, let us define

$$X_j^{(i)} = \begin{cases} 1 & \text{if } e \text{ is big-small} \\ 0 & \text{if } e \text{ is big-big} \end{cases}$$

for the j^{th} edge we choose from B_i and define

$$\alpha_i = \sum_j X_j^{(i)} / |S_i|,$$

the average of the $X_j^{(i)}$s.

To check whether this will give a good estimate, let us first optimistically assume that all nodes in B_i have the same degree d. We then have that

$$\Pr[\text{“big-small” edge } e \text{ in } B_i \text{ is chosen}] = \frac{1}{|B_i|}\frac{1}{d},$$

where there is only a $1/|B_i|$ chance we choose a node adjacent to e because only one end of the edge can be in B_i given that e is a big-small edge. We then have that

$$\Pr[X_j^{(i)} = 1] = E[X_j^{(i)}] = \frac{T_i}{d \cdot |B_i|},$$

where T_i is the number of big-small edges in B_i. Thus, $E[X_j^{(i)}]$ is exactly what we would want it to be, and thus α_i would be a good estimate for $\frac{T_i}{d \cdot |B_i|}$ given enough samples.

[5] Note that when sampling random edges from B_i, we mean random “outgoing” edges. Thus, an edge between two nodes in B_i is considered two separate edges for the sake of this sampling.

4.1.2 The General Case

In general it is not true that all edges in B_i have the same degree, but it is true that they are all within a factor of $(1+\beta)$. Thus, we have that

$$\frac{1}{|B_i|}\frac{1}{(1+\beta)^i} \leq \Pr[\text{“big-small” edge } e \text{ in } B_i \text{ is chosen}] \leq \frac{1}{|B_i|}\frac{1}{(1+\beta)^{i-1}},$$

and that

$$\frac{T_i}{|B_i|(1+\beta)^i} \leq E[X_j^{(i)}] \leq \frac{T_i}{|B_i|(1+\beta)^{i-1}},$$

which implies that

$$E[X_j^{(i)}]|B_i|(1+\beta)^{i-1} \leq T_i \leq E[X_j^{(i)}]|B_i|(1+\beta)^i.$$

Thus, by estimating $E[X_j^{(i)}]$ within a factor of $(1+\epsilon)$ we can get an estimate of T_i/n via $\alpha_i\rho_i(1+\beta)^{i-1}$ within a factor of $(1+\epsilon)(1+\beta)$.

4.1.3 Errors in Our Final Algorithm

As a final count of errors in our last algorithm, we have

- A multiplicative factor of $(1+\epsilon)$ from estimating the ρ_i terms
- A multiplicative factor of $(1+\epsilon)$ from estimating the α_i terms
- An additive factor of ϵn from not counting the small-small edges at all