Sampling Weighted Edges with Proportional Probability

Julie Luo
Advisor: Daniel Spielman

December 2018

Abstract

Random edge sampling is a useful technique that can be used to get an idea of what a large graph looks like. This project was based on Eden and Rosenbaum’s paper “On Sampling Edges Almost Uniformly” from the 2018 Simplicity in Algorithms conference, in which they improved upon previous algorithms to approximately uniformly randomly sample edges from an unweighted graph. We extend this problem by considering the problem of sampling a weighted edge with probability proportional to its weight from an unknown weighted graph, \( G = (V, E, w) \), where \( w : E \rightarrow \mathbb{R} \) is a weight function that maps each edge to some real number, which we call the edge’s weight. We are assuming the graph is too large to store in local memory, so access to the graph is provided only through three queries: (1) uniform vertex queries, (2) weighted vertex queries, and (3) edge existence queries. We describe a simple algorithm that, when successful, returns an edge \( e \in E \) with probability approximately \( \frac{w(e)}{\sum_{e \in E} w(e)} \), in \( O(n) \) time.

1 Introduction

Suppose we have a graph \( G = (V, E, w) \), with \( w : E \rightarrow \mathbb{R} \) a weight function for the edges, that is too large to fit in local memory. We can access the graph through certain queries to the database: (1) sample a vertex uniformly at random (uniform vertex queries), (2) sample a vertex with probability proportional to its weighted degree (weighted vertex queries), and (3) given two vertices, return whether or not an edge exists between them (edge existence queries). The first query type is granted from the standard bounded degree graph query model of Goldreich and Ron [2]. The second type can be implemented using a preprocessing technique called alias sampling, which we will describe in the next section. The third type, although not in Goldreich and Ron’s standard query model, is not an unreasonable assumption of the database functionality.

We describe an \( O(n) \) time algorithm for sampling a weighted edge from a graph with probability proportional to its weight, with some probability of success.

1.1 Algorithm Overview

All sampling is done with replacement. We create two sets of vertices through different methods of sampling, construct the edge set between the sets of vertices, and then cut down that edge set.
Then, we return an edge from the final edge set uniformly at random.

**Algorithm:**

1. Sample vertices uniformly $\sqrt{n}$ times with replacement (query 1). Put these into a set $U$.

2. Sample vertices with probability proportional to its weighted degree $\sqrt{n}$ times with replacement (query 2). Put these vertices in a set $P$.

3. Construct an edge set $E_S = \{(u, v) | u \in U, v \in P\}$. That is, the set of edges such that one vertex is in $U$ and the other is in $P$ (query 3).

4. For each $e$ in $E_S$, we move the edge into a new set $E_F$ with some probability. If we don’t move the edge to $E_F$, we discard it.

5. Choose some edge from $E_F$ with uniform probability. Return this edge.

After cutting down our first set, $E_S$, we label this new set $E_F$ simply for ease of notation.

### 1.2 Alias Sampling

The alias sampling method is a method used to sample from a discrete probability function. The general method uses a preprocessing technique that separates the items we are sampling (in this case, vertices) into "buckets", storing the values of each bucket in an "alias table", and then rolling a fair die and biased coin to determine what bucket we fall into. The alias table is then used to determine what item that corresponds to. Thus, this preprocessing step will take $O(n)$ time, and then each query to the table will take $O(1)$ time.

The general idea of alias sampling is as follows. We have two tables: a probability table, and an "alias" table. The probability table contains $n$ columns (given $n$ vertices we want to put into the table), each column divided into two sections. The height of all the columns is the average probability of all the items: $\frac{1}{n}$. We divide our items into two groups: (1) items with probability less than the average probability, and (2) items with probability greater than or equal to the average probability. For each column, we take an item from group (1), and put it into the column. We fill up the rest of the column with the necessary amount from an item from group (2). The item from group (2) will have some leftover value, and we will put that back into its corresponding group. We repeat until all the items have been put into the probability table. The alias table keeps track of which item each area of the table corresponds to.

Then, we roll a fair die to determine which column to go to, and flip a fair coin to determine if we want to choose the top or bottom section of the column. The resulting item is chosen with its corresponding probability.

For example, take a 6-sided weighted dice. For simplicity, assume one of the faces, say face F, is weighted to come up with probability $\frac{1}{2}$, and the others, faces A-E, come up with probability $\frac{1}{10}$. We want to use alias sampling to mimic this behavior. The average probability is $\frac{1}{6}$, so group (1) consists of faces A-E, and group (2) is just face F.

Starting from column 1 in the probability table, we put in an item from group (1), say face A for the first column, and then fill the rest of the bar with the corresponding amount from face F. After filling out column 1, we have face F in group (2) with remaining probability $\frac{2}{5}$, and faces B-E
in group (1). After finishing filling out the table, it should look as follows:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>

Then, we will get faces A-E with probability:

\[
\text{Pr}[\text{return A}] = \text{Pr} [\text{Fair dice returns col 1}] \times \text{Pr} [\text{Biased coin returns side A}] = \frac{1}{6} \times \frac{3}{5} = \frac{1}{10}
\]

Similarly, we have:

\[
\text{Pr}[\text{return F}] = 5 \times \frac{1}{6} \times \left( \frac{2}{5} + \frac{1}{6} \right) = \frac{1}{2}
\]

We use alias sampling for the vertices in the graph. Each vertex corresponds to a separate "item" in the alias table, and the probability of choosing some vertex \( u \in V \) is \( \frac{d(u)}{\sum_{v \in V} d(u)} \). This is precisely what we need to run the weighted vertex query.

At some point, the graph will need to be read into the database. During this process, we can run the preprocessing step needed for alias sampling.

### 1.3 Related work

This project was inspired by Talya Eden and Will Rosenbaum’s "On Sampling Edges Almost Uniformly" [1], in which they improved upon and simplified edge sampling procedures in previously suggested uniform edge sampling techniques. Their approximation guarantees pointwise distance to uniformity is strictly stronger than any of the previous guarantees, and the algorithm depends on only a constant number of previous queries, and poly-logarithmic space, and thus can be easily parallelized. These queries were also from the standard bounded degree query graph model of Goldreich and Ron.

Sampling uniformly random edges can be applied to graph sampling techniques. When a graph is too large to store in local memory, we can sample edges to get a good idea of what the graph looks like. A natural extension to this problem is to consider weighted graphs.
2 Algorithm

Let \( G = (V, E, w) \) be an undirected graph with \( n = |V| \) vertices, and \( m = |E| \) edges.

We define \( D = \sum_{u \in V} d(u) \), the sum of all the weighted degrees of all the vertices. By the handshake lemma, we have \( D = 2 \sum_{e \in E} w(e) \).

**Algorithm 1** Sampling edges with Probability Proportional to Its Weight

1: Sample vertices uniformly for set \( U \) \((1) \sqrt{n}\) times.
2: Sample vertices with probability proportional to its degree for set \( P \) \((2) \sqrt{n}\) times.
3: Construct \( E_S = \{ (u,v) | u \in U, v \in P \} \)
4: for \( e = (u,v) \in E_S \) do
5: Move \( e \) to new set \( E_F \) with probability \( \frac{w(e)}{d(u)+d(v)} \)
6: Else, discard \( e \).
7: Choose \( e \) from final set \( E_F \) with uniform probability.
8: Return \( e \)

3 Analysis

First, we will go through the proof of correctness to show upon success, the algorithm returns an edge with the correct probability. Then, we will go through the time analysis and the probability of success of the algorithm.

3.1 Proof of Correctness

By the handshake lemma, \( D = 2 \sum_{e \in E} w(e) \). Thus, we want the algorithm to return edge \( e \) with probability: \( \Pr \{ \text{return } e \} = \frac{w(e)}{\sum_{e \in E} w(e)} = \frac{2w(e)}{D} \)

Claim 1. The algorithm returns edge \( e \) with probability:

\[
\frac{2w(e)}{(\alpha\beta)^2 D} \leq \Pr \{ \text{return } e \} \leq \frac{(\alpha\beta)^2 2w(e)}{D},
\]

, for \( \alpha \to 1 \) as \( n \to \infty \), and \( \beta \to 1 \) for most edges, as \( n \to \infty \).

First, we review these bounds:

\[
e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!}
\]

For \( x \geq 0 \), we have:

\[
e^x \geq 1 + x
\]

As a lower bound, we have:

\[
e^{-x} \leq 1 - x + \frac{x^2}{2}
\]
If $x \leq 1$:
\[
e^{-x} \leq 1 - \frac{x}{2}
\]

To tighten this bound, we replace 2 with $\alpha$:
\[
e^{-x} \leq 1 - \frac{x}{\alpha}
\]

This holds true for all $\alpha \geq \frac{x}{1-e^{-x}}$.

**Lemma 2.** The probability an edge $e = (u, v)$ is in edge set $E_S$ is approximately:

\[
\frac{d(u) + d(v)}{(\alpha \beta) D} \leq \Pr[e \in E_S] \leq \frac{(\alpha \beta) (d(u) + d(v))}{D}
\]

**Proof.** First, we look at the probability some vertex $v$ is in the first vertex set $U$.

\[
\Pr[v \in U] = 1 - \Pr[v \notin U]
\]

\[
\Pr[v \notin U] = \left(1 - \frac{1}{n}\right)^{\sqrt{n}} \leq \left(e^\frac{\alpha}{n}\right)^{\sqrt{n}} = \left(e^{\frac{\alpha}{\sqrt{n}}}\right) \leq 1 - \frac{1}{\alpha \sqrt{n}}
\]

This holds true for $\alpha = \frac{1}{\sqrt{n}(1-e^{-1/\sqrt{n}})}$.

For the lower bound, we have:

\[
\Pr[v \notin U] = \left(1 - \frac{1}{n}\right)^{\sqrt{n}} \geq \left(e^\frac{\alpha}{n}\right)^{\sqrt{n}} = \left(e^{\frac{\alpha}{\sqrt{n}}}\right) \geq 1 - \frac{1}{\alpha \sqrt{n}}
\]

This results in:

\[
\frac{1}{\alpha \sqrt{n}} \leq \Pr[v \in U] \leq \frac{\alpha}{\sqrt{n}}
\]

We follow the same steps to find the probability some vertex $v$ is in $P$.

\[
\Pr[v \in P] = 1 - \Pr[v \notin P]
\]

\[
\Pr[v \notin P] = \left(1 - \frac{d(v)}{D}\right)^{\sqrt{n}} \leq \left(e^{\frac{\beta d(v)}{D}}\right)^{\sqrt{n}} = \left(e^{\frac{\beta d(v) \sqrt{n}}{D}}\right) \leq 1 - \frac{d(v) \sqrt{n}}{\beta D}
\]

This is true for $\beta = \frac{d(u) \sqrt{n}}{D(1-e^{\frac{d(u) \sqrt{n}}{D}})}$.

This results in:

\[
\frac{d(u) \sqrt{n}}{\beta D} \leq \Pr[v \in U] \leq \frac{\beta d(u) \sqrt{n}}{D}
\]
We want to ensure the probability is not greater than 1, so we can more precisely write:

$$\min \left( \frac{d(u) \sqrt{n}}{\beta D}, 1 \right) \leq \Pr [v \in U] \leq \min \left( \frac{\beta d(u) \sqrt{n}}{D}, 1 \right)$$

Now, we find the probability some edge $e = (u, v) \in E_S$.

$$\Pr [e \in E_S] = \Pr [u \in U] \Pr [v \in P] + \Pr [v \in U] \Pr [u \in P] - \Pr [u \in U, P] \Pr [v \in U, P]$$

Because the probability $u$ and $v$ are in both sets is very small, we can ignore the last term in the sum:

$$\Pr [e \in E_S] \approx \Pr [u \in U] \Pr [v \in P] + \Pr [v \in U] \Pr [u \in P]$$

Then, plugging in the probabilities we found above:

$$\frac{d(u) + d(v)}{\frac{1}{\alpha \beta} D} \leq \Pr [e \in E_S] \leq \frac{(\alpha \beta) (d(u) + d(v))}{D}$$

This proves lemma 2.

The probability that some edge $e \in E_F$ is simply $\Pr [e \in E_S] \left( \frac{w(e)}{w(u) + w(v)} \right)$. This gives us lemma 3.

**Lemma 3:** The probability some edge $e = (u, v) \in E_F$ is approximately:

$$\frac{w(e)}{(\alpha \beta) D} \leq \Pr [e \in E_F] \leq \frac{(\alpha \beta) w(e)}{D}$$

What’s left is to find the probability some edge $e$ is returned. The returned edge is chosen uniformly at random from the final edge set $E_F$. So, $\Pr [\text{return } e] = \Pr [e \in E_F] * \frac{1}{|E_F|}$

$$\mathbb{E} [|E_F|] = \sum_{e \in E} \Pr [e \in E_F] \leq \sum_{e \in E} \frac{(\alpha \beta) w(e)}{D} = \frac{\alpha \beta}{2}$$

From this, we have

$$\frac{2}{\alpha \beta} \leq \mathbb{E} [|E_F|] \leq 2(\alpha \beta)$$

Finally, we prove the claim.

$$\frac{2w(e)}{(\alpha \beta)^2 D} \leq \Pr [\text{return } e] \leq \frac{(\alpha \beta)^2 2w(e)}{D}$$

$$\alpha = \frac{1}{\sqrt{n}(1-e^{-1/\sqrt{n}})}, \text{ which approaches 1 as } n \text{ approaches } \infty.$$  

$$\beta = \frac{d(u) \sqrt{n}}{D(1-e^{-d(u)/\sqrt{n}})}.$$

This approaches 1 as $n$ approaches $\infty$ for all $\frac{d(u)}{D} > \frac{1}{\sqrt{n}}$.  

$$6$$
3.2 Time Analysis

Sampling vertices takes $O(\sqrt{n})$ time, as we run queries (1) and (2) $\sqrt{n}$ times. Then, we have sets $U$ and $P$ of size $|U|, |P| \leq \sqrt{n}$. To go through each pair of vertices and check if an edge exists between them will take at most $O(n)$ time. $|E_S| \leq n$, so to go through each edge in $E_S$ will take at most $O(n)$ time. Finally, to sample an edge from $\tilde{E}_F$ uniformly at random will take $O(1)$ time. Thus, the entire algorithm will take at most $O(n)$ time to run once.

3.3 Probability of Success

To determine the expected number of times we need to run this algorithm in order for it to successfully return an edge, we must determine the probability of success of one run of the algorithm. Currently, this is ongoing research.

We are attempting to use the second moment method to find this probability. Recall Chebyshev’s Inequality:

Chebyshev: $\Pr[|X - \mu| \geq k] \leq \frac{\sigma^2}{k^2}$, where $\sigma^2$ is the variance of the random variable $X$, and $\mu$ is the expected value.

To begin, we define $X = |E_F|$, and $x_i$ an indicator variable for edge $i$. We let $x_i = 1$ if $e_i \in E_F$, and 0 otherwise. So, $X = \sum_{i=1}^m x_i$.

We want want to find $\Pr[X > 0]$. Notice that if we set $k = E(X)$ in Chebyshev’s inequality, we have:

$$\Pr[X \leq 0] \leq \frac{\text{Var}(X)}{E(X)^2}$$

We know $E(X) = \frac{1}{2}$.

$$\text{Var}(X) = E(X^2) - E(X)^2$$

$$= E\left(\left(\sum_i x_i^2\right) - \left(\sum_i x_i\right)^2\right)$$

$$= E\left(\sum_{i,j} x_ix_j\right) - \sum_i E(x_i)^2$$

$$= \left(\sum_i E(x_i^2) - \sum_i E(x_i)^2\right) + \left(\sum_{i\neq j} E(x_i)E(x_j)\right)$$

$$= \sum_i \text{Var}(x_i) + \sum_{i\neq j} \text{Cov}(x_i, x_j)$$
Calculating the variance of each $x_i$ is simple after making the following observation:

$x_i$ takes values 0 or 1, so $x_i^2 = x_i$. So, $\mathbb{E}(x_i^2) = \mathbb{E}(x_i)$

$$\text{Var}(x_i) = \mathbb{E}(x_i^2) - \mathbb{E}(x_i)^2 = \mathbb{E}(x_i) - \mathbb{E}(x_i)^2 = \frac{w(e_i)}{D} - \frac{w(e_i)^2}{D^2}$$

The covariance of $x_i$ and $x_j$ is 0 when the edges are not correlated - i.e., when the edges don’t share a common vertex.

When the edges do share a common vertex (say, $u$), the covariance is as follows:

$$\text{Cov}(x_i, x_j) = E(x_i x_j) - E(x_i)E(x_j)$$

$$\approx \Pr \left[ e_i \in E_F \cap e_j \in E_F \right] - \left( \frac{w(e_i)}{D} \right) \left( \frac{w(e_j)}{D} \right)$$

$$= \left( \frac{1}{\sqrt{n}} \right) \left( \frac{d(v_i) \sqrt{n}}{D} \right) \left( \frac{d(v_j) \sqrt{n}}{D} \right) + \left( \frac{d(u) \sqrt{n}}{Dn} \right) - \left( \frac{w(e_i)}{D} \right) \left( \frac{w(e_j)}{D} \right)$$

Finally, what’s left is to find the sum of the covariances for all $x_i \neq x_j$, and sum it with the sum of the variances for all $x_i$. This will give us $\text{Var}(X)$, which we can use to find the probability the size of the set $E_F$ is 0.

### 4 Conclusion / Future Work

The probability of success calculations are all but complete. Next steps for these calculations start with assuming each vertex in the graph has a maximum (unweighted) vertex. This might give some bound to work with for the covariance. Then, we might be able to extend it to all cases.

We would also like to further restrict the bounds on the probability of returning edge $e$ given a success. Currently, $\beta$ is only bounded for certain vertices.

Finally, we would like to improve the runtime for the algorithm. The current runtime is limited by the creation of, and then parsing through edge set $E_S$, which takes $O(n)$ time and should be optimized.

This project gives a promising preliminary algorithm for a weighted edge sampling problem. We described an algorithm that, upon success, runs in $O(n)$ time and returns an edge with the desired probability. Further research could work on improving upon and tweaking this algorithm.
5 Acknowledgments

Many thanks to Professor Daniel Spielman, for guiding me throughout this project and sharing his incredible wisdom on graphs and algorithms, as well as teaching me much of the foundations of algorithms, and finally for this invaluable introduction to the possibilities of computer science research. I am extremely grateful. Further gratitude to my teammates and coaches on the Yale Women’s Golf Team, with whom I spend the most time and receive the greatest support, and without whom I would not have made it to this point today.

6 References


