AN INTRODUCTORY SURVEY OF RANDOMIZED GRAPH ALGORITHMS

SAMANTHA REY

Abstract. Randomized algorithms are noteworthy for their ability to present simple solutions that are efficient with high probability and can withstand adversarial input. This paper serves as an exploratory introduction to the topic of randomized algorithms, focusing specifically on graph algorithms. We introduce three different randomized graph algorithms and analyze their expected running times. We begin with an algorithm that specifies randomized routing protocol for a computer network, continuing on to an analysis Karger's algorithm and it's uses for both finding and counting global-minimum cuts, finishing with an algorithm for determining graph connectivity.

Contents

1. Randomized Routing

In this section, we present an oblivious randomized routing protocol for a network of $N = 2^n$ processors in an n-dimensional boolean hypercube that aims to solve the *permuatation routing problem* [1]. A routing protocol is considered "oblivious" when the algorithm chooses routes independently for each processor, where each processor i's route depends only on it's own destination $d(i)$ and does not depend on the destination $d(j)$ of any other processor j. Deterministic oblivious routing protocols are weak to adversarial inputs, with there being an instance of the problem that always requires $\Omega\left(\sqrt{(N/n)}\right)$ steps to complete. However, because the routing protocol we present is randomized, on any input, our algorithm resolves in $O(n)$ steps with high probability and is provably superior under conditions of adversarial input. Presentation of this problem adapts from section 4.2 of [1].

For this problem, we take the following model of a parallel computer

> We model a network of N parallel processors in a computer as a directed graph on N nodes. Each node represents a single processor, and possesses a unique identifying number between 0 and $N-1$, and a queue from which the processor sends out packets. The processors communicate with each other by sending information in the form of "packets" over their communication links. Each edge in the graph represents a communication link in the computer. That is, an edge from processor i to j represents a communication link by which processor i can send packets to processor j. When routing packets, the computer adheres to the following:
>
> (1) All communication between processors proceeds in a sequence of synchronous steps, which can be represented by a sequence of numbers all between 0 and $N-1$ that shows in order each processor a packet visits as it travels to its destination.
> (2) Each link can carry at most 1 packet in a step.
> (3) During a step, a processor can send at most one packet to each of its neighbors.
> (4) As packets arrive at a processor, the processor places them in its FIFO ordered queue in the order which they arrived

Date: 5/14/2018.

(5) During each step, for each communication link attached to our processor i with destination processor j, if there is any packet in the queue to be routed next in its sequence to j, then i transmits that packet to j and removes the packet transmitted from its queue.
(6) If for a certain communication link attached to i with destination j, there exists more than one packet in the queue of i to be transmitted next to j, then processor i removes only the first packet in the queue and transmits that packet.

For the purpose of our analysis, we will be using a n dimensional hypercube to model the processor network of our computer, such that $N = 2^n$. Therefore, each processor's identifying number can be represented by an n bit binary string. In the hyper cube, a communication link exists between all processor nodes that differ by one bit in the binary string representation of their id number. Each processor is connected to n other processors in the cube. For example, in the 3-dimensional hypercube, processor 2 has 3-bit binary id 010. Processor 2 is therefore connected to processors 0, 3, and 6 because the 3-bit binary ids of these processors are 000, 011, and 110 respectively, which all differ from 010 by one exactly one bit in their binary string representation.

Now, we consider the *permutation routing problem* on our model [1]. Each processor has a packet to be delivered to some processor in the network. Let v_i denote the packet originating at processor i and let $d(i)$ denote the the destination processor for packet v_i. In the permutation routing problem, for all $0 \leq i \leq N-1$, the $d(i)$s form a a permutation of $\{0, ..., N-1\}$. That is, each of the N packets routes to a unique destination processor and each of the processors is the destination of exactly one packet. For the permutation routing problem, we seek an algorithm that can route any arbitrary permutation of destination processors. The algorithm must specify a route for each packet, with the goal of minimizing the number of steps required for all packets to reach their destination.

Our proposed solution to the permutation routing problem is the following **Randomized Routing** algorithm. The first subroutine of which is:

Algorithm 1: bitFixRoute(i, v_i, j)

Input: Current processor i, Packet v_i, Destination processor j
1 **if** $i = j$ **then**
2 | **End**;
3 **end if**
4 $k = 1$;
5 **while** $i[k] = j[k]$ **do**
6 | k++;
7 **end while**
8 $i' = \text{concat}(i[1:k], !i[k], i[k+1:n+1])$;
9 send v_i to i';
10 **bitFixRoute**(i', v_i, j);

Let i and j be two processors ids in our N node processor network. Suppose we were to route a packet v_i from i to j using the **bitFixRoute**(i, v_i, j). Let the kth bit be the lowest indexed bit on which the binary strings i and j differ. We send the packet down the communication link to the processor i' with the same binary id representation as i, except the kth bit is flipped. To this end, the binary id of i' is the concatenation $i[1:k]$, $!i[k]$, and $i[k+1:n+1]$, where $i[\ell, \ell']$ is the substring from the ℓ^{th} character to the $\ell' - 1^{st}$ character of the binary string representation of i and $!i[\ell]$ is the ℓ^{th} of the string representation of i flipped. Now, we send v_i to processor i', let $i = i'$, and repeat the process until $i = j$. So, packet v_i will eventually reach j from i using this bit fixing strategy. To this end, the subroutine **bitFixRoute** moves a packet v_i from its starting processor i to it's destination processor j by changing the bits of i from left to right as needed to match j, where each movement of v_i at line 9 represents a change in the initial id bit string of i to each i'

which eventually will become j.

Algorithm 2: Randomized Routing

1 For each processor $0 \leq i \leq N-1$, pick a random processor $\sigma(i)$ from $\{0, ..., N-1\}$ with a uniform distribution;
2 Simultaneously Route each v_i to $\sigma(i)$ using **bitFixRoute**$(i, v_i, \sigma(i))$;
3 Once all v_i arrive at their $\sigma(i)$, simultaneously route each v_i from $\sigma(i)$ to their $d(i)$ using **bitFixRoute**$(\sigma(i), v_i, d(i))$;

Lemma 1.1. *Let e' and e'' be any two edges in the hypercube. Let $\mathcal{T}(e)$ be the the number of packet routing paths in the cube that use edge e. Then $\mathbb{E}[\mathcal{T}(e')] = \mathbb{E}[\mathcal{T}(e'')] = \frac{1}{2}$.*

Proof. For all $0 \leq i \leq N-1$, let ρ_i be the routing path taken by v_i from processor i to processor $\sigma(i)$. Then for an edge e in the hypercube,

$$\mathbb{E}[\mathcal{T}(e)] = \sum_{i=0}^{N-1} \mathbb{E}\Big[\mathbb{1}[e \in \rho_i]\Big].$$

For a given path ρ_i, $\mathbb{E}\Big[\mathbb{1}[e \in \rho_i]\Big]$ is equal to $\Pr(e \in \rho_i)(1) + \Pr(e \notin \rho_i)(0) = \Pr(e \in \rho_i)$. Therefore

$$\mathbb{E}[\mathcal{T}(e)] = \sum_{i=0}^{N-1} \Pr(e \in \rho_i).$$

As far as our routing protocol is concerned, taking an edge in the hypercube is equivalent to switching a bit in a starting processor's n-bit binary id representation. Moving a packet along a communication link moves the packet to a processor with the same binary string id except with a change on the first, second, third, etc. bit in the id bit string. Therefore, each edge encodes a change on a particular index bit in the id bit string and encodes either a switch from 0 to 1 or a switch from 1 to 0.

Let $e = (v, w)$ be an arbitrary edge in the hypercube that goes from processor v to processor w that encodes an arbitrary switch on an arbitrary bit index $1 \leq \mu \leq n$. Because the bit fixing protocol progressively changes bits from right to left, we know a packet will only traverse the edge if the μ-th though the n-th bit of the starting processor's ids are the same as that of v. Therefore, there are $2^{\mu-1}$ possible packets that can use e as there are exactly $2^{\mu-1}$ processors that have identical μ-th through n-th bits in their initial id bit string.

If we consider the bit fixing protocol again, the probability that any of these $2^{\mu-1}$ packets takes edge e is dependent on the probability that the first $\mu - 1$ bits in the destination processor match that of v, and the μ-th bit of the destination processor does not match that of v. Because the random destination processor is chosen uniformly amongst all possible destination processors, the probability that any particular bit is a 0 in the destination processor is $\frac{1}{2}$, as is the probability that any particular bit is a 1. Because the bits that appear in the destination processor are independent of each other, the probability that a particular k bits are in the destination processor is 2^{-k}. The probability that any particular one of these $2^{\mu-1}$ packets' paths uses e is $2^{-\mu}$. Therefore, we know there are a total of $2^{\mu-1}$ packets that each have a probability of $2^{-\mu}$ of taking e, and some remaining $2^n - 2^{\mu-1}$ packets that have probability 0 of taking e. Thus,

$$\begin{aligned}\mathbb{E}[\mathcal{T}(e)] &= \sum_{i=0}^{N-1} \Pr(e \in \rho_i) \\ &= 2^{\mu-1}(2^{-\mu}) + (2^n - 2^{\mu-1})(0) \\ &= \frac{1}{2}.\end{aligned}$$

□

While routing the packets to their destinations, a packet is always transmitted down the the next edge in it's path unless it is waiting in the queue of a processor because another packet is being transmitted along said edge. Each time step that a packet spends waiting in a queue rather than being transmitted over an edge, until it reaches it's final destination, contributes to the delay of that packet. To this end, given an arbitrary packet v_i with path $\rho_i = (e_1, e_2, ..., e_\ell)$, if v_i reaches it's destination at timestep j, then the total delay for that packet is $j - \ell$. We will show that this delay can be bounded by the the number of intersecting paths with ρ_i, but first we must prove the following.

Lemma 1.2. *Let v_i and $v_{i'}$ be packets being routed from starting processors i and i' respectively to destination processors h and h' respectively in our hypercube such that $i \neq i'$. Let $\rho_i = (e_1, e_2, ..., e_\ell)$ and $\rho_{i'} = (e'_1, e'_2, ..., e'_\ell)$ be the paths of these packets as determined by the* **bitFixRoute** *protocol. Suppose ρ_i and $\rho_{i'}$ intersect. Let $(e_j \in \rho_i) = (e'_{j'} \in \rho_{i'})$ be the first edge shared by both ρ_i and $\rho_{i'}$ such that e_j is the j-th edge of ρ_i and the j'-th edge of $\rho_{i'}$. The first point at which paths ρ_i and $\rho_{i'}$ diverge after sharing an edge is the edge with the lowest k such that $(e_{j+k} \in \rho_i) \neq (e'_{j'+k} \in \rho_{i'})$. At the edge at which paths ρ_i and $\rho_{i'}$ diverge, the two paths will never re-converge and share no remaining edges.*

Proof. For any processor i, routing a packet v_i can be treated conceptually the same as changing the bits from left to right of the initial id bit string of i to those of the destination processor g's id bit string. To this end, v_i being in the queue of processor h represents the intermediate state of the bit string when changing between i and g. Furthermore, v_i taking edge $e = (g_1, g_2)$ represents switching a bit in the id that moves the intermediate state of the id from g_1 to g_2.

Let $e_{j+k} = (g_1, g_2)$ and $e'_{j'+k} = (g'_1, g'_2)$. Because edge e_{j+k} is the first edge where the paths of ρ_i and $\rho_{i'}$ diverge after previously having converged, we know that both $g_1 = g'_1$. Furthermore, since the paths diverge at e_{j+k} we know $g_2 \neq g'_2$. Let e_{j+k} encode a change on the μ-th bit and let $e'_{j'+k}$ encode a change on the μ'-th bit. Since the paths diverge at these edges, we know $\mu \neq \mu'$. WLOG let $\mu < \mu'$. Therefore, the first $\mu' - 1$ bits of g'_2 must be the same as the first $\mu' - 1$ bits of g_2, except for the μ-th bit. The $\mu - th$ bit of g'_2 was not flipped, unlike the μ-th bit of g_2 which was flipped from g_1. The only way the two paths converge again to take the same edge is if the state of the intermediate processor id's match again. Therefore, if the path of $v_{i'}$ meets back up with v_i after it diverges, the μ-th bit of the id of i' must be flipped later to match that of i. However, the bit fixing protocol switches bits incrementally from lower index to higher index, so the μ-th bit would be flipped before the μ'-th bit by the protocol. Therefore, the only way for the path of ρ_i and $\rho_{i'}$ to share edges again would contradict the bit fixing protocol. So it must be that the paths of v_i and $v_{i'}$ cannot re-converge.

□

Lemma 1.3. *Let v_i be a packet being routed from processor i to processor j via path ρ_i. Let $\mathcal{P}$ be the set of processors other than i which have packets with routes that pass through at least one edge in path $\rho_i = (e_1, e_2, ..., e_\ell)$. Let D_i be the delay of packet v_i. Then, the delay D_i is bounded such that $D_i \leq |\mathcal{P}|$.*

Proof. To prove that D_i is bounded by $|\mathcal{P}|$, we will show that there exists an injection from each delay of v_i to a unique element in $\mathcal{P}$.

Let v be a packet with starting processor in $\mathcal{P}$. We say v has left ρ_i just after the last time step at which it traverses any edge in ρ_i ends. We define $L(v, t)$, the function for the lag of v at time step t, as follows. Let e_j be the last edge traversed or being traversed by v in ρ_i at time step t, where $j = 0$ if v has yet to traverse an edge in ρ_i at time step t. Define $L(v, t) = t - j$. Therefore, $D_i = L(v_i, t_f) = t_f - \ell$ where t_f is the final time step before v_i reaches it's destination.

At any time step t where $L(v, t) - 1 = L(v, t - 1)$, we know there must be some other packet from $\mathcal{P}$ traversing $e_{t-L(v,t)+1}$ as otherwise v would be traversing that edge and its lag would not have increased. Let this packet be v'. So, $L(v', t) = L(v, t - 1)$, and there exists some packet v' with lag $L(v, t) - 1$. Therefore, at any particular time step t_k where $L(v_i, t_k) - 1 = L(v_i, t_k - 1)$, we know there exists some packet at time step t_k with lag $L(v_i, t_k) - 1$.

Let t^* be the last time step during which which any packet using an edge in ρ_i has lag $L(v_i, t_k) - 1$. Let v^* be a packet such that $L(v^*, t^*) = L(v_i, t_k) - 1$. Let e_{j^*} be the last edge traversed or being traversed by v^* at time step t^* such that $j^* = t^* - L(v^*, t^*)$. We claim that v^* traverses e_{j^*} at time t^*.

Suppose for the sake of contradiction that v^* does not traverse e_{j^*} at time t^*. Then, v^* must be waiting in a queue to take e_{j^*+1} as it has yet to leave ρ_i. So, at time step $t^* + 1$, v^* either traverses edge e_{j^*+1} or waits in the queue for another time step. If v^* traverses e_{j^*+1} at time $t^* + 1$, then

$$L(v^*, t^* + 1) = (t^* + 1) - (j^* + 1) = t^* - j^* = L(v_i, t_k) - 1,$$

which contradicts t^* as the last time step at which any packet in ρ_i has lag $L(v_i, t_k) - 1$. On the other hand, if e_{j^*+1} waits in the queue for another time step, then

$$L(v^*, t^* + 1) = (t^* + 1) - (j^*) = L(v^*, t^*) + 1 = L(v_i, t_k)$$

which, by our earlier result, means that during time $t^* + 1$, some packet in ρ_i has lag $L(v_i, t_k) - 1$, again contradicting t^* as the last time step at which any packet in ρ_i has lag $L(v_i, t_k) - 1$. Thus, v^* traverses e_{j^*} at time t^*. We now claim that v^* leaves ρ_i at the finish of time step t^*. Suppose for the sake of contradiction that v^* does not leave ρ_i at the finish of time step t^*. Then, some packet traverses e_{j^*+1} at timestep $t^* + 1$, contradicting t^* as the last time step at which any packet in ρ_i has lag $L(v_i, t_k) - 1$.

Therefore, for all $1 \leq q \leq L(v_i, t_f)$, a packet leaves ρ_i with lag q. Since paths never re-converge after diverging, each of these packets must be unique. For each delay increase, there is a unique processor from $\mathcal{P}$ whose packet leaves ρ_i. Thus, there exists an injection from each delay of v_i to an element in $\mathcal{P}$, and the total delay must be bounded by $|\mathcal{P}|$.

□

Theorem 1.4. *Let D_i be the delay incurred by the path ρ_i when routing from i to $\sigma(i)$. Then $\mathbb{E}[D_i] \leq \frac{3n}{4}$.*

Proof. As established in lemma 1.3, the delay D_i incurred for a packet originating from processor i is at most $|\mathcal{P}|$, where $\mathcal{P}$ is the set of processors other than i which have packets with routes that pass through at least one edge in ρ_i. Since $|\mathcal{P}|$ is the total number of paths that intersect path ρ_i, we can represent $|\mathcal{P}|$ as

$$\sum_{j \in \{0,\ldots,n-1\} \setminus i} H_{ij}, \tag{1.5}$$

where H_{ij} is the indicator variable that at least one edge in ρ_j intersects with some edge in ρ_i. Thus, we can say for all $0 \leq i \leq N - 1$

$$\sum_{j \in \{0,\ldots,n\} \setminus i} H_{ij} \leq \sum_{k=1}^{\ell} \mathcal{T}(e_k), \tag{1.6}$$

since each ρ_j that intersects with ρ_i must contribute at least 1 to some $\mathcal{T}(e_k)$.

Now, to bound the expected delay in a path.

$$\begin{aligned} \mathbb{E}[D_i] &\leq \mathbb{E}\Big[\sum_{j \in \{0,\ldots,n\} \setminus i} H_{ij} \Big] \\ &\leq \mathbb{E}\Big[\sum_{e \in \rho_i} \mathcal{T}(e) \Big]. \end{aligned}$$

Since ρ_i is itself a random variable, we use the law of total expectation to show the following

$$\mathbb{E}\Big[\sum_{e \in \rho_i} \mathcal{T}(e) \Big] = \sum_{\rho \in P} \Big(\Pr\Big(\rho_i = \rho\Big) \sum_{e \in \rho} \mathbb{E}\Big[\mathcal{T}(e) | \rho_i = \rho \Big] \Big), \tag{1.7}$$

where P is the set of all possible paths originating from processor i.

We know $e \in \rho$, $\mathbb{E}[\mathcal{T}(e)|\rho_i = \rho]$ will not be the same as $\mathbb{E}[\mathcal{T}(e)]$. So we calculate the following. Fixing $e \in \rho$,

$$\begin{aligned}\mathbb{E}\Big[\mathcal{T}(e)\Big|\rho_i=\rho\Big] &= \mathbb{E}\Big[\mathbb{1}[e\in\rho_i]+\sum_{j\in\{0,\dots,n-1\}\setminus i}\mathbb{1}[e\in\rho_j]\Big|\rho_i=\rho\Big]\\ &= \mathbb{E}\Big[\mathbb{1}[e\in\rho_i]\Big|\rho_i=\rho\Big]+\mathbb{E}\Big[\sum_{j\in\{0,\dots,n-1\}\setminus i}\mathbb{1}[e\in\rho_j]\Big|\rho_i=\rho\Big]\\ &= 1+\sum_{j\in\{0,\dots,n-1\}\setminus i}\mathbb{E}\Big[\mathbb{1}[e\in\rho_j]\Big|\rho_i=\rho\Big].\end{aligned}\tag{1.8}$$

Since the paths are chosen independently of each other, whether or not we have any edge $e\in\rho_j$ is independent of whether we have $\rho_i=\rho$. Therefore,

$$\begin{aligned}\mathbb{E}\Big[\mathcal{T}(e)\Big|\rho_i=\rho\Big] &= 1+\sum_{j\in\{0,\dots,n-1\}\setminus i}\mathbb{E}\Big[\mathbb{1}[e\in\rho_j]\Big|\rho_i=\rho\Big]\\ &= 1+\sum_{j\in\{0,\dots,n-1\}\setminus i}\mathbb{E}\Big[\mathbb{1}[e\in\rho_j]\Big]\\ &\le 1+\sum_{j\in\{0,\dots,n-1\}}\mathbb{E}\Big[\mathbb{1}[e\in\rho_j]\Big]\\ &= 1+\mathbb{E}[\mathcal{T}(e)]\\ &= \frac{3}{2}.\end{aligned}\tag{1.9}$$

We can take our result from equation 1.9 to say

$$\begin{aligned}\sum_{\rho\in P}\Big(\Pr\Big(\rho_i=\rho\Big)\sum_{e\in\rho}\mathbb{E}\Big[\mathcal{T}(e)|\rho_i=\rho\Big]\Big) &\le \sum_{\rho\in P}\Big(\Pr\Big(\rho_i=\rho\Big)\sum_{e\in\rho}\frac{3}{2}\Big).\\ &= \sum_{\rho\in P}\Big(\Pr\Big(\rho_i=\rho\Big)|\rho|\frac{3}{2}\Big)\\ &= \sum_{\ell=0}^{n}\Big(Pr(|\rho_i|=\ell)\frac{3}{2}\ell\Big).\end{aligned}\tag{1.10}$$

We know from the bit fixing protocol the length of the path is the same as the number of bits that need to be changed between the id representations of i and $\sigma(i)$. Therefore, the number of paths of length ℓ is the number of combinations of ℓ bit changes that can be made. There are $\binom{n}{\ell}$ paths of length ℓ, and a total of 2^n paths. So,

$$\Pr(|\rho_i|=\ell)=\frac{\binom{n}{\ell}}{2^n},$$

and therefore,

$$\begin{aligned}\mathbb{E}[D_i] &\le \sum_{\ell=0}^{n}\Big(Pr(|\rho_i|=\ell)\frac{3}{2}\ell\Big)\\ &= \sum_{\ell=0}^{n}\Big(\frac{\binom{n}{\ell}}{2^n}\frac{3}{2}\ell\Big)\\ &= \frac{3}{2^{n+1}}\sum_{\ell=0}^{n}\binom{n}{\ell}\ell\\ &= \frac{3}{2^{n+1}}\cdot n2^{n-1}\\ &= \frac{3n}{2^2}\\ &= \frac{3n}{4}.\end{aligned}\tag{1.11}$$

□

Theorem 1.12. *Let i be any processor with packet v_i. Let D_i be the totaly delay incurred when routing v_i from i to intermediate destination processor $\sigma(i)$. Then* $\Pr(D_i \geq 6n) \leq 2^{-7n}$.

Proof. By theorem 1.4 the expected delay incurred when routing for all packets is bounded from above by $\frac{3n}{4}$. Since the D_i themselves are bounded above by the sum of the independent 0-1 valued random variables H_{ij}, we can run a Chernoff bound on the individual D_i. So we get the probability that the delay of the packet going out of processor i, D_i, is greater than its expected value μ by a factor of $(1+\delta)$ is

$$\begin{aligned} \Pr(D_i \geq (1+\delta)\mu) &\leq \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^{\mu} \\ &\leq \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^{\frac{3n}{4}}. \end{aligned} \tag{1.13}$$

We have $6n = 8\frac{3n}{4}$. Therefore, if we make $\delta = 7$, then

$$\begin{aligned} \Pr(D_i \geq (1+7)\mu) &\leq \Pr\left(D_i \geq (8)\frac{3n}{4}\right) \\ &\leq \left[\frac{e^7}{8^8}\right]^{\frac{3n}{4}} \\ &= \left[\frac{e^{\frac{21n}{4}}}{2^{18n}}\right] \\ &\leq \left[\frac{(2^2)^{\frac{21n}{4}}}{2^{18n}}\right] \\ &\leq 2^{-7n}. \end{aligned} \tag{1.14}$$

□

Theorem 1.15. *The algorithm* **Randomized Routing** *runs in less than* $14n$ *steps with probability greater than* $1 - 2^{-6n+1}$.

Proof. First, we consider the probability that the first routing step, when routing all packets from their starting processor i to their intermediate processor $\sigma(i)$, takes $7n$ or more time steps. Let A_1 be the event that the first routing step takes at least $7n$ time steps. By theorem 1.12, the probability that any processor i experiences a delay of greater than $6n$ steps is less than 2^{-7n}. Since the length of a packet's path is at most length n, the probability that any processor takes $7n$ or more time steps to route to it's intermediate destination must be less than 2^{-7n}.

Now, in the event that the first routing step takes at least $7n$ steps, we know that at least one processor must have taken greater than $7n$ steps to complete their route. Let A_1^j be the event that processor j takes at least $7n$ steps to route to it's final destination. We can therefore consider the event A_1 that the first routing step takes at least $7n$ steps to be

$$A_1 = \bigcup_{i=0}^{N-1} A_1^i.$$

So,

$$\begin{aligned} \Pr(A_1) &\leq \sum_{i=0}^{N-1} \Pr(A_1^i) \\ &\leq \sum_{i=0}^{N-1} 2^{-7n} \\ &= 2^n(2^{-7n}) \\ &= 2^{-6n}. \end{aligned}$$

Now, we consider the probability that the second routing step, when routing all the packets from their intermediate processor $\sigma(i)$ to their destination processor $d(i)$, takes $7n$ or more time steps. Let A_2 be the event that the second routing step takes at least $7n$ time steps. The second routing step is nearly the same as the first routing step, except reversed so that the random processor is the starting processor. We can determine by the same analysis of the first routing step that the probability of A_2 is less than or equal to 2^{-6n}.

Let A be the event that the **Randomized Routing** algorithm runs in at least $14n$ time steps. Should event A occur, we know that the first routing step has taken at least $7n$ steps or the second routing step has taken at least $7n$ steps. So,

$$A \subseteq A_1 \cup A_2,$$

and therefore,

$$\begin{aligned} \Pr(A) &\leq \Pr(A_1 \cup A_2) \\ &\leq \Pr(A_1) + \Pr(A_2) \\ &= 2^{-6n+1}. \end{aligned}$$

Thus, the probability that **Randomized Routing** runs in less than $14n$ steps is

$$\begin{aligned} \Pr(\overline{A}) &= 1 - \Pr(A) \\ &\geq 1 - 2^{-6n+1}. \end{aligned}$$

□

2. Minimum Cuts and Karger's Algorithm

In this section, we present Karger's algorithm [2] and its applications to minimum cut problems on undirected graphs. Despite the simplicity of Karger's algorithm, the algorithm can be used to both find and count global minimum cuts with high probability in polynomial time. As a consequence of Karger's algorithm, we can also easily place a bound on the number of minimum cuts that can emerge in an undirected graph. Here we present a generalized variant of Karger's. To this end, while it may not be clear from our generalized construction of the algorithm, there are implementations of Karger's algorithm that run in $O(|E|)$ time and space, where E is the set of edges in the input graph. The presentation of our analysis derives from [2] and [3]

Karger's algorithm is a randomized algorithm that attempts to find the minimum cut in a connected undirected graph $G = (V, E)$, with any number of duplicate edges between vertices. A minimum cut is defined as a partition of the vertex set V into two disjoint vertex subsets $(S^*, \overline{S^*})$ such that $\overline{S^*} = V \setminus S^*$, both $S^* \neq \varnothing$ and $\overline{S^*} \neq \varnothing$, and the number of edges that have one vertex in S^* and one vertex in $\overline{S^*}$ is minimal. First we define the subroutines, then Karger's algorithm is as follows.

Algorithm 3: Subroutine **Collapse**(G, u, v)

```
struct {
    set vertices;

    /* A superNode is a vertex in the graph that contains a set of vertices. During the initial state of
       the graph G, we consider each vertex to be a superNode with only itself in its vertex set.   */
    /*                                                                                              */
} superNode;
```

Input: Super nodes u and v in an undirected connected graph $G = (V, E)$ with no self-loops

Let $w =$ **new** *superNode*;
$V = V \cup \{w\}$;
w.vertices $= u$.vertices $\cup$ v.vertices;
for *all edges* $e = (u, v) = (v, u)$ **do**
    Remove e from E;
end for
for *all edges that* $e = (u, u')$*, where* u' *is any vertex* **do**
    $e' = (w, u')$;
    $E = E \cup \{e'\}$;
    Remove e' from E;
end for
for *all edges that* $e = (v, v')$*, where* v' *is any vertex* **do**
    $e' = (w, v')$;
    $E = E \cup \{e'\}$;
    Remove e' from E;
end for
Remove u and v from V;

Algorithm 4: Karger's Algorithm

Input: Undirected connected graph $G = (V, E)$ with no self loops, where E is the set of edges and V is the set of vertices in G

Output: $S \subset V$ such that the partition $(S, \overline{S})$ defines a cut of G

while $|V| > 2$ **do**
    Select a random edge $e = (u, v) \in E$;
    Collapse(u, v)**;**
end while
Let the remaining two nodes be v_1 and v_2;
Let $S = \{$the original vertices that comprise supernode $v_1\}$;
Output: S;

Lemma 2.1. *Let $G_i = (V_i, E_i)$ be the state of the graph $G = (V, E)$ after the i^{th} iteration of the collapsing procedure loop in Karger's algorithm from lines 1 to 4. Let S_i be any set of vertices such that $(S_i, \overline{S_i})$ is a partition that defines a cut on the graph of G_i, and let C_i be the set of edges across the cut. Then, given the cut defined by $(S_i, \overline{S_i})$ on G_i, there exists a cut defined by some partition $(S, \overline{S})$ on G with set C of edges across the cut such that (i) $\bigcup_{v \in S_i} v.vertices = \bigcup_{v \in S} v.vertices$, (ii) $\bigcup_{v \in \overline{S_i}} v.vertices = \bigcup_{v \in \overline{S}} v.vertices$, and (iii) $|C_i| = |C|$.*

Proof. We will prove this claim by induction. We start with base case $i = 0$. The state of G_0 is the state of G before any iteration of the collapsing loop. So, $G = G_0$. Trivially, any cut defined by the partition

$(S_0 = S, \overline{S_0} = \overline{S})$ on G_0 is the same cut on G. Therefore,

$$\bigcup_{v \in S_0} v.\text{vertices} = \bigcup_{v \in S} v.\text{vertices},$$

$$\bigcup_{v \in \overline{S_0}} v.\text{vertices} = \bigcup_{v \in \overline{S}} v.\text{vertices},$$

and the size of the cut defined by the partition created by $(S_0, \overline{S_0})$ is the same as the size of the cut defined by the partition created by $(S, \overline{S})$, and our claim holds.

Now, for the inductive case, assume that our claim holds for all $i < k$, show that our claim holds when $i = k$. Let $S_k, \overline{S_k}$ define a cut on G_k. Now, from the end the $k-1^{st}$ iteration to the end of the k^{th} iteration of the collapsing procedure loop, we know some nodes u_k and v_k were collapsed into a single node w_k such that $w_k.\text{vertices} = u_k.\text{vertices} \cup v_k.\text{vertices}$. We know w_k is exclusively either in S_k or $\overline{S_k}$, so WLOG assume $w_k \in S_k$ as otherwise we would just consider $\overline{S_k}$.

Let $S_{k-1} = (S_k \setminus \{w_k\}) \cup \{u_k, v_k\}$. None of the nodes other than u_k, v_k, and w_k were added or removed between the $k-1^{st}$ and k^{th} iterations of the collapsing procedure loop, $\overline{S_{k-1}} = \overline{S_k}$. So trivially,

$$\bigcup_{v \in \overline{S_k}} v.\text{vertices} = \bigcup_{v \in \overline{S_{k-1}}} v.\text{vertices}.$$

Furthermore, it also follows that

$$\bigcup_{v \in S_k \setminus \{w_k\}} v.\text{vertices} = \bigcup_{v \in S_{k-1} \setminus \{u_k, v_k\}} v.\text{vertices},$$

since none of those node's vertices sets were changed between the iterations. Therefore, because $w_k.\text{vertices} = u_k.\text{vertices} \cup v_k.\text{vertices}$ (line 6 of subroutine **Collapse**), it so follows that

$$\bigcup_{v \in S_k} v.\text{vertices} = \bigcup_{v \in S_{k-1}} v.\text{vertices}.$$

Let, $C_k \subseteq E_k$ be the set of edges across the cut defined by partition $(S_k, \overline{S_k})$. let $C_{k-1} \subseteq E_{k-1}$ be the set of edges across the cut defind by the partition $(S_{k-1}, \overline{S_{k-1}})$. We split C_k into disjoint subsets $C_k^{\varnothing}$ and $C_k^{w_k}$, where $C_k^{w_k}$ is the set of edges in C_k that have w_k as an endpoint and $C_k^{\varnothing}$ is the set of edges in C_k that do not have w_k as an endpoint. We also split C_{k-1} into disjoint subsets $C_{k-1}^{\varnothing}, C_{k-1}^{u_k}, C_{k-1}^{v_k}, C_{k-1}^{u_k, v_k}$ where $C_{k-1}^{u_k}$ is the set of edges in C_{k-1} with u_k and not v_k as an endpoint, $C_{k-1}^{v_k}$ is the set of edges in C_{k-1} with v_k and not u_k as an endpoint, $C_{k-1}^{u_k, v_k}$ is the set of edges with both u_k and v_k as endpoints, and $C_{k-1}^{\varnothing}$ is the set of edges in C_{k-1} that have neither u_k nor v_k as endpoints.

Because the only edges and nodes that were changed or affected in any manner between iterations $k-1$ and k of the collapsing procedure loop were nodes u_k, v_k, or w_k and any edges that had these as an endpoint, $|C_{k-1}^{\varnothing}| = |C_k^{\varnothing}|$. Since u_k and v_k are on the same side of the partition in S_{k-1}, $C_{k-1}^{u_k, v_k} = \varnothing$, so $|C_{k-1}^{u_k, v_k}| = 0$. Let $e' = (a', b') \in (C_{k-1}^{u_k} \cup C_{k-1}^{v_k}$ be any edge where $a' = u_k$ or $a' = v_k$ and b' is any vertex. Since e' is an edge across the cut, we know $b' \in \overline{S_{k-1}}$ and therefore $b' \in \overline{S_k}$. Furthermore, since $a' = u_k$ or $a' = v_k$, we know during the k^{th} iteration of the collapsing procedure loop we will remove edge e' from the graph and add a new edge $e'' = (w_k, b')$ to the graph. Because $w_k \in S_k$, e'' is an edge across the cut, so $e'' \in C_k^{w_k}$. So, for each edge in $e' \in (C_{k-1}^{u_k} \cup C_{k-1}^{v_k})$ in the graph of G_{k-1}, we remove e' and add a new edge $e'' \in C_k^{w_k}$, so $|(C_{k-1}^{u_k} \cup C_{k-1}^{v_k})| \leq |C_k^{w_k}|$.

Now, consider any edge $e'' = (w_k, b'') \in C_k^{w_k}$. Since e'' has w_k as an endpoint, then during the k^th iteration, it was added to the graph only after some edge $e' = (u_k, b'')$ or $e' = (v_k, b'')$ was removed from the graph. Since $e'' \in C_k^{w_k}$, then $b'' \in \overline{S_k}$, which means also $b'' \in \overline{S_{k-1}}$. As we already know, both $u_k \in S_{k-1}$ and $v_k \in S_{k-1}$. Therefore, $e' \in (C_{k-1}^{u_k} \cup C_{k-1}^{v_k})$. Thus, for ever edge e'' added to the graph $C_k^{w_k}$, there was some edge removed from the graph in $C_{k-1}^{u_k} \cup C_{k-1}^{v_k}$, so $|C_k^{w_k} \leq (C_{k-1}^{u_k} \cup C_{k-1}^{v_k})$.

Thus, $|C_{k-1}| = |C_k|$. Because the cut $(S_k, \overline{S_k}$ on G_k with set of edges C_k across the cut maps to a cut defined by partition $(S_{k-1}, \overline{S_{k-1}})$ on G_{k-1} with set C_{k-1} of edges across the cut such that $\bigcup_{v \in S_k} v.\text{vertices} = \bigcup_{v \in S_{k-1}} v.\text{vertices}$, $\bigcup_{v \in \overline{S_k}} v.\text{vertices} = \bigcup_{v \in \overline{S_{k-1}}} v.\text{vertices}$ and $|C_k| = |C_{k-1}|$, by the inductive hypothesis there must exist some vertex set S such that the cut $(S_k, \overline{S_k}$ on G_k maps to some cut defined by partition $(S, \overline{S})$ on G with set C of edges across that cut where $\bigcup_{v \in S_k} v.\text{vertices} = \bigcup_{v \in S} v.\text{vertices}$, $\bigcup_{v \in \overline{S_k}} v.\text{vertices} = \bigcup_{v \in \overline{S}} v.\text{vertices}$ and $|C_k| = |C|$.

□

Due to the collapsing procedure defined by algorithm **Collapse**, not all graph cuts in the graph of G can necessarily be returned by Karger's algorithm. So we prove the following.

Lemma 2.2. *Let $(S, \overline{S})$ be any vertex partition that defines a cut on an undirected connected graph $G = (V, E)$, where C is the set of edges across the cut such that $C = \{\forall e = (u, v) \in E \mid u \in S, v \in \overline{S}$ or $v \in S, u \in \overline{S}\}$. Let G' be the graph after we collapse all edges between vertices in S and edges between vertices in $\overline{S}$, such that the only edges that are not collapsed and remain in G' are those in C. The cut defined by the partition $(S, \overline{S})$ is returned by Karger's when run on G with probability $p > 0$ iff G' consists of exactly two super nodes.*

Proof. First, suppose G' collapses to exactly two super nodes. Let E_S be a set of edges such that, for each unordered pair of vertices (u, v) in S that share an edge, E_S contains one edge $e = (u, v)$. Let $E_{\overline{S}}$ be a set of edges such that, for each unordered pair of vertices (u, v) in $\overline{S}$ that share an edge, $E_{\overline{S}}$ contains one edge $e = (u, v)$. We can consider an instance of Karger's algorithm that collapses first the edges in E_S and then the edges in $E_{\overline{S}}$. Since each edge is selected randomly and uniformly, we know there is a non-zero probability that this choice of edges to collapse occurs in that order. Furthermore, the collapsing procedure will stop after these edges are collapsed, and Karger's will return the set of vertices contained in one of the supernodes, which we know to be S or $\overline{S}$. Therefore, running Karger's on G' returns the cut defined by $(S, \overline{S})$ with a non-zero probability.

Now, assume $(S, \overline{S})$ does not reduce to 2 super nodes, then either the collapse of S or $\overline{S}$ is not a single super node. Since Karger's algorithm only returns cuts consisting of two super-nodes, it cannot return $(S, \overline{S})$.
□

Lemma 2.3. *Let $(S, \overline{S})$ be the vertex partition that defines a minimum cut on a connected undirected graph G. The cut defined by partition $(S, \overline{S})$ is returned by Karger's algorithm when run on G with probability $p > 0$.*

Proof. Let C be the set of edges across the cut such that $C = \{\forall e = (u, v) \in E \mid u \in S, v \in \overline{S}$ or $v \in S, u \in \overline{S}\}$. Let G' be the graph after we collapse all edges between vertices in S and edges between vertices in $\overline{S}$, such that the only edges that are not collapsed and remain in G' are those in C. We know by lemma 2.2, the cut defined by S is returned with probability $p > 0$ iff G' consists of exactly two super nodes.

Suppose for the sake of contradiction that G' does not consist of exactly two super nodes. Therefore, either the collapse of S or $\overline{S}$ is not a single super node. WLOG assume it's S. Let A be the vertices contained by one of these super nodes of S. Because G' is the collapsed graph of the partition of S, there are no remaining edges between the super nodes of S. Furthermore because G is connected, we know that, since the super nodes of S are not connected to each other, the super nodes of S must be connected by edges to the super nodes of $\overline{S}$. Now, we consider cut defined by the vertex partition between $S \setminus A$ and $\overline{S} \cup A$. The weight of this cut must be the weight of the cut defined by S, subtracted by the weight of the edges between the super node containing A and the other super nodes in $\overline{S}$, which is lower than the weight of the cut defined by $(S, \overline{S})$. However, this contradicts the minimality of the cut defined by $(S, \overline{S})$. Thus, our supposition must be incorrect, and G' consists of exactly two super nodes, which means cut S is returned by Karger's with probability $p > 0$.

□

Lemma 2.4. *Given any unweighted undirected graph $G = (V, E)$, let v be a random vertex chosen uniformly from V. Let $\deg(v)$ be the degree of vertex v. Then $\mathbb{E}[\deg(v)] = \frac{2|E|}{|V|}$.*

Proof.

$$\begin{aligned}\mathbb{E}[\deg(v)] &= \sum_{u\in V} \Pr(v=u)\deg(u) \\ &= \sum_{u\in V} \frac{1}{|V|}\deg(u) \\ &= \frac{1}{|V|}\sum_{u\in V}\deg(u) \\ &= \frac{2|E|}{|V|}\end{aligned} \tag{2.5}$$

□

Lemma 2.6. *Let $S^* \subset V$ be a vertex set such that the partition $(S^*, \overline{S^*})$ defines a minimum cut of graph $G = (V, E)$, and let C^* define the set of edges across the cut with one vertex in S^* and one vertex in $\overline{S^*}$. Then, $|C^*| \leq \frac{2|E|}{|V|}$.*

Proof. Let v be any vertex in V. Now, consider the edges across cut C_v defined by the vertex partition of $\{v\}$ and $\overline{\{v\}} = V \setminus \{v\}$. We know the size of this cut $|C_v| = \deg(v)$ since all of the edges that cross the partition must be the edges which have v as one endpoint.

Suppose that $\deg(v) \leq \frac{2|E|}{|V|}$. Since there exists a cut C_v of size at most $\frac{2|E|}{|V|}$, then the the size of the minimum cut $|C^*|$ must be no greater than $\frac{2|E|}{|V|}$.

Now, suppose on the otherhand that $\deg(v) > \frac{2|E|}{|V|}$. By Lemma 2.4 we know that the average vertex degree is $\frac{2|E|}{|V|}$. Because $\deg(v) > \frac{2|E|}{|V|}$, we know there must exist some other vertex v' that witnesses this average such that $\deg(v') < \frac{2|E|}{|V|}$. Now, we consider the vertex cut $C_{v'}$ between $\{v'\}$ and $\overline{\{v'\}} = V \setminus \{v'\}$. As already established, the size of this cut will be $\deg(v')$ which we know to be less than $\frac{2|E|}{|V|}$. It so follows that since there exists a cut of size less than $\frac{2|E|}{|V|}$, then the size of the minimum cut $|C^*| \leq |C_{v'}| < \frac{2|E|}{|V|}$.

□

Theorem 2.7. *Given a connected graph $G = (V, E)$ with no self loops, let C^* be the set of edges across any minimum cut of G defined by partition $(S^*, \overline{S^*})$. Let p^* be the probability that Karger's algorithm successfully outputs S^*. Then $p^* \geq \frac{2}{n^2}$, where $n = |V|$.*

Proof. Given the simple manner by which Karger's algorithm chooses which vertex partition to return, it's clear that Karger's algorithm returns a particular minimum cut if and only if none of the the edges in the cut are randomly chosen for contraction at step 3 in the algorithm. So,

$$p^* = \Pr(\text{no edge across cut } C^* \text{ is chosen by the algorithm}).$$

Since there are n vertices, we know the algorithm contracts exactly $n-2$ edges before reaching 2 super nodes. Thus,

$$\begin{aligned}&\Pr(\text{no edge across cut } C^* \text{ is chosen by the algorithm}) \\ &= \prod_{i=1}^{n-2} \Pr(\text{no edge across } C^* \text{ is chosen during the } i^{th} \text{ edge contraction} \mid \\ &\qquad \text{no other edges had been chosen from the cut}) \\ &= \prod_{i=1}^{n-2} \Big(1 - \Pr(\text{an edge across } C^* \text{ is chosen during the } i^{th} \text{ edge contraction} \mid \\ &\qquad \text{no other edges had been chosen from the cut})\Big).\end{aligned}$$

If C^* is a minimum cut in G in it's initial state, it must also be the minimum cut in G in its state after any subsequent iterations of the vertex collapsing procedure loop. For the sake of contradiction, suppose that after the j^{th} iteration of the procedure loop, there was another cut C' on the current state of the graph such that $|C^!| < |C^*|$. By lemma 2.1, there would also need to be some cut on the input graph of G with size $|C^!|$, thereby contradicting the minimality of C^*. Thus, C^* must be a minimum cut in G in its state after any subsequent iterations of the vertex collapsing procedure loop.

By lemma 2.6, we know that in any graph $G' = (V', E')$ with minimum cut defined by partition $(S', \overline{S'})$ with edges across the cut C', the size of the minimum cut $|C'| \leq \frac{2|E'|}{|V'|}$. So, by simple algebra $|E| \geq \frac{|C'||V'|}{2}$. Furthermore, we know that each time we perform the collapsing procedure, we collapse two nodes into one, thereby reducing the total number of the nodes in the graph by 1. Therefore, during the i^{th} iteration of the collapsing procedure loop, there are $n - i + 1$ nodes in the graph of G at the time. Let $G_j = (E_j, V_j)$ be the state of G during the j^{th} iteration of the collapsing procedure loop. We can therefore say that during the i^{th} iteration of the loop when we're choosing which edge we will collapse, the size of the edge set E_i is

$$\begin{aligned} |E_i| &\geq \frac{|C^*||V_i|}{2} \\ &= \frac{|C^*|(n-i+1)}{2}, \end{aligned}$$

and therefore

$$\begin{aligned} &\Pr(\text{an edge across } C^* \text{ is chosen during the } i^{th} \text{ edge contraction} | \\ &\qquad \text{no other edges had been chosen from the cut}) = \frac{|C^*|}{|E_i|} \\ &\qquad \leq \frac{|C^*|}{|C^*|(n-i+1)/2} \\ &\qquad = \frac{2}{n-i+1}. \end{aligned} \tag{2.8}$$

So,

$$\begin{aligned} p^* &\geq \prod_{i=1}^{n-2} 1 - \Pr(\text{an edge } C^* \text{ is chosen during the } i^{th} \text{ edge contraction } | \\ &\qquad \text{no other edges had been chosen from the cut}) \\ &\geq \prod_{i=1}^{n-2} 1 - \frac{2}{n-i+1} \\ &= \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right) \ldots \left(1 - \frac{2}{4}\right)\left(1 - \frac{2}{3}\right) \\ &= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right) \ldots \left(\frac{2}{4}\right)\left(\frac{1}{3}\right) \\ &= \frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}} \\ &\geq \frac{2}{n^2}. \end{aligned} \tag{2.9}$$

□

Corollary 2.10. *Let p be the probability that the vertex partition $(S', \overline{S'})$ found by Karger's algorithm is a minimum cut. Then $p \geq \frac{2}{n^2}$.*

Proof. This is largely a consequence of theorem 2.7. Karger's algorithm can find a particular minimum cut with probability $p^* \geq \frac{2}{n^2}$. Since there can be more than one minimum cut in a graph, it follows that $p \geq p^* \geq \frac{2}{n^2}$.

□

Theorem 2.11. *Let S be a set of vertices such that the partition $(S, \overline{S})$ defines a minimum cut on an unweighted undirected graph G. For any $0 < \delta \leq 1$, running k iterations of Karger's algorithm on $G = (V, E)$ will return S at some point with probability $p > 1 - \delta$, where $k \geq \frac{-n^2}{2} \log(\delta)$ and $n = |V|$.*

Proof. Let X_i be a random variable that indicates whether S is returned by Karger's algorithm on the i^{th} iteration of Karger's algorithm. Let $X = \sum_{i=1}^{k} X_i$. Then S is returned by one of the k iterations of Karger's iff $X \geq 1$. Therefore, the probability that S is returned by Karger's is

$$\Pr(X \geq 1) = 1 - \Pr(X = 0) = 1 - \Pr\left(\bigcap_{i=1}^{k} X_i = 0\right).$$

Because each iteration of Karger's is independent of the others and all have identical probability for finding the cut,

$$\Pr\left(\bigcap_{i=1}^{k} X_i = 0\right) = \prod_{i=0}^{k} \Pr(X_i = 0) = \Big(\Pr(X_j = 0)\Big)^k,$$

for any $1 \leq j \leq k$. We know by theorem 2.7, that the probability p^* that a particular minimum cut is returned by Karger's is $p^* \geq \frac{2}{n^2}$. So, the probability p' that S is not returned by Karger's alogrithm is $p' \leq 1 - \frac{2}{n^2}$. Therefore,

$$\Big(\Pr(X_j = 0)\Big)^k \leq \Big(1 - \frac{2}{n^2}\Big)^k,$$

which we can rewrite as

$$\left(\left(1 - \frac{1}{\frac{1}{2}n^2}\right)^{\frac{1}{2}n^2}\right)^{\frac{k}{\frac{1}{2}n^2}}.$$

For all $m > 1$, we have $0 < (1 - \frac{1}{m})^m \leq \frac{1}{e}$. So, for all graphs with more than 2 vertices,

$$\begin{aligned} \Big(\Pr(X_j = 0)\Big)^k &\leq \left(\left(1 - \frac{1}{\frac{1}{2}n^2}\right)^{\frac{1}{2}n^2}\right)^{\frac{k}{\frac{1}{2}n^2}} \\ &\leq \Big(\frac{1}{e}\Big)^{\frac{2k}{n^2}} \\ &= \exp\left(\frac{-2k}{n^2}\right). \end{aligned} \tag{2.12}$$

Now, substituting $k \geq \frac{-n^2}{2} \log(\delta)$, we have

$$\begin{aligned} \Big(\Pr(X_j = 0)\Big)^k &\leq \exp\left(\frac{-2k}{n^2}\right) \\ &\leq \exp\Big(\log(\delta)\Big) \\ &= \delta. \end{aligned}$$

Thus, when $k \geq \frac{-n^2}{2} \log(\delta)$, for any $0 < \delta \leq 1$,

$$\Pr(X \geq 1) = 1 - \Big(\Pr(X_j = 0)\Big)^k \geq 1 - \delta.$$

□

Corollary 2.13. *A particular minimum cut $(S, \overline{S})$ on G is found with high probability using $O(n^2)$ iterations of Karger's algorithm.*

Proof. By theorem 2.11, we know that for any $0 < \delta \leq 1$, a particular minimum cut of G is found in at least $\frac{-n^2}{2} \log(\delta)$ iterations of Karger's algorithm with probability $p > 1 - \delta$, which is

$$O\left(\frac{-n^2}{2} \log(\delta)\right) = O\left(n^2 \log\left(\frac{1}{\delta}\right)\right) = O\Big(n^2\Big)$$

iterations of Karger's algorithm.

□

Theorem 2.14. *There are no more than $\binom{n}{2}$ minimum cuts in any undirected unweighted graph $G = (V, E)$ where $|V| = n$*

Proof. Let $\mathscr{C}$ be the set of all possible cuts that can be found by partition returned from Karger's algorithm. Consider the the cut C resulting from the vertex partition returned by Karger's algorithm. The events $(\mathscr{c} = C)$ for all $\mathscr{c} \in \mathscr{C}$ form a partition of the probability space. Therefore,

$$\sum_{\mathscr{c} \in \mathscr{C}} \Pr(\mathscr{c} = C) = 1.$$

By theorem 2.7, we know

$$\Pr(C^* = C) \geq \frac{1}{\binom{n}{2}},$$

for any minimum cut C^*. Let $\mathscr{C}^*$ be the set of all minimum cuts in G. So,

$$\begin{aligned} 1 &= \sum_{\mathscr{c} \in \mathscr{C}} \Pr(\mathscr{c} = C) \\ &\geq \sum_{\mathscr{c}^* \in \mathscr{C}^*} \Pr(\mathscr{c}^* = C) \\ &\geq |\mathscr{C}^*| \frac{1}{\binom{n}{2}}. \end{aligned} \tag{2.15}$$

Therefore, $|\mathscr{C}^*| \leq \binom{n}{2}$.

□

The bound $\binom{n}{2}$ is also tight. Take for example the graph of the n-vertex cycle. We know the size of the minimum cut is 2. Taking any two distinct edges in the cycle, there exists a valid cut on the cycle in which these two edges are the only two edges across the cut. Therefore, we can create a total of $\binom{n}{2}$ minimum cuts in the graph of the n-vertex cycle.

Theorem 2.16. *Let $\mathcal{S}$ be the set of all vertex partitions on an unweighted undirected graph G such that (i) for all $(S, \overline{S}) = (\overline{S}, S) \in \mathcal{S}$, (ii) the partition $(S, \overline{S})$ defines a minimum cut. For any $0 < \delta \leq 1$, running k iterations of Karger's algorithm on $G = (V, E)$ will return all S such that $(S, \overline{S}) \in \mathcal{S}$ during the k iterations with probability $p > 1 - \delta$, where $k \geq \frac{-n^2}{2} \log\left(\frac{\delta}{\binom{n}{2}}\right)$ and $n = |V|$.*

Proof. Assume there are m minimum cuts on the graph of G. Let X be random variable indicating the event that the vertex sets that define all m cuts of G are returned during the k iterations of Karger's. Assign an arbitrary unique id from 1 to m to each minimum cut. Let X_i be the random variable indicating the event that the the vertex set that defines the i^{th} cut of G is returned during the k iterations of Karger's. So,

$$\Pr(X = 1) = \Pr\left(\bigcap_{i=0}^{m} X_i = 1\right) = 1 - \Pr\left(\overline{\bigcap_{i=0}^{m} X_i = 1}\right) = 1 - \Pr\left(\bigcup_{i=0}^{m} X_i = 0\right),$$

and therefore,

$$\Pr\left(\bigcup_{i=0}^{m} X_i = 0\right) \leq \sum_{i=0}^{m} \Pr(X_i = 0).$$

By equation 2.12, we know for any graph G with at least 2 vertices, the probability that a particular minimum cut is not found over k iterations of Karger's algorithm is less than $e^{\frac{-2k}{n^2}}$. So,

$$\begin{aligned} \Pr\left(\bigcup_{i=0}^{m} X_i = 0\right) &\leq \sum_{i=0}^{m} \Pr(X_i = 0) \\ &= m \exp\left(\frac{-2k}{n^2}\right). \end{aligned}$$

As proven in theorem 2.14, there are no more than $\binom{n}{2}$ minimum cuts in G. Thus,

$$\begin{aligned}\Pr\left(\bigcup_{i=0}^{m} X_i = 0\right) &\leq m\exp\left(\frac{-2k}{n^2}\right)\\ &\leq \binom{n}{2}\exp\left(\frac{-2k}{n^2}\right).\end{aligned} \tag{2.17}$$

Now, substituting $k \geq \frac{-n^2}{2}\log\left(\frac{\delta}{\binom{n}{2}}\right)$,

$$\begin{aligned}\Pr\left(\bigcup_{i=0}^{m} X_i = 0\right) &\leq \binom{n}{2}\exp\left(\frac{-2k}{n^2}\right)\\ &\leq \binom{n}{2}\exp\left(\log\left(\frac{\delta}{\binom{n}{2}}\right)\right)\\ &= \delta.\end{aligned} \tag{2.18}$$

Thus, when $k \geq \frac{-n^2}{2}\log\left(\frac{\delta}{\binom{n}{2}}\right)$), for any $0 < \delta \leq 1$,

$$\Pr(X = 1) = 1 - \Pr\left(\bigcup_{i=0}^{m} X_i = 0\right) \geq 1 - \delta.$$

□

Corollary 2.19. *Every minimum cut in G is found with high probability using $O(n^2 \log n)$ iterations of Karger's algorithm.*

Proof. By theorem 2.16, we know that for any $0 < \delta \leq 1$, every minimum cut in G is found in at least $\frac{-n^2}{2}\log\left(\frac{\delta}{\binom{n}{2}}\right)$ iterations of Karger's algorithm with probability $p > 1 - \delta$, which is

$$O\left(\frac{-n^2}{2}\log\left(\frac{\delta}{\binom{n}{2}}\right)\right) = O\left(n^2\log\left(\frac{\binom{n}{2}}{\delta}\right)\right) = O\Big(n^2\log n\Big)$$

iterations of Karger's algorithm.

□

3. Random Walks and Graph Connectivity

In this section, we present a very simple algorithm to determine graph connectivity on an undirected graph. The algorithm takes the rather brute force approach of simply taking random edges until the destination vertex is reached, if ever. However, the analysis reveals that despite the simplicity of the method, the algorithm runs in polynomial time with high probability. Although our analysis primarily focuses on demonstrating the probabilistic run time of the algorithm, there exists a simple construction of our algorithm that runs in log-space. However, the details of such a machine construction are not pertinent to the paper and left as an exercise to the reader. The algorithm and analysis thereof adapts largely from sections 16.1 and 16.2 of [4].

We take the following notion of a random walk. Let $G = (V, E)$ be an undirected graph. Let $\Gamma(u)$ denote the set neighboring vertices of u in G. We define a neighboring vertex as any vertex v such that there exists an edge $e = (u, v)$. Now, we define a T-step random walk starting at v to be a sequence of random variables $X_0, ..., X_T$ such that for all $0 \leq i \leq T$, (i) $X_i \in V$, (ii) $X_{i+1} \in \Gamma(X_i)$ and X_{i+1} is randomly chosen uniformly from $\Gamma(X_i)$, and (iii) $X_0 = v$. A T-step random walk is a path that starts at vertex v where each edge taken in the path is chosen at random.

Now, let us introduce the *undirected path problem*. Given an undirected graph G, a starting vertex $s \in V$, and an ending vertex $t \in V$, the *undirected path problem* answers whether or not there exists a path in G connecting s to t. We will show that this problem can be solved in probabilistic polynomial time. However,

before that, we must introduce the following graph reduction algorithm.

The intent of algorithm **selfLooping4Regular** is to create a 4 regular graph wherein each vertex has at least one self-looping edge. A 4-regular graph is a graph in which every vertex has a degree of 4. We consider a self-loop to contribute an added degree of one to a vertex. We impose a labeling on our vertices from the numbers 1 to n, where $|V| = n$.

Algorithm 5: selfLooping4Regular(G)

Input: An undirected graph G
Output: Undirected graph G' such that G' is a 4-Regular graph such that each vertex $v' \in G'$ has a self-loop

1 Let $G' = (V', E')$ initially be the empty graph;
2 **for** *each vertex* $v_u \in G$ **do**
3 | add a cycle c_u of size n to G';
4 **end for**
5 **for** *each unique edge* $e = (v_i, v_j) = (v_j, v_i) \in G$ *such that* $v \neq v$ **do**
6 | add edge $e' = (c_i^j, c_j^i)$ to G';
7 | //Define c_a^b to be the b^{th} vertex of cycle a^{th} cycle c_a in G'
8 **end for**
9 **for** *all* $v' \in G'$ **do**
10 | **while** $\deg(v') < 3$ **do**
11 | | add a self looping edge $e'' = (v', v')$ to G';
12 | **end while**
13 | add a self looping edge $e'' = (v', v')$ to G';
14 **end for**
15 **Output**: G';

Lemma 3.1. *G' is a 4-Regular graph such that each vertex $v' \in G'$ has a self-loop.*

Proof. Because the algorithm in the for loop at line 13 clearly adds a self-loop to each vertex $v' \in V'$, the claim that each vertex has a self loop is easily and trivially proven.

Now, we only need to prove the claim that each vertex $v' \in G'$ has degree 4. Suppose for the sake of contradiction that there exists some vertex v'' such that $\deg(v'') \neq 4$. Clearly, $\deg(v'') > 4$ as during the while loop at line 10, we ensure the degree of every vertex is at least 3 before adding the last self loop to the vertex. Therefore, if $\deg(v'') \neq 4$, then $\deg(v'') > 4$. So, prior to the for loop at lines 9 through 14, $\deg(v'') > 3$. Suppose $v'' = c_{j'}^{i'}$. Because each vertex in G' is in a cycle of n vertices, we know two of the edges that go out of the vertex are the edges that connect v'' to it's cycle, so we know v'' as an edge connecting it to $c_{j'}^{i'-1 \mod n}$ and an edge connecting it to $c_{j'}^{i'+1 \mod n}$. Since $\deg(v'') > 3$ before the for loop at line 9, we know there must be at least 2 more edges connected to v'' that are not the edges that connect v'' to it's cycle and were added in the for loop at line 9. Let $e'' = (v'', c_{j''}^{i''})$, and $e^! = (v'', c_{j^!}^{i^!})$ be any two of those edges.

For any i, j, k, ℓ, the algorithm only adds an edge between the vertices c_j^i and c_ℓ^k if $j \neq \ell$. Also, the algorithm adds an edge between two vertices c_j^i and c_ℓ^k iff there exists an edge between v_j and v_ℓ in G. Furthermore, considering the manner in which the algorithm adds edges to G', if an edge between v_j and v_ℓ exists in the graph, then the algorithm adds the edge between the ℓ^{th} vertex in the j^{th} cycle and the j^{th} vertex of the ℓ^{th} cycle of G'. Therefore, if there is an edge between c_j^i and c_ℓ^k, then $j \neq \ell$, $i = \ell$ and $k = j$. This implies then that if edges $e'' = (v'' = c_{j'}^{i'}, c_{j''}^{i''})$ and $e^! = (v'', c_{j^!}^{i^!})$ are in G', then $i' = j'' = j^!$ and $j' = i'' = i^!$. This also implies that an edge between $v_{j'}$ and $v_{i'}$ exists in the original graph G. Since the algorithm only adds one edge between cycles for each edge in G, then e'' and $e^!$ must be the same edge.

Thus, since any choice of two edges coming out of v'' that are not the neighboring cycle edges of v'' are the same, then it must be that there is only one other edge going out of v'', a contradiction. Thus, there exists no such vertex in v'' such that $\deg(v'') \neq 4$, which means G' is a 4-regular graph.

□

Lemma 3.2. *Vertex s is connected to vertex t in the graph of G iff cycle c_s is connected to cycle c_t in the graph of G'.*

Proof. If s and t are connected in G, then there exists some path $P = p_1 \to p_2 \to ... \to p_\ell$ connecting s and t such that $p_1 = s$, $p_\ell = t$ and, for all $1 \leq i \leq \ell - 1$, there exists an edge $e = (p_i, p_{i+1}) \in E$. WLOG, let P be the shortest such path between s and t in G, so there's no self-loops in P. Because there's an edge between $e = (p_i, p_{i+1})$ for all $1 \leq i \leq \ell - 1$, then there must be an edge $e' = (c_{p_i}^{p_{i+1}}, c_{p_{i+1}}^{p_i}) \in G'$. Furthermore, we know we can build a path P_j from any vertex in the cycle c_j to any other vertex in the cycle c_j as all of the vertices in the cycle are connected. Thus, we can construct a new path $P' = c_{p_1}^{p_2} \to c_{p_2}^{p_1} \to P_{p_2} \to c_{p_2}^{p_3} \to c_{p_3}^{p_2} \to P_{p_3} \to ... \to P_{p_{\ell-1}} \to c_{p_{\ell-1}}^{p_\ell} \to c_{p_\ell}^{p_{\ell-1}}$. Thus, there exists a path P' that connects cycle $c_{p_1} = c_s$ to cycle $c_{p_\ell} = c_t$.

On the other hand, if s and t are not connected, there won't be a path connecting c_s to c_t. Suppose there were such a path P' connecting c_s to c_t. Then we could define another path P wherein all edges connecting vertices in the same cycle were removed, and every edge that connected some vertex c_i^j to some vertex $c_{i'}^{j'}$ such that $i \neq i'$ was replaced with an edge between vertices (i, i') in the graph G. By the definition of the algorithm, if such an edge between vertices c_i^j and $c_{i'}^{j'}$ exists in G', then some edge exists between vertices i and i' exists in G. Therefore, P' would be a valid path in G from vertex s to vertex t. However, no such path exists between s and t in G, which is a contradiction. Thus, our supposition that there exists a path between c_s and c_t must be false.

□

Lemma 3.3. *the graph reduction* **selfLooping4Regular***(G) is polynomial time constructible.*

Proof. In the for loop at line 2, for each vertex in G, we add a cycle of n vertices to G', which has $n - 1$ edges. Assuming it takes some constant number of steps to add a vertex and an edge to a graph, each cycle takes $O(n + n - 1) = O(n)$ steps to add itself to G'. Since the for loop runs once for each vertex in G, then the total time complexity of this for loop is $O(n^2)$.

Next, in the for loop at line 5, we add an edge to G' for each unique edge in G that is not a self loop. There are no more than n^2 unique edges in an undirected graph. Again assuming a constant number of time-steps to add an edge, the loop has a time complexity of $O(n^2)$.

Lastly, for the for loop at line 9, we add self loops to each vertex in G'. We know for each vertex in G, we added a cycle of n vertices to G'. Therefore, there are n^2 vertices in G'. Again assuming adding an edge is a constant time operation, the loop has a time complexity of $O(n^2)$.

Thus, the reduction has a total time complexity of $O(3n^2) = O(n^2)$.

□

Algorithm 6: undirectedPath(G', s, t)

Input: An undirected 4-regular graph G' with self loops at each vertex, a start vertex s, and an end vertex t

Output: Accept, if t is the ending vertex of the walk, otherwise output **Reject**

```
currentVertex = s;
for i = 0, i < 10dn^3 log n, i++ do
    randomly choose an e from the set of edges going out of currentVertex;
    Let e = (currentVertex, nextVertex);
    currentVertex = nextVertex;
end for
if nextVertex = t then
    Output: Accept;
else
    Output: Reject;
end if
```

Before we begin our analysis, we must consider the following. Let G be any arbitrary n vertex graph. Let A be the normalized adjacency matrix of G. Then A is of the form

$$\begin{bmatrix} \frac{|E_{1,1}|}{d_1} & \frac{|E_{2,1}|}{d_2} & \dots & \frac{|E_{n,1}|}{d_n} \\ \frac{|E_{1,2}|}{d_1} & \frac{|E_{2,2}|}{d_2} & \dots & \frac{|E_{n,2}|}{d_n} \\ & \dots\dots\dots\dots & & \\ \frac{|E_{1,n}|}{d_1} & \frac{|E_{2,n}|}{d_2} & \dots & \frac{|E_{n,n}|}{d_n} \end{bmatrix}$$

where $E_{i,j}$ is the set of edges in G that go from vertex v_i to vertex v_j and d_ℓ is the degree of the vertex v_ℓ.

Notice that $\frac{|E_{i,j}|}{d_i} = A_{j,i}$ is the probability that, in a random walk, the next vertex in the walk is v_j given that the current vertex of the walk is v_i. The ℓ^{th} column vector in A, A^ℓ is the probability vector for outcome of the next step of the random walk given that the current vertex is v_ℓ. For all ℓ, the L_1-norm of A^ℓ is 1. To this end, A is a Markov Chain encoding for the possible state outcomes of a random walk.

Now, consider the following operation. Let $\mathbf{p}$ be a probability vector such that is a positive column vector with an L_1-norm of 1. We multiply $A\mathbf{p}$ to get

$$\begin{bmatrix} A_{1,1} & A_{1,2} & \dots & A_{1,n} \\ A_{2,1} & A_{2,2} & \dots & A_{2,n} \\ & \dots\dots\dots\dots & & \\ A_{n,1} & A_{n,2} & \dots & A_{n,n} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ \cdot \\ p_n \end{bmatrix} = \begin{bmatrix} A_{1,1}p_1 + A_{1,2}p_2 + \dots + A_{1,n}p_n \\ A_{2,1}p_1 + A_{2,2}p_2 + \dots + A_{2,n}p_n \\ \dots\dots\dots\dots\dots\dots \\ A_{n,1}p_1 + A_{n,2}p_2 + \dots + A_{n,n}p_n \end{bmatrix} = \mathbf{q}$$

If we let $\mathbf{p}$ encode the probability vector for state of the current step of the walk such that for all $1 \le i \le n$ the i^{th} coordinate of $\mathbf{p}$, $\mathbf{p}_i$, is the probability that the current vertex is v_i, then for all $1 \le j \le n$

$$\begin{aligned} \mathbf{q}_j &= \sum_{k=1}^{n} A_{j,k}\mathbf{p}_k \\ &= \sum_{k=1}^{n} \Pr\begin{pmatrix} \text{next vertex in the walk is } v_j | \\ \text{current vertex in the walk is } v_k \end{pmatrix} \Pr\Big(\text{current vertex in the walk is } v_k\Big) \\ &= \Pr(\text{next vertex in the walk is } v_j) \end{aligned}$$

Therefore, $\mathbf{q}$ encodes the probability vector for the state of the next step of the walk. Now, if we take the operation $A\mathbf{q} = \mathbf{q}^2$, then $\mathbf{q}^2$ would clearly encode the probability vector for the state of the step after the next step of the walk, that is two steps after the current state of the walk encoded by $\mathbf{p}$. Therefore, $\mathbf{q}^i$, the probability vector for the state of the walk i steps after the current step of the walk, is encoded by

$$\mathbf{q}^i = A\mathbf{q}^{i-1} = A^2\mathbf{q}^{i-2} = ... = A^{i-1}\mathbf{q} = A^i\mathbf{p}$$

Let $\{\mathbf{e}^i\}_{i=1}^n$ represent the standard basis of $\mathbb{R}^n$ such that each $\mathbf{e}^i$ is an n-dimensional column vector with zero's in every coordinate except for the i^{th} coordinate, which is 1. If we assume $\mathbf{e}^i$ to represent a probability

vector for the state of a step of the random walk, then, during that step, we know the current vertex of the walk is guaranteed to be v_i. To this end, if $\mathbf{e}^i$ is the initial state of a random walk, then

$$A^T\mathbf{e}^i \tag{3.4}$$

is the probability distribution of the state of a T step random walk starting at vertex v_i in an undirected graph G.

Now, for the following definitions, let $G = (V, E)$ be any d-regular undirected graph with n vertices such that $n = |V|$ and $A = A(G)$ is the normalized adjacency matrix of G.

Definition 3.5. We define the vector $\mathbf{1}$ to be the n dimensional column vector with the value $1/n$ at every coordinate.

Consider the vector

$$A\mathbf{1} = \begin{bmatrix} A_{1,1} & A_{1,2} & \dots & A_{1,n} \\ A_{2,1} & A_{2,2} & \dots & A_{2,n} \\ & \dots\dots\dots & & \\ A_{n,1} & A_{n,2} & \dots & A_{n,n} \end{bmatrix} \begin{bmatrix} 1/n \\ 1/n \\ . \\ 1/n \end{bmatrix}$$
$$= \begin{bmatrix} A_{1,1}1/n + A_{1,2}1/n + \dots + A_{1,n}1/n \\ A_{2,1}1/n + A_{2,2}1/n + \dots + A_{2,n}1/n \\ \dots\dots\dots\dots\dots \\ A_{n,1}1/n + A_{n,2}1/n + \dots + A_{n,n}1/n \end{bmatrix}.$$

Since G is a d-regular matrix, we know each $A_{i,j} = \frac{|E_{j,i}|}{d}$. So,

$$A\mathbf{1} = \begin{bmatrix} \frac{|E_{1,1}|+|E_{2,1}|+\dots+|E_{n,1}|}{dn} \\ \frac{|E_{1,2}|+|E_{2,2}|+\dots+|E_{n,2}|}{dn} \\ \dots\dots\dots \\ \frac{|E_{1,n}|+|E_{2,n}|+\dots+|E_{n,n}|}{dn} \end{bmatrix}.$$

For all $1 \le i \le n$, $|E_{1,i}| + |E_{2,i}| + ... + |E_{n,i}| = \sum_{j=1}^{n} |E_{j,i}|$. Again, since G is a d-regular matrix, then we know

$$\sum_{j=1}^{n} |E_{j,i}| = d.$$

So,

$$A\mathbf{1} = \begin{bmatrix} \frac{d}{dn} \\ \frac{d}{dn} \\ . \\ \frac{d}{dn} \end{bmatrix} = \mathbf{1}$$

Thus, $\mathbf{1}$ is an eigenvector of $A = A(G)$ with a corresponding eigenvalue of 1 for any d-regular undirected graph G with n vertices.

Lemma 3.6. *For any d-regular undirected graph G with n vertices, $\mathbf{1}$ is the eigenvector with the largest absolute valued eigenvalue of A, where A is the normalized adjacency matrix of G.*

Proof. Suppose $\mathbf{v}$ is any eigenvector of A with eigenvalue λ. Then

$$A\mathbf{v} = \lambda\mathbf{v},$$

and therefore, for all $1 \le i \le n$

$$\sum_{j=1}^{n} A_{i,j}\mathbf{v}_j = \lambda\mathbf{v}_i.$$

Because $\mathbf{v}$ is an eigenvector of A, we know that $\mathbf{v}$ cannot be the 0 vector. Let the k^{th} coordinate of $\mathbf{v}$ be a coordinate such that for all $1 \le i \le n$, $|\mathbf{v}_i| \le |\mathbf{v}_k|$. We know $\mathbf{v}_k > 0$. So,

$$\lambda = \sum_{j=1}^{n} A_{k,j}\frac{\mathbf{v}_j}{\mathbf{v}_k},$$

and therefore

$$\begin{aligned}|\lambda| &= \left|\sum_{j=1}^{n} A_{k,j}\frac{\mathbf{v}_j}{\mathbf{v}_k}\right| \\ &\leq \sum_{j=1}^{n}\left|A_{k,j}\frac{\mathbf{v}_j}{\mathbf{v}_k}\right| \\ &= \sum_{j=1}^{n}\left|A_{k,j}\right|\left|\frac{\mathbf{v}_j}{\mathbf{v}_k}\right| \\ &\leq \sum_{j=1}^{n}\left|A_{k,j}\right|.\end{aligned}$$

We already know L_1-norm for each column vector in A is 1. Since A is the adjacency matrix for an undirected graph, we know A is symmetric. The L_1-norm of each row vector in A must also then be 1. Thus,

$$\begin{aligned}|\lambda| &\leq \sum_{j=1}^{n}\left|A_{k,j}\right| \\ &= 1.\end{aligned}$$

□

For any d-regular undirected graph G with n vertices, we know that $A(G)$ is symmetric. Therefore, we can find an orthogonal basis of n eigenvectors $\mathbf{v}^1, \mathbf{v}^2, ..., \mathbf{v}^n$ with corresponding eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$. WLOG assume $|\lambda_1| \geq |\lambda_2| \geq ... \geq |\lambda_n|$. As a consequence of lemma 3.6, we know $\lambda_1 = 1$. Furthermore, there exists some eigenvectors $\mathbf{v}^2, \mathbf{v}^3, ..., \mathbf{v}^n$ of A which are all mutually orthogonal with $\mathbf{1}$.

Lemma 3.7. *For any d-regular undirected graph G with n vertices with normalized adjacency matrix $A(G) = A$, suppose vector $\mathbf{v} \perp \mathbf{1}$. Then $A\mathbf{v} \perp \mathbf{1}$.*

Proof. Let $A^\dagger$ be the transpose of A. Since G is an undirected graph, we know that A is a symmetric matrix. Therefore, $A^\dagger = A$. Now we take the dot product of $\mathbf{1}$ and $A\mathbf{v}$ to show

$$\begin{aligned}\langle \mathbf{1}, A\mathbf{v}\rangle &= \langle A^\dagger \mathbf{1}, \mathbf{v}\rangle \\ &= \langle A\mathbf{1}, \mathbf{v}\rangle \\ &= \langle \mathbf{1}, \mathbf{v}\rangle \\ &= 0.\end{aligned}$$

Since, $\langle \mathbf{1}, A\mathbf{v}\rangle = 0$, it must be that $A\mathbf{v} \perp \mathbf{1}$.

□

Definition 3.8. Let M be a normalized adjacency matrix, let $\mathbf{1}^\perp$ be the set of vectors perpendicular to $\mathbf{1}$ such that for any vector $\mathbf{v}^\perp \in \mathbf{1}^\perp, \langle \mathbf{v}^\perp, \mathbf{1}\rangle = 0$. For a vector $\mathbf{u}$ and for $p > 1$, let $||\mathbf{u}||_p$ denote the L_p-norm of $\mathbf{u}$. We define the parameter $\lambda(M)$ as the maximum value of $||M\mathbf{v}||_2$ over all vectors $\mathbf{v} \in \mathbf{1}^\perp$ with $||\mathbf{v}||_2 = 1$.

Lemma 3.9. *Let M be a normalized adjacency matrix. Then, for all $\mathbf{v} \perp \mathbf{1}$, $||M\mathbf{v}||_2 \leq \lambda(M)||\mathbf{v}||_2$.*

Proof. Let $\mathbf{w} = \frac{\mathbf{v}}{||\mathbf{v}||_2}$. Then

$$\begin{aligned}||\mathbf{w}||_2 &= \left|\left|\frac{\mathbf{v}}{||\mathbf{v}||_2}\right|\right|_2 \\ &= 1.\end{aligned}$$

By the definition 3.8 of $\lambda(M)$, we know that $||A\mathbf{w}||_2 \leq \lambda(M)$, so

$$\begin{aligned}\lambda(M) &\geq ||A\mathbf{w}||_2 \\ &= \left|\left|A\frac{\mathbf{v}}{||\mathbf{v}||_2}\right|\right| \\ &= \frac{1}{||\mathbf{v}||_2}||A\mathbf{v}||_2,\end{aligned}$$

and therefore

$$\lambda(M)||\mathbf{v}||_2 \geq ||A\mathbf{v}||_2.$$

□

Lemma 3.10. *For any d-regular undirected graph G with n vertices and normalized adjacency matrix $A(G) = A$, let $\mathbf{p}$ be any probability distribution vector over the vertices. Then*

$$||A^T\mathbf{p} - \mathbf{1}||_2 \leq (\lambda(A))^T.$$

Proof. By lemma 3.9 we know for any vector $\mathbf{v} \perp \mathbf{1}$, we have $||A\mathbf{v}||_2 \leq \lambda(A)||\mathbf{v}||_2$. Furthermore, by lemma 3.7, we know that for any vector $\mathbf{v} \perp \mathbf{1}$, we have $A\mathbf{v} \perp \mathbf{1}$. So, for all $T' \geq 1$ we must have

$$A^{T'}\mathbf{v} \perp \mathbf{1}.$$

Therefore,

$$\begin{aligned}||A^T\mathbf{v}||_2 &= ||A(A^{T-1}\mathbf{v})||_2 \leq \lambda(A)||A^{T-1}\mathbf{v}||_2 \\ &= \lambda(A)||A(A^{T-2}\mathbf{v})||_2 \leq \lambda(A)^2||A^{T-2}\mathbf{v}||_2 \\ &... \\ &= \lambda(A)^{T-1}||A\mathbf{v}||_2 \leq \lambda(A)^T||\mathbf{v}||_2.\end{aligned}$$

Now, we can break the probability distribution vector $\mathbf{p}$ into its components parallel and orthogonal to $\mathbf{1}$ such that $\mathbf{p} = \alpha 1 + \mathbf{p}'$ where $\mathbf{p}' \perp \mathbf{1}$. So,

$$\begin{aligned}0 &= \langle \mathbf{1}, \mathbf{p}' \rangle \\ &= \sum_{i=1}^{n} \mathbf{1}_i \mathbf{p}'_i \\ &= \sum_{i=1}^{n} \frac{1}{n}\mathbf{p}'_i \\ &= \sum_{i=1}^{n} \mathbf{p}'_i.\end{aligned}$$

Because we know $\mathbf{p}$ is a probability distribution, we know the sum of its coordinates $\sum_{i=1}^{n} \mathbf{p}_i = 1$. For all $1 \leq i \leq n$ each coordinate $\mathbf{p}_i = \alpha\mathbf{1}_i + \mathbf{p}'_i = \frac{\alpha}{n} + \mathbf{p}'_i$. So,

$$\begin{aligned}1 &= \sum_{i=1}^{n} \left(\frac{\alpha}{n} + \mathbf{p}'_i\right) \\ &= \sum_{i=1}^{n} \frac{\alpha}{n} + \sum_{i=1}^{n} \mathbf{p}'_i \\ &= \alpha + \sum_{i=1}^{n} \mathbf{p}'_i \\ &= \alpha.\end{aligned}$$

We can break down the operation $A^T\mathbf{p}$ into the components of $\mathbf{p}$ to show that

$$A^T\mathbf{p} = A^T(\mathbf{1} + \mathbf{p}') = \mathbf{1} + A^T\mathbf{p}',$$

which implies that $A^T\mathbf{p} - \mathbf{1} = A^T\mathbf{p}'$.

Because $\mathbf{1}$ and $\mathbf{p}'$ are orthogonal components of $\mathbf{p}$, by the pythagorean theorem, we have $||\mathbf{p}||_2 = ||\mathbf{1}||_2 + ||\mathbf{p}'||_2$, and therefore $||\mathbf{p}'||_2 \leq ||\mathbf{p}||_2$. Then,

$$\begin{aligned}\sum_{i=1}^{n} \mathbf{p}_i^2 &\leq \sum_{i=1}^{n} \mathbf{p}_i \\ (||\mathbf{p}||_2)^2 &\leq 1 \\ \sqrt{(||\mathbf{p}||_2)^2} &\leq \sqrt{1} \\ ||\mathbf{p}||_2 &\leq 1,\end{aligned}$$

and therefore,

$$||A^T\mathbf{p} - \mathbf{1}||_2 = ||A^T\mathbf{p}'||_2 \leq \lambda(A)^T||\mathbf{p}'|| \leq (\lambda(A))^T.$$

□

Lemma 3.11. *For every d-regular undirected connected graph G with n vertices, self-loops at each vertex, and normalized-adjacency matrix A,* $\lambda(A) \leq 1 - \frac{1}{8dn^3}$.

Proof. Let $\mathbf{u}$ be any vector such that $\mathbf{u} \perp \mathbf{1}$ and $||\mathbf{u}||_2 = 1$. Let $\mathbf{v} = A\mathbf{u}$. To prove this claim, we first show that $1 - (||\mathbf{v}||_2)^2 = \sum_{i,j} A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2$.

$$\begin{aligned}\sum_{i,j} A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2 &= \sum_{i=1}^{n}\sum_{j=1}^{n} A_{i,j}\mathbf{u}_i^2 - 2\sum_{i,j} A_{i,j}\mathbf{u}_i\mathbf{v}_j + \sum_{j=1}^{n}\sum_{i=1}^{n} A_{i,j}\mathbf{v}_j^2 \\ &= \sum_{i=1}^{n} \mathbf{u_i}^2 \sum_{j=1}^{n} A_{i,j} - 2\sum_{i,j} A_{i,j}\mathbf{u}_i\mathbf{v}_j + \sum_{j=1}^{n} \mathbf{v}_j^2 \sum_{i=1}^{n} A_{i,j} \\ &= \sum_{i=1}^{n} \mathbf{u_i}^2 - 2\sum_{i,j} A_{i,j}\mathbf{u}_i\mathbf{v}_j + \sum_{j=1}^{n} \mathbf{v}_j^2 \\ &= (||\mathbf{u}||_2)^2 - 2\sum_{i,j} A_{i,j}\mathbf{u}_i\mathbf{v}_j + (||\mathbf{v}||_2)^2.\end{aligned} \tag{3.12}$$

Now, if we expand $\sum_{i,j} A_{i,j}\mathbf{u}_i\mathbf{v}_j$, we get

$$\begin{aligned}\sum_{i,j} A_{i,j}\mathbf{u}_i\mathbf{v}_j &= \sum_{j=1}^{n} \mathbf{v}_j \sum_{i=1}^{n} A_{i,j}\mathbf{u}_i \\ &= \sum_{j=1}^{n} \mathbf{v}_j A^\dagger \mathbf{u}_j \\ &= \sum_{j=1}^{n} \mathbf{v}_j A\mathbf{u}_j \\ &= \langle \mathbf{v}, A\mathbf{u}\rangle \\ &= \langle \mathbf{v}, \mathbf{v}\rangle \\ &= (||\mathbf{v}||_2)^2.\end{aligned}$$

We substitute this result into equation 3.12 to get

$$(||\mathbf{u}||_2)^2 - 2\sum_{i,j} A_{i,j}\mathbf{u}_i\mathbf{v}_j + (||\mathbf{v}||_2)^2 = (||\mathbf{u}||_2)^2 - 2(||\mathbf{v}||_2)^2 + (||\mathbf{v}||_2)^2 = (||\mathbf{u}||_2)^2 - (||\mathbf{v}||_2)^2.$$

Given that $||\mathbf{u}||_2 = 1$,

$$(||\mathbf{u}||_2)^2 - (||\mathbf{v}||_2)^2 = 1 - (||\mathbf{v}||_2)^2.$$

Now, we will show $1 - (||\mathbf{v}||_2)^2 = \sum_{i,j} A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2 \geq \frac{1}{d4n^3}$. Because the sum $\sum_{i,j} A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2$ has no non-negative terms, if at least one $A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2$ term is greater than or equal to $\frac{1}{d4n^3}$ then we have

sufficiently proven that $A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2 = 1 - (||\mathbf{v}||_2)^2 \geq 1$.

Suppose for the sake of contradiction that for all i, j, $A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2 < \frac{1}{d4n^3}$. Because every vertex in G has a self loop, we know that, for all $1 \leq i \leq n$, $A_{i,i} = 1$. So, in order for our supposition to hold, it must be that for all i, $(\mathbf{u}_i - \mathbf{v}_i)^2 < \frac{1}{4n^3}$, and therefore $|\mathbf{u}_i - \mathbf{v}_i| < \frac{1}{2n^{1.5}}$.

Let $f : [1 : n] \to [1 : n]$ be a function that maps some input i to the index of the i^{th} largest coordinate of $\mathbf{u}$. Because $\mathbf{u} \perp \mathbf{1}$, we know $\langle \mathbf{u}, \mathbf{1} \rangle = 0$, which means $\sum_{i=1}^{n} \mathbf{u}_i = 0$. Therefore, it must be that $\mathbf{u}_{f(1)} \geq 0 \geq \mathbf{u}_{f(n)}$. Furthermore, because $||\mathbf{u}||_2 = 1$, then we know some coordinate $\mathbf{u}_i^2 \geq \frac{1}{n}$. Thus, either $\mathbf{u}_{f(1)} \geq \frac{1}{\sqrt{n}}$ or $\mathbf{u}_{f(n)} \leq -\frac{1}{\sqrt{n}}$, which implies $\mathbf{u}_{f(1)} - \mathbf{u}_{f(n)} \geq \frac{1}{\sqrt{n}}$.

Now, because we know there are $n-1$ coordinates ordered between $\mathbf{u}_{f(1)}$ and $\mathbf{u}_{f(n)}$, we know there exists some k such that the difference between two of the consecutively ordered coordinates

$$|\mathbf{u}_{f(k)} - \mathbf{u}_{f(k+1)}| \geq \frac{1}{\sqrt{n}(n-1)} = \frac{1}{n^{1.5} - \sqrt{n}} \geq \frac{1}{n^{1.5}}.$$

Now, we define the partition $S = \{f(1), ..., f(k)\}$ and let $\overline{S} = [n] \setminus S$. For any $i \in S$, $j \in \overline{S}$, it must hold that $\mathbf{u}_i - \mathbf{u}_j \geq \frac{1}{n^{1.5}}$. Because G is connected, we know there exists some edge (i', j') such that $f(i') \in S$ and $f(j') \in \overline{S}$.

By the triangle inequality,

$$|\mathbf{u}_{i'} - \mathbf{v}_{j'}| \geq |\mathbf{u}_{i'} - \mathbf{v}_{j'}| - |\mathbf{u}_{j'} - \mathbf{v}_{j'}|.$$

Furthermore, by our earlier supposition, we assume $|\mathbf{u}_{j'} - \mathbf{v}_{j'}| < \frac{1}{2n^{1.5}}$. So,

$$\begin{aligned} |\mathbf{u}_{i'} - \mathbf{v}_{j'}| &\geq |\mathbf{u}_{i'} - \mathbf{v}_{j'}| - |\mathbf{u}_{j'} - \mathbf{v}_{j'}| \\ &\geq |\mathbf{u}_{i'} - \mathbf{v}_{j'}| - \frac{1}{2n^{1.5}} \\ &\geq \frac{1}{n^{1.5}} - \frac{1}{2n^{1.5}} \\ &= \frac{1}{2n^{1.5}}. \end{aligned}$$

Because there is an edge between vertices $v_{i'}$ and $v_{j'}$ in G, it must be that $A_{i',j'} \geq \frac{1}{d}$. Thus,

$$A_{i',j'}(\mathbf{u}_{i'} - \mathbf{v}_{j'})^2 \geq \frac{1}{d}(\frac{1}{2n^{1.5}})^2 = \frac{1}{d4n^3},$$

which, finishing our argument by contradiction, implies

$$\frac{1}{d4n^3} \leq \sum_{i,j} A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2 = 1 - (||\mathbf{v}||_2)^2.$$

Since $\frac{1}{d4n^3} \leq 1 - (||\mathbf{v}||_2)^2$, we know

$$(||\mathbf{v}||_2)^2 \leq 1 - \frac{1}{d4n^3}.$$

Now, consider

$$(1 - \frac{1}{d8n^3})^2 = 1 - \frac{1}{d4n^3} + \frac{1}{(d8n^3)^2}.$$

It so follows that

$$(||\mathbf{v}||_2)^2 \leq 1 - \frac{1}{d4n^3} \leq (1 - \frac{1}{d8n^3})^2,$$

and therefore

$$||\mathbf{v}||_2 \leq 1 - \frac{1}{d8n^3}.$$

For all $\mathbf{u}' \perp \mathbf{1}$ with an L_2-norm of one, $||A\mathbf{u}||_2 = \lambda(A)$. Therefore, because $\mathbf{u}$ is a vertex with $\mathbf{u} \perp \mathbf{1}$ and $||\mathbf{u}||_2 = 1$, and we know $||\mathbf{v}||_2 = ||A\mathbf{u}||_2 \leq 1 - \frac{1}{d8n^3}$, it must hold that $\lambda(A) \leq 1 - \frac{1}{d8n^3}$.

□

Theorem 3.13. *Let G be any d-regular undirected graph with n vertices such that each vertex in G has a self-loop and normalized adjacency matrix A. Let s be a vertex in G. Let $T > 14dn^3 \log n$, $n \geq 16$, and let X^T denote the probability distribution for the current vertex of the T^{th} step in a random walk from s. Then, for every j connected to s, $\Pr(X^T = j) \geq \frac{1}{2n}$.*

Proof. Let $\mathbf{p}$ be any probability distribution vector. By lemma 3.10, we know that $||A^T\mathbf{p} - \mathbf{1}||_2 \leq \lambda(A)^T \leq (1 - \frac{1}{d8n^3})^T$.
So, we take

$$(1 - \frac{1}{d8n^3})^T \leq (1 - \frac{1}{d8n^3})^{14dn^3 \log n}$$
$$= \exp\left(14dn^3 \log n \log\left(1 - \frac{1}{d8n^3}\right)\right) \quad = n^{-\frac{7}{4}}.$$

Because $d > 1$ and $n \geq 1$, we can use $\log(1 + x) \leq x$ for $x > -1$ to show

$$(1 - \frac{1}{d8n^3})^T \leq \exp\left(14dn^3 \log(n) \log\left(1 - \frac{1}{d8n^3}\right)\right)$$
$$\leq \exp\left(14dn^3 \frac{-1}{d8n^3} \log n\right)$$
$$= \exp\left(\log n^{-\frac{7}{4}}\right).$$

Since $n \geq 16$,

$$\frac{1}{n^{\frac{7}{4}}} \leq \frac{1}{2n^{1.5}}.$$

Now, let vector $\mathbf{u}$ be an n coordinate vector with the term $\frac{1}{\sqrt{n}}$ in every coordinate. Since $\frac{1}{2} + \frac{1}{2} = 1$, we will apply Hölder's inequality for $p = q = 2$ to $\mathbf{u}$ and $(A^T\mathbf{p} - \mathbf{1})$ to get that

$$||(A^T\mathbf{p} - \mathbf{1})\mathbf{u}||_1 \leq ||A^T\mathbf{p} - \mathbf{1}||_2 ||\mathbf{u}||_2,$$

which we can expand to show that

$$||(A^T\mathbf{p} - \mathbf{1})\mathbf{u}||_1 = \sum_{i=1}^{n} \left|(A^T\mathbf{p} - \mathbf{1})_i \mathbf{u}_i\right|$$
$$= \frac{1}{\sqrt{n}} \sum_{i=1}^{n} \left|(A^T\mathbf{p} - \mathbf{1})_i\right|$$
$$= \frac{1}{\sqrt{n}} ||A^T\mathbf{p} - \mathbf{1}||_1$$
$$\leq ||A^T\mathbf{p} - \mathbf{1}||_2 ||\mathbf{u}||_2$$
$$= ||A^T\mathbf{p} - \mathbf{1}||_2 \sqrt{\sum_{i=1}^{n} \mathbf{u}_i^2}$$
$$= ||A^T\mathbf{p} - \mathbf{1}||_2 \sqrt{\frac{1}{n} \sum_{i=1}^{n} 1}$$
$$= ||A^T\mathbf{p} - \mathbf{1}||_2.$$

Thus, $\frac{1}{\sqrt{n}} ||A^T\mathbf{p} - \mathbf{1}||_1 \leq ||A^T\mathbf{p} - \mathbf{1}||_2$. Since $||A^T\mathbf{p} - \mathbf{1}||_2 \leq \frac{1}{2n^{1.5}}$, it then follows that $||A^T\mathbf{p} - \mathbf{1}||_1 \leq \frac{1}{2n}$.

Because $||A^T\mathbf{p} - \mathbf{1}||_1 = \sum_{i=1}^{n} |(A^T\mathbf{p} - \mathbf{1})_i| \leq \frac{1}{2n}$, then we know for all j connected to our start vertex s that

$$|(A^T\mathbf{p})_j - \mathbf{1}_j| = |(A^T\mathbf{p})_j - \frac{1}{n}| \leq \frac{1}{2n},$$

and therefore

$$\frac{1}{n} - (A^T\mathbf{p})_j \leq \frac{1}{2n},$$

which implies that $(A^T\mathbf{p})_j \geq \frac{1}{2n}$. The j^{th} coordinate of $A^T\mathbf{p}$ represents the probability that the walk is then currently at vertex j during the T^{th} step of the random walk. So, it follows that for all j connected to s, $\Pr[X_T = j] \geq \frac{1}{2n}$.

□

Corollary 3.14. *For an undirected d-regular graph $G = (V, E)$ with $n = |V| \geq 16$, given any $0 < \delta \leq 1$,* **undirectedPath***(G, s, t) will output* **Accept** *if vertices s and t in G are connected in time $O(n^4 \log n)$ with probability $p > 1 - \delta$ after running a number $k > -2n\log(\delta)$ iterations of* **undirectedPath***(G, s, t).*

Proof. Let X_i be a random variable that indicates whether **undirectedPath**(G, s, t) outputs **Accept** on some i^{th} iteration of running the algorithm. Let $X = \sum_{i=1}^{k} X_i$. Then **Acccept** is returned by one of the k iterations of the algorithm iff $X \geq 1$. Therefore, the probability that the algorithm accepts over k iterations is

$$\Pr(X \geq 1) = 1 - \Pr(X = 0) = 1 - \Pr\left(\bigcap_{i=1}^{k} X_i = 0\right).$$

Because each iteration of **undirectedPath** is independent of the others and all have identical probability of ending on vertex t an outputting **Accept**,

$$\Pr\left(\bigcap_{i=1}^{k} X_i = 0\right) = \prod_{i=0}^{k} \Pr(X_i = 0) = \Big(\Pr(X_j = 0)\Big)^k,$$

for any $1 \leq j \leq k$. We know by theorem 3.13, that the probability p^* that t is the final vertex in the walk of **undirectedPath**(G, s, t) given that s and t are connected is $p^* \geq \frac{1}{2n}$. So, the probability p' that t is the ending vertex of the walk is $p' \leq 1 - \frac{1}{2n}$. Therefore,

$$\Big(\Pr(X_j = 0)\Big)^k \leq \Big(1 - \frac{1}{2n}\Big)^k,$$

which we can rewrite as

$$\left(\left(1 - \frac{1}{2n}\right)^{2n}\right)^{\frac{k}{2n}}.$$

For all $m > 1$, we have $0 < (1 - \frac{1}{m})^m \leq \frac{1}{e}$. So,

$$\begin{aligned}\Big(\Pr(X_j = 0)\Big)^k &\leq \left(\left(1 - \frac{1}{2n}\right)^{2n}\right)^{\frac{k}{2n}} \\ &\leq \left(\frac{1}{e}\right)^{\frac{k}{2n}} \\ &= \exp\left(-\frac{k}{2n}\right).\end{aligned} \tag{3.15}$$

Now, substituting $k \geq -2n\log(\delta)$, we have

$$\begin{aligned}\Big(\Pr(X_j = 0)\Big)^k &\leq \exp\left(-\frac{k}{2n}\right) \\ &\leq \exp\Big(\log(\delta)\Big) \\ &= \delta.\end{aligned}$$

Thus, when $k \geq -2n\log(\delta)$, for any $0 < \delta \leq 1$,

$$\Pr(X \geq 1) = 1 - \Big(\Pr(X_j = 0)\Big)^k \geq 1 - \delta.$$

Because each run **undirectedPath**(G, s, t) uses $O(n^3 \log n)$ steps, then running $-2n\log(\delta)$ trials of **undirectedPath**(G, s, t) runs in $O(-2n\log(\delta) \cdot n^3 \log n) = O\Big(2n \log\left(\frac{1}{\delta}\right) \cdot n^3 \log n\Big) = O(n^4 \log n)$ time.

□

Acknowledgments. I would like to thank my thesis advisor Andrew Drucker for supporting my research and cultivating my interest in the theory of algorithms. I'd also like Eric Thoma for countless hours of conversations and advice, as well as for being a genuinely great friend; it's unlikely I'd even understand math without his help. I'd also like to thank Christopher Rey, Ann Shih, Zachary Levine, Caitlin Sheehan, Kim Vu, Kanisha Williams, and countless others for their unwavering support and belief in me, even when my own wavered.

References

[1] Motwani, Rajeev & Raghavan, Prabhakar (1995). Randomized Algorithms (1st ed.). New York, NY: Cambridge University Press.

[2] Karger, David (1993). "Global Min-cuts in RNC and Other Ramifications of a Simple Mincut Algorithm". Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms.

[3] Arora, Sanjeev, Princeton University, cited 2018: "Lecture 2: Karger's Min Cut Algorithm". [Available online at https://www.cs.princeton.edu/courses/archive/fall13/cos521/lecnotes/lec2final.pdf.]

[4] Arora, Sanjeev & Barak, Boaz. (2009). Computational Complexity: a Modern Approach. New York, NY: Cambridge University Press.