Exploring Random Walks on Graphs for Protein Function Prediction

Angela M. Dahl

Bowdoin College, angeladahl18@gmail.com

Exploring Random Walks on Graphs for Protein Function Prediction

An Honors Paper for the Department of Mathematics

By Angela M. Dahl

Bowdoin College, 2018

Contents

Acknowledgements

This work would not have been possible without the guidance of Professor Amanda Redlich, who has never failed to provide wisdom, patience, and encouragement throughout this process. She went out of her way to pursue this project with me, and for that I am deeply grateful. I would also like to thank the entire Bowdoin math department, in particular Professors Thomas Pietraho and Jack O'Brien, for helping me discover a love for mathematics and for continually challenging me to become the student I am today. Of couse, I would be nowhere without the endless support of my friends and family. Thanks to Hallie Lam for sitting through countless hours of practice presentations and bad jokes about random walks. Finally, thanks to my mom, Lorie DuBois, who has always encouraged me to pursue what I love, and whose intelligence and independence inspires me every day.

1 Background

1.1 Problem: Protein function prediction

With the completion of the Human Genome Project in 2003, scientists have finally identified all of the approximately 20,500 genes in the human body [17]. Genes act as a sort of blueprint for the human body and its functions, providing the code to build proteins, which put those genes into action. Proteins come in many forms – antibodies, enzymes, messengers, structural components, and transport and storage proteins, to name a few – and they work together to carry out virtually every biological process that sustains human life [21]. While we now know the size of the human genome, we are much less certain about the number of proteins it ultimately produces; a single gene may code for a hundred different proteins, and as a result, researchers estimate that there may be millions, or even billions, of different proteins in the human body [27]. Furthermore, although the processes they carry out *together* are well-studied, we still know relatively little about the biological functions of individual proteins. This is an important question: if we knew more about the functions of specific proteins, then we could better understand diseases such as cancer that disrupt normal protein function, and ultimately we could produce better, more targeted treatments to these diseases [32]. Consequently, protein function prediction has become a major problem in modern bioinformatics.

It's important to note that what is meant by a protein's "biological function" can be somewhat ambiguous. While a protein's genetic sequence and structure are easily identified and categorized, its function can be described in many ways, such as by its biochemical activity or its place in a signaling pathway. Following other classical papers on protein function prediction ([10], [29], [31]), we'll be using the Munich Information Center for Protein Sequences Functional Catalogue (MIPS FunCat), a system that classifies protein function at increasing levels of specificity using a controlled set of vocabulary [28]. Additionally, we'll primarily focus on proteins in the *Saccharomyces cerevisiae* protein network. Commonly known as baker's yeast, *S. cerevisiae* has become the standard model organism for studies in proteomics, as its genome has been fully mapped since 1996 and the biological functions of its proteins are well known [16].

Of the protein sequences we have successfully identified within the genome, less than 1% of them have “experimentally verified” biological functions; about 64% have functions that are “inferred” from proteins that appear to be similar; and the remaining third are considered to have an “unknown” or “uncharacterized” function. Furthermore, even the proteins whose functions we have inferred may not have accurate annotations. Most of these inferences are based on the idea that two proteins with similar genetic sequences (and therefore a similar evolutionary history, in theory) will likely have a similar function [12]. However, a single protein can have many functions, and proteins that are related evolutionarily may still develop unrelated functions, as even a tiny, single change to a protein’s DNA sequence can have enormous effects on its folding patterns and function [2]. Conversely, completely *unrelated* proteins may nevertheless evolve to have the same function. As a result, simply comparing genetic sequences is not always a reliable way to gauge how similar two proteins are in function, and a better method is needed [12].

The problem of predicting protein function has been approached from many angles, including comparing individual genetic sequences, as explained above, or protein structure. While these methods achieve about 60% and 67% accuracy, respectively, we’re still searching for simpler, more reliable ways of predicting function. This paper focuses on network-based methods, a relatively novel and very promising approach that takes advantage of the wealth of information we have on the interactions that occur among proteins. We’re able to study a protein-protein interaction (PPI) network by modeling as a mathematical graph, in which each vertex represents a protein and each edge represents an interaction between two proteins. By analyzing PPI networks on a global scale, we’re often able to obtain more accurate results than with methods comparing the characteristics of single proteins [12].

To better understand the research behind network-based protein function prediction, it’s important that we identify some important structural features that make PPI networks unlike other types of networks. First, biologists have observed that proteins organize themselves into complexes that work together to carry out biological processes. Since many proteins have multiple functions or participate in multiple biological processes, these complexes are often fluid. In some cases, groups of two or more proteins are *always* found together; these groups, called “modules,” participate as a functional unit across multiple complexes [13]. Another key feature of the yeast PPI network is the existence of “hub” proteins, which play an important role within their complexes. A hub protein is defined as a protein that interacts with at least five other proteins in the PPI network [15]. In most cases, the removal of a single protein will not have a significant effect on the biological processes in which it participates; however, the removal of a hub protein is almost always very distruptive, signaling the biological importance of these high-degree proteins [18]. Importantly, most hub proteins have different functions from the proteins they interact with – for example, one common hub protein simply helps others fold properly, and another binds to many different proteins while assisting in the translation of DNA to RNA [9].

The existence of hub proteins significantly complicates the way PPI networks behave, and, consequently, it becomes more difficult to use what we know about the network to predict the function of unknown proteins, which is our ultimate goal. This is largely because clustering algorithms, commonly used in other types of networks, simply *don’t work for PPI networks.* Typically, in cases such as social networks, we’re able to break a network down into “clusters” to uncover information about the structure of the network or the identity of unknown nodes [29]. In general, clustering algorithms identify clusters by finding the optimal partition of the network into groups whose connections are maximized within the group and minimized outside of the group. When done right, clustering reveals groups that are highly “naturally associated” within groups but relatively distinct between groups [1]. For instance, in a social network, clustering will reveal close-knit groups of friends or groups of people who share a common interest [26].

When studying PPI networks, clustering no longer works quite how we want it to. As we’ve seen, proteins naturally organize themselves into modules that work together, and clustering works well to uncover these modules [13]. When we want to predict the function of individual proteins within these modules, however, clustering algorithms are no longer effective. This is because clustering typically relies on a key assumption: that an edge between vertices in a network implies that they are similar. In social networks, for example, similar nodes (people) will naturally have more connections between them. People are usually friends because they share common interests, so once clustering reveals these friend groups, we can assume that a person we don’t know much about will have similar interests to other people in their friend group. Unfortunately for PPI networks, an edge does *not* imply similarity, but rather interaction, and one protein simply interacting with another may tell us nothing at all about how similar the two are in function. As we've seen, hub proteins interact with many different types of proteins that are often completely unrelated in function. As a result, over 95% of nodes in a PPI network have a shortest-path distance of only two or three apart, so two proteins being "close" in distance tells us almost nothing [9]. As a consequence, clustering algorithms fail. It turns out that "guilt by association" (an algorithm in which an unknown protein is simply assigned the most common function of its known neighbors) was found to be almost universally more effective when tested against six standard clustering algorithms on the yeast PPI network [29].

So what do we do instead? Is there a better way to predict protein function than by simple "guilt by association"? We would like to find a better way to measure protein similarity that doesn't use clustering but that is still more nuanced than shortest-path distance. As we've seen, most proteins are very "close" together within PPI networks because most of them interact with hub proteins. Consequently, we can better predict a protein's function by looking not at which hub proteins it interacts with, but rather at the *low-degree* proteins it interacts with, which are likely to have more specialized functions. Ideally, we'd like to find a similarity metric that captures the idea that similar proteins tend to share similar low-degree neighbors and follow similar patterns of interaction overall [9].

1.2 Diffusion State Distance: A possible solution

In 2013, a new metric called Diffusion State Distance was introduced by Cao et al. to accomplish just that. Diffusion State Distance (DSD) relies on the notion of a "random walk" on a mathematical graph; in this case, our graph is the PPI network, in which the vertices are proteins and the edges represent interactions between two proteins. A random walk is exactly what it sounds like: pick a vertex in the graph and start "walking" randomly along the edges from vertex to vertex. The basic premise of DSD is this: if a random walk on the PPI network starting from one protein follows a similar route to a random walk starting from another protein, then the functions of those two proteins are therefore also similar [9].

To see this idea in action, let's consider an example (Figure 1). Proteins i and j both interact with a hub protein of degree 5, and j and k both interact with a protein of degree 2. By our reasoning above, j and k should be more similar in function than i and j because they share a low-degree neighbor in common rather than a hub, which is likely to have a different function from either of them. Although $d(i, j) = d(j, k) = 2$ using traditional shortest-path distance, we'd like DSD to tell us that j and k are "closer" (in function) than i and j are. Sure enough, a random walk starting from either j or k will highlight that low-degree neighbor and other low-degree vertices near them by spending more time on their leg of the graph before walking around the rest of the network. On the other hand, a random walk started from protein i will spend more time on its own leg with its own low-degree neighbors. Notice that once *any* walk reaches the hub, it will continue to traverse the rest of the network at random, so random walks from i, j, and k will all look the same after reaching the hub. Thus, when we compare the pathways of random walks from i, j, and k, they will differ only by the amount of time they spend on their respective starting legs. As a result, the pathway of a random walk from i will be quite different from the pathway of a walk from j, while the pathway of a walk from k will be similar to the pathway of a walk from j. Therefore, we'll find that $DSD(i, j) > DSD(k, j)$.

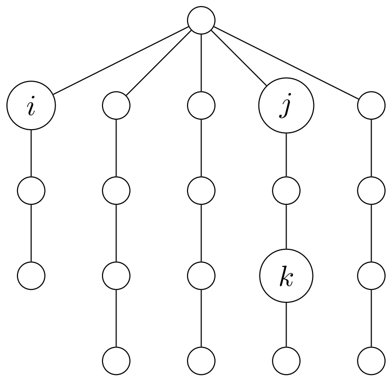


Figure 1: $DSD(i, j) > DSD(k, j)$.

In Section 3, we'll formalize the mathematics of DSD by defining a way to store the pathway of a random walk, which will then allow us to compare the pathways of two random walks to measure their similarity [9]. While DSD is a useful way to begin thinking about random walks on protein networks, the original research in this paper is focused on a newer metric called Exit Frequency Distance (EFD). Essentially, EFD is a refined version of DSD that compares the pathways of random walks that end at distributions, rather than random walks that continue for a set number of steps. Before discussing DSD and EFD more formally, however, we'll first need to build up a better understanding of random walks and how they behave on graphs.

2 Random walks on graphs

2.1 Introduction to random walks

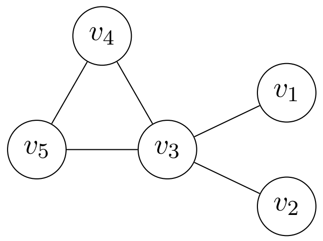
Random walks have been studied by probability theorists for decades, and they have applications in many areas of mathematics and physics. In 1905, statistician Karl Pearson introduced the idea of a random walk by proposing that one choose a random angle, walk for ℓ yards, then choose another random angle and walk for ℓ yards, repeating the process n times [14]. The classical random walk involves randomly traversing an infinite graph: as a simple example, consider standing on the integer number line at $x = 0$, then flipping a coin and moving +1 on heads and -1 on tails. We can now study the behavior of the resulting walk, such as where we expect it to go and how long it will take to get there. Of course, random walks can be considerably more complex; for example, they are famously used to model Brownian motion, a phenomenon in which particles suspended in fluids and gases appear to move erratically throughout the space containing them [19]. Random walks can be performed on many different mathematical spaces, but here we'll focus on random walks on graphs. To start, we'll define a graph's transition matrix, which will govern the pathway of a random walk.

Definition 1. [22] Consider a finite graph $G = (V, E)$ with vertex set V and unweighted edges E. For the purposes of this paper, we will only consider connected graphs; if a graph is disconnected, then the following definitions and results will apply for each connected component. The **transition matrix** M of G gives the probability of going from each vertex i to another vertex j in one step, given by

$$M_{ij} = \begin{cases} 1/d(i) & \text{if } (i,j) \in E(G) \\ 0 & \text{otherwise} \end{cases}, \sum_{j \in N(i)} M_{ij} = 1,$$

where $d(i)$ is the degree of vertex i and $N(i)$ is the set of neighboring vertices of i.

Intuitively, if a random walk is currently on vertex i, then there is an equal probability of going to any adjacent vertex j, so the probability of transitioning from i to j is $1/d(i)$. Additionally, we assume that a random walk will always keep moving from one vertex to another (it will never remain on the same vertex at both time t and $t+1$), so the sum of the transition probabilities from i to its neighbors is 1. Figure 2 gives an example.



$$M = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix}$$

Figure 2: A graph and its transition matrix.

We can now define a random walk on a graph G.

Definition 2. [22] Consider a graph G with transition matrix M. Given a starting vertex v_0, a **random walk** on G is a sequence of vertices $v_0, v_1, \ldots, v_n$ such that if the walk is at node v_t at time t, then it moves to a neighbor of v_t according to M_t.

A random walk is a reversible finite Markov chain, as the next state of the walk is dependent only on the current state.

2.2 Distributions on graphs

Definition 3. [22] Consider a random walk on a graph $G = (V, E)$ with starting vertex v_0. The **distribution** P_t on G is a vector of length $|V|$ that gives the probability of the walk being at each vertex in V at time t.

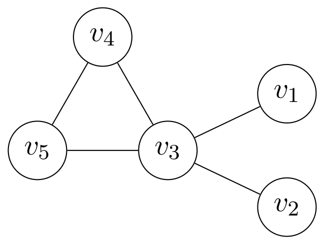
Distributions come in many shapes and sizes, depending on the structure of the graph, where the walk started, how long it's been going on for, and so on. A distribution might be spread evenly among all vertices in the graph, for instance, or it might be concentrated on just a single vertex (with probability 1) or two vertices (with probability one-half each, or perhaps p and $1 - p$). If a graph has distribution P_t at time t, then the distribution one step later can be calculated as

$$P_{t+1} = M^T P_t. \tag{1}$$

The starting state of a random walk may also be drawn from an initial distribution P_0. If we know P_0, then we can calculate the distribution of the random walk at any time t as

$$P_t = \left(M^T\right)^t P_0. \tag{2}$$

Figure 3 gives an example of the distributions on a graph for the first few steps of a random walk whose initial state is drawn from the uniform distribution. As Figure 3 demonstrates, as the random walk continues on this graph, it is more likely to be on a higher-degree vertex than a lower-degree vertex at any given time.



$$P_0 = \begin{pmatrix} \frac{1}{5} \\ \frac{1}{5} \\ \frac{1}{5} \\ \frac{1}{5} \\ \frac{1}{5} \end{pmatrix} \qquad P_1 = \begin{pmatrix} \frac{1}{20} \\ \frac{1}{20} \\ \frac{3}{5} \\ \frac{3}{20} \\ \frac{3}{20} \end{pmatrix} \qquad P_2 = \begin{pmatrix} \frac{3}{20} \\ \frac{3}{20} \\ \frac{1}{4} \\ \frac{9}{40} \\ \frac{9}{40} \end{pmatrix} \qquad P_3 = \begin{pmatrix} \frac{1}{16} \\ \frac{1}{16} \\ \frac{11}{20} \\ \frac{13}{80} \\ \frac{13}{80} \end{pmatrix}$$

Figure 3: Distributions on a graph.

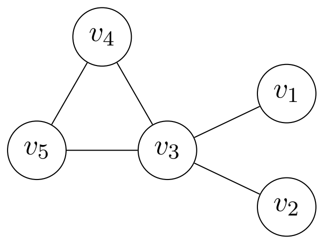
An important distribution that we'll return to frequently is the stationary distribution, denoted π, at which the distribution on the graph does not change by taking another step.

Definition 4. [22] Let P_t be the distribution on a graph G at time t. A distribution is **stationary** if $P_t = P_{t+1}$.

Every graph G has a unique stationary distribution π whose entries are given by

$$\pi_v = \frac{d(v)}{2|E|}, \tag{3}$$

where $d(v)$ is the degree of vertex v and $|E|$ is the number of edges in G. For a graph has transition matrix M, π is also the left eigenvector of M [22]. Figure 4 shows a graph and its stationary distribution.



$$\pi = \begin{pmatrix} \frac{1}{10} \\ \frac{1}{10} \\ \frac{2}{5} \\ \frac{1}{5} \\ \frac{1}{5} \end{pmatrix}$$

Figure 4: A graph and its stationary distribution π.

For non-bipartite graphs, a random walk will always tend to π as $t \to \infty$. For non-bipartite graphs such as trees, a "lazy" random walk, which moves to a new state with probability $\frac{1}{2}$ and remains in the current state with probability $\frac{1}{2}$, will also tend to π; laziness simply doubles the expected length of the walk [3].

2.3 Hitting and access time

Definition 5. [22] **Hitting time** $H(i, j)$ is the expected number of steps it takes for a random walk starting from vertex i to reach vertex j.

Hitting time is an important concept when studying random walks on graphs because it gives us a sense of the "distance" between two vertices in terms of time, revealing properties about the graph that might not be obvious to the naked eye. For example, say we have two vertices i and j with a shortest-path distance of 2, but the vertex separating them has degree 5 (in the context of a PPI network, this would be a hub protein) (Figure 5). Although i and j are close in shortest-path distance, the hitting time $H(i, j)$ will be relatively high because once a random walk from i reaches the hub, it only has a 20% chance of choosing vertex j from there. As a result, we would expect a walk from i to spend more time randomly traversing the rest of the graph before it returns to the hub and eventually ends up at j.

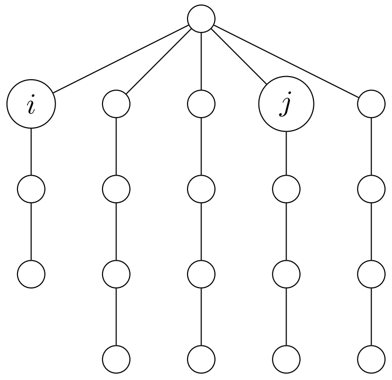


Figure 5: $H(i,j) = 34$.

On the other hand, if i and j are separated by a degree-3 vertex, then the hitting time will be shorter, as a random walk from i has a much simpler route to reach j (Figure 6).

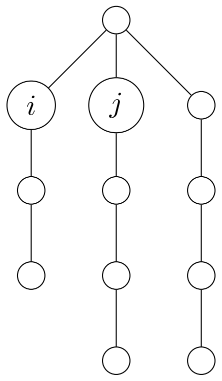


Figure 6: $H(i,j) = 20$.

In general, hitting time is difficult to calculate, though an explicit formula does exist (see [25]). While hitting time is a powerful basic ingredient in calculations involving random walks, we often choose to study general walks between distributions rather than between specific starting and ending vertices. We call the the expected time between distributions the access time, which can be calculated using hitting time. Note that access time uses the same notation as hitting time, with Greek letters representing distributions rather than Roman letters representing single vertices, as in hitting time.

Definition 6. [22] **Access time** $H(\sigma, \tau)$ is the expected number of steps it takes for a random walk with starting distribution σ to reach target distribution τ.

The access time between a single vertex (that is, the distribution concentrated on a single vertex) and a distribution can be calculated as

$$H(\sigma, i) = \sum_{k \in V} \sigma_k H(k, i) \tag{4}$$

or

$$H(i, \tau) = \max_{j \in V} \big(H(i, j) - H(\tau, j) \big). \tag{5}$$

We can use these formulas to calculate the access time between two general distributions. For any initial distribution σ and target distribution τ, Lovász and Winkler [25] show that

$$H(\sigma, \tau) = \max_{j \in V} \big(H(\sigma, j) - H(\tau, j) \big). \tag{6}$$

2.4 Stopping rules and exit frequencies

Random walks to target distributions are typically governed by stopping rules, which tell the walk when to stop. This will be a key idea when we introduce Exit Frequency Distance in Section 4, which utilizes stopping rules to halt random walks at specific distributions. Lovász and Winkler [23] provide numerous results relating to stopping rules, some of which are presented below.

Definition 7. [23] Consider a random walk on a graph G with starting state σ. A **stopping rule** $\Gamma(\sigma, \tau)$ gives the probability of continuing the walk at any given time and halts the walk once it reaches state τ.

For example, a stopping rule for a walk on the graph in Figures 2 - 4 might be: "pick a vertex at random, and walk until you hit vertex v_1." This gives us a stopping rule from the uniform distribution $(\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5})$ to the singleton distribution concentrated on v_1 $(1, 0, 0, 0, 0)$. If we happened to pick v_1 to start with, then we'd be done, and our walk would be of length 0. On the other hand, if we started on v_5, then it will take us significantly longer to reach v_1, as the route it takes will be random. The mean length tells us how long we can expect the walk to take.

Definition 8. [23] The **mean length** of a stopping rule $\Gamma(\sigma, \tau)$ is its expected duration, given by

$$\sum_{i,j} \sigma_i \tau_j H(i, j).$$

There can be many different stopping rules that produce a walk from σ to τ – for example, we might use the stopping rule "pick a vertex at random, and walk until you hit vertex v_1 for the second time." This stopping rule has the same starting and ending distributions, but its mean length will be much longer. Often, we care only about the most efficient, or optimal, route from σ to τ.

Definition 9. [23] A stopping rule $\Gamma(\sigma, \tau)$ is **optimal** if its mean length is minimal over all stopping rules from σ to τ.

It's also helpful to know the pathway that we can expect a random walk to take to get from one distribution to another. To get an idea of this pathway, we can look at a stopping rule's exit frequencies.

Definition 10. [23] Given a stopping rule $\Gamma(\sigma, \tau)$, its **exit frequencies** $\{x_i\}_{i \in V}$ are the expected number of times a walk from σ leaves each vertex $i \in V$ on its way to τ.

The mean length of a stopping rule is the sum of its exit frequencies. It turns out that the exit frequencies of an optimal stopping rule $\Gamma(\sigma, \tau)$ are uniquely determined by σ and τ and can be calculated using access time [24]. For a vertex $i \in V$, the exit frequency x_i for a stopping rule from σ to τ is given by

$$x_i(\sigma, \tau) = \pi_i(H(\sigma, \tau) + H(\tau, i) - H(\sigma, i)). \tag{7}$$

Using Equation 7, we can think about an exit frequency x_i for a stopping rule $\Gamma(\sigma, \tau)$ as the expected number of extra steps that would be required to first walk from σ to τ, then to vertex i from there ($H(\sigma, \tau) + H(\tau, i)$), rather than walking straight from σ to i ($H(\sigma, i)$) [4]. Exit frequencies are an important concept that we will return to later as we define Exit Frequency Distance, which uses exit frequencies to compare the pathways of random walks from different vertices.

2.5 The forget distribution

Hitting and access time give us the most basic language with which to talk about the behavior of random walks on graphs, and we can now use them to define some other properties of random walks. As mentioned above, for most graphs, a random walk will tend to the stationary distribution π as $t \to \infty$. However, there are other important distributions that a walk can hit before it reaches π.

One well-studied distribution is the forget distribution μ, in which the random walk has "forgotten" where it started, and the distribution no longer reflects the starting state of the walk. To better understand the forget distribution, we'll introduce some ideas related to a walk "forgetting" where it started.

Definition 11. [25] The **mixing time** T_{mix} is the expected time for a random walk to reach the stationary distribution π from any starting state σ.

$$T_{\text{mix}} = \max_{\sigma} H(\sigma, \pi)$$

Definition 12. [25] The **forget time** T_{forget} is the minimum expected time for a random walk to reach the same distribution (not necessarily the stationary distribution) from any starting state σ.

$$T_{\text{forget}} = \min_{\tau} \max_{\sigma} H(\sigma, \tau)$$

Mixing time and forget time are closely related but distinct: while mixing time is concerned with how long it takes, on average, for any walk to reach the stationary distribution π, forget time focuses on the average time it takes any walk to simply reach the *same* distribution τ, which might happen before it reaches π. It follows that

$$T_{\text{forget}} \leq T_{\text{mix}}.$$

Definition 13. [25] The **forget distribution** μ is the distribution minimizing $T_{\text{forget}} = \min_{\tau} \max_{\sigma} H(\sigma, \tau)$.

The forget distribution is the first distribution reached that no longer reflects where the walk began – when a walk has reached μ, it is equally likely to have started on any vertex. It turns out that the forget distribution is unique for every graph and can be calculated as

$$\mu_i = \pi_i \big(1 + \sum_{k \in V} M_{ik} H(k, \pi) - H(i, \pi)\big). \tag{8}$$

This formula tells us how much the mixing time from vertex i to π differs from the average mixing time over all vertices, then weights that difference for vertex i based on the probability given by π_i (just as the stationary distribution gives higher weights to vertices with higher degree, the forget distribution will also have higher probabilities of being at higher-degree vertices). Lovász and Winkler show that this formula achieves $\min_\tau \max_\sigma H(\sigma, \tau)$ [25].

3 Diffusion State Distance

Now that we've built up a working understanding of random walks on graphs, we can return to Diffusion State Distance (DSD). Recall from Section 1.2 that DSD, introduced by Cao et al. in 2013 [9], provided a novel way of measuring protein similiarity (their "distance" in the PPI network) by comparing the pathway of a random walk from one protein with the pathway of a random walk from another protein, the idea being that the more similar their pathways, the more similar they are in function. DSD turned out to be a huge success in predicting protein function; when substituted for shortest-path distance in classical network-based methods of functional annotation in the yeast PPI network, DSD attained universally more accurate results. Much work has been done to further refine DSD, and the original research later in this paper is focused on Exit Frequency Distance, another metric inspired by DSD. Since EFD uses many of the same concepts as DSD, we'll first build up the mathematics of DSD before going over the details of EFD.

3.1 Formal definition

We'll store information about the pathway of a random walk of length k in a vector denoted $He^{\{k\}}(i)$. Essentially, this keeps a tally of the number of times we expect to visit each vertex j during a random walk for k steps from vertex i.

Definition 14. [9] Given some integer $k > 0$, define $He^{\{k\}}(i, j)$ to be the expected number of times that a random walk starting at vertex i and proceeding for k steps will visit vertex j.

Notice that $He^{\{k\}}(i, j)$ is similar to an exit frequency, only instead of fixing a target distribution, we're fixing the length of the random walk. As a result, our walk is not governed by a typical stopping rule, so we can't simply calculate the exit frequency for each vertex. Instead, we calculate $He^{\{k\}}(i, j)$ using the formula

$$He^{\{k\}}(i, j) = \sum_{\ell=0}^{k} M_{ij}^{\{\ell\}}, \tag{9}$$

where $M_{ij}^{\{\ell\}}$ gives the probability of being at vertex j on step ℓ of a random walk starting at i. Notice that we cannot only use $He^{\{k\}}(i,j)$ to measure the similarity of proteins i and j, since if j were a hub protein adjacent to i, then we would find that i is very similar to j. To solve this problem, we'll instead compare the values of $He^{\{k\}}(i,v)$ with $He^{\{k\}}(j,v)$ for all $v \in V(G)$.

For a graph on n vertices, let $He^{\{k\}}(i)$ be the n-dimensional vector in which each entry j corresponds to the expected number of times a random walk starting from i and proceeding for k steps visits vertex j.

$$He^{\{k\}}(i) = (He^{\{k\}}(i,v_1), He^{\{k\}}(i,v_2), \ldots, He^{\{k\}}(i,v_n)).$$

We'll define DSD_k for k between vertices u and v by taking the L_1 norm of the vectors $He^{\{k\}}(u)$ and $He^{\{k\}}(v)$.

$$DSD_k(u,v) = \|He^{\{k\}}(u) - He^{\{k\}}(v)\|_1.$$

This tells us how different the pathways of random walks from u and v are by giving us the number of steps by which they differ. Cao et al. show that for any fixed k, DSD is a metric that converges as k goes to infinity. We can therefore obtain a final definition of DSD independent from k.

Definition 15. [9] The **Diffusion State Distance** between vertices u and v is given by

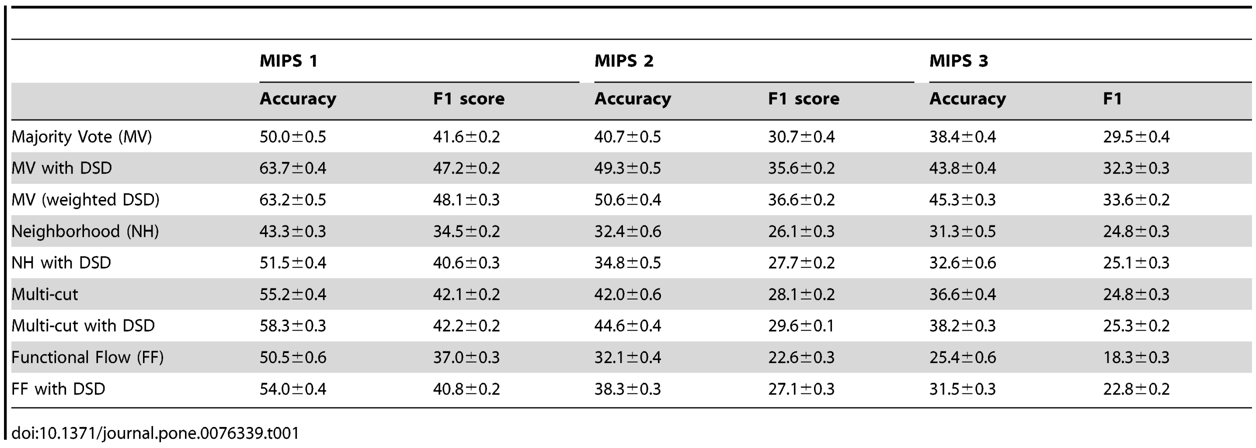
$$DSD(u,v) = \lim_{k\to\infty} \|He^{\{k\}}(u) - He^{\{k\}}(v)\|_1.$$

3.2 Performance

The *S. cerevisiae* PPI network was used test how well Diffusion State Distance performs against existing metrics. Data on protein interactions was taken from version 3.2.102 of the BioGRID network [30], which contains data on 128,643 interactions between proteins in the yeast PPI network. After selecting the largest connected component, the final PPI network consisted of 74,310 interactions between 4990 proteins [9]. Protein function was assigned based on the MIPS Functional Catalogue (FunCat) [28], and DSD was tested against other metrics at the first three levels of the MIPS hierarchy, each providing increasingly specific functional categories. To put DSD to use, Cao et al. created several functional annotation algorithms based on DSD, including the DSD Neighborhood Majority Voting algorithm. Based on the simple "guilt by association" algorithm described in Section 1.1, the t nearest neighbors $v_1, \ldots, v_t$ of a node u under the DSD metric "vote" on the function of u, either all with equal weight or with weight $1/DSD(u, v_i)$ for each neighbor v_i.

Remarkably, both unweighted and weighted Neighborhood Majority Vote with DSD outperformed every other network-based method of functional annotation when tested on the yeast PPI network (Table 1), while being significantly simpler to calculate. At the most general level of function, weighted Majority Vote with DSD obtained 63.2 $\pm$ 0.5% accuracy versus 55.2 $\pm$ 0.4 under the next-best algorithm; at the most specific level of function, weighted Majority Vote with DSD obtained 45.3 $\pm$ 0.3% accuracy versus 38.4 $\pm$ 0.4% under the next-best algorithm [9].

Table 1: Performance of standard functional annotation metrics vs. metrics with DSD in the yeast PPI network [9, Table 1].



	MIPS 1		**MIPS 2**		**MIPS 3**	
	Accuracy	**F1 score**	**Accuracy**	**F1 score**	**Accuracy**	**F1**
Majority Vote (MV)	50.0±0.5	41.6±0.2	40.7±0.5	30.7±0.4	38.4±0.4	29.5±0.4
MV with DSD	63.7±0.4	47.2±0.2	49.3±0.5	35.6±0.2	43.8±0.4	32.3±0.3
MV (weighted DSD)	63.2±0.5	48.1±0.3	50.6±0.4	36.6±0.2	45.3±0.3	33.6±0.2
Neighborhood (NH)	43.3±0.3	34.5±0.2	32.4±0.6	26.1±0.3	31.3±0.5	24.8±0.3
NH with DSD	51.5±0.4	40.6±0.3	34.8±0.5	27.7±0.2	32.6±0.6	25.1±0.3
Multi-cut	55.2±0.4	42.1±0.2	42.0±0.6	28.1±0.2	36.6±0.4	24.8±0.3
Multi-cut with DSD	58.3±0.3	42.2±0.2	44.6±0.4	29.6±0.1	38.2±0.3	25.3±0.2
Functional Flow (FF)	50.5±0.6	37.0±0.3	32.1±0.4	22.6±0.3	25.4±0.6	18.3±0.3
FF with DSD	54.0±0.4	40.8±0.2	38.3±0.3	27.1±0.3	31.5±0.3	22.8±0.2

doi:10.1371/journal.pone.0076339.t001

While DSD clearly outperforms other network-based methods of protein function prediction, it still falls somewhat short of the accuracy achieved by biology-based methods such as comparing protein structure. However, we often have little data about the specific features (such as folding patterns) that are needed to make predictions about proteins. Network-based methods like DSD are able to solve this problem by annotating the entire network at once rather than comparing individual protein sequences or structures, resulting in far simpler calculations with minimal loss of accuracy.

3.3 Refining Diffusion State Distance

The results in Section 3.2 above prove DSD to be a very promising metric for use in predicting protein function. Furthermore, we obtained these encouraging results using only a simple, undirected model of the yeast PPI network that gave equal weight to all edges in the BioGRID network. In reality, our understanding of the PPI network is much more nuanced: we're more confident about some interactions than about others, some interactions are naturally directed, and we know that certain interactions tend to send off a chain of interactions to carry out a full biological process. The natural next step, then, would be to fine-tune DSD to take these details into account.

In 2015, Cao et al. modified DSD to incorporate edge weights and explicit pathways, creating new metrics cDSD, caDSD, and capDSD [8]. First, with cDSD, they looked at the PPI data from BioGRID, and they modified the weight of each edge based on how many experiments endorsed that interaction and whether each experiment studied many interactions at once or focused on fewer proteins. Next, with caDSD, they located all interactions endorsed by the KEGG PATHWAY database [20], which are based on rigorous and reliable experimental data, and gave them an edge weight of 1, regardless of their support in BioGRID. Finally, to create capDSD, they looked again to KEGG for information on the fixed pathways that chains of protein interactions are known to follow, and they again modified edge weights to favor those known pathways. Not surprisingly, these changes resulted in increased accuracy at all three levels of the MIPS hierarchy when tested on the yeast PPI network. The best-performing metric was weighted Majority Vote with capDSD, which attained $68.09 \pm 0.49\%$ accuracy at the first level (Table 2) [8].

Table 2: Performance of standard functional annotation metrics vs. metrics with cDSD, caDSD, and capDSD in the yeast PPI network [8, Table 2].

	MIPS 1		MIPS 2		MIPS 3	
	Accuracy	F1 score	Accuracy	F1 score	Accuracy	F1
Majority Vote (MV)	50.08 ± 0.72	41.45 ± 0.40	40.69 ± 0.49	30.85 ± 0.33	38.03 ± 0.37	29.50 ± 0.14
MV with original DSD	62.96 ± 0.45	47.40 ± 0.28	49.41 ± 0.65	35.71 ± 0.33	43.87 ± 0.47	32.33 ± 0.18
MV with cDSD	66.16 ± 0.56	49.10 ± 0.24	53.08 ± 0.54	38.12 ± 0.16	47.73 ± 0.56	35.13 ± 0.33
MV with caDSD (directed edges)	67.61 ± 0.56	50.37 ± 0.22	59.11 ± 0.67	41.58 ± 0.19	52.14 ± 0.55	38.09 ± 0.16
MV with caDSD (no directed edges)	67.61 ± 0.42	50.36 ± 0.24	59.11 ± 0.57	41.57 ± 0.25	52.13 ± 0.56	38.07 ± 0.21
MV with capDSD	67.60 ± 0.37	50.28 ± 0.27	59.46 ± 0.57	41.58 ± 0.22	52.97 ± 0.59	38.19±0.23
Weighted MV (WMV) with original DSD	63.40 ± 0.51	48.29 ± 0.25	50.69 ± 0.82	36.74 ± 0.36	45.20 ± 0.58	33.72 ± 0.27
WMV with cDSD	67.07 ± 0.45	50.12 ± 0.35	54.82 ± 0.56	39.53 ± 0.18	49.56 ± 0.49	36.71 ± 0.32
WMV with caDSD (directed edges)	68.69 ± 0.40	51.48 ± 0.29	60.96 ± 0.51	43.13 ± 0.23	54.51 ± 0.51	39.91 ± 0.28
WMV with caDSD (no directed edges)	68.68 ± 0.41	51.48 ± 0.25	60.96 ± 0.53	43.13 ± 0.22	54.51 ± 0.46	39.90 ± 0.32
WMV with capDSD	**68.90 ± 0.49**	**51.61 ± 0.21**	**61.82 ± 0.59**	**43.54 ± 0.26**	**56.16 ± 0.59**	**40.42 ± 0.35**
Multi-way Cut (GMC)	55.31 ± 0.41	42.18 ± 0.29	42.02 ± 0.43	28.21 ± 0.36	36.69 ± 0.50	24.98 ± 0.21
GMC with original DSD	58.36 ± 0.32	42.51 ± 0.19	44.63 ± 0.32	29.51 ± 0.27	38.20 ± 0.40	25.49 ± 0.22
GMC with cDSD	61.11 ± 0.37	42.85 ± 0.23	47.11 ± 0.35	30.52 ± 0.25	40.83 ± 0.61	26.66 ± 0.22
GMC with caDSD (directed edges)	62.71 ± 0.30	43.46 ± 0.24	52.59 ± 0.25	32.47 ± 0.30	44.29 ± 0.63	28.46 ± 0.19
GMC with caDSD (no directed edges)	62.76 ± 0.31	43.45 ± 0.25	52.61 ± 0.25	32.50 ± 0.30	44.31 ± 0.63	28.46 ± 0.19
GMC with capDSD	62.44 ± 0.31	43.43 ± 0.17	52.30 ± 0.46	32.48 ± 0.31	44.18 ± 0.59	28.34 ± 0.32
Functional Flow (FF)	50.48 ± 0.48	37.17 ± 0.25	32.57 ± 0.48	22.64 ± 0.32	25.29 ± 0.39	18.27 ± 0.14
FF with original DSD	53.58 ± 0.36	40.75 ± 0.11	38.20 ± 0.65	26.71 ± 0.29	30.70 ± 0.45	22.29 ± 0.28
FF with cDSD	57.78 ± 0.49	42.82 ± 0.27	42.17 ± 0.58	29.29 ± 0.38	35.68 ± 0.48	25.72 ± 0.17
FF with caDSD (directed edges)	60.09 ± 0.55	44.81 ± 0.24	49.73 ± 0.41	33.89 ± 0.32	40.82 ± 0.60	28.94 ± 0.27
FF with caDSD (no directed edges)	60.18 ± 0.47	44.80 ± 0.20	49.67 ± 0.51	33.89 ± 0.28	40.82 ± 0.51	28.97 ± 0.23
FF with capDSD	58.98 ± 0.53	43.80 ± 0.27	49.32 ± 0.61	33.32 ± 0.29	41.04 ± 0.33	28.83 ± 0.33

4 Exit Frequency Distance

With Diffusion State Distance, we've found a way to predict protein function that yields significantly more accurate results than existing network-based methods. However, we would like to be as accurate as possible if we hope to use our functional annotations in a laboratory or clinical setting, and we're simply not where we'd like to be. The results clearly improved when we added edge weights and fixed pathways in the modified versions of DSD, but can we do better? Consider this: what if instead of letting our random walks go to infinity (as in DSD), we made use of a stopping rule to halt the random walks at some distribution that *we* get to specify, say a certain vertex or set of vertices? Exit Frequency Distance (EFD) was conceived as another approach to fine-tuning DSD, and it gives us a mathematical way to incorporate our knowledge of important proteins or subnetworks within a PPI network, so we can focus on more meaningful random walks to pre-specified target distributions. Since EFD is focused on random walks to a target distribution, we can build on the extensive research already done on stopping rules and exit frequencies (Section 2.4) to study EFD's behavior.

Introduced by Beveridge et al. in 2016 [5], EFD is a metric that measures the similarity of two proteins u and v based on the similarity of the pathways of random walks from u and v to some pre-specified target distribution τ. Essentially, EFD is an extension of DSD that halts the random walk at a distribution τ instead of taking the limit as the number of steps goes to infinity. Since EFD uses random walks to target distributions, we can now use exit frequencies rather than $He^{\{k\}}$ to keep track of the pathway of the walk. The work of Beveridge et al. on Exit Frequency Distance is summarized below.

4.1 Formal definition

Consider a stopping rule $\Gamma(v_i, \tau)$ for a random walk from vertex v_i to target distribution τ, and let $x_k(v_i, \tau)$ be the optimal exit frequency for vertex v_k in that walk (optimal exit frequencies can be calculated using equation 7 in Section 2.4). Let X_τ be a matrix of exit frequences of Γ whose entries are given by

$$X_\tau(i, j) = x_j(v_i, \tau).$$

Definition 16. [5] Given a matrix of exit frequences X_τ for a stopping rule $\Gamma(v_i, \tau)$, the **Exit Frequency Distance** between vertices u and v is given by

$$EFD_\tau(u, v) = \|(\mathbf{1}_u - \mathbf{1}_v)X_\tau\|$$

where $\mathbf{1}_i X_\tau$ gives the row of exit frequencies for a walk from vertex i to distribution τ.

4.2 Connection to Diffusion State Distance

The discrete Green's function provides the mathematical link between DSD and EFD. Introduced by George Green in 1828, Green's functions are widely used for solving differential equations but appear in many areas of mathematics. The discrete Green's function $\mathbb{G}$ is useful for studying discrete Markov chains, such as random walks on graphs, and is defined using the discrete Laplace operator $\triangle$ [11].

Definition 17. [4] For a graph G on n vertices, let D be the $n \times n$ diagonal matrix of vertex degrees with $D_{ii} = d(v_i)$, and let A be the adjacency matrix of G. The **discrete Laplace operator** is the $n \times n$ matrix $\triangle$ given by

$$\triangle = I - D^{-1}A.$$

The Laplace operator is related directly to random walks on graphs: $D^{-1}A$ is the transition matrix for the graph G, since entry $M_{ij} = 1/d(i)$ if $(i,j) \in E(G)$ and 0 otherwise. Therefore, for a graph with transition matrix M, we can write the discrete Laplace operator as

$$\triangle = I - M.$$

Definition 18. [4] The **discrete Green's function** is the unique matrix satisfying

$$\mathbb{G}\triangle = I - \mathbf{1}\pi^T,$$
$$\mathbb{G}\mathbf{1} = \mathbf{0}.$$

Fortunately, we have a much simpler formula for Green's function in terms of hitting times. If

$$H(\pi, j) = \sum_{i \in V} \pi_i H(i,j)$$

is the expected time for a walk starting from a random vertex to reach vertex j, then

$$\mathbb{G}(i,j) = \pi_j \big(H(\pi, j) - H(i,j)\big).$$

Boehnlein et al. [6] have shown that

$$DSD(u,v) = \|(\mathbf{1}_u - \mathbf{1}_v)\mathbb{G}\|.$$

In Exit Frequency Distance, then, we have simply replaced Green's function with the matrix of exit frequences X_τ:

$$EFD_\tau(u,v) = \|(\mathbf{1}_u - \mathbf{1}_v)X_\tau\|.$$

This connection between DSD and EFD serves as another motivating factor for studying EFD. Since the two metrics are mathematically very similar, we have reason to believe that EFD will be as useful as DSD is for measuring protein similarity. With EFD, we simply go one step further and specify a target distribution, which we hope will give us even better results than DSD.

4.3 Performance

For the first test of EFD's performance in 2016, the stationary distribution π was chosen as the target distribution. As in the DSD experiments, to assign protein function, EFD_π was substituted for traditional shortest-path distance in the Majority Vote algorithm: every other node in the network was ranked in terms of EFD_π from a given protein, then the closest 10 nodes "voted" on its functional label. This algorithm was compared against unweighted and weighted Majority Vote (MV) using the usual shortest-path distance, as well as against Majority vote using cDSD. When tested on the *S. cerevisiae* PPI network, EFD_π was found to perform with similar accuracy to cDSD and to outperform both unweighted and weighted Majority Vote in predicting protein function (Table 3) [5].

Table 3: Performance of standard functional annotation metrics vs. metrics with EFD_π in the yeast PPI network [5, Table 1].

Method	1st Level		2nd Level		3rd Level	
	Accur.	F1	Accur.	F1	Accur.	F1
MV	0.5065	0.4211	0.3977	0.3064	0.3677	0.2919
wMV	0.5445	0.4451	0.4475	0.3411	0.4208	0.3308
EFD_π	0.6664	0.4943	0.5420	0.3901	0.4912	0.3634
cDSD	0.6689	0.4917	0.5528	0.3999	0.5015	0.3716

These results support our conjecture that DSD and EFD behave similarly on PPI networks, which is not surprising given their mathematical similarities discussed above. However, they only tell us what happens when we use the stationary distribution π as our target distribution. Since any random walk (on a non-bipartite graph) will already tend to π, it doesn't take into account any other knowledge we might have about a graph or its special properties – so what about other target distributions? A natural second choice for our target distribution is the forget distribution, another distribution whose properties relating to exit frequencies are well-studied.

Recall from Definition 13 that the forget distribution μ is reached when the distribution no longer reflects where the random walk began. Since π is the limiting distribution for a random walk, μ is always reached *before* π (recall from Section 2.5 that $T_{\text{forget}} \leq T_{\text{mix}}$). While using π as our target distribution tells us how a random walk behaves over a long period of time, then, using μ as our target distribution allows us to see what happens earlier on in the walk. In the context of PPI networks, it makes sense to look at the early behavior of a random walk, since random walks from different starting proteins will be most different at the beginning, while over time, they will start to look more and more similar as they approach π.

In general, the forget distribution is distinct from the stationary distribution, so we would expect EFD_μ and EFD_π to yield different results, as the exit frequencies for a random walk to μ should be distributed differently from those of a random walk to π. However, in preliminary experiments measuring EFD_μ on PPI networks, EFD_μ appeared to actually equal EFD_π (the exit frequencies for random walks to μ were the same as the exit frequencies for random walks to π). Further investigation proved that $\text{EFD}_\mu = \text{EFD}_\pi$ on any tree. These unusual experimental results suggest that PPI networks might behave similarly to trees, a property that might help us understand how PPI networks behave under network-based metrics like EFD [7].

5 Random walks on trees

5.1 Hitting time on trees

Based on the fact that $\text{EFD}_\mu = \text{EFD}_\pi$ on trees, we'd like to further investigate the special properties of trees that might lead to this unique behavior. It turns out that there is a mathematical reason for this, related to the access times $H(i, \pi)$ and $H(i, \mu)$ on trees, which will be discussed in more detail below. Since it appears that EFD_μ also equals EFD_π on the yeast PPI network, we hope to use the insight we gain on trees to better inform our decision of which target distribution to use when calculating Exit Frequency Distance.

Fortunately, trees are much more straightforward to study than other, more complicated types of graphs. Because trees are acyclic, there exists a unique shortest path between any two vertices, a fact that makes hitting time significantly easier to calculate. Beveridge demonstrates that hitting time in a tree can be computed using a simple formula dependent only on distance [3]. For vertices i, j, and k in a tree, let

$$\ell(i,k;j) = \frac{1}{2}(d(i,j) + d(k,j) - d(i,k))$$

be the length of the intersection of the (i,j)-path and the (k,j)-path. Then the hitting time from i to j is

$$H(i,j) = \sum_{k \in V} \ell(i,k;j)d(k), \tag{10}$$

where $d(k)$ is the degree of vertex k [3].

Let's walk through an example to see equation 10 in action. Consider a simple tree on three "legs" exending from a single "root" vertex v_0 (Figure 7). We'll call these legs 1, 2, and 3, and we'll label their leaves ℓ_1, ℓ_2, and ℓ_3, respectively. (Section 6 will introduce this labeling system more formally, and Section 7 will consider properties of this particular type of tree in more depth.) In this example, let's compute the hitting time $H(i,j)$ between the vertices labeled i and j in Figure 7.

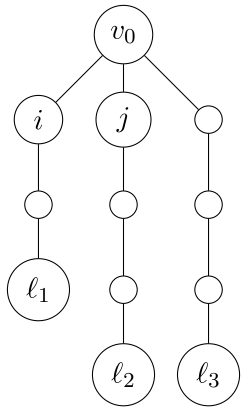


Figure 7: An a,b,b spider.

To calculate $H(i,j)$, we can first split the vertices into three groups based on degree: the three leaves have degree 1, the root has degree 3, and every other vertex has degree 2 (for simplicity of notation in the following calculations, we'll refer to any arbitrary vertex with degree 2 as k_2). So our sum becomes

$$H(i,j) = \ell(i,\ell_1;j) + \ell(i,\ell_2;j) + \ell(i,\ell_3;j) + 3\ell(i,v_0;j) + 2\sum_{k_2 \in V} \ell(i,k_2;j). \tag{11}$$

Now, let's find the length of the overlap of each $k - j$ path with the $i - j$ path.

degree-1 vertices:	$\ell(i,\ell_1;j) = 2$
	$\ell(i,\ell_2;j) = 0$
	$\ell(i,\ell_3;j) = 1$
degree-3 vertex:	$\ell(i,v_0;j) = 1$
degree-2 vertices:	For k_2 on leg 1, $\ell(i,k_2;j) = 2$
	For k on leg 2, $\ell(i,k;j) = 0$
	For k on leg 3, $\ell(i,k;j) = 1$

There are two degree-2 vertices on leg 1 and three each on legs 2 and 3. We now have all the elements we need to calculate the hitting time from i to j, so we can simply plug them into equation 11 to obtain

$$H(i,j) = 3(1) + 2 + 0 + 1 + 2\big(2(2) + 3(1)\big) = 20.$$

5.2 Centers of trees

To better understand how a tree's stucture affects the behavior of a random walk, let's now consider what it means for a vertex to be at the "center" of a tree in terms of time rather than traditional distance. The "average" center of a tree is simply the vertex that minimizes the average hitting time from all other vertices.

Definition 19. [3] The **average center** of a tree is the vertex j satisfying

$$\min_{j \in V} \sum_{i \in V} \pi_i H(i,j).$$

Sometimes we care more about how long it takes for a random walk to reach the farthest corners of a graph rather than how long it takes to get to any vertex on average. The "extremal" center, or focus, of a tree is the vertex that minimizes its maximum hitting time – that is, the hitting time from its farthest-away, or pessimal, vertex is minimized.

Definition 20. [3] For a vertex $j \in V$, a vertex j' is j-**pessimal** if it satisfies

$$H(j', j) = \max_{i \in V} H(i, j).$$

Definition 21. [3] The **focus** of a tree is the vertex a satisfying

$$H(a', a) = min_{j \in V} H(j', j) = \min_{j \in V} \max_{i \in V} H(i, j).$$

Every tree has either one focus or two adjacent foci [3]. It's valuable to know the location of the focus because it can reveal nuances in the behavior of random walks on graphs that might not be immediately apparent. For example, Figure 8 shows two trees that differ by only a single vertex, yet their foci have different locations.

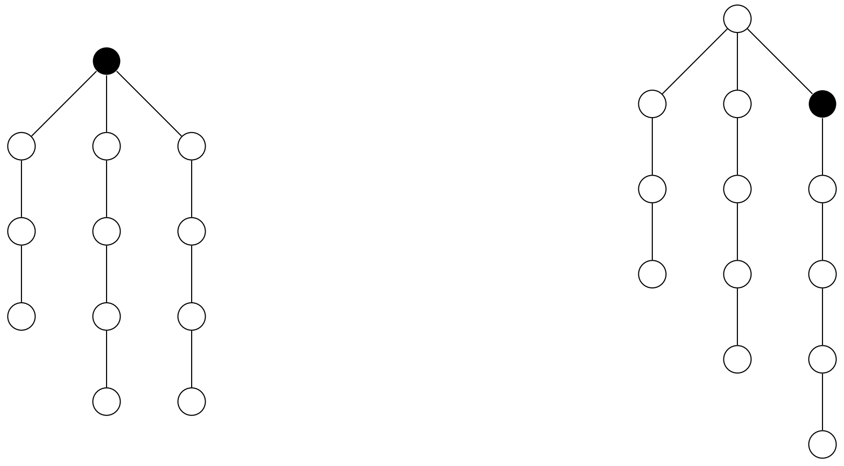


Figure 8: Two trees and their foci, shaded in black.

Now that we're familiar with the idea of the focus, we can return once again to the forget distribution μ, which turns out to have some properties that are particularly interesting on trees. It's been proven that for any tree, μ is concentrated on its focus (or foci) [3]. This makes sense: from Definition 12, we know that μ minimizes the expected time of a random walk starting from the worst-possible starting state, and similarly, the focus is defined as having the minimum time from its furthest-away vertex. Not only is μ concentrated on the focus, but furthermore, for all $i \in V$,

$$H(i,\pi) = H(i,\mu) + H(\mu,\pi). \tag{12}$$

In terms of stopping rules, this means that following an optimal stopping rule from a vertex i to μ, and then another optimal stopping rule from μ to π, will produce an optimal stopping rule from i to π. This result is vital: it tells us that on any tree, a random walk will *always* hit the forget distribution "on the way" to the stationary distribution, which is *not* true for graphs in general. We now have more insight into why $\mathrm{EFD}_\mu = \mathrm{EFD}_\pi$ on trees: no matter which vertex a random walk starts from, it will always have the same exit frequencies between μ and π, so when calculating EFD_π between two vertices, any exit frequencies gained after the random walks reach μ will cancel.

6 Simplified tree vertex labeling

When calculating hitting time on a tree, it will be helpful to have a better way to measure distance between vertices that doesn't rely on repeatedly counting by hand the length of the shortest path connecting them. Here, we'll introduce a new system of vertex notation that simplifies this calculation by assigning "coordinates" to the vertices of a tree, allowing us to measure distance between vertices by simply taking the L_1 norm of the difference of their coordinates.

We can think of a tree as consisting of a set of "legs" extending from some "root" vertex. The basic idea is that a vertex's coordinate will tell us how many steps down each leg we must walk in order to find that vertex. To assign coordinates, we'll first number the vertices (arbitrarily), which will act as a tiebreaker to help us decide which vertices correspond to which leg. We'll call the highest-degree vertex (with the lowest number, if there are multiple) the "root," to which we'll assign a coordinate of zeros $(0, 0, \ldots, 0)$. Next, we'll give each vertex a coordinate by tracing its path from the root, counting the number of steps we take down each leg and adding to a new entry of the coordinate every time we branch off onto a new leg.

When we hit a point where the tree branches into multiple new legs, how will we decide which one is a continuation of the current leg and which ones are "new" legs? This is where the vertex numbers come in: all vertices in the path from the root to the lowest-numbered leaf will be on leg 1; all vertices in the path from the root to the second lowest-numbered leaf, *not* including those already on leg 1, will be on leg 2; and so on (so all vertices in the path from the root to leaf k, not including those already on a lower-numbered leg, will be on leg k). Once we've assigned a coordinate to a vertex, we can now find exactly its place in the tree and its distance from the root, which is simply the sum of its entries. For a tree with m leaves, each coordinate will be a vector in m dimensions, so we can now think of the tree as existing in $\mathbb{R}^m$, with each new leg branching off into a new dimension. This will be a helpful way to think about trees with three legs (which we'll examine in more detail in Section 7); their legs will simply run along the x, y, and z axes in $\mathbb{R}^3$.

6.1 Important terms and notation

Now that we're familiar with the general idea behind vertex coordinates, we'll work through the process of assigning vertex coordinates more formally. Before we start, let's define some terms and notation we'll be using. Consider a tree $G = (V, E)$ on n numbered vertices $\{v_1, v_2, \dots, v_n\}$, with v_i corresponding to the vertex labeled with number i. Suppose our tree has m leaves; for ease of notation, denote the set of leaves $\{\ell_1, \dots, \ell_m\}$, so that if ℓ_i and ℓ_j correspond to vertices $v_{i'}$ and $v_{j'}$, respectively, then $i' < j' \Leftrightarrow i < j$. Each vertex will eventually be assigned a coordinate $c(v_i) = (x_1, \dots, x_m)$ in m dimensions, with $c(v_i)_k$ corresponding to the kth entry in the coordinate.

Definition 22. For a tree on n labeled vertices $\{x_1, \dots, x_n\}$, the **root** is the highest-degree vertex with the lowest index. Denote the root v_0.

$$v_0 = \min_i \{v_i \in V \mid d(v_i) = \max_{v \in V} d(v)\}.$$

To assign vertex coordinates, we'll need to have a specific way of defining what a "leg" of a tree is. Ultimately, every leg will end in a leaf, so a tree with m leaves will have m legs. Since our graph is a tree, there is a unique path between any two vertices; therefore, we can partition the vertices into legs by looking at the paths between the root and each leaf.

Definition 23. For a tree with root v_0 and vertices $\{v_i\}$, P_{v_k} denotes the unique path of vertices from v_0 to vertex v_k.

The path between the root and each leaf ℓ_i will thus be denoted P_{ℓ_i}. Notice that these paths may intersect; in Figure 11 below, for example, $P_{\ell_3} \cap P_{\ell_4} = \{v_1, v_5, v_6\}$. We'll now define the legs of a tree by removing these overlaps, so leg 1 will simply contain all vertices in P_{ℓ_1}, leg 2 will contain all vertices in $P_{\ell_2} \setminus P_{\ell_1}$, leg 3 will contain all vertices in $P_{\ell_3} \setminus \big(P_{\ell_2} \cup P_{\ell_1}\big)$, and so on.

Definition 24. For a tree with paths P_{ℓ_i} between the root and its leaves, **leg** k is the set of all vertices in P_{ℓ_k}, minus the vertices in P_{ℓ_i} for all $i < k$.

$$\text{leg } k = \big\{ v \in V \mid v \in P_k \setminus \big(\bigcup_{i=1}^{k-1} P_i \big) \big\}.$$

6.2 Assigning vertex coordinates

For each vertex $v_i \in V$, we'll assign a unique coordinate $c(v_i) = (x_1, \ldots, x_m)$. The process is simple:

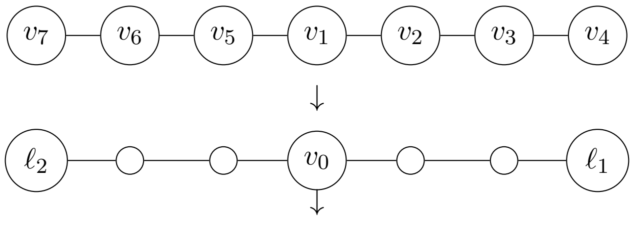
1. Locate the path P_{v_i} from v_0 to v_i.
2. Set each entry $c(v_i)_k$ to the length of the overlap of P_{v_i} with leg k.

$$c(v_i)_k = |P_{v_i} \cap (\text{leg } k)|.$$

We can now easily locate any vertex v_i in the tree by going $c(v_i)_1$ steps down leg 1, $c(v_i)_2$ steps down leg 2, $c(v_i)_3$ steps down leg 3, and so on, tracing a continuous path from the v_0 to v_i. Notice that if P_{v_i} does not intersect leg k, then $c(v_i)_k = 0$.

Let's walk through an example of assigning vertex coordinates. A tree with two leaves will have coordinates in $\mathbb{R}^2$ (Figure 9). We'll first number each vertex arbitrarily, then choose the vertex of degree two with the lowest number (which will be somewhere in the middle of the path) to be the root $v_0 = (0, 0)$. From there, all vertices on one side of the root (leading to the lower-numbered leaf) will have coordinate $(x_1, 0)$, with x_1 corresponding to their distance from the root, and all vertices on the other side of the root (leading to the higher-numbered leaf) will have coordinate

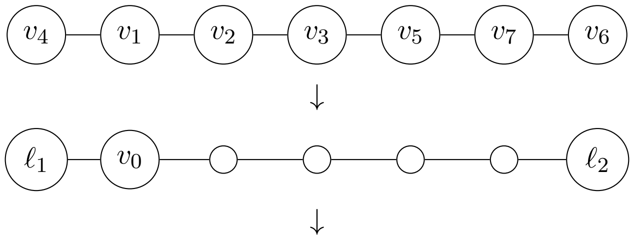
$(0, x_2)$, with x_2 similarly corresponding to their distance from the root. Since each vertex will be assigned a coordinate in two dimensions, we can also think of our tree as the number line in $\mathbb{R}^2$, with one leg corresponding to the x-axis and the other corresponding to the y-axis.



$(0,3)$ - $(0,2)$ - $(0,1)$ - $(0,0)$ - $(1,0)$ - $(2,0)$ - $(3,0)$

Figure 9: A tree and its corresponding coordinates in $\mathbb{R}^2$.

Note that the resulting set of vertex coordinates is not unique, as they depend entirely on the original numbering of the vertices. If we numbered the vertices in a different way, as in Figure 10, then we would end up with different vertex coordinates. Importantly, for any tree whose vertices have *already* been numbered (though arbitrarily), its vertex coordinates are still unique, and the distance between vertices can still be measured using L_1 distance (Theorem 1).



$(1,0)$ - $(0,0)$ - $(0,1)$ - $(0,2)$ - $(0,3)$ - $(0,5)$ - $(0,4)$

Figure 10: A tree and its corresponding coordinates in $\mathbb{R}^2$.

As another example, the tree in Figure 11 will have coordinates in $\mathbb{R}^4$. As above, we'll first number the vertices of the tree, then we'll choose the lowest-numbered vertex of degree three to be the root $(0, 0, 0, 0)$. From there, we'll assign coordinates to the vertices by counting their distance down each leg, starting with those leading to the lowest-numbered leaf and working up from there.

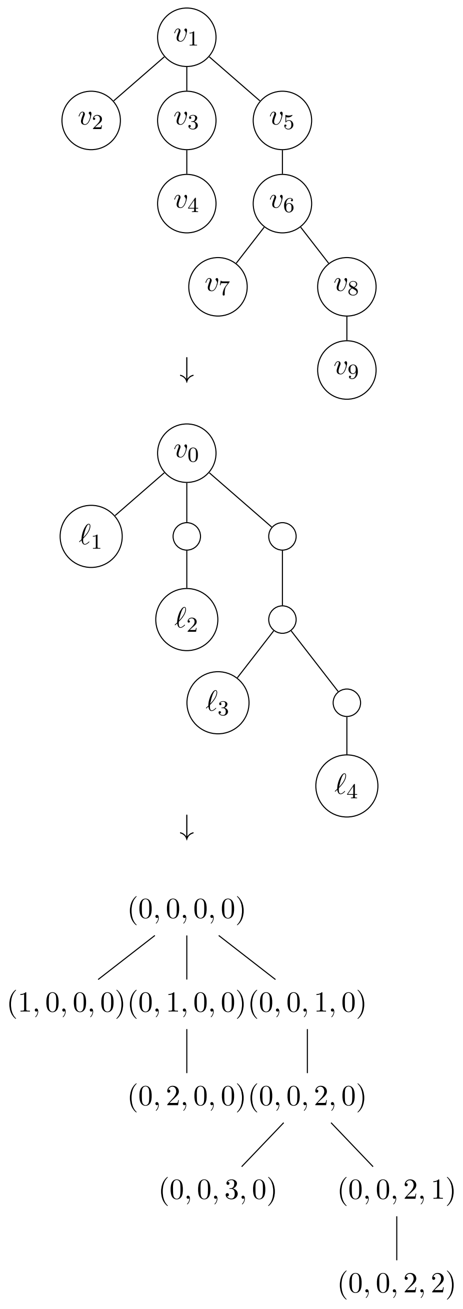


Figure 11: A tree and its corresponding coordinates in $\mathbb{R}^4$.

Theorem 1. *Consider a tree on* n *numbered vertices* $\{v_1, \ldots, v_n\}$ *with* m *leaves* $\{\ell_1, \ldots, \ell_m\}$*. The above method assigns a coodinate* $c(v_i)$ *in* m *dimensions to each vertex* v_i *so that:*

1. *Each vertex has one unique coordinate, and*

2. *We can measure the distance between two vertices by taking the difference in the* L_1 *norm of*

the difference of their coordinates:

$$d(v_i, v_j) = \sum_{k=1}^{m} |c(v_i)_k - c(v_j)_k|.$$

Proof.

1. By our algorithm, we assign a coordinate $c(v_i)$ to a vertex v_i so that each entry $c(v_i)_k$ corresponds to v_i's distance down each leg k of the tree (this is equivalent to $|P_{v_i} \cap \text{ leg } k|$). Suppose two vertices v_i, v_j have the same coordinate: then every entry in $c(v_i)$ is the same as every entry in $c(v_j)$, so $c(v_i)_k = c(v_j)_k \ \forall k \in \{1, \ldots, m\}$. It follows that the path from v_0 to v_i is the same as the path from v_0 to v_j, because we can locate both v_i and v_j by going $c(v_i)_1 = c(v_j)_1$ steps down leg 1, $c(v_i)_2 = c(v_j)_2$ steps down leg 2, and so on. Since we're working on a tree, there is a unique shortest path between any two vertices, so $v_i = v_j$.

   Thus, every coordinate is unique.

2. Why does the L_1 norm of the difference of two coordinates give us the distance between their corresponding vertices? Let's think about the way we assign coordinates. As above, for vertex v_i, each entry $c(v_i)_k$ is the distance down leg k, so we can find the location of v_i by going $c(v_i)_k$ steps down each leg $k \in \{1, \ldots, m\}$. For two different vertices v_i, v_j, $|c(v_i)_k - c(v_j)_k|$ gives us the number of steps on leg k by which P_{v_i} and P_{v_j} differ. By taking $\sum_{k=1}^{m} |c(v_i)_k - c(v_j)_k|$, then, we get the total number of vertices by which P_{v_i} and P_{v_j} differ.

   This number is the same as $d(v_i, v_j)$. Why? The set of vertices by which P_{v_i} and P_{v_j} differ is in fact the path from v_i to v_j. If P_{v_i} and P_{v_j} meet at the root v_0, then this path is simply $P_{v_i} \sqcup P_{v_j}$, and

$$|P_{v_i} \sqcup P_{v_j}| = |P_{v_i}| + |P_{v_j}| = d(v_i, v_0) + d(v_0, v_j) = d(v_i, v_j).$$

   If they meet at some other vertex (call this vertex v_0' for now) then this path is $\left(P_{v_i} \cup P_{v_j}\right) \setminus P_{v_0'}$, and

$$|(P_{v_i} \cup P_{v_j}) \backslash P_{v_0'}| = |(P_{v_i} \backslash P_{v_0'}) \cup (P_{v_j} \backslash P_{v_0'})| = |P_{v_i} \backslash P_{v_0'}| + |P_{v_j} \backslash P_{v_0'}| = d(v_i, v_0') + d(v_0', v_j) = d(v_i, v_j).$$

Therefore, in both cases, the number of vertices by which P_{v_i} and P_{v_j} differ is $d(v_i, v_j)$, which is the length of the path from v_i to v_j, and we conclude that $d(v_i, v_j) = \sum_{k=1}^{m} |c(v_i)_k - c(v_j)_k|$.

$\square$

7 Exploring simple trees

When studying target distributions for EFD on trees, it has been helpful to begin by studying the properties of random walks on simple trees, so we can eventually apply that knowledge to more complicated trees. The simplest type of tree is a path (as in Figure 12), and random walks on paths are very straightforward: they can only move up or down the path. The next step, then, is to study trees on three "legs," essentially paths with an extra path attached somewhere in the middle. For simplicity, we'll call these types of trees **spiders** (Figure 13 below gives an example). These trees turn out to be significantly more complicated than paths. For example, consider the forget distribution, which is a target distribution we're particularly interested in on trees. From Section 5, we know that μ is concentrated on a tree's focus, which is the vertex whose hitting times to its pessimal vertices are minimized (Definition 21). On a path, the focus is simply the middle vertex (or, if $|V|$ is even, then the two middle vertices will be the foci) (Figure 12).

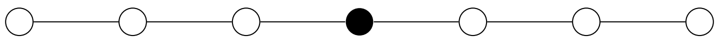


Figure 12: A path and its focus, shaded in black.

Once we look at trees on three legs, however, the placement of the focus is much less intuitive; Figure 8, for example, shows that even a slight change in the lengths of a tree's legs can cause the location of the focus to shift. In this section, we'll show that if the two longer legs of a spider are of equal length, then the focus is guaranteed to be at the root (Theorem 2). From now on, we'll call these special types of spiders a, b, b spiders, since their legs are of length a, b, and b. We'll denote the root of this tree v_0 and its leaves ℓ_1, ℓ_2, and ℓ_3.

Definition 25. An a, b, b **spider** is a tree made up of three paths of length a, b, and b extending from a single degree-3 vertex, with $2 < a < b$.

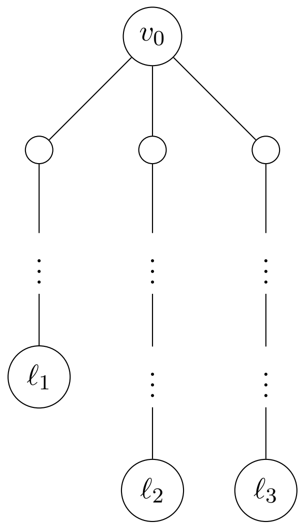


Figure 13: An a, b, b spider.

Since we want to prove that the root v_0 is the focus, we'll be calculating the maximum hitting time from every vertex in the tree and showing that v_0 minimizes this value, thus achieving $\min_{j \in V} \max_{i \in V} H(i, j)$. Hitting time is calculated using distance (equation 10), so this will be a great time to use our new vertex labeling scheme from Section 6. By assigning a coordinate (x, y, z) to each vertex, we can easily measure distance between two vertices by taking the L_1 distance of their coordinates. We can now also think of an a, b, b spider as running along the three axes in $\mathbb{R}^3$ (Figure 14).

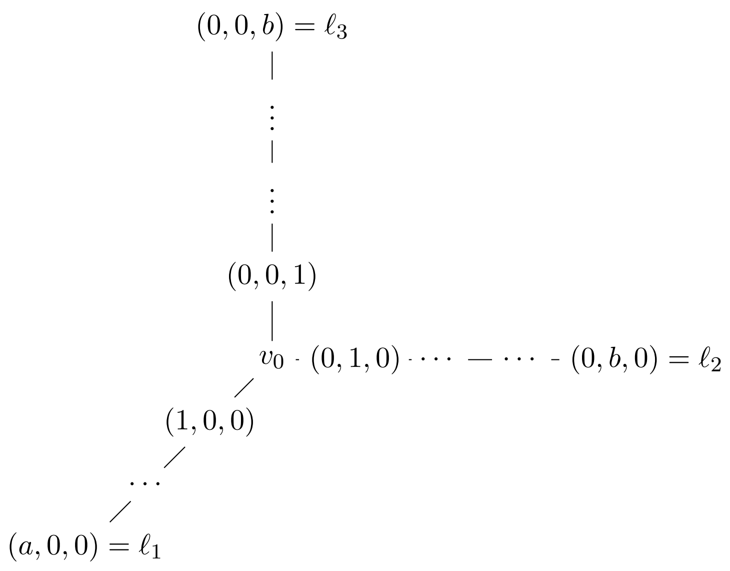


Figure 14: An a, b, b spider in $\mathbb{R}^3$.

We'll now refer to each leg as leg 1 (with length a), 2 (with length b), and 3 (with length b), where $\ell_1 = (a, 0, 0)$, $\ell_2 = (0, b, 0)$, and $\ell_3 = (0, 0, b)$; the root is $v_0 = (0, 0, 0)$. For our a, b, b spider, we can now easily calculate the distance between two vertices $i = (x_1, y_1, z_1)$ and $j = (x_2, y_2, z_2)$ by taking their L_1 norm:

$$d(i, j) = |x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2|.$$

From here on, we'll continue to use coordinate notation (x, y, z) to refer to vertices in our graph, with the exception of the root v_0 and the leaves ℓ_1, ℓ_2, and ℓ_3.

7.1 Proof: The focus of an a, b, b spider is the root

Theorem 2. *For an a, b, b spider as defined above, the focus is the root v_0.*

$$H(v_0', v_0) = \min_{j \in V} \max_{i \in V} H(i, j).$$

Proof. We want to show that $H(v_0', v_0)$, which achieves $\max_{i \in V} H(i, v_0)$, is smaller than $H(v_i', v_i)$ for every other vertex v_i. To do so, we'll need to be able to calculate the hitting time between any two vertices in our tree. In Section 5, we already worked through an example of calculating hitting time on a general a, b, b spider, so we can follow the same process as before to calculate $H(i,j)$. Recall that when a tree has three legs, the formula for hitting time is

$$H(i,j) = \ell(i,\ell_1;j) + \ell(i,\ell_2;j) + \ell(i,\ell_3;j) + 3\ell(i,v_0;j) + 2\sum_{k_2 \in V} \ell(i,k_2;j),$$

where $\ell(i,k;j)$ is the length of the overlap of the $i-j$ and $k-j$ paths.

First, we'll calculate $max_{i\in V} H(i,v_0)$, and then we'll show that $max_{i\in V} H(i,v_0) \leq \max_{i\in V} H(i,j), \forall j \in V$, thus proving that v_0 achieves $\min_{j\in V} \max_{i\in V} H(i,j)$.

7.1.1 Finding $max_{i\in V} H(i,v_0)$

We want to find the hitting time from v_0's furthest-away (pessimal) vertex or vertices to v_0 by finding the maximum hitting time $H(i,v_0)$ over all $i \in V$. Let's consider three cases: i may be on leg 1, 2, or 3. To calculate $H(i,v_0)$ for each case of i, we'll have to find $\ell(i,k;v_0)$ for every other vertex k on the graph, which includes:

- the degree-1 vertices (ℓ_1, ℓ_2, ℓ_3),
- the degree-3 vertex (v_0), and
- the degree-2 vertices (all other vertices in the graph, denoted generally as k_2).

Note that $\ell(i,k_2;v_0)$ will be different based on the location of each degree-2 vertex. If k_2 is on the same leg as i but closer to the leaf, then $\ell(i,k_2;v_0) = d(i,v_0)$; if k_2 is on the same leg as i but closer to v_0, then $\ell(i,k;v_0) = d(k,v_0)$; and if k_2 is on a different leg from i, then $\ell(i,k_2;v_0) = 0$.

Once we've found $\ell(i,k;v_0)$ for each $i,k \in V$, the last piece of information we need is the number of degree-2 vertices in the graph so we can properly sum them. Since we're working on an a, b, b spider, we know that the total number of degree-2 vertices is $a + 2b - 2$. We'll need to break this total down further to account for the different possible placements of the degree-2 vertices described above.

For each case of i that follows, we'll calculate each of these components $\ell(i,k;v_0)$, then put them together to find $H(i,v_0)$.

1. i **on leg 1**

   For any i on leg 1 with $d(i,v_0)=x$, we can refer to i using its (general) coordinate $(x,0,0)$. We'll now find $\ell(i,k;v_0)$ for all vertices k in our graph by considering carefully the length of the overlap of the $i-j$ and $k-j$ path for every k.

degree-1 vertices:	$\ell(i,\ell_1;v_0)=d(i,v_0)=x$
	$\ell(i,\ell_2;v_0)=\ell(i,\ell_3;v_0)=0$
degree-3 vertex:	$\ell(i,v_0;v_0)=0$
degree-2 vertices:	For k on leg 1 between i and ℓ_1: $\ell(i,k;v_0)=d(i,v_0)=x$
	For k on leg 1 between i and v_0, including i: $\ell(i,k;v_0)=d(k,v_0)$
	For k on leg 2 or 3: $\ell(i,k;v_0)=0$

   Now, our challenge is to count the number of degree-2 vertices on leg 1 either side of i and v_0 so we can properly sum the lengths of their overlaps to find $\sum_{k_2\in V}\ell(i,k_2;v_0)$.

   - For k between i and ℓ_1, not including i, our sum is

   $$\begin{aligned}\sum_{k_2\in V}\ell(i,k_2;j)&=x\cdot(\text{number of degree-2 vertices between } i \text{ and } \ell_1)\\&=x(a-d(i,v_0)-1)\\&=x(a-x-1).\end{aligned}$$

   - For k between i and v_0, including i, our sum is

   $$\begin{aligned}\sum_{k_2\in V}\ell(i,k_2;v_0)&=d((1,0,0),v_0)+d((2,0,0),v_0)+\cdots+d(i,v_0)\\&=1+\cdots+x\\&=\frac{x(x+1)}{2}.\end{aligned}$$

Putting this together, we find that for $i = (x, 0, 0)$ on leg 1, hitting time can be calculated as

$$\begin{aligned} H(i, v_0) &= x + 2\Big(x(a - x - 1) + \frac{x(x+1)}{2}\Big) \\ &= x + 2x(a - x - 1) + x(x + 1) \\ &= x(2a - x). \end{aligned}$$

2. i **on leg 2**

For any i on leg 2 with $d(i, v_0) = y$, we can refer to i as $(0, y, 0)$. Again, we'll find $\ell(i, k; v_0)$ for all vertices k on the graph, considering the same cases as above.

degree-1 vertices:	$\ell(i, \ell_2; v_0) = d(i, v_0) = y$
	$\ell(i, \ell_1; v_0) = \ell(i, \ell_3; v_0) = 0$
degree-3 vertex:	$\ell(i, v_0; v_0) = 0$
degree-2 vertices:	For k on leg 2 between i and ℓ_2: $\ell(i, k; v_0) = d(i, v_0) = y$
	For k on leg 2 between i and v_0, including i: $\ell(i, k; v_0) = d(k, v_0)$
	For k on leg 1 or 3: $\ell(i, k; v_0) = 0$

As we did with i on leg 1, we'll now find $\sum_{k_2 \in V} \ell(i, k_2; j)$ for all degree-2 vertices in the graph. For k between i and ℓ_2, not including i, our sum is

$$\begin{aligned} \sum_{k_2 \in V} \ell(i, k_2; v_0) &= y \cdot (\text{number of degree-2 vertices between } i \text{ and } \ell_2) \\ &= y(b - d(i, v_0) - 1) \\ &= y(b - y - 1). \end{aligned}$$

For k between i and v_0, including i, our sum is

$$\begin{aligned} \sum_{k_2 \in V} \ell(i, k_2; v_0) &= d\big((0, 1, 0), v_0\big) + d\big((0, 2, 0), v_0\big) + \cdots + d(i, v_0) \\ &= 1 + \cdots + y \\ &= \frac{y(y+1)}{2}. \end{aligned}$$

Thus, for $i = (0, y, 0)$ on leg 2, we have

$$\begin{aligned} H(i, v_0) &= y + 2\Big(y(b - y - 1) + \frac{y(y+1)}{2}\Big) \\ &= y + 2y(b - y - 1) + y(y + 1) \\ &= y(2b - y). \end{aligned}$$

3. i **on leg 3**

   For i on leg 3 with $d(i, v_0) = z$, we can write i as $(0, 0, z)$. Since leg 3 is identical in structure to leg 2, we find by symmetry that

$$H(i, v_0) = z(2b - z).$$

Maximizing $H(i, v_0)$

We have now found specific hitting time formulas for any $i \in V$:

- $i = (x, 0, 0)$ on leg 1: $H(i, v_0) = x(2a - x)$
- $i = (0, y, 0)$ on leg 2: $H(i, v_0) = y(2b - y)$
- $i = (0, 0, z)$ on leg 3: $H(i, v_0) = z(2b - z)$

Now, we'll maximize these hitting times, keeping in mind the constraints of our a, b, b spider, in which $2 < x \leq a$, $2 < y \leq b$, $2 < z \leq b$, and $2 < a < b$.

- i on leg 1: $\frac{d}{dx}(x(2a - x)) = 2a - 2x = 0 \Rightarrow x = a \Rightarrow max_i H(i, v_0) = a^2$
- i on leg 2: $\frac{d}{dy}(y(2b - y)) = 2b - 2y = 0 \Rightarrow y = b \Rightarrow max_i H(i, v_0) = b^2$
- i on leg 3: $\frac{d}{dz}(z(2b - z)) = 2b - 2z = 0 \Rightarrow z = b \Rightarrow max_i H(i, v_0) = b^2$

Since $a < b$, we find that $max_{i \in V} H(i, v_0) = b^2$ is achieved by $i = \ell_2$ and $i = \ell_3$.

7.1.2 Finding $max_{i \in V} H(i,j)$ for all other $j \in V$

Again, recall equation 11 for hitting time on an a, b, b spider:

$$H(i,j) = \ell(i, \ell_1; j) + \ell(i, \ell_2; j) + \ell(i, z_0; j) + 3\ell(i, v_0; j) + 2 \sum_{k_2 \in V} \ell(i, k_2; j).$$

As in Section 7.1.1 above, we'll consider various cases of i and j to find a formula for $H(i,j)$ for any pairing of i and j. In particular, our calculations will be different depending on whether i and j are on the same leg or if they're located on different legs. Furthermore, for i and j on the same leg, we'll need to consider extra sub-cases based on whether i or j is located closer to v_0, since this will affect how the $i-j$ and $k-j$ paths will overlap for different k.

1. **i and j on leg 1**

   (a) **j closer to v_0**

   As in the previous section, we'll find $\ell(i,k;j)$ for each case of k (degree 1, 2, or 3), and then we'll put those lengths into the above formula to find hitting time between any i, j on leg 1. For i, j on leg 1 with $d(i, v_0) = x_1$ and $d(j, v_0) = x_2$, we'll write $i = (x_1, 0, 0)$ and $j = (x_2, 0, 0)$. In this case, $x_1 > x_2$ (since j is closer to v_0), and $d(i,j) = x_1 - x_2$. Now, let's examine the lengths of the overlaps of the $i-j$ and $k-j$ paths for every vertex k in our graph.

degree-1 vertices:	$\ell(i, \ell_1; j) = d(i,j) = x_1 - x_2$
	$\ell(i, \ell_2; j) = \ell(i, \ell_3; j) = 0$
degree-3 vertex:	$\ell(i, v_0; j) = 0$
degree-2 vertices:	For k between i and ℓ_1: $\ell(i,k;j) = d(i,j) = x_1 - x_2$
	For k between i and j, including i: $\ell(i,k;j) = d(k,j)$
	For k between j and v_0 or on another leg: $\ell(i,k;j) = 0$

   Now, let's find $\sum_{k_2 \in V} \ell(i, k_2; j)$ for all degree-2 vertices in the graph.

For k between i and ℓ_1, not including i, our sum is

$$\begin{aligned}\sum_{k_2 \in V} \ell(i, k_2; j) &= (x_1 - x_2) \cdot (\text{number of degree-2 vertices between } i \text{ and } \ell_1) \\ &= (x_1 - x_2)(a - d(i, v_0) - 1) \\ &= (x_1 - x_2)(a - x_1 - 1).\end{aligned}$$

For k between i and j, including i, our sum is

$$\begin{aligned}\sum_{k_2 \in V} \ell(i, k_2; j) &= 1 + \cdots + d(i, j) \\ &= 1 + \cdots + (x_1 - x_2) \\ &= \frac{(x_1 - x_2)(x_1 - x_2 + 1)}{2}.\end{aligned}$$

Summing over all vertices, we find that

$$\begin{aligned}H(i, j) &= (x_1 - x_2) + 2\Big((a - x_1 - 1)(x_1 - x_2) + \frac{(x_1 - x_2)(x_1 - x_2 + 1)}{2}\Big) \\ &= (x_1 - x_2)(1 + 2a - 2x_1 - 2 + x_1 - x_2 + 1) \\ &= (x_1 - x_2)(2a - x_1 - x_2).\end{aligned}$$

(b) i **closer to** v_0

In this case, for i, j on leg 1 with i closer to v_0 (instead of j), we'll still write $i = (x_1, 0, 0)$ and $j = (x_2, 0, 0)$. Now, we have $x_2 > x_1$ since i is closer to v_0, and $d(i, j) = x_2 - x_1$. This time, we need to be careful when thinking about the lengths of the overlaps of the paths; they'll be different since i and j have switched positions.

degree-1 vertices:	$\ell(i, \ell_1; j) = 0$
	$\ell(i, \ell_2; j) = \ell(i, \ell_3; j) = d(i, j) = x_2 - x_1$
degree-3 vertex:	$\ell(i, v_0; j) = d(i, j) = x_2 - x_1$
degree-2 vertices:	For k between j and ℓ_1: $\ell(i, k; j) = 0$
	For k between i and j, including i: $\ell(i, k; j) = d(k, j)$
	For k between i and v_0 or on another leg: $\ell(i, k; j) = d(i, j) = x_2 - x_1$

For k between i and j, including i, we have

$$\begin{aligned}\sum_{k_2 \in V} \ell(i, k_2; j) &= 1 + \cdots + d(i,j) \\ &= 1 + \cdots + (x_2 - x_1) \\ &= \frac{(x_2 - x_1)(x_2 - x_1 + 1)}{2}.\end{aligned}$$

For k between i and v_0 or on another leg, we have

$$\begin{aligned}\sum_{k_2 \in V} \ell(i, k_2; j) &= (x_2 - x_1) \cdot (\text{number of degree-2 vertices between } i \text{ and } v_0 \text{ or on another leg}) \\ &= (x_2 - x_1)\big((b-1) + (b-1) + (d(i, v_0) - 1)\big) \\ &= (x_2 - x_1)(2b - 2 + x_1 - 1) \\ &= (x_2 - x_1)(2b - 3 + x_1).\end{aligned}$$

Thus, we find that

$$\begin{aligned}H(i,j) &= 2(x_2 - x_1) + 3(x_2 - x_1) + 2\Big(\frac{(x_2 - x_1)(x_2 - x_1 + 1)}{2} + (x_2 - x_1)(2b - 3 + x_1)\Big) \\ &= (x_2 - x_1)(5 + x_2 - x_1 + 1 + 4b - 6 + 2x_1) \\ &= (x_2 - x_1)(4b + x_1 + x_2).\end{aligned}$$

2. **i and j on leg 2**

   (a) **j closer to v_0**

   For i and j on leg 2 with $d(i, v_0) = y_1$ and $d(j, v_0) = y_2$, we'll write $i = (0, y_1, 0)$ and

$j = (0, y_2, 0)$. In this case, we have $y_1 > y_2$ since j is closer to the v_0, and $d(i,j) = y_1 - y_2$.

degree-1 vertices:	$\ell(i, \ell_1; j) = 0$
	$\ell(i, \ell_2; j) = d(i,j) = y_1 - y_2$
	$\ell(i, \ell_3; j) = 0$
degree-3 vertex:	$\ell(i, v_0; j) = d(i,j) = 0$
degree-2 vertices:	For k between i and ℓ_2: $\ell(i,k;j) = d(i,j) = y_1 - y_2$
	For k between i and j, including i: $\ell(i,k;j) = d(k,j)$
	For k between j and v_0 or on another leg: $\ell(i,k;j) = 0$

For k between i and ℓ_2, not including i, we have

$$\begin{aligned}\sum_{k_2 \in V} \ell(i, k_2; j) &= (y_1 - y_2) \cdot (\text{number of degree-2 vertices between } i \text{ and } \ell_2) \\ &= d(i,j)(b - d(i, v_0) - 1) \\ &= (y_1 - y_2)(b - y_1 - 1).\end{aligned}$$

For k between i and j, including i, we have

$$\begin{aligned}\sum_{k_2 \in V} \ell(i, k_2; j) &= 1 + \cdots + d(i,j) \\ &= 1 + \cdots + (y_1 - y_2) \\ &= \frac{(y_1 - y_2)(y_1 - y_2 + 1)}{2}.\end{aligned}$$

Thus, we find that

$$\begin{aligned}H(i,j) &= (y_1 - y_2) + 2\Big((y_1 - y_2)(b - y_1 - 1) + \frac{(y_1 - y_2)(y_1 - y_2 + 1)}{2}\Big) \\ &= (y_1 - y_2)(1 + 2b - 2y_1 - 2 + y_1 - y_2 + 1) \\ &= (y_1 - y_2)(2b - y_1 - y_2).\end{aligned}$$

(b) i **closer to** v_0

In this case, for i, j on leg 2 with i closer to v_0, we'll still write $i = (0, y_1, 0)$ and

$j = (0, y_2, 0)$. Now, we have $y_2 > y_1$ since i is closer to v_0, and $d(i,j) = y_2 - y_1$.

degree-1 vertices:	$\ell(i, \ell_1; j) = d(i,j) = y_2 - y_1$
	$\ell(i, \ell_2; j) = 0$
	$\ell(i, \ell_3; j) = d(i,j) = y_2 - y_1$
degree-3 vertex:	$\ell(i, v_0; j) = d(i,j) = d(i,j) = y_2 - y_1$
degree-2 vertices:	For k between j and ℓ_2: $\ell(i,k;j) = 0$
	For k between i and j, including i: $\ell(i,k;j) = d(k,j)$
	For k between i and v_0 or on another leg: $\ell(i,k;j) = d(i,j) = y_2 - y_1$

For k between i and j, including i, we have

$$\begin{aligned}\sum_{k_2 \in V} \ell(i, k_2; j) &= 1 + \cdots + d(i,j) \\ &= 1 + \cdots + (y_2 - y_1) \\ &= \frac{(y_2 - y_1)(y_2 - y_1 + 1)}{2}.\end{aligned}$$

For k between i and v_0 or on another leg, our sum is

$$\begin{aligned}&(y_2 - y_1) \cdot (\text{number of degree-2 vertices between } i \text{ and } v_0 \text{ or on another leg}) \\ &= (x_2 - x_1)\big((a-1) + (b-1) + (d(i,v_0) - 1)\big) \\ &= (x_2 - x_1)(a + b - 2 + y_1 - 1) \\ &= (x_2 - x_1)(a + b - 3 + y_1).\end{aligned}$$

Thus, we find that

$$\begin{aligned}H(i,j) &= 2(y_2 - y_1) + 3(y_2 - y_1) + 2\Big(\frac{(y_2 - y_1)(y_2 - y_1 + 1)}{2} + (y_2 - y_1)(a + b - 3 + y_1)\Big) \\ &= (y_2 - y_1)(5 + y_2 - y_1 + 1 + 2a + 2b - 6 + 2y_1) \\ &= (y_2 - y_1)(2a + 2b + y_1 + y_2).\end{aligned}$$

3. **i and j on leg 3**

For i and j on leg 3 with $d(i, v_0) = z_1$ and $d(j, v_0) = z_2$, we can write $i = (0, 0, z_1)$ and $j = (0, 0, z_2)$. The hitting times will be the same as those in the previous case since legs 2 and 3 are both of length b. Therefore, by symmetry, we have

- j closer to v_0: $H(i, j) = (z_1 - z_2)(2b - z_1 - z_2)$
- i closer to v_0: $H(i, j) = (z_2 - z_1)(2a + 2b + z_1 + z_2)$

4. **i on leg 1, j on leg 2**

Now that i and j are on different legs, our hitting times calculations will be even trickier. In this case, for i on leg 1 and j on leg 2 with $d(i, v_0) = x$ and $d(j, v_0) = y$, we can write $i = (x, 0, 0)$ and $j = (0, y, 0)$.

degree-1 vertices:	$\ell(i, \ell_1; j) = d(i, j) = x + y$
	$\ell(i, \ell_2; j) = 0$
	$\ell(i, \ell_3; j) = d(j, v_0) = y$
degree-3 vertex:	$\ell(i, v_0; j) = d(j, v_0) = y$
degree-2 vertices:	For k between i and ℓ_1: $\ell(i, k; j) = d(i, j) = x + y$
	For k between i and j, including i: $\ell(i, k; j) = d(k, j)$
	For k between j and ℓ_2: $\ell(i, k; j) = 0$
	For k on leg 3: $\ell(i, k; j) = d(j, v_0) = y$

For k on leg 1 between i and ℓ_1, not including i, we have

$$\begin{aligned}\sum_{k_2 \in V} \ell(i, k_2; j) &= (y_1 - y_2) \cdot (\text{number of degree-2 vertices between } i \text{ and } \ell_1) \\ &= (x + y)(a - d(i, v_0) - 1) \\ &= (x + y)(a - x - 1).\end{aligned}$$

For k between i and j, we add the distances $d(k, j)$ of all degree-2 vertices between i and j, including i, subtracting $d(v_0, j)$ since v_0 has degree 3. Therefore, we have

$$\begin{aligned}\sum_{k_2 \in V} \ell(i, k_2; j) &= 1 + \cdots + d(i,j) - d(v_0, j) \\ &= 1 + \cdots + (x+y) - y \\ &= \frac{(x+y)(x+y+1)}{2} - y.\end{aligned}$$

Finally, for k on leg 3, we have

$$\begin{aligned}\sum_{k_2 \in V} \ell(i, k_2; j) &= y \cdot (\text{number of degree-2 vertices on leg 3}) \\ &= y(b-1).\end{aligned}$$

Summing over all vertices, we find that

$$\begin{aligned}H(i,j) &= (x+y) + y + 3y + 2\Big((x+y)(a-x-1) + \frac{(x+y)(x+y+1)}{2} - y + y(b-1)\Big) \\ &= (x+y)(1 + 2a - 2x - 2 + x + y + 1) + y(4 - 2 + 2b - 2) \\ &= (x+y)(2a - x + y) + 2by.\end{aligned}$$

5. ***i* on leg 1, *j* on leg 3**

For i on leg 1 and j on leg 3 with $d(i, v_0) = x$ and $d(j, v_0) = z$, we can write i as $(x, 0, 0)$ and j as $(z, 0, 0)$. The hitting times will be the same as above since legs 2 and 3 are both of length b. Therefore, by symmetry, we have

$$H(i,j) = (x+z)(2a - x + z) + 2bz.$$

6. ***i* on leg 2, *j* on leg 1**

For i on leg 2 and j on leg 1, we'll write $i = (0, y, 0)$ and $j = (x, 0, 0)$. This case looks similar to when i was on leg 1 and j was on leg 2, but this time, i and j have flipped, resulting in a different hitting time since j is now located on the shorter leg, so there will be a different number of vertices on either side of j.

degree-1 vertices:	$\ell(i,\ell_1;j) = d(i,j) = 0$
	$\ell(i,\ell_2;j) = d(i,j) = x+y$
	$\ell(i,\ell_3;j) = d(j,v_0) = x$
degree-3 vertex:	$\ell(i,v_0;j) = d(j,v_0) = x$
degree-2 vertices:	For k between i and ℓ_2: $\ell(i,k;j) = d(i,j) = x+y$
	For k between i and j, including i: $\ell(i,k;j) = d(k,j)$
	For k between j and ℓ_1: $\ell(i,k;j) = 0$
	For k on leg 3: $\ell(i,k;j) = d(j,v_0) = x$

For k on leg 2 between i and ℓ_2, not including i, we have

$$\begin{aligned}\sum_{k_2\in V} \ell(i,k_2;j) &= (x+y)\cdot(\text{number of degree-2 vertices between } i \text{ and } \ell_2)\\ &= (x+y)(b-d(i,v_0)-1)\\ &= (x+y)(b-y-1).\end{aligned}$$

For k between i and j, including i, we again sum $d(k,j)$ for all degree-2 vertices between i and j, subtracting $d(v_0,j)$ since v_0 has degree 3. Therefore, we have

$$\begin{aligned}\sum_{k_2\in V} \ell(i,k_2;j) &= 1+\cdots+d(i,j)-d(v_0,j)\\ &= 1+\cdots+(x+y)-x\\ &= \frac{(x+y)(x+y+1)}{2}-x.\end{aligned}$$

For k on leg 3, we have

$$\begin{aligned}\sum_{k_2\in V} \ell(i,k_2;j) &= x\cdot(\text{number of degree-2 vertices on leg 3})\\ &= x(b-1).\end{aligned}$$

Thus, we find that

$$\begin{aligned} H(i,j) &= (x+y)+x+3x+2\Big((x+y)(b-y-1)+\frac{(x+y)(x+y+1)}{2}-x+x(b-1)\Big) \\ &= (x+y)(1+2b-2y-2+x+y+1)+x(4-2+2b-2) \\ &= (x+y)(2b+x-y)+2bx. \end{aligned}$$

7. i **on leg 3,** j **on leg 1**

For i on leg 3 and j on leg 1 with $d(i,v_0)=z$ and $d(j,v_0)=x$, we'll write $i=(0,0,z)$ and $j=(x,0,0)$. The hitting times will be the same as the above case since legs 2 and 3 are of the same length. Therefore, by symmetry, we have

$$H(i,j)=(x+z)(2b+x-z)+2bx.$$

8. i **on leg 2,** j **on leg 3**

For i on leg 2 and j on leg 3 with $d(i,v_0)=y$ and $d(j,v_0)=z$, we'll write $i=(0,y,0)$ and $j=(0,0,z)$.

degree-1 vertices:	$\ell(i,\ell_1;j)=d(j,v_0)=z$
	$\ell(i,\ell_2;j)=d(i,j)=y+z$
	$\ell(i,\ell_3;j)=0$
degree-3 vertex:	$\ell(i,v_0;j)=d(j,v_0)=z$
degree-2 vertices:	For k between i and ℓ_2: $\ell(i,k;j)=d(i,j)=y+z$
	For k between i and j, including i: $\ell(i,k;j)=d(k,j)$
	For k between j and ℓ_3: $\ell(i,k;j)=0$
	For k on leg ℓ_1: $\ell(i,k;j)=d(j,v_0)=z$

For k on leg 2 between i and ℓ_2, not including i, we have

$$\sum_{k_2 \in V} \ell(i, k_2; j) = (y+z) \cdot (\text{number of degree-2 vertices between } i \text{ and } \ell_2)$$
$$= (y+z)(b - d(i, v_0) - 1)$$
$$= (y+z)(b - y - 1).$$

For k between i and j, including i, we have

$$\sum_{k_2 \in V} \ell(i, k_2; j) = 1 + \cdots + d(i,j) - d(j, v_0)$$
$$= 1 + \cdots + (y+z) - z$$
$$= \frac{(y+z)(y+z+1)}{2} - z.$$

For k on leg 1, we have

$$\sum_{k_2 \in V} \ell(i, k_2; j) = z \cdot (\text{number of degree-2 vertices on leg 1})$$
$$= z(a-1).$$

Thus, we find that

$$\begin{aligned} H(i,j) &= z + (y+z) + 3z + 2\Big((y+z)(b-y-1) + \frac{(y+z)(y+z+1)}{2} - z + z(a-1)\Big) \\ &= (y+z)(1 + 2b - 2y - 2 + y + z + 1) + z(4 - 2 + 2a - 2) \\ &= (y+z)(2b - y + z) + 2az. \end{aligned}$$

9. ***i* on leg 3, *j* on leg 2**

For i on leg 3 and j on leg 2 with $d(i, v_0) = z$ and $d(j, v_0) = y$, we'll write $i = (0, 0, z)$ and $j = (0, y, 0)$. The hitting times will be the same as above since legs 2 and 3 are of the same length. Therefore, by symmetery, we have

$$H(i,j) = (y+z)(2b - y + z) + 2az.$$

Maximizing $H(i,j)$

Our goal is to ultimately find $\min_{j\in V}\max_{i\in V} H(i,j)$. Now that we've found hitting time formulas for all possible placements of i and j, we want to find the maximum hitting time to a vertex j in each case above by taking their derivative with respect to i under the constraints $2 \leq x \leq a$, $2 \leq y \leq b$, $2 \leq z \leq b$, and $2 \leq a \leq b$. From there, we'll find the maximum hitting time to a vertex j on each leg. Since we'll want to maximize each formula with respect to i, it will be helpful to sort them by each case of j (Table 4).

j	j coordinates	i	i coordinates	closer to v_0?	$H(i,j)$
leg 1	$(x_2,0,0)$	leg 1	$(x_1,0,0)$	j	$(x_1-x_2)(2a-x_1-x_2)$
				i	$(x_2-x_1)(4b+x_1+x_2)$
	$(x,0,0)$	leg 2	$(0,y,0)$	–	$(x+y)(2b+x-y)+2bx$
	$(x,0,0)$	leg 3	$(0,0,z)$	–	$(x+z)(2b+x-z)+2bx$
leg 2	$(0,y_2,0)$	leg 2	$(0,y_1,0)$	j	$(y_1-y_2)(2b-y_1-y_2)$
				i	$(y_2-y_1)(2a+2b+y_1+y_2)$
	$(0,y,0)$	leg 3	$(0,0,z)$	–	$(y+z)(2b-y+z)+2az$
	$(0,y,0)$	leg 1	$(x,0,0)$	–	$(x+y)(2a-x+y)+2by$
leg 3	$(0,0,z_2)$	leg 3	$(0,0,z_1)$	j	$(z_1-z_2)(2b-z_1-z_2)$
				i	$(z_2-z_1)(2a+2b+z_1+z_2)$
	$(0,0,z)$	leg 2	$(0,y,0)$	–	$(y+z)(2b-y+z)+2az$
	$(0,0,z)$	leg 1	$(x,0,0)$	–	$(x+z)(2a-x+z)+2bz$

Table 4: Hitting times between any two vertices i,j on an a,b,b spider.

Now, we'll maximize hitting time between any vertices i and j on each leg under the given constraints (Table 5). From there, we'll find the maximum hitting time to any vertex j on each leg (Table 6).

j	j coordinates	i	i coordinates	closer to v_0?	$\max_{i\in V} H(i,j)$
leg 1	$(x_2,0,0)$	leg 1	$(x_1,0,0)$	j	$(x_2-a)^2$
				i	$x_2^2+4bx_2-(4b+1)$
	$(x,0,0)$	leg 2	$(0,y,0)$	–	$x^2+4bx+b^2$
	$(x,0,0)$	leg 3	$(0,0,z)$	–	$x^2+4bx+b^2$
leg 2	$(0,y_2,0)$	leg 2	$(0,y_1,0)$	j	$(y_2-b)^2$
				i	$y_2^2+(2a+2b)y_2-(2a+2b+1)$
	$(0,y,0)$	leg 3	$(0,0,z)$	–	$-y^2+2by+2ab+3b^2$
	$(0,y,0)$	leg 1	$(x,0,0)$	–	$y^2+(2a+2b)y+a^2$
leg 3	$(0,0,z_2)$	leg 3	$(0,0,z_1)$	j	$(z_2-b)^2$
				i	$z_2^2+(2a+2b)z_2-(2a+2b+1)$
	$(0,0,z)$	leg 2	$(0,y,0)$	–	$z^2+(2a+2b)z+b^2$
	$(0,0,z)$	leg 1	$(x,0,0)$	–	$z^2+(2a+2b)z+a^2$

Table 5: Maximum hitting times to any vertex j from i on each leg of an a,b,b spider.

j	$\max_{i\in V} H(i,j)$
leg 1	$x^2+4bx+b^2$
leg 2	$-y^2+2by+2ab+3b^2$
leg 3	$-z^2+2bz+2ab+3b^2$

Table 6: Maximum hitting times to any vertex j on an a,b,b spider.

7.1.3 Finding $\min_{j\in V}\max_{i\in V} H(i,j)$

We're almost there! Recall that $max_{i\in V}H(i,v_0)=b^2$, as calculated above. Under the constraints of our a,b,b spider, we know that

$$\begin{aligned}
b^2 &\le x^2+4bx+b^2 && \Rightarrow max_{i\in V}H(i,v_0)\le max_{i\in V}H(i,j \text{ on leg 1})\\
b^2 &\le -y^2+2by+2ab+3b^2 && \Rightarrow max_{i\in V}H(i,v_0)\le max_{i\in V}H(i,j \text{ on leg 2})\\
b^2 &\le -z^2+2bz+2ab+3b^2 && \Rightarrow max_{i\in V}H(i,v_0)\le max_{i\in V}H(i,j \text{ on leg 3})
\end{aligned}$$

Thus, maximum hitting time to v_0 is less than the maximum hitting time to any other vertex j in our spider, and we have proven that the root v_0 achieves $\min_{j\in V}\max_{i\in V} H(i,j)$ and is the focus of an a,b,b spider.

□

7.2 Implications and future directions

We've now shown that for an a, b, b spider, the focus is the root. Additionally, we've found an easier way of measuring distance between the vertices of our tree (Section 6), which we can use to calculate hitting time on trees such as a, b, b spiders. Following from our proof above, we can conclude that on an a, b, b spider, the forget distribution μ is the singleton distribution concentrated on the root. This tells us that even a slight change in the structure of a tree can shift the focus from a higher-degree vertex to a lower-degree vertex. In terms of PPI networks, this can be significant: if one leg is just a little longer, then the forget distribution may go from being concentrated on a hub protein (the root) to being concentrated on a less "important" protein down one of the legs.

From our proof above, we can also see that on a, b, b spiders, the focus has two pessimal vertices ℓ_2 and ℓ_3. Further experimentation has suggested that when we extend one of the legs of our spider so that all three legs have different lengths, there are sometimes three pessimal vertices (located on all three leaves) rather than two – that is, all three leaves are equally "far away" from the focus of these spiders. It seems that the location of the focus is key to understanding the number and location of pessimal vertices, but we have yet to identify exactly how the leg lengths affect this.

These results bring to mind a number of questions to explore in the future, many of which we now have the tools to address. First, can we further generalize Theorem 2 for *all* spiders with $2 \leq a \leq b$? Furthermore, under which conditions on leg lengths are there two or three pessimal vertices, and how does this relate to the placement of the focus? From there, we would like to extend our study of trees to more complicated types of trees, such as those with different leg placement or additional legs. A final (major) question is whether other target distributions exhibit unique properties on other types of graphs, just as the forget distribution is concentrated on the focus for a, b, b spiders. Ultimately, we hope that our work will better inform our choice of target distribution for EFD, perhaps based on the specific structure of the PPI network, so we can continue to improve our predictions of protein function.

References

[1] Michael R. Anderberg, *Cluster analysis for applications*, Probability and Mathematical Statistics: A Series of Monographs and Textbooks, Academic Press, New York, 1973.

[2] C. Atilgan, Z.N. Gerek, S.B. Ozkan, and A.R. Atilgan, *Manipulation of conformational change in proteins by single-residue perturbations*, Biophysical Journal **99** (2010), 933–943.

[3] Andrew Beveridge, *Centers for random walks on trees*, SIAM J. Discrete Math **23** (2009), no. 1, 300–318.

[4] ———, *A hitting time formula for the discrete green's function*, Combinatorics, Probability and Computing **25** (2016), 362–379.

[5] Andrew Beveridge, Mengfei Cao, Amanda Redlich, and Lenore Cowen, *Designing exit frequency distance measures for biological networks*, SIAM Workshop on Network Science 2016, Society for Industrial and Applied Mathematics, 2016, pp. 35–36.

[6] Edward Boehnlein, Peter Chin, Amit Sinha, and Linyan Lu, *Computing diffusion state distance using greens function and heat kernel on graphs*, Algorithms and Models for the Web Graph, Lecture Notes in Computer Science, vol. 8882, Springer International Publishing, 2014, pp. 79–97.

[7] Mengfei Cao, Lenore J. Cowen, Andrew Beveridge, and Amanda Redlich, Personal communication.

[8] Mengfei Cao, Christopher M. Pietras, Xian Feng, Kathryn J. Doroschak, Thomas Schaffner, Jisoo Park, Hao Zhang, Lenore J. Cowen, and Benjamin J. Hescott, *New directions for diffusion-based network prediction of protein function: incorporating pathways with confidence*, Bioinformatics **30** (2014), i219–i227.

[9] Mengfei Cao, Hao Zhang, Jisoo Park, Noah M. Daniels, Mark E. Crovella, Lenore J. Cowen, and Benjamin Hescott, *Going the distance for protein function prediction: A new distance metric for protein interaction networks*, PLOS One **8** (2013), no. 10.

[10] Hon Nian Chua, Wing-Kin Sung, and Limsoon Wong, *Exploiting indirect neighbours and topological weight to predict protein function from proteinprotein interactions*, Bioinformatics **22** (2006), no. 13, 1623–1630.

[11] Fan Chung and S.-T. Yau, *Discrete green's functions*, Journal of Combinatorial Theory **A** (2000), no. 91, 191–214.

[12] Serkan Erdin, Andreas Martin Lisewski, and Olivier Lichtarge, *Protein function prediction: Towards integration of similarity metrics*, Current Opinion in Structural Biology **21** (2011), no. 2, 180–188.

[13] Anne Claude Gavin, Patrick Aloy, Paola Grandi, Roland Krause, Markus Boesche, Martina Marzioch, Christina Rau, Lars Juhl Jensen, et al., *Proteome survey reveals modularity of the yeast cell machinery*, Nature **400** (2006), no. 30, 631–636.

[14] Ivor Grattan-Guinness, *The rainbow of mathematics: A history of the mathematical sciences*, W. W. Norton & Company, 2000.

[15] Jing-Dong J. Han, Nicolas Bertin, Tong Hao, Debra S. Goldberg, et al., *Evidence for dynamically organized modularity in the yeast protein-protein interaction network*, Nature **430** (2004), 88–93.

[16] Peter E. Hodges, William E. Payne, and James I. Garrels, *The yeast protein database (ypd): A curated proteome database for saccharomyces cerevisiae*, Nucleic Acids Research **26** (1998), no. 1, 68–72.

[17] National Human Genome Research Institute, *An overview of the human genome project*, `https://www.genome.gov/12011238/an-overview-of-the-human-genome-project/`, 2016, [Online; accessed 28 December 2017].

[18] H. Jeong, S.P. Mason, and A.-L. Barabási, *Lethality and centrality in protein networks*, Nature **411** (2001), 41–42.

[19] Mark Kac, *Random walk and the theory of brownian motion*, The American Mathematical Monthly **54** (1947), no. 7, 369–391.

[20] Minoru Kanehisa and Sasumu Goto, *Kegg: Kyoto encyclopedia of genes and genomes*, Nucleic Acids Research **28** (2000), no. 1, 27–30.

[21] U.S. National Library of Medicine Lister Hill National Center for Biomedical Communications, *How genes work*, `https://ghr.nlm.nih.gov/primer/basics/gene`, 2017, [Online; accessed 27 December 2017].

[22] László Lovász, *Random walks on graphs: A survey*, Combinatorics: Paul Erdős is eighty, vol. 2, Janos Bolyai Mathematical Society, 1993, pp. 1–46.

[23] László Lovász and Peter Winkler, *Efficient stopping rules for markov chains*, Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing (New York, NY), ACM, 1995, pp. 76–82.

[24] ———, *Mixing of random walks and other diffusions on a graph*, Surveys in Combinatorics (Peter Rowlinson, ed.), London Mathematical Society Lecture Notes Series 218, Cambridge University Press, 1995, pp. 119–154.

[25] ———, *Reversal of markov chains and the forget time*, Combinatorics, Probability and Computing **7** (1998), no. 2, 189–204.

[26] Nina Mishra, Robert Schreiber, Isabelle Stanton, and Robert E Tarjan, *Clustering social networks*, Algorithms and Models for the Web-Graph (Heidelberg Berlin) (Anthony Bonato and Fan R. K.Chung, eds.), Springer Berlin Heidelberg, 2007, pp. 56–67.

[27] Elena A. Ponomarenko, Ekaterina V. Poverennaya, Ekaterina V. Ilgisonis, Mikhail A. Pyatnitskiy, Arthur T. Kopylov, Victor G. Zgoda, Andrey V. Lisitsa, and Alexander I. Archakov, *The size of the human proteome: The width and depth*, International Journal of Analytical Chemistry **2016** (2016).

[28] Andreas Ruepp, Alfred Zollner, Dieter Maier, Kaj Albermann, Jean Hani, Martin Mokrejs, Igor Tetko, Ulrich Guldener, Gertrud Mannhaupt, Martin Munsterkotter, and H. Werner Mewes, *The funcat, a functional annotation scheme for systematic classification of proteins from whole genomes*, Nucleic Acids Research **32** (2004), no. 18, 5339–5545.

[29] Jimin Song and Maya Singh, *How and when should interactome-derived clusters be used to predict functional modules and protein function?*, Bioinformatics **25** (2009), no. 23, 3142–3150.

[30] Christ Stark, Bobby-Joe Breitkreutz, Teresa Rugely, Lorrie Boucher, Ashton Breitkreutz, and Mike Tyers, *Biogrid: A general repository for interaction dataseta*, Nucleic Acids Research **34** (2–6), no. Database Issue, D535–D539.

[31] Alexei Vazquez, Alessandro Flammini, Amos Maritan, and Alessandro Vespignani, *Global protein function prediction from protein-protein interaction networks*, Nature Biotechnology **21** (2003), 697–700.

[32] K.M. Verspoor, *Roles for text mining in protein function prediction*, Methods in Molecular Biology **1159** (2014), 95–108.