A Fast Method to Calculate Hitting Time Distribution for a Random Walk on Connected Undirected Graph

Preprint · August 2019

DOI: 10.48550/arXiv.1908.09644

1 author:

Enzhi Li
Amazon
10 PUBLICATIONS **6** CITATIONS

A Fast Method to Calculate Hitting Time Distribution for a Random Walk on a Connected and Undirected Graph

Enzhi Li, Zhengyi Le
Suning R&D Center, Palo Alto, USA
(Dated: October 31, 2019)

With the advent of increasingly large graphs, we need to find a quick and reliable method to measure the distance and similarity between any pair of nodes on a large graph. One way to measure the distance is by performing random walks on graph, and people already designed plenty of algorithms to accomplish this goal. However, most of the implementations of random walk algorithms are computationally expensive due to the use of Monte Carlo simulations which could be pretty time-consuming. Here, we propose an alternative measure of the distance between any pair of nodes on a connected and undirected graph using the notion of hitting time for a random walk. We also give an analytical solution to the hitting time distribution of a random walk on graph. This analytical method, which can be conveniently implemented using SciPy linear algebra packages, is more time-saving to run and yields more accurate results than that obtained via Monte Carlo simulations. It is further noted that the hitting times also provide a glimpse of the community structure of a graph. This algorithm for measuring the distance between any pair of nodes is specifically devised to measure the influence of a fraudulent user upon all the other users that co-occur in the same social network. We employ our algorithm to weed out potential fraudsters from among tens of thousands of online retail users.

I. INTRODUCTION

With the advent of the era of social media, graph theory has become an increasingly important tool for studying user behavior in such fields as social networks, academic citation, online retails, web analysis, etc. Various graph algorithms have been proposed to attack the problems people encounter during the study of graphs. For example, in order to find the web pages that are of uttermost importance and relevance to a set of keywords, people designed the PageRank algorithm to rank the tens of millions of web pages that are available on line[1]. Nodes on a connected graph may form tightly bound communities, which can be detected using an algorithm that aims to maximize the modularity of each potential cluster[2–4]. We can equally well study the division of a connected graph into weakly linked communities via the Laplacian matrix, which is derived from a graph's adjacency matrix[5]. It is also of significant interest to find a metric that can measure the distance between any two nodes on a connected graph, and the node2vec method is there to our resort[6]. All of these algorithms depend on a direct or indirect invocation of a graph's adjacency matrix, an invocation that is understandable due to the convenience of the adjacency matrix in uniquely identifying a graph, whether be it directed or undirected.

The detection of potential fraudsters in a social network is an imperative task for online retailers and requires the development of an efficient algorithm for performing label propagation in a connected graph. Users of online retailers may have connections with each other, and we can capture these connections by analyzing users' behavior. For example, different users may share the same shipping address, or they may log into their accounts via the same device. By analyzing data like this, we can create a social network of the users in which each user is represented by a node in a graph. If two users share the same login device, we can link an edge between these two users. By this way, we can create a graph that represents the social links between users, who may be normal or fraudulent. In order to protect the normal users from being exploited by fraudsters, we need to find a method to identify these fraudsters and build an aegis for the normal users against them. One way to identify these fraudsters is to employ a set of rules and see who has violated them. However, due to the possibly large number of fraudsters in a social network, it is impractical to try to weed out all fraudsters purely by hand. Here, we will perform a label propagation in a social network so that once we have found out a handful of fraudsters via rules, we can continue to find more potential fraudsters automatically. The propagation of labels in a social network requires a precise definition of the similarity between users. In this paper, we will explore an algorithm that could measure the distance (and thus similarity) between any pair of nodes in a graph. By calculating the distances between nodes in a graph, we can gain a deeper and more intuitive understanding about the similarities between apparently disconnected social network users since the similarity between a pair of users should be inversely proportional to their distance.

There are already well developed algorithms for measuring the distance between nodes in a graph, such as the geodesic path distance, or the node2vec algorithms. However, these algorithms either utterly disregards the graph structure or is too time-consuming to run. Here, we propose an algorithm for measuring the distance between a pair of nodes by considering the concept of hitting time for a random walk on a graph. Hitting time of a random walk is the number of steps traversed by a random walker before it hits a pre-specified target node

for the first time. This hitting time can be either obtained by Monte Carlo simulation which is again much too time-consuming to be practical for a large graph or can be exactly calculated by an analytical formula which we will derive in this paper. We will validate our algorithm by applying it to some real world problems that we encountered in our daily work. All of these test cases will be presented in detail in the main text of this paper.

The organization of the paper is as follows. In section II, we will give a brief review of the previous methods for calculating the distance between a pair of nodes. We will summarize the advantages and disadvantages of these methods, and explain why we want to propose an alternative method to calculate the distance using the notion of hitting times in random walks[7]. After outlining our algorithm in section III and detailing the algorithm in section IV, we continue to apply our analytical and numerical methods to small and huge graphs respectively, in section V. We also highlight the asymmetricity of our distance function under the exchange of its two arguments in the same section. In section VI, we make a comparison of our method with other existing methods. Finally, we make a conclusion in section VII.

II. BACKGROUND

This paper explores the influence of a node on another node, or on a specific set of nodes in a graph. PageRank algorithm is a convenient method that can measure how influential and important a single node is to the graph as a whole, whereas sometimes we also need to know how influential a node is to another specific node, a situation that could arise when we want to know how susceptible a community of nodes is to the presence of a labeled user in a social network. For example, we can create a connected graph whose nodes represent the users of an online retailer. If we already added a label to a user in the social network, then we want to know how that label will propagate among the other users that cooccur with the labeled user. Intuitively, this process of label propagation depends on the distance between each pair of users in the social network. The shorter the distance between two nodes, the more similar these two nodes are to each other, and thus the easier it is to propagate a label from one user to another. Nowadays, there are already plenty of algorithms that can measure the distance or similarity between each pair of nodes in a graph. However, there is no universal definition of this distance function, and for each specific case, people can devise their own version of distance function. Two distance functions that have gained much popularity are the geodesic distance[8] and the cosine distance which is a byproduct of node2vec algorithm[6]. The geodesic distance between two nodes in an undirected graph is defined as the length of the shortest path connecting these two nodes. The geodesic distance, which is calculated by finding the shortest distance from one node say A to anther node say B using Dijkstra's algorithm for sparse graph or Floyd's algorithm for dense graph, is a deterministic algorithm. In this algorithm, we are considering a deterministic walk on the graph. Due to the non-randomness of this algorithm, when we employ it to find the distance between two nodes in a graph, we have failed to capture a significant part of the known information about the graph. The disregarding of the rich structures of a graph from which we could have extracted a huge amount of precious information about the relationship between a pair of nodes constitutes one major disadvantage of this algorithm. In node2vec algorithm, we calculate the distance between two nodes by first mapping each node in the graph into a dense vector using the word2vec method[9], and then using the cosine distance between two mapped dense vectors as the distance between a pair of nodes. This algorithm, which can be considered as an extension of the word2vec algorithm to graphs, requires the pre-existence of a node corpus that can only be generated by performing tens of thousands of random walks on a graph. The generation of this corpus is pretty time-consuming and memory-intensive, thus precluding its application to extremely large graphs.

Another thing that is noteworthy is that both of these two algorithms yield symmetric distance functions for any pair of nodes in an undirected graph. The distance function is symmetric in the sense that the distance from node A to node B is guaranteed to be identical to the distance from node B to node A. However, even for an undirected graph such as the friendship social network of Facebook, it is unreasonable to believe the distance from an influential user to an obscure user should be the same as the distance from an obscure user to an influential user. Since not all users of a social network share identical reputation and influence, we claim that the relationships between social network users are non-equivalent, non-reflective and asymmetric. Thus, *a good definition of distance function between two nodes of a graph should take account of this non-equivalence, non-reflectivity and asymmetricity of relationships even for undirected graphs.*

As we have noted above, a deterministic walk on a graph tends to be blind to the rich structure of a graph. Therefore, here in this paper, we will focus our attention on random walks on graphs. There are many scenarios for performing random walks on graphs. One such scenario starts from a node, say A, and selects a node, say B, as its target, and performs a multitude of random walks starting from A and counts how many times this random walker encounters node B within a pre-specified number of steps. This encountering frequency for the random walk provides a measure of the distance between nodes A and B. The larger the frequency, the shorter the distance between A and B. This method of of measuring the distance between two nodes, although valid in some sense, has several drawbacks, the most prominent of which is its strong dependence upon such capricious parameters as the maximum number of nodes each random walk is allowed to traverse, and the number of random

walks to be performed for the encountering frequency to be statistically stable and meaningful. Another weak point of this method is that it is again much too time-consuming to perform enough number of random walks to gain a statistically significant result for two nodes that are located afar in a huge graph. The application of this random walk scenario to a small sized graph is no less troublesome due to the fact that a random walker starting from one node in a connected graph is guaranteed to reach any other node in the same graph as long as the random walk lasts long enough, thus rendering all the distances between any pair of nodes almost the same.

Considering the time-expensiveness of performing sufficiently large number of random walks on a large graph and the strong dependence of the final results on the hard-to-select hyper-parameters, we prefer to find an alternative method that can deliver an exact solution to the random walk problem, thus avoiding this lengthy and tedious process of Monte Carlo simulations from the beginning. For sake of concreteness, consider a graph in which we have labeled some nodes as "black", some as "white", and some as "unknown", as detailed in Ref. [10]. We can estimate the color of the unknown nodes either by performing a Monte Carlo simulation or by solving a discrete Laplacian equation. The inference of the colors of these unknown nodes is equivalent to performing label propagation in a graph. It is shown in Ref. [10] that solution of discrete Laplacian equation gives us more accurate results using far less time. Unfortunately, solution of Laplacian equations requires the pre-existence of boundary conditions, which are not always available[11]. The black and white labels in Ref. [10] are the boundary conditions for a direct solution of Laplacian equation to be feasible. However, if all the known labels are marked black, then the only thing that a solution of Laplacian equation can tell us is that all the colors of the unknown nodes should be black, which is practically useless to us. For example, in order to quantify the influence of a black node on the other nodes that cooccur in a social network, we also need the existence of at least one node that is explicitly labeled as "white", a label that is not always available.

In order to avoid these conundrums, here we propose a new algorithm that can measure the distance between any two nodes in a graph by giving an exact solution to a random walk problem on undirected graphs, just like the analytical solution of discrete Laplacian equation for color inference as described above. **The advantage of this algorithm** is that the final result is uniquely obtained by solving a sparse linear system, thus releasing us of the unnecessarily thorny duty of selecting a set of appropriate parameters, and saving us tens of thousands of CPU hours from performing Monte Carlo simulations thanks to the highly efficient numerical linear algebra packages that are readily available for performing sparse matrix multiplications. **Another advantage of this algorithm** is that it gives us a distance function that is asymmetric between a pair of nodes, reflecting the reality that users in a social network generally have non-equivalent and non-reflective relationships with each other.

III. PROPOSED METHOD

In this section, we will propose an analytical method for finding expected hitting times of a random walk on an undirected and connected graph G. The connectedness of the graph does not constitute a major restriction to our method due to the availability of efficient algorithms for finding connected components of an undirected graph.

The adjacency matrix of this graph is A, which is a $|\mathbb{V}| \times |\mathbb{V}|$ matrix ($\mathbb{V}$ is the set of vertices in the graph, and $|\mathbb{V}|$ is the cardinality of the set), with matrix elements $A_{ij} = 1$ if there is an edge between node i and node j, otherwise $A_{ij} = 0$. Because we are considering a social network of users who would have relationships only with others, we demand that the graph in this paper should be simple, meaning that none of the nodes are self-looped. The matrix dimension $|\mathbb{V}|$ is the number of nodes in the graph, and the number of non-zero matrix elements of A gives us the edge number. A is symmetric if graph G is undirected, or else it is generally non-symmetric.

In this paper, we are trying to calculate the probability of reaching a target node from any other node in the graph, whereas in a directed graph, a node may not be reachable from another node, thus here we only consider undirected and connected graphs for which the adjacency matrix is always symmetric. Furthermore, our method also applies to weighted graphs, for which $A_{ij} = w > 0$ if the edge connecting node i to j has weight w, and $A_{ij} = 0$ if there is no edge between nodes i and j.

A random walk from a start node to a target node on a graph is defined as follows:

Algorithm 1 Random walk on a graph

Require: An undirected and connected Graph G
1: **procedure** RANDOMWALK(s, t) ▷ s is the start node, and t is the target node.
2: $c \leftarrow s$
3: **repeat**
4: $r \leftarrow$ a random neighbor of c
5: $c \leftarrow r$
6: **until** $c = t$
7: **end procedure**

For a random walk that starts from node s, **the hitting time is defined as the number of steps needed for the random walker to reach a target node t for the first time**. According to this definition, the hitting time is a random variable that depends on the graph structure, the starting node s, and the target node t. Therefore, we can denote the hitting time as $N_t^{(s)}$.

Denote the probability of hitting target t after exactly

n steps starting from node s as

$$x_n^{(s)} = P(N_t^{(s)} = n) \tag{1}$$

Assume that node s has m neighboring nodes, of which at most one is the target node t. We enumerate these m nodes using indices $i_s = 1, 2, ..., m$. Then the probability $P(N_t^{(s)} = n)$ can be recursively represented as

$$P(N_t^{(s)} = n) = \sum_{\substack{i_s=1 \\ i_s \neq t}}^{m} \frac{w_{s,i_s}}{W_s} P(N_t^{(i_s)} = n-1) \tag{2}$$

In the above equation, $W_s = \sum_{i_s=1}^{m} w_{s,i_s}$ is the total weight associated with node s, $\frac{w_{s,i_s}}{W_s}$ represents the probability for the random walker to make a transition from node s to one of its neighbors i_s, and $P(N_t^{(i_s)} = n-1)$ is the probability of reaching target node t from node i_s after exactly $n-1$ steps. Since we are calculating the probability of reaching target t for the first time from node s after exactly n steps, the probability of reaching target starting from the target itself is zero for any non-zero number of steps, i.e., $P(N_t^{(t)} = n) = 0, \forall n > 0$. Thus, we demand that the random walker in the above recursive equation should not make a transition to the target node t even if t is one of the neighbors of node s, which justifies our notation $i_s \neq t$ in the summation subscript. If we scan all possible starting vertices s, we can obtain a simultaneous system of difference equations for the hitting probabilities $P(N_t^{(i)} = n), \forall i \in \mathbb{V}, i \neq t$, where $\mathbb{V}$ is the set of all vertices in graph G. Specifically, if a vertex j has only one single neighbor, and this very neighbor is just our target t, then we can directly write out the probability of reaching target t after exactly $n > 0$ steps starting from node j as $P(N_t^{(j)} = n) = \delta_{n,1}$, with $\delta_{n,1}$ being the Kronecker δ function. Since we already know the probability distribution of hitting times for such queer nodes which we call *adherents* to target t, we can ignore those nodes when establishing the simultaneous system of difference equations for $P(N_t^{(s)} = n)$.

Definition 1. A node in graph G is called an ***adherent*** to target node t if and only if this node has the target node as its only neighbor.

With these notations, we can write out a simultaneous system of difference equations for the probabilities of hitting target node t for the first time with exactly $n > 0$ steps after starting from different nodes $i \in \mathbb{V}$ as

$$P(N_t^{(i)} = n) = \sum_{\substack{j=1 \\ j \neq t}}^{|\mathbb{V}|} B_{ij} P(N_t^{(j)} = n-1), \tag{3}$$

where we have imposed the restriction that the starting node i should not be equal to t, and should not be an *adherent* to target t. **The B matrix is called probability transition matrix**, the elements of which are

$$B_{ij} = \begin{cases} \frac{w_{ij}}{\sum_{i'} w_{ii'}}, i' \in \{neighbors\ of\ i\} & A_{ij} \neq 0; \\ 0 & A_{ij} = 0. \end{cases} \tag{4}$$

Most of the time, B is a sparse matrix. Note that although in an undirected graph the adjacency matrix A is always symmetric, the probability transition matrix is generally non-symmetric. Moreover, due to the exclusion of the target node t in the definition of probability transition matrix, the sum of matrix elements for each row in B is not necessarily equal to 1. In fact, for a connected undirected graph, there is at least one row of B whose sum is less than 1. The rule is that $\sum_j B_{ij} = 1$ if the target node t is not a neighbor of node i; otherwise $\sum_j B_{ij} < 1$. Since we have excluded the target node t and all its adherent nodes from the set of starting nodes, the matrix B has a dimension that is smaller than that of matrix A. For an undirected graph, the matrix B is guaranteed to be square due to the fact that a random walker starting from a node that is not the target cannot possibly reach an adherent node to the target. The fact that matrix B has rows with sum that are less than 1 means that this matrix is not a Markov matrix, and that all of its eigenvalues have a magnitude smaller than 1. As a result, the spectral radius of matrix B is also smaller than 1. We will take advantage of this fact later in this paper.

We can consider the hitting time $N_t^{(i)}$ as the ith component of a column vector $\boldsymbol{N}_t$. Our aim in this paper is to study the probability distribution of this random vector. Once we have already created the probability transition matrix B, we can directly write out the expectation values of $\boldsymbol{N}_t$ as

$$\langle \boldsymbol{N}_t \rangle = \sum_{n=0}^{\infty} B^n \mathbf{1}, \tag{5}$$

where $\mathbf{1}$ is a column vector of which each element is 1, i.e, $\mathbf{1} = \begin{pmatrix} 1 & 1 & \dots & 1 \end{pmatrix}^{\mathrm{T}}$. Since the spectral radius of B is smaller than 1, the summation of power series in Eq. [5] will converge. By terminating the summation at a power that is high enough, we can obtain numerical results for the expected hitting times with arbitrary precision. We will show that the expected hitting times can be used to measure the distance between two nodes in an undirected graph. We can also obtain higher order moments of $\boldsymbol{N}_t$, the formulae for which are no more complicated than the one in Eq. [5]. We will show how to calculate all the moments of $\boldsymbol{N}_t$ in the next section.

IV. THEORETICAL PROOF

In the previous section, we have given a formula for calculating the expected hitting times from an arbitrary node to a target node in an undirected graph. In this section, we will give the necessary mathematical details for

obtaining that formula. Actually, we will overshoot this goal by giving a generating function whose derivatives give us moments of any order for hitting time distribution.

Previously, we have obtained a recursive equation for $P(N_t^{(i)} = n), n \geq 2$, which is the probability for a random walker to hit target node t starting from node s after exactly n steps. By introducing the notation $\boldsymbol{X}_n^{(i)} = P(N_t^{(i)} = n)$, we can rewrite Eq. [3] into matrix form as

$$\boldsymbol{X}_n = B\boldsymbol{X}_{n-1}, n \geq 2 \tag{6}$$

An iterative solution to Eq. [6] is

$$\boldsymbol{X}_n = B^{n-1}\boldsymbol{X}_1, n \geq 1 \tag{7}$$

The initial probability vector $\boldsymbol{X}_1$ can be conveniently obtained by the observation that $X_1^{(i)} = 0$ if the target node t is not a neighbor of node i, and that $X_1^{(i)} = w_{it}/w_{ii'}, i' \in \{neighbors\ of\ i\}$ if the target node t is a neighbor of node i. Now that we have known matrix B and the initial probability vector $\boldsymbol{X}_1$, we can calculate all the hitting probabilities for any valid starting node.

Although we can calculate all the hitting probabilities, most of the time, we are more interested in the observable quantities associated with these probability distributions. We can calculate the moments of the probability distribution by invoking their definitions, which are

$$\langle N_t^{(i)m} \rangle = \sum_{n=1}^{\infty} P(N_t^{(i)} = n) n^m := \sum_{n=1}^{\infty} X_n^{(i)} n^m \tag{8}$$

The expectation and variance of the first hitting time starting from any node i can be easily calculated from the first and second moments of the hitting probability distribution. At first sight, it seems that we need to know all the hitting probabilities before we can calculate their moments. However, we can exploit the fact the spectral radius of matrix B is less than 1 and directly calculate the moments vector $\langle \boldsymbol{N}_t^m \rangle$ from the recursive Eq. [6]. To accomplish this, we need to first define the characteristic function for the probability density function $f(x)$ as

$$\hat{f}(\omega) = \int_{x \in \mathbb{R}} f(x) e^{\mathrm{i}\omega x} dx \tag{9}$$

For a discrete series like $X_n^{(i)}$, the probability density function is

$$f^{(i)}(x) = \sum_{n=1}^{\infty} X_n^{(i)} \delta(x - n), \tag{10}$$

where $\delta(x-n)$ is the Dirac δ function with the property that for any continuous function $f(x)$, we always have

$$\int_{x \in \mathbb{R}} f(x)\delta(x - x_0)dx = f(x_0). \tag{11}$$

The characteristic function of $f^{(i)}(x)$ is

$$\begin{aligned} \hat{f}^{(i)}(\omega) &= \int_{x \in \mathbb{R}} f^{(i)}(x) e^{\mathrm{i}\omega x} dx \\ &= \sum_{n=1}^{\infty} X_n^{(i)} e^{\mathrm{i}\omega n} \end{aligned} \tag{12}$$

If we further define $z = e^{\mathrm{i}\omega}$, then the characteristic function can be more compactly rewritten as

$$\tilde{f}^{(i)}(z) = \sum_{n=1}^{\infty} X_n^{(i)} z^n \tag{13}$$

We can read off the expectation value and variance of hitting probabilities $X_n^{(i)}$ from the first and second derivatives of $\tilde{f}(z)$ as

$$\begin{aligned} &\langle N_t^{(i)} \rangle = \frac{d}{dz}\Big(\tilde{f}^{(i)}(z)\Big)\Big|_{z=1} \\ &\langle N_t^{(i)2} \rangle = \frac{d^2}{dz^2}\Big(\tilde{f}^{(i)}(z)\Big)\Big|_{z=1} + \frac{d}{dz}\Big(\tilde{f}^{(i)}(z)\Big)\Big|_{z=1} \\ &\mathrm{Var}(N_t^{(i)}) = \langle N_t^{(i)2} \rangle - \langle N_t^{(i)} \rangle^2 \end{aligned} \tag{14}$$

If we consider $N_t^{(i)}$ as the ith component of $\boldsymbol{N}_t$, and $N_t^{(i)2}$ as the ith component of $\boldsymbol{N}_t^2$ which is a component-wise square of vector $\boldsymbol{N}_t$, then the above three relations can be simplified into the form

$$\langle \boldsymbol{N}_t \rangle = \tilde{\boldsymbol{f}}'(1) \tag{15}$$

$$\langle \boldsymbol{N}_t^2 \rangle = \tilde{\boldsymbol{f}}''(1) + \tilde{\boldsymbol{f}}'(1) \tag{16}$$

$$\mathrm{Var}(\boldsymbol{N}_t) = \langle \boldsymbol{N}_t^2 \rangle - \langle \boldsymbol{N}_t \rangle^2 \tag{17}$$

Here, we have defined a vector function $\tilde{\boldsymbol{f}}(z)$ as

$$\tilde{\boldsymbol{f}}(z) = \sum_{n=1}^{\infty} \boldsymbol{X}_n z^n \tag{18}$$

Since the coefficients of $\tilde{f}^{(i)}(z)$ are the hitting probabilities $X_n^{(i)}$, we call it the *generating function* of hitting probabilities. Plug Eq. [6] into the above definition, and we get

$$\begin{aligned} \tilde{\boldsymbol{f}}(z) &= \Big(\sum_{n=1}^{\infty} z^n B^{n-1}\Big)\boldsymbol{X}_1 \\ &= z(I - zB)^{-1}\boldsymbol{X}_1 \end{aligned} \tag{19}$$

The second line of the above equation stems from the fact that $|z| = 1$ and that the spectral radius of matrix B is smaller than 1.

We have already known how to calculate the probability transition matrix B and the initial probability vector $\boldsymbol{X}_1$, we can in principle calculate exactly the generating function from Eq. [19]. However, calculating the inverse of matrix $I - zB$ is no easy task, especially when the graph is huge. Moreover, the fact that the sparsity of

matrix B which we should take full advantage of can get lost after matrix inversion compels us to shun the idea of directly inverting matrix $I - zB$ to calculate hitting probability moments. Therefore, we have devised a trick for finding probability moments without resort to matrix inversion. For this purpose, we rewrite Eq. [19] as

$$(I - zB)\tilde{\boldsymbol{f}}(z) = z\boldsymbol{X}_1 \tag{20}$$

Performing first order derivative of both sides with respect to z and setting $z = 1$ yields

$$-B\tilde{\boldsymbol{f}}(1) + (I - B)\tilde{\boldsymbol{f}}'(1) = \boldsymbol{X}_1 \tag{21}$$

By definition, $\tilde{\boldsymbol{f}}(1) = \begin{pmatrix}1 & 1 & 1... & 1\end{pmatrix}^{\mathrm{T}}$, and $\tilde{\boldsymbol{f}}'(1)$ gives us the first order moment of $\boldsymbol{X}_n$, which is equal to

$$\begin{aligned}\tilde{\boldsymbol{f}}'(1) &= (I - B)^{-1}(B\tilde{\boldsymbol{f}}(1) + \boldsymbol{X}_1) \\ &= (I - B)^{-1}\tilde{\boldsymbol{f}}(1)\end{aligned} \tag{22}$$

The second line of the above equation is due to the identity that $\tilde{\boldsymbol{f}}(1) = B\tilde{\boldsymbol{f}}(1) + \boldsymbol{X}_1$, which can be easily verified by plugging $z = 1$ into Eq. [19]. We can avoid inverting sparse matrices, an operation that will destroy the sparsity of a matrix, by noting that Eq. [22] can be rewritten as (remember that the spectral radius of B is smaller than 1)

$$\tilde{\boldsymbol{f}}'(1) = \sum_{n=0}^{\infty} B^n \tilde{\boldsymbol{f}}(1) \tag{23}$$

It is noteworthy that the first order moments of hitting probabilities starting from each valid vertex are independent of the initial probability vector $\boldsymbol{X}_1$, and depend only on the probability transition matrix B. Taking the second order derivative of Eq. [20] yields

$$\begin{aligned}\tilde{\boldsymbol{f}}''(1) &= 2B(I - B)^{-2}\tilde{\boldsymbol{f}}(1) \\ &= 2\sum_{n=1}^{\infty} nB^n \tilde{\boldsymbol{f}}(1)\end{aligned} \tag{24}$$

The pseudocode for calculating mean and variance of hitting times is shown below:

Algorithm 2 Hitting time calculation algorithm

Require: probability transition matrix B must be square
Require: max iteration number N must be positive
Require: error limit ϵ must be positive

procedure HittingTimeCalculator(B, N, ϵ)
  $i \leftarrow 0$
  $d \leftarrow B.dimension$ ▷ Dimension of matrix B
  ones $\leftarrow$ *vector of all 1's, shape = (d, 1)*
  zeros $\leftarrow$ *vector of all 0's, shape = (d, 1)*
  power $\leftarrow$ *ones*
  $\mu \leftarrow$ *ones* ▷ μ: expectation of hitting times
  var $\leftarrow$ *zeros* ▷ *var*: variance of hitting times
  while $i \leq N$ **do**
    $i \leftarrow i + 1$
    power $\leftarrow B *$ *power* ▷ Matrix multiplication
    $\mu \leftarrow \mu +$ *power*
    var $\leftarrow$ *var* $+ i *$ *power*
    error $\leftarrow$ norm of $i *$ *power*
    if *error* $< \epsilon$ **then**
      break
    end if
  end while
  var $\leftarrow 2 *$ *var*
  var $\leftarrow$ *var* $+ \mu - \mu^2$(element wise square)
  return μ, *var*
end procedure

Higher order derivatives of Eq. [20] yield higher order moments. Now we have already developed the algorithm for calculating the moments of hitting probabilities using both analytical and numerical methods, next we will illustrate the effectiveness of this algorithm using both small and huge graphs.

The significance of the first order moment lies in that it is a measure of the distance from a starting node to a target node. If we already know that the target node is a fraudulent user in the social network, then we can infer that the nodes with average distance smaller than a threshold value could be considered to be potential fraudulent users. From intuition, we make a claim that the smaller the distance between any two nodes, the more similar they are to each other.

V. EXPERIMENTAL EVIDENCE

A. An analytical calculation of hitting time distribution on a simple graph

In this section, we will show how to calculate the hitting time distribution on a small graph using analytical methods. This graph contains five nodes, which are denoted as $0, 1, 2, 3, 4$, as shown in Fig. [1].

The adjacency matrix of this graph is

$$A = \begin{pmatrix}0 & 1 & 1 & 0 & 0\\ 1 & 0 & 1 & 0 & 0\\ 1 & 1 & 0 & 1 & 0\\ 0 & 0 & 1 & 0 & 1\\ 0 & 0 & 0 & 1 & 0\end{pmatrix} \tag{25}$$

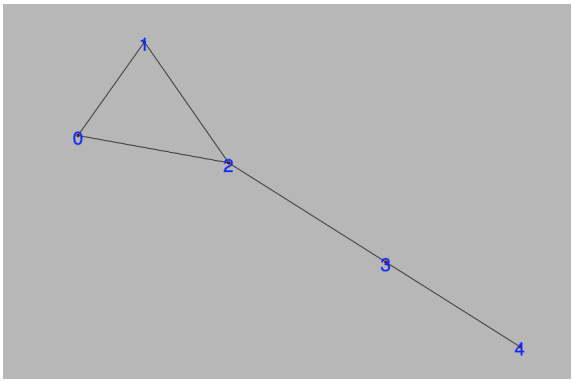

FIG. 1. An undirected graph that is small enough to be solved using analytical formulae. We use node 3 as our target node.

Our target node is 3, and we want to calculate the probability of hitting the target starting from each vertex for the first time after exactly n steps. Since node 4 is an adherent to target 3, we can directly write out its hitting probability as

$$P(N_3^{(4)} = n) = \delta_{n,1} \tag{26}$$

For nodes 0, 1, 2, we can define the probabilities of starting from each node and ending at node 3 after exactly n steps. We encapsulate these probabilities into a column vector as

$$\boldsymbol{X}_n = \begin{pmatrix} P(N_3^{(0)} = n) \\ P(N_3^{(1)} = n) \\ P(N_3^{(2)} = n) \end{pmatrix} \tag{27}$$

The probability transition matrix is

$$B = \begin{pmatrix} 0 & 1/2 & 1/2 \\ 1/2 & 0 & 1/2 \\ 1/3 & 1/3 & 0 \end{pmatrix} \tag{28}$$

The probability vector satisfies this equation

$$\boldsymbol{X}_n = B\boldsymbol{X}_{n-1}, n \geq 2, \tag{29}$$

with initial condition

$$\boldsymbol{X}_1 = \begin{pmatrix} 0 \\ 0 \\ 1/3 \end{pmatrix} \tag{30}$$

The moment generating function for $\boldsymbol{X}_n$ is

$$\begin{aligned} \tilde{\boldsymbol{f}}(z) &= z(I - zB)^{-1}\boldsymbol{X}_1 \\ &= \frac{z}{3}\frac{1}{1 - \frac{7}{12}z^2 - \frac{z^3}{6}} \begin{pmatrix} \frac{z}{2} + \frac{z^2}{4} \\ \frac{z}{2} + \frac{z^2}{4} \\ 1 - \frac{z^2}{4} \end{pmatrix} \end{aligned} \tag{31}$$

We can easily get the first order moments by differentiating the above equation with respect to z at $z = 1$, the results of which are

$$\tilde{\boldsymbol{f}}'(1) = \begin{pmatrix} \langle N_3^{(0)} \rangle \\ \langle N_3^{(1)} \rangle \\ \langle N_3^{(2)} \rangle \end{pmatrix} = \begin{pmatrix} 9 \\ 9 \\ 7 \end{pmatrix} \tag{32}$$

The above result means that the average distance from nodes 0 and 1 to node 3 are equal, both being 9, and the average distance from node 2 to node 3 is 7. We interpret these results as demonstrating that nodes 0 and 1 have equal distance to node 3, whereas node 2 is nearer to node 3 than both 1 and 2. Node 4, being an adherent to target node 3, always has an average distance 1 to the target. These results are consistent with our intuition and signify to us that node 4 is most susceptible to the influence of node 3, node 2 is second most susceptible, and nodes 0 and 1 are least susceptible to its influence.

In order to make a further test of the analytical results, we also perform a Monte Carlo simulation for the random walk on this graph. In the Monte Carlo program, we use node 3 as our target node, and start each random walk from node 0, 1, and 2. Each random walk terminates at node 3 after some number of steps. By repeating this process thousands of times, we obtain the average number of steps required before the random walker finally reaches the target. For each node in the set $\{0, 1, 2\}$, we perform 10^6 random walks, each of which yields a step number, and then we calculate the mean value of these 10^6 numbers. The Monte Carlo simulation results we get are pretty similar to the analytical results, which are shown together in Table I. We can see that the Monte Carlo simulation results are consistent with our analytical results, with relative errors being approximately 10^{-3}. We do not expect Monte Carlo simulation to give us high precision numerical results, and the final results of Monte Carlo simulations may vary slightly for different random number generators.

Using the algorithm outlined in the previous section, we can equally well compute the average step number starting from each node 0, 1, 2 using numerical methods. We can use either Eq. [22] or Eq. [23] for this purpose, because the probability transition matrix is small enough for the direct inversion of matrix to be feasible. However, Eq. [22] is no longer practical when the graph is large, and thus even for this small graph, we still prefer to use Eq. [23], where we need to calculate the sum of an infinite power series. Due to the quick convergence of this series, we artificially impose a cutoff condition such that the summation series should terminate if the norm of the summand vector is smaller than a pre-specified error limit ϵ, which we choose to be 10^{-13} here. We run the Python program listed in the previous section on macOS Mojave, and obtain numerical results that are shown together with analytical results and Mont Carlo simulation results in Table I. It is clear that the numerical results obtained using our algorithm have a much higher precision than that of the Monte Carlo simulation results.

$\langle N_3^{(i)} \rangle$	Analytical	Monte Carlo	Numerical
$\langle N_3^{(0)} \rangle$	9	9.00323	8.99999999999999
$\langle N_3^{(1)} \rangle$	9	8.96693	8.99999999999999
$\langle N_3^{(2)} \rangle$	7	7.00000000000002	7.000000000000002

TABLE I. Random walk results for the graph shown in Fig. 1. $\langle N_t^{(i)} \rangle$ is the expected hitting time for a random walk that starts from node i and ends at target node t. Here, $t = 3$. The above table lists the expected hitting times for random walkers to first reach target node 3 starting from nodes 0, 1, and 2, respectively. We obtain these results using three methods: analytical, Monte Carlo simulation, and numerical. Both Monte Carlo simulation results and numerical computation results are consistent with analytical results, although the numerical results have much higher precision, which justifies our introduction of the numerical algorithm for attacking this problem.

B. Numerical computation of hitting time distribution on large graphs

When dealing with large graphs, it is both tedious and impractical to get an analytical formula as Eq. [31]. Instead, we will resort to Eq. [23] to find numerical values of hitting times. Another method to find the hitting times is to use Monte Carlo simulation, although we will see that for a large graph, the running time of Monte Carlo simulation is much longer than the numerical method, thus making the Monte Carlo simulation an inferior alternative compared to Eq. [23]. In this section, we apply the numerical method and Monte Carlo simulation method to a connected graph as shown in Fig. [2].

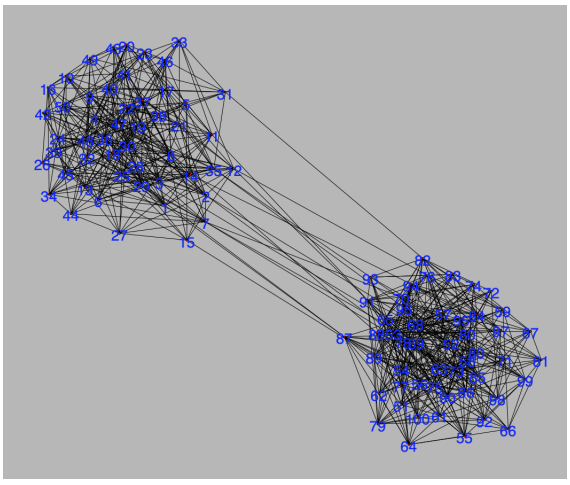

FIG. 2. An undirected graph with 100 vertices and 740 edges. This graph contains two communities, and is generated by the rule that each pair of vertices within the same community is connected by an edge with probability $p_{\text{in}} = 0.3$, whereas each pair of vertices from different communities is connected by an edge with probability $p_{\text{out}} = 0.02$.

We will calculate the hitting times from each vertex in the graph to the target node which is chosen to be node 1. In order to visualize the results, we sort the vertices in the graph according to their hitting times to the target node. In Fig. [3], we plot the results from Monte Carlo simulation method and numerical method. We can see from the figure that these two methods give almost the same results, although we know that results from Monte Carlo simulation have a precision that is much lower that obtained from numerical method. Another weak point of Monte Carlo simulation is that it is much more time-consuming than the numerical method. In order to obtain the results shown in Fig. [3], we need to run 10^5 random walks from each vertex in the graph except the target node, and the whole process takes about 157 seconds, whereas in the numerical method, we only need to compute the power series in Eq. [23] up to 6180 terms, and it takes only 3.06 seconds to obtain results with machine precision. Actually, the larger the graph, the more time-saving the numerical method is compared to the Monte Carlo simulation method.

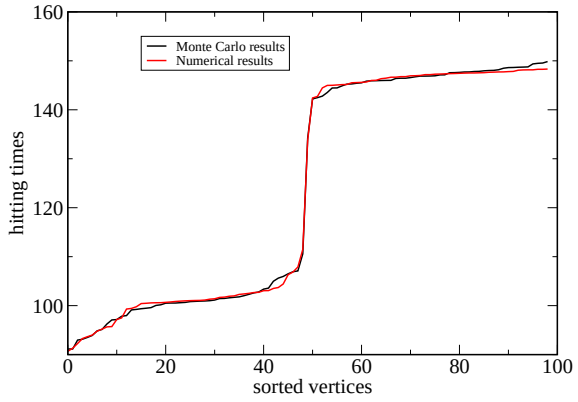

FIG. 3. An undirected graph with 100 edges and 740 edges. This graph contains two communities, and is generated by the rule that each pair of vertices within the same community is connected by an edge with probability $p_{\text{in}} = 0.3$, whereas each pair of vertices from different communities is connected by an edge with probability $p_{\text{out}} = 0.02$.

Another feature that is worth mentioning in Fig. [3] is that we can distinguish the two communities in the original graph by looking at the distribution of hitting times with respect to vertices. It is clear that there is a transition region which connects two plateaus in the hitting time vs. sorted vertices curve. The two plateaus correspond to the two communities in the graph. We can interpret the emergence of these two plateaus by noting that a random walker that starts from within one of the two communities tend to get trapped in a community. Once the random walker gets trapped in a community, the hitting times for each vertex in the community would not change substantially from vertex to vertex, which gives rise to a plateau in the curve. However, as soon as

the random walker finds a bridge leading from one community to another, it will make a rapid transition across the two communities, and consequently the hitting times experience a significant change. Thus, the calculation of hitting times provides us a tool for community detection as a by-product. However, we should make a caveat that this method of community detection is usable only when the number of communities in the graph is small enough and the communities are clearly separated from each other. Or else, this method of community detection is not as good as the ones compiled in Ref. [5].

C. Directional distances and non-reciprocal relationships

The distance between a pair of nodes can be thought of as a function that maps two nodes to a non-negative real number whose value represents the distance between these two nodes, i.e., $d : (u, v) \mapsto \mathbb{R}^+$, where u, v are two nodes in a graph. Previously, people define the distance function between a pair of nodes with the implicit assumption that this function should remain invariant with the exchange of its two arguments, which means $d(u, v) = d(v, u)$. This property, which we call the symmetricity of the distance function, is however undesirable for our situation where we want to study the influence of a node A upon another node B. It is a truth universally acknowledged that not all users in a social network are equally influential, and thus we do not expect the mutual influence between a pair of nodes to be the same regardless of the direction. Based on this consideration, we dictate that the distance function for our case should be directional even if we restrict our attention to undirected graphs. Take the undirected graph shown in Fig. 4 as a concrete example. In this graph, where all edges are assumed to be have equal weight, we do not expect the distance from node 10 to node 11 to be identical to the distance from node 11 to node 10. It is obvious from the figure that node 10, which belongs to a clique that consists of nodes $1-10$, should have a strong association with nodes $1-9$ and a ridiculously tenuous association with node 11, although node 11 is a direct neighbor of node 10. On the other hand, node 11, which has only three direct neighbors, would have an appreciably strong association with node 10. Therefore, we claim based on this graph that node 10 would exert a stronger influence on node 11 than the influence that node 11 would exert on node 10. In other words, the affection from node 11 to node 10 tend not to be reciprocated. Our assertion on the non-reciprocal relationship between node 10 and node 11 is buttressed by numerical results according to which the distance from node 10 to 11 is 91 whereas the distance from node 11 to 10 is 7, indicating that node 11 is more loyal to 10 than node 10 is to loyal to 11. Due to the fact that in most real world examples the relationships and influences are generally non-reciprocal and non-equivalent, we claim that our directional distance function is more suitable for describing the influence that one node exerts upon another one than other symmetric distance functions .

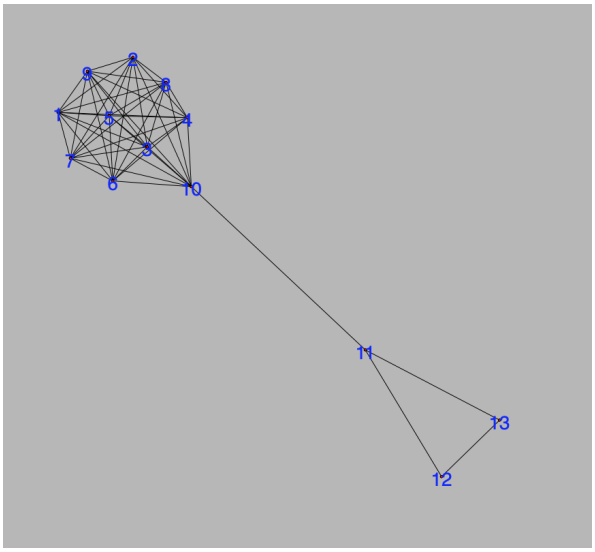

FIG. 4. A figure that illustrates the directedness or asymmetricity of the distance function between a pair of nodes. In this figure, we do not expect the distance from node 10 to 11 to be identical to the distance in the reverse direction due to the fact that node 10 is more closely associated with nodes $1-9$ than with node 11, whereas node 11, being less social, has a pretty strong association with node 10. Here, we assume all the edges to have equal weight.

VI. COMPARISON WITH EXISTING METHODS

VII. CONCLUSION AND FUTURE WORK

In this paper, we have derived an analytical formula for calculating the hitting time from a starting vertex to a target vertex in a connected undirected graph. This method relies on the probability transition matrix that can be calculated conveniently from the graph's adjacency matrix. We also propose a quick method for implementing this formula using Python code, without the need to invert a possibly huge sparse matrix. Since hitting time is a core concept in random walk which can directly be simulated using Monte Carlo method, we can obtain an approximate value of the hitting time using Monte Carlo simulation. We tested our formula by applying the analytical method and Monte Carlo simulations to undirected connected graphs, and show that these two methods can give similar results within tolerance of error. The advantage of the analytical method over Monte Carlo simulation is that the former is much quicker and more accurate than the latter. Our calculation of the hitting times for vertices in a graph can also give us a glimpse of the community structures of the graph on

which we perform random walks, although this method for detecting communities is not as good as other existing algorithms when the communities in graph are numerous and not clearly separated.

ACKNOWLEDGMENTS

[1] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[2] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70:066111, Dec 2004.

[3] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006.

[4] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E*, 74:036104, Sep 2006.

[5] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

[6] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[7] Sheldon M Ross, John J Kelly, Roger J Sullivan, William James Perry, Donald Mercer, Ruth M Davis, Thomas Dell Washburn, Earl V Sager, Joseph B Boyce, and Vincent L Bristow. *Stochastic processes*, volume 2. Wiley New York, 1996.

[8] Bart Baesens, Veronique Van Vlasselaer, and Wouter Verbeke. *Fraud analytics using descriptive, predictive, and social network techniques: a guide to data science for fraud detection*. John Wiley & Sons, 2015.

[9] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[10] Peter G Doyle and J Laurie Snell. Random walks and electric networks. *arXiv preprint math/0001057*, 2000.

[11] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pages 912–919, 2003.