Computation of Node Distances on Hypergraphs

Enzhi Li
Amazon
San Diego, CA, USA
lienzhi@amazon.com

Bilal Fadlallah
Amazon
Seattle, WA, USA
bhf@amazon.com

ABSTRACT

A hypergraph is a generalization of a graph that arises naturally when we consider attribute-sharing among entities. Although a hypergraph can be converted into a graph by expanding its hyperedges into fully connected subgraphs, going the reverse way is computationally complex and NP-complete. We hence hypothesize that a hypergraph contains more information than a graph. Moreover, it is more convenient to manipulate a hypergraph directly, rather than expanding it into a graph. An open problem in hypergraphs is how to accurately and efficiently calculate their node distances. Once node distances are defined, we can find a node's nearest neighbors, and perform label propagation on hypergraphs using a K-nearest neighbors (KNN) approach. In this paper, we propose two methods to achieve this. In the first, we compute expected hitting times of random walks on a hypergraph as node distances; in the second, we generalize the DeepWalk method to hypergraphs. We note that simple random walks (SRW) cannot accurately describe highly complex real-world hypergraphs, which motivates us to introduce frustrated random walks (FRW) to better describe them. Using real-world datasets, we show that FRW and DeepWalk can beat SRW with a large margin. For large and sparse hypergraphs, our method for computing the expected hitting times of random walks is approximately linear in time complexity, rendering it superior to the DeepWalk method.

ACM Reference Format:
Enzhi Li and Bilal Fadlallah. 2023. Computation of Node Distances on Hypergraphs. In *Proceedings of XXX (Conference acronym 'XX)*. ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

A graph is a useful data structure for describing complex relations in real world. In a graph, nodes are connected by edges, and each edge contains exactly two nodes. A hypergraph is an extension of ordinary graphs where each hyperedge of a hypergraph may contain an arbitrary number of nodes (see Fig. 4 in appendix for an illustration). In the special case where each hyperedge contains just two nodes, the hypergraph reduces to a graph. Each hyperedge of a hypergraph is considered a clique in a graph. We can therefore convert a hypergraph to a graph by expanding each hyperedge. However, hypergraphs contain more information than what can be encapsulated into its expanded graphs. If we simply expand a hypergraph into a graph, the explicit information of the cliques is lost. Although the conversion from hypergraphs to graphs is reversible in theory, the NP-completeness of detecting maximal cliques in a graphs renders it irreversible in practice. Furthermore, the expansion of hyperedges can be computationally expensive and memory-intensive considering that a hyperedge of N nodes yields $N(N-1)/2$ undirected edges in a graph. It is hence advantageous to directly work with hypergraphs whenever possible rather than operating on their expanded graph derivation.

One of the central problems in graph theory is the quantitative determination of graph node distances. The literature has multiple algorithms that achieve this, such as DeepWalk[1], graphSAGE[2], hitting times of random walks[3] and frustrated random walks[4], among others. We can easily generalize DeepWalk to hypergraphs, yet for the other algorithms, such generalization is highly non-trivial. In this paper, we generalize frustrated random walks to hypergraphs, and show that the approach is on par with the performance of DeepWalk even for complicated hypergraphs. There are four reasons that motivate using frustrated random walks (FRW) as opposed to DeepWalk. First, in applications where the goal is to find the nearest neighbors of a few nodes in a large hypergraph, FRW is a preferable and faster option than DeepWalk. Second, FRW conveniently gives a closed-form and interpretable solution for node distances, a solution that is impossible to obtain using any deep-learning based method. Third, FRW does not require any parameter tuning, whereas DeepWalk requires heavy investment in parameter tuning. Finally, and unlike in DeepWalk, the node distances of FRW are asymmetric, rendering it more suitable to describe real-world relationships which are generally asymmetric and non-reflective.

A hypergraph consists of nodes and hyperedges. Each hyperedge is a subset of the node set $\mathbb{V}$. For clarity purpose, we use in this paper Latin letters to indicate node indices and Greek letters to indicate hyperedge indices. The incidence matrix of a hypergraph is defined as:

$$e_{i\alpha} = \begin{cases} w, \text{if vertex } v_i \in E_\alpha \\ 0, \text{otherwise} \end{cases} \tag{1}$$

In the above definition, w is the weight of vertex v_i in hyperedge E_α. If the hypergraph is unweighted, then $w = 1$ always holds. If we think of v_i as a member of a community E_α, then $e_{i\alpha}$ can be thought of as the loyalty of v_i to E_α. By definition, the degree of a node v_i is $D_i = \sum_\alpha e_{i\alpha}$, and the degree of a hyperedge is $\delta_\alpha = \sum_i e_{i\alpha}$. We can think of δ_α as the adhesiveness of E_α.

Researchers have long used random walks to study hypergraphs. In Ref. [5], the authors generalized spectral clustering [6–8] from graphs to hypergraphs and gave their algorithm a random walk interpretation. It is however noted that the hypergraph random walks defined in Ref. [5] is no different from the random walks performed

Conference acronym 'XX, August 06–10, 2023, Long Beach, CA

ACM ISBN 978-1-4503-XXXX-X/23/08…$15.00
https://doi.org/XXXXXXX.XXXXXXX

on expanded graphs [9, 10]. Since a hyperedge in a hypergraph represents a more adhesive community, it is argued in [11] that a random walker roaming on a hypergraph should show preference towards hyperedges of higher degrees (stronger adhesiveness). Timoteo Carletti *et al.* thus proposed an alternative random walk algorithm that takes the hyperedge degree into account. In this paper, we will generalize the method described in Ref. [11] to weighted hypergraphs, and introduce frustrated random walks on hypergraphs. We will show that for scale-free hypergraphs, frustrated random walks is more suitable for computing node distances.

2 A UNIFIED FRAMEWORK FOR CALCULATING EXPECTED HITTING TIMES OF RANDOM WALKS

With the abundance of research on random walks on hypergraphs, we note that almost all of the previous works have focused on the Laplacian matrix. Laplacian matrices invoke spectral clustering[6, 8] which naturally leads to image segmentation and community detection in hypergraphs. Furthermore, the eigenvectors of a Laplacian matrix yield node embeddings which enable us to apply powerful machine learning algorithms to perform node classification on hypergraphs. Since the study of the Laplacian matrix yields information about the long-term stationary distribution of particles that are randomly walking on hypergraphs, we label this approach of studying random walks the stationary approach. On the other hand, we can also extract much information about hypergraphs by focusing on the diffusion process itself, and we label this way of studying random walks the dynamic approach. In this paper, we will focus on the hitting times of random walks. Given a hypergraph H, we select a node t as target, and perform random walks starting from any other node s. The hitting time for this process is defined as the number of steps a random walker needs to traverse before it hits the target t for the first time. By definition, the hitting time on a given hypergraph is a random variable that depends on the hypergraph structure, the starting node s and target node t, and can thus be denoted as $N_t^{(s)}$. The expectation of $N_t^{(s)}$ provides a natural measurement of the distance from s to t. Note that the distance here is asymmetric, meaning that the distance from s to t is not guaranteed to be identical to that from t to s. Since we want to use the expected hitting time to measure the closeness between two persons in the real world, and we know that the real-world human-human relationships are generally asymmetric, this lack of symmetry is thus a blessing to us.

The dynamics of random walks on hypergraphs are fully captured by the transition matrix, and different definitions of transition matrices produce different results. We calculate hypergraph node distances to find nearest neighbors of a node. To accomplish this, we select a node as the target, and calculate the expected hitting times starting from all other nodes. We then rank all those nodes according to their distances to the target. Intuitively, we expect that the shorter a node's distance to the target, the closer they should be. If we represent real-world human-human interactions using a hypergraph, we could find a person's close friends from among his/her nearest neighbors in the hypergraph.

2.1 Simple random walks on hypergraphs

As already stated, the calculation of hypergraph hitting times requires specification of the random walk algorithm. To capture the preference of a random walker towards a more cohesive hyperedge, we define transition probability from node i to node j as

$$T^S_{ij,i\neq j} = \frac{\sum_\alpha \left(\delta_\alpha - e_{i\alpha}\right) \min\{e_{i\alpha}, e_{j\alpha}\}}{\sum_j \sum_\alpha \left(\delta_\alpha - e_{i\alpha}\right) \min\{e_{i\alpha}, e_{j\alpha}\}} \tag{2}$$

The mechanism behind Eq. (2) is that a random walker standing on site i would randomly select one of its neighbors j towards which it makes a transition. A transition from i to j is possible only when both nodes belong to the same hyperedge. In a hypergraph that represents real-world human-human interactions, the node weight $e_{i\alpha}$ measures the loyalty of the node to the hyperedge which is interpreted as a community. Intuitively, two loyal nodes tend to have high frequency interactions, and two disloyal nodes tend not to interact with each other. The interaction frequency between a loyal node say i and a disloyal node say j depends on how disloyal j is to the community. Thus, when calculating the transition probability from i to j, we demand that the probability should be proportional to the lesser of the two node weights, i.e., $\min\{e_{i\alpha}, e_{j\alpha}\}$, rather than their product $e_{i\alpha}e_{j\alpha}$. The proportionality coefficient $\delta_\alpha - e_{i\alpha}$ in front of $\min\{e_{i\alpha}, e_{j\alpha}\}$ reflects the tendency of a node to stay within a more cohesive community (a hyperedge of higher degree). The denominator in Eq. (2) is the normalization constant.

2.2 Frustrated random walks on hypergraphs

Although simple random walks as described in the previous subsection is a good algorithm for exploring hypergraph properties and can be used to generate random paths for DeepWalk, we find that the hitting times derived from this algorithm is not a good measure of node distances when the hypergraph is heavily weighted and scale-free[12]. The fact that most real-world human-human interactions can be recast into a scale-free network compels us to develop an alternative random walk scenario which we call the *frustrated random walks* algorithm[4], an algorithm that takes into account how people interact with each other in the real world. Real world networks are mostly dominated by the Mathew effect. For example, in a social network, the more followers you have, the easier it is for you to attract new followers, a phenomenon that is called preferential attachment[12]. The advantage of frustrated random walks over simple random walks is that it can yield node distances that are consistent with human judgment even for heavily weighted, scale-free networks. For networks that are lightly weighted or non-scale-free, the results from frustrated and simple random walks are similar to each other.

In order to measure node distances on hypergraphs, we set one node (call it t) as the target, and perform random walks starting from another node (call it s). The expected hitting time of this random walk process is interpreted as the distance from s to t. Intuitively, the shorter the distance, the closer the relationship between s and t. We can think of this random walk process as the donation of a gift. If one person has one gift to donate, she tends to donate it to her close friend. Similarly, when a person is given more than one gift, she also tends to accept her close friend's. With this observation,

we decompose the gift-donating process into two parts: one for proposing the donation, and another for accepting the donation. By setting an acceptance threshold for the gift donation, we have *frustrated* the gift donation process. In this scenario, a transition is possible only when a proposal is made and then accepted. Therefore, in frustrated random walks, the transition probability is the product of the proposal probability and the acceptance probability, which is

$$T^F_{ij,i\neq j} = \frac{\sum_\alpha \left(\delta_\alpha - e_{i\alpha}\right) \min\{e_{i\alpha}, e_{j\alpha}\}}{\sum_j \sum_\alpha \left(\delta_\alpha - e_{i\alpha}\right) \min\{e_{i\alpha}, e_{j\alpha}\}} \times \frac{\sum_\beta \left(\delta_\beta - e_{j\beta}\right) \min\{e_{j\beta}, e_{i\beta}\}}{\sum_k \sum_\beta \left(\delta_\beta - e_{j\beta}\right) \min\{e_{j\beta}, e_{k\beta}\}} \tag{3}$$

Another major difference between simple random walks and frustrated random walks is that in the former case, the proposal to make a transition is always accepted and thus $T^S_{ii} = 0, \forall i \in \mathbb{V}$, whereas in the latter case, there is a non-zero probability for the proposal to be declined, and thus T^F_{ii} is generally non-zero. The diagonal values of T^F matrix can be easily calculated from the observation that $\sum_j T^F_{ij} = 1, \forall i, j \in \mathbb{V}$, from which we have

$$T^F_{ii} = 1 - \sum_{j\neq i} T^F_{ij}, \forall i \in \mathbb{V}. \tag{4}$$

All of these differences combined impact significantly on the calculation of average hitting times and ultimately the ranking of nodes with respect to a target node. We will show later, using real-world data, that frustrated random walks can better capture human-human interactions in the real world than simple random walks.

2.3 A Unified framework for calculating expected hitting times

In the previous subsections, we outline simple random walks and frustrated random walks on hypergraphs, and give formulae for calculating their corresponding transition probabilities. In this section, we will continue to calculate their average hitting times from the transition probabilities. It will be seen that although the transition probabilities differ from each other, the calculation of expected hitting times can be formulated in a unified framework.

For a connected hypergraph H, denote $N^{(s)}_t$ as the hitting time of a random walk process that starts from node s, with node t as the target. We use $P(N^{(s)}_t = n)$ to denote the probability of reaching target node t from node s after exactly n steps. Since the problem is trivial for the case when $s = t$, here we demand that the starting node s should differ from the target node t. Due to the Markov property of the random walk process, whether it be simple or frustrated, we can establish a recurrence equation for the probability, which is

$$P(N^{(s)}_t = n) = \sum_{i_s\neq t} T_{s,i_s} P(N^{(i_s)}_t = n-1), n \geq 2 \tag{5}$$

A physical interpretation of the above equation is that we decompose the the random walk process into two steps. First, we make a transition from node s to one of its neighbors i_s; second, due to the Markov property of the random walks, the whole process restarts all over again with i_s as the new starting node, with the condition that now the random walker needs to reach the target with exactly $n-1$ steps since one step has already been taken. Thus, Eq. (5) is a recurrence equation for the probabilities $P(N^{(s)}_t = n)$. In this equation, we have imposed the condition that $n \geq 2$ because $P(N^{(j)}_t = 0) = 0$ for any node $j \neq t$, and the recursion process terminates immediately when the random walker hits the target. From the transition matrix, we can readily read out the probability $P(N^{(s)}_t = 1)$, which is the initial condition for the recurrence relation in Eq. (5).

Eq. (5) is essentially a system of first order difference equations with constant coefficients. For simplicity of notation, we can understand $N^{(i)}_t$ as the ith component of a column vector $\boldsymbol{N}_t$, and $P(N^{(i)}_t = n)$ as the ith component of a column vector $\boldsymbol{X}_n = P(\boldsymbol{N}_t = n)$. With these symbols, we can rewrite Eq. (5) in an abstract form as

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

Here, B is the coefficient matrix with elements depending on the transition probabilities and the target node t.

2.3.1 Calculation of B matrix for simple random walks. For simple random walks, $B_{ij} = T^S_{ij}, \forall i, j \in \mathbb{V}-\{t\}$, and $B_{it} = 0, \forall i \in \mathbb{V}-\{t\}$ by definition of Eq. (5). Still from Eq. (5), we have that the left subscript of B, which is the index of the starting node for the random walk, can never be equal to t. In contrast with the transition matrix T^S, matrix B is not a Markov matrix because the summation of each row is not always equal to one. This non-Markov property of B originates from the existence of the target node t. The rule is that if t is not a neighbor of node i, then $\sum_j B_{ij} = 1$, or else $\sum_j B_{ij} = 1 - T^S_{it}$. Because of this, the spectral radius of B is always smaller than one.

We have established Eq. (5) to calculate the probability distribution of hitting times using any non-target node s as our starting node. However, if the starting node s is pretty special in that it has only one single neighbor, and this very neighbor is precisely the target node t, then no recurrence equation is needed since we already have $P(N^{(s)}_t = n) = \delta_{n,1}, \forall n \geq 1$. We call such a node an *adherent* node of target node t, and exclude these nodes from the set of starting nodes (For example, node 4 in Fig. 4 is an adherent to target node 3.).

Definition 1. *In random walks, a node is called adherent to the target if the node has the target as its only neighbor.*

Since we need to exclude target node t and its adherent nodes from the starting node set, the dimension of matrix B is

$$N_B = |\mathbb{V} - \{t\}| - |\{v_i : T^S_{it} = 1\}|, \tag{7}$$

where $|\mathbb{S}|$ indicates the cardinality of a set $\mathbb{S}$. Obviously, for simple random walks, the hitting time from an adherent to the target is always equal to one.

2.3.2 Calculation of B matrix for frustrated random walks. The calculation of B matrix elements for frustrated random walks is similar to that for simple random walks. Here, we still have $B_{ij} = T^F_{ij}, \forall i, j \in \mathbb{V}-\{t\}$ and $B_{it} = 0, \forall i \in \mathbb{V}-\{t\}$. The existence of target node t still renders B a non-Markov matrix in that $\sum_j B_{ij} = 1$ if

i and t do not lie simultaneously in any hyperedge, and $\sum_j B_{ij} = 1 - T_{it}^F$ otherwise. The only difference is that due to the imposition of the acceptance threshold for the frustrated random walks, the transition probability from an *adherent* to the target no longer obeys the sharp δ distribution, but a geometric distribution, i.e., $P(N_t^{(adherent)} = n) = (1-p)^{n-1}p$, where p is the probability of t accepting the transition from the adherent. Consequently, the expected hitting time from the adherent to the target is $1/p$.

2.3.3 Numerical computation of expected hitting times. Now that we have calculated the elements of B matrix, we can continue to solve Eq. (6) as

$$\boldsymbol{X}_n = B^{n-1}\boldsymbol{X}_1, n \geq 1. \tag{8}$$

Here, $\boldsymbol{X}_1$ is the probability of hitting the target in the first step, a probability that can be conveniently read out from the transition matrix. By definition, the expected hitting times from an arbitrary node to the target node is

$$\mathbb{E}\boldsymbol{N}_t = \sum_{n=1}^{\infty} nP(\boldsymbol{N}_t = n) := \sum_{n=1}^{\infty} n\boldsymbol{X}_n \tag{9}$$

Plugging Eq. (8) into the above definition yields an expression for the expected hitting times that depends both on the initial condition $\boldsymbol{X}_1$ and the coefficient matrix B, that is,

$$\mathbb{E}\boldsymbol{N}_t = \sum_{n=1}^{\infty} nB^{n-1}\boldsymbol{X}_1 \tag{10}$$

The above equation involves an infinite summation and the powers of the B matrix, and is thus difficult to compute numerically. However, we can devise an ingenious method to convert the above equation into an equivalent linear system of equations which we can efficiently solve using highly optimized linear algebra packages. To do this, we first define the probability generating function, which is

$$\boldsymbol{f}(z) = \sum_{n=1}^{\infty} \boldsymbol{X}_n z^n, |z| \leq 1. \tag{11}$$

It is easy to show that the convergence radius of the above power series is larger than 1. More precisely, we can pinpoint the convergence radius of the series in Eq. (11) at $R = 1/\rho(B)$, where $\rho(B)$ is the spectral radius of the coefficient matrix B. $\rho(B)$ is guaranteed to be smaller than one by the Gershgorin circle theorem[13]. With the help of Eq. (8), the probability generating function can be evaluated exactly as

$$\begin{aligned}\boldsymbol{f}(z) &= \sum_{n=1}^{\infty} B^{n-1}\boldsymbol{X}_1 z^n \\ &= z(I - zB)^{-1}\boldsymbol{X}_1, |z| \leq 1.\end{aligned} \tag{12}$$

Here, the summation can be evaluated in closed form because the convergence radius of the power series is larger than one. Still due to the fact that convergence radius is larger than 1, we can differentiate the infinite series in Eq. (11) term by term at $z = 1$ and obtain an equivalent expression for the expected hitting times as

$$\begin{aligned}\boldsymbol{f}'(z=1) &= \sum_{n=1}^{\infty} nB^{n-1}\boldsymbol{X}_1 z^{n-1}\Big|_{z=1} \\ &= \mathbb{E}\boldsymbol{N}_t\end{aligned} \tag{13}$$

Therefore, once we have the probability generating function, we can easily calculate the expected hitting times. An example of calculating expected hitting times using this method is given in Appendix. However, when the dimension of B is huge, analytical evaluation of Eq. (11) is computationally prohibitive, which compels us to resort to numerical methods. Multiplying both sides of Eq. (12) with $I - zB$ and taking first order derivative with respect to z at $z = 1$ yields

$$(I - B)\boldsymbol{f}'(z=1) = \boldsymbol{f}(z=1) \tag{14}$$

By definition, $\boldsymbol{f}(z=1)$, which is the sum of all probabilities, is a column vector with each element being 1. Therefore, to calculate expected hitting times $\mathbb{E}\boldsymbol{N}_t = \boldsymbol{f}'(z=1)$, we only need to solve the linear equation (14), which is easy to do numerically. It is noteworthy that in Eq. (14), the coefficient matrix B depends both on the hypergraph structure and the target node t, and thus target information is implicit in that equation. For real-world hypergraph data, the coefficient matrix in Eq. (14) is generally large and sparse. The best method to numerically solve a large and sparse system is the conjugate gradient method[14] or its variants, which we will use throughout our paper.

Our method for calculating hypergraph node distances can be summarized in Algorithm 1. For sake of simplicity, we only show how to calculate expected hitting times of simple random walks. The extension of this algorithm to frustrated random walks is straightforward.

Algorithm 1 CalculateExpectedHittingTime(H, t)

Input: hypergraph H
target node t
Return: expected hitting times N_t

1: Calculate transition probabilities using Eq. (2).
2: Set node t as the target.
3: Calculate transition matrix B from hypergraph structure and target node t, following the procedures in Section 2.3.1.
4: $d \leftarrow B.dimension$
5: $ones \leftarrow$ *vector of all 1's*, $shape = (d, 1)$
6: Solve $(I - B)x = ones$ using conjugate gradient method.
7: Obtain expected hitting times $N_t \leftarrow x$

3 EXPERIMENTAL RESULTS

In the previous sections, we have outlined two random walk algorithms, the simple random walk (SRW) and frustrated random walk (FRW), and showed that we can compute their expected hitting times in a unified framework. In this section, we will use real-world hypergraph data sets to show that we can use the expected hitting times as node distances to find each node's nearest neighbors. We will also compare their results with that of DeepWalk.

3.1 Hypergraphs with ground-truth node labels

In this section, we will test our methods using two hypergraph data sets with ground truth node labels. We construct the first dataset from arXiv and get the second dataset from Ref. [15].

First, we create hypergraphs using articles published in arXiv under the category of physics from 2017 to 2020. Hypergraph nodes represent articles, and articles written by the same author are collected into the same hyperedge. From the hypergraph, we extract its largest connected component, which contains 52144 nodes and 45188 hyperedges. The node and hyperedge degree distributions are shown in Fig. 1. In this data set, node (article) degree represents the number of authors of the article. Here, the node degree distribution just approximately follows the power law due to the fact that in modern world most of papers have more than one author. However, when we plot the hyperedge (author) degree distribution, the power law emerges since there is neither upper limit nor lower limit on the number of articles that can be authored by the same researcher.

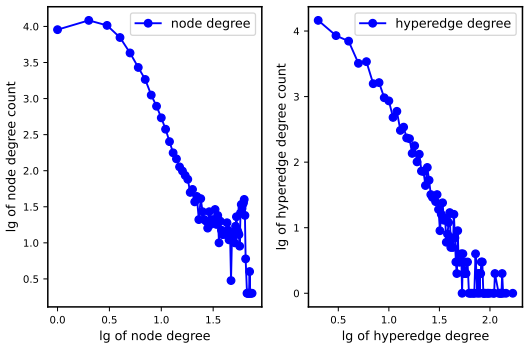

Figure 1: Log-log plot of node and hyperedge degree distribution for arXiv data set.

Each arXiv article has one or more subjects, which we can use as its ground truth label. Using the methods described in this paper, we can calculate hypergraph node distances, and thus find each article's nearest neighbors. Intuitively, the shorter the distance between two articles, the more similar they should be to each other. We can quantify two articles' similarity by calculating the Jaccard similarity between their subjects. For comparison, we find each article's nearest neighbors using SRW, FRW and DeepWalk, and calculate the mean Jaccard similarity between the target article and its top ten nearest neighbors. For one specific article, each method gives a mean Jaccard similarity score. The higher the mean Jaccard similarity, the better the method. We randomly select 12 articles, and for each method, calculate these 12 articles' average Jaccard similarity score. The average score for each method is shown in Table 1.

From table 1, we can see that FRW has the highest Jaccard similarity, which is shown in bold type in the table. Although FRW performs obviously better than SRW, and is almost on par with DeepWalk, we claim that the biggest advantage of FRW over SRW is that it can best describe node similarities for heavily-weighted

Method name	SRW	FRW	DeepWalk
Jaccard similarity	0.4092	**0.4615**	0.4599

Table 1: Comparison of Jaccard similarities of three methods. Twelve articles are randomly selected, and each article's top ten nearest neighbors are calculated using three different methods. For each article, we calculate the mean Jaccard similarity between the target article's subjects and its neighboring article's subjects. Finally, for each method, we calculate the mean Jaccard similarity of the twelve articles.

and scale-free hypergraphs, the results for which will be shown in the next section.

The second dataset, called trivago-clicks dataset[16], is constructed by authors of Ref. [15] from users' browsing behavior in trivago while the users are trying to book a hotel via trivago.com. In this hypergraph dataset, nodes represent accommodations (mostly hotels) the users browse, and hyperedges represent a user's browsing history in the same browsing session before the user clicks out (an order is placed). Each node (accommodation) is located in a country, which can be used as the node's label. The dataset contains 172,738 nodes and 233,202 hyperedges. The number of distinct node labels (countries or regions) is 160. The node degree follows a power law, yet the node degrees are generally small, ranging from 1 to 339. The node degree log-log distribution is shown in Fig. 2.

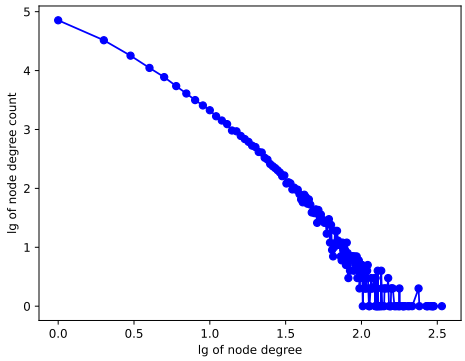

Figure 2: Log-log plot of node degree distribution for trivago data set.

Using SRW, FRW and DeepWalk, we can calculate each node's nearest neighbors. Generally, we expect a node and its nearest neighbors to be located in the same country, thus having the same node label. We can quantify this similarity by counting what is the proportion of nearest neighbors having the same label as that of the target node. We still randomly select 12 nodes from this hypergraph, and calculate the similarity between a target node and its top 100 nearest neighbors using the three methods. The results are shown in Table 2. For this dataset, the results from the three methods are almost the same, with more than 97% nearest neighbors having the same label as that of the target node, indicating the effectiveness of all three methods in detecting nearest neighbors.

Method name	SRW	FRW	DeepWalk
Similarity	0.9708	0.9775	**0.9792**

Table 2: Comparison of similarities resulting from SRW, FRW and DeepWalk methods. A larger similarity indicates a better method.

3.2 Heavily-weighted and Scale-free hypergraphs

A hypergraph is called scale-free if its node degree distribution follows a power law. A hypergraph is called heavily weighted if its node degrees range through several orders of magnitude. Power law distribution is ubiquitous in real world, e.g., the wealth distribution among the world population, the number of followers of social media accounts, the citation times of scientific papers, etc. According to Ref. [12], when constructing a graph, if we follow the preferential attachment principle, which means we prefer to attach a newly created node to an existing node of higher degree, then the resultant graph will be a scale-free graph. A scale-free hypergraph can also be constructed in this manner. In a scale-free graph or hypergraph, the nodes accumulate their degrees according to the Matthew effect, meaning that the higher a node's degree, the easier it is for its degree to get even higher. Frustrated random walk method is just designed to capture the Matthew effect. Consequently, the advantage of FRW is most obvious for heavily-weighted and scale-free hypergraphs, as we will show in this section.

We first show our results on a hypergraph that we created from *Dream of the Red Chamber*, a novel written by Cao Xueqin. We represent novel characters as hypergraph nodes, and each scene in the novel as a hyperedge. Characters are collected into a hyperedge if they appear in the same scene in the novel. Node degree indicates the number of times a node (character) appears in the novel. Just as we expected, the node degree distribution follows a power law, as shown in Fig 3. The node degrees range from 1 to 954, meaning the hypergraph is heavily weighted. We will show that for such a hypergraph, the FRW method is most suitable for calculating node distances.

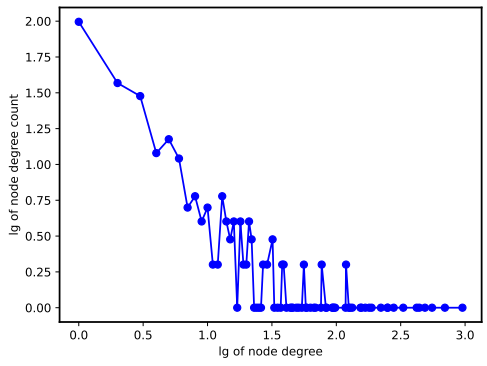

Figure 3: Log-log plot of node degree distribution in *Dream of the Red Chamber*.

Unlike the arXiv hypergraph, here we do not have ground truth labels for nodes, and thus here we will compare our random walk results against DeepWalk results. The fact that DeepWalk results are consistent with human judgement justifies our employment of DeepWalk results as our benchmark. We first run DeepWalk on the hypergraph to obtain a node ranking with respect to a target node, and then use SRW and FRW methods to get nearest neighbors of the target. Thus, with respect to each target node, we obtain three rankings using three methods. We then calculate the rank difference between SRW/FRW results and DeepWalk results. To do this, we get the top 10 neighbors of a target node using the SRW/FRW method, find the mean DeepWalk ranking for these 10 neighbors, and finally obtain for each target node the difference between mean DeepWalk rankings and mean SRW/FRW rankings. We summarize the rank differences for several nodes in table 3. To make the results representative enough, we selected three kinds of nodes (characters) from the novel: the central characters of highest node degrees, the supporting characters of medium node degrees, and the marginal characters of low node degrees. It can be seen from the table that for this data set, FRW is much better at finding nearest neighbors of central nodes than SRW. For marginal nodes of low node degrees, the difference between FRW and SRW is negligible.

As a second test, we create a hypergraph using *Harry Potter* novels written by J.K. Rowling. Nodes represent novel characters, and characters that appear in the same scene are collected into the same hyperedge. We use SRW, FRW, and DeepWalk to find the nearest neighbors of Harry Potter, and list our results in Table 4. From the table, we can see that the results from FRW and DeepWalk are consistent with human judgment, yet SRW results are not what we expect. This corroborates our assertion that FRW is more suitable than SRW for describing real-world human relationships.

4 TIME COMPLEXITY OF THE EXPECTED HITTING TIME METHOD

In Algorithm 1, we have outlined our method for calculating expected hitting times. For large hypergraphs, the most time consuming part of the algorithm is the numerical solution of equation $(I - B)x = \mathbf{1}$. For large and sparse coefficient matrix B, we can only solve this equation using conjugate gradient method or its variants. Conjugate gradient method involves a series of iteration cycles, and the most time consuming part of each cycle is the evaluation of Bv, where B is the probability transition matrix and v is a dense vector. Thus, the time complexity of our method as described in Algorithm 1 is $N_{iter} \times O(Bv)$, where N_{iter} is the iteration number in conjugate gradient method and $O(Bv)$ is the time complexity of evaluating Bv. We will determine $O(Bv)$ first.

From Eq. (2) and Eq. (3), we can see that each of the summation $\sum_j B_{ij}v_j$ requires $D_i + 1$ operations, where D_i is the node degree of vertex i. Thus, the total number of operations required for computing Bv is $\sum_i (D_i + 1) = 2E + V$, where E is the edge number (the number of edges if we expand all hyperedges in the hypergraph) and V is the node number. We thus have $O(Bv) = 2E + V$.

To determine N_{iter}, we need to note that conjugate gradient method is guaranteed to give the exact solution after exactly N_B iterations, where $N_B \lesssim V$ is the dimension of the coefficient matrix. However, for most cases, conjugate gradient method already gives

Target node	Jia Baoyu	Wang Xifeng	Lin Daiyu	Jia Zheng	Jia Tanchun	Jia Huan	Jia Yucun	Zhen Shiyin	Wang Yitie
Node degree	954	554	488	279	188	94	56	22	1
SRW rank diff	99.44	86.33	41.22	74.56	53.33	43.33	0.22	0.0	0.67
FRW rank diff	7.33	0.78	4.11	4.44	4.67	16.44	0.33	0.0	2.33

Table 3: Rank difference between SRW/FRW and DeepWalk for different target nodes in *Dream of the Red Chamber* data set. Target nodes are ranked from high to low node degrees. For central nodes of high degrees, the differences between FRW and DeepWalk ranking results are much smaller than that for SRW, indicating that the ranking results of FRW are more consistent with DeepWalk results. Yet for marginal nodes of low degrees, the ranking results from SRW, FRW and DeepWalk are equally good.

SRW	FRW	DeepWalk
Hokey	Ron Weasley	Ron Weasley
Morfin Gaunt	Hermione Granger	Hermione Granger
Merope Gaunt	Severus Snape	Albus Dumbledore
Marge Dursley	Sirius Black	Severus Snape
Mafalda Hopkirk	Fred Weasley	Ginny Weasley
Ignotus Peverell	George Weasley	Minerva McGonagall
Helena Ravenclaw	Albus Dumbledore	Voldemort
Cole	Rubeus Hagrid	Rubeus Hagrid
Mary Cattermole	Ginny Weasley	Fred Weasley
Mary Riddle	Draco Malfoy	Sirius Black

Table 4: Nearest neighbors of Harry Potter according to different methods.

results of high precision when $N_{iter} \ll N_B$. We can thus consider N_{iter} to be a small constant number compared to N_B.

We can thus get the time complexity of our algorithm as $O(2E + V)$, with the understanding that N_{iter} is a small constant number. For real-world hypergraphs, we generally have $E \propto V$. In such a case, the time complexity can be simplified to $O(V)$. Thus, the time complexity of our method is approximately linear with respect to the hypergraph size. This linear time complexity for the computation of expected hitting times of random walks is a significant advantage it has compared to the DeepWalk method.

5 COMPARISON OF RUNNING SPEEDS

In the previous sections, we have shown that to find a node's nearest neighbors using SRW/FRW methods, we only need to construct the transition matrix from hypergraph structure and the target node, and solve a system of linear equations using conjugate gradient method, the time complexity of which is approximately linear. However, if we want to accomplish this using DeepWalk, we need to map all the nodes to vectors, calculate the cosine distances between the target node vector and all the other node vectors, and finally rank all the other nodes with respect to the target according to cosine distances. The node mapping could be pretty time consuming, thus rendering the DeepWalk inferior in speed compared to SRW/FRW methods.

To compare the running speeds of these methods, we test our program on three data sets (arXiv, *Dream of Red Chamber*, and *Harry Potter*) using a Linux system with 64 processors. When running DeepWalk, we generate random paths by walking 3200 steps starting from each node in the hypergraph, and use word2vec[17, 18] method to map each node to a vector of size 128. To accelerate the program, the generation of random paths is fully parallelized. To obtain the running speeds for SRW and FRW methods, we randomly select some nodes as targets, and calculate the mean and standard deviation of the times spent on these nodes. The timing results are shown in Table 5.

As can be seen from the table, the running speeds of SRW and FRW are much faster than DeepWalk. SRW is even faster than FRW, yet as we have shown in the previous sections, for complicated hypergraphs, SRW cannot yield results that are on par with DeepWalk or FRW.

Data set	arXiv	*Dream of Red Chamber*	*Harry Potter*
DeepWalk	841	8.28	5.72
SRW	3.79 ± 0.17	0.012 ± 0.0002	0.0086 ± 0.0003
FRW	20.12 ± 0.4	0.026 ± 0.0005	0.018 ± 0.012

Table 5: Running times of DeepWalk, SRW and FRW on three data sets. All the numbers have seconds as unit.

6 CONCLUSION

In this paper, we have introduced two methods for computing node distances, one based on random walk method, and another based on DeepWalk method. Random walk method includes both simple and frustrated random walks, the expected hitting times of which can be exactly computed in a unified framework. We used the expected hitting times of both random walk scenarios to measure node distances on hypergraphs, and tested our methods on several real-world datasets. We showed that for lightly weighted hypergraphs, simple random walk, frustrated random walk and DeepWalk give similar results, whereas for scale-free and heavily weighted hypergraphs, frustrated random walk is more suitable for calculating node distances. We finally analyzed the time complexity of our random walk method, and showed that it is approximately of linear time complexity.

REFERENCES

[1] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.
[2] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[3] Douglas J Klein and Milan Randić. Resistance distance. *Journal of mathematical chemistry*, 12:81–95, 1993.
[4] Enzhi Li and Zhengyi Le. Frustrated random walks: A faster algorithm to evaluate node distances on connected and undirected graphs. *Phys. Rev. E*, 102:052135, Nov 2020.
[5] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. *Advances in neural information processing systems*, 19:1601–1608, 2006.
[6] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.
[7] Marina Meilă and Jianbo Shi. Learning segmentation by random walks. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, pages 837–843, 2000.
[8] Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.
[9] Sameer Agarwal, Kristin Branson, and Serge Belongie. Higher order learning with graphs. In *Proceedings of the 23rd international conference on Machine learning*, pages 17–24, 2006.
[10] Uthsav Chitra and Benjamin Raphael. Random walks on hypergraphs with edge-dependent vertex weights. In *International Conference on Machine Learning*, pages 1172–1181. PMLR, 2019.
[11] Timoteo Carletti, Federico Battiston, Giulia Cencetti, and Duccio Fanelli. Random walks on hypergraphs. *Phys. Rev. E*, 101:022308, Feb 2020.
[12] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.
[13] Carl D Meyer. *Matrix analysis and applied linear algebra*, volume 188. Siam, 2023.
[14] Magnus R Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems1. *Journal of Research of the National Bureau of Standards*, 49(6), 1952.
[15] Philip S Chodrow, Nate Veldt, and Austin R Benson. Hypergraph clustering: from blockmodels to modularity. *Science Advances*, 2021.
[16] Austin R. Benson. trivago-clicks dataset, 2021.
[17] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
[18] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546*, 2013.

A COMPUTATION OF EXPECTED HITTING TIMES ON AN ARTIFICIAL HYPERGRAPH

In this section, we illustrate the procedures of calculating expected hitting times of frustrated random walks on an artificial hypergraph. Monte Carlo simulations are used to validate our numerical results. The expected hitting times of simple random walks can be calculated in a similar manner. We use the hypergraph in Fig. 4 as our example.

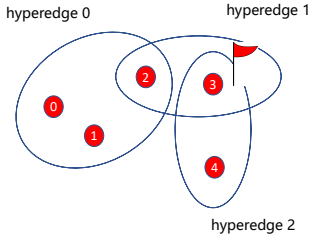

Figure 4: An example hypergraph

In this hypergraph, there are five nodes and three hyperedges. Its incidence matrix is

$$e = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \tag{15}$$

We use node 3 as the target node, and calculate the expected hitting times of frustrated random walks starting from nodes {0, 1, 2, 4}. First note that, according to our definition in section 2.3, node 4 is an adherent to target node 3, since node 4 has the target node as its only neighbor. From Eq. (3), we can see that the transition probability from node 4 to 3 is $p = \frac{1}{2}$. Thus, the expected hitting time from node 4 to 3 is $\mathbb{E}N_3^{(4)} = \frac{1}{p} = 2$. Next we focus on the calculation of expected hitting times starting from nodes {0, 1, 2}.

From Eq. (3) and Eq. (5), we can establish a system of difference equations for the hitting time probabilities, which is

$$\boldsymbol{X}_n = B\boldsymbol{X}_{n-1}, n \geq 2, \tag{16}$$

where the hitting time probability vector is

$$\boldsymbol{X}_n = \begin{pmatrix} P(N_3^{(0)} = n) \\ P(N_3^{(1)} = n) \\ P(N_3^{(2)} = n) \end{pmatrix}, \tag{17}$$

and the probability transition matrix is

$$B = \begin{pmatrix} 11/20 & 1/4 & 1/5 \\ 1/4 & 11/20 & 1/5 \\ 1/5 & 1/5 & 1/2 \end{pmatrix}. \tag{18}$$

To solve Eq. (16), we need the initial condition which is $\boldsymbol{X}_1 = \begin{pmatrix} 0 & 0 & \frac{1}{10} \end{pmatrix}^T$. The probability generating function as

$$\begin{aligned} \boldsymbol{f}(z) &= z(I - zB)^{-1}\boldsymbol{X}_1 \\ &= \frac{1}{z(16z - 65) + 50} \begin{pmatrix} z^2 \\ z^2 \\ (5 - 4z)z \end{pmatrix} \end{aligned} \tag{19}$$

Taking the first order derivative of the above equation, we can obtain the expected hitting times as

$$\mathbb{E}\boldsymbol{N}_3 = \boldsymbol{f}'(z = 1) = \begin{pmatrix} 35 \\ 35 \\ 30 \end{pmatrix} \tag{20}$$

We can similarly obtain the above results using Eq. (14), which is

$$\mathbb{E}\boldsymbol{N}_3 = (I - B)^{-1}\boldsymbol{f}(z = 1) = \begin{pmatrix} 35 \\ 35 \\ 30 \end{pmatrix} \tag{21}$$

To validate the above results, we write a Monte Carlo program to simulate the random walk process. Running the Monte Carlo simulation 100K times, we obtain the mean value of the hitting times starting from nodes {0, 1, 2, 4}. Monte Carlo simulation results and exact results are shown together in Table 6. From the table, we can see that exact results are consistent with Monte Carlo simulation results, which justifies our computational method. For large hypergraphs, it is impractical to run Monte Carlo simulations

Expected hitting times	Analytical result	Monte Carlo result
$\mathbb{E}N_3^{(0)}$	35	34.91879
$\mathbb{E}N_3^{(1)}$	35	35.00548
$\mathbb{E}N_3^{(2)}$	30	30.03275
$\mathbb{E}N_3^{(4)}$	2	1.99335

Table 6: Expected hitting times of frustrated random walks on hypergraph 4 with node 3 as target, starting from nodes {0, 1, 2, 4}, using both analytical and Monte Carlo simulation methods.

because it is too time consuming. In this case, we will only use numerical methods to compute the expected hitting times.