Random-walk Based Approximate k-Nearest Neighbors Algorithm for Diffusion State Distance [⋆]

Lenore J. Cowen[1], Xiaozhe Hu[2], Junyuan Lin[3], Yue Shen[4], and Kaiyi Wu[2]

[1] Department of Computer Science, Tufts University, Medford MA 02155, USA
`lenore.cowen@tufts.edu`
[2] Department of Mathematics, Tufts University, Medford MA 02155, USA
`xiaozhe.hu,kaiyi.wu@tufts.edu`
[3] Department of Mathematics, Loyola Marymount University, Los Angeles, CA 90045, USA
`junyuan.lin@lmu.edu`
[4] Department of Mathematics, Florida State University, Tallahassee, FL 32306, USA
`yshen@math.fsu.edu`

Abstract. Diffusion State Distance (DSD) is a data-dependent metric that compares data points using a data-driven diffusion process and provides a powerful tool for learning the underlying structure of high-dimensional data. While finding the exact nearest neighbors in the DSD metric is computationally expensive, in this paper, we propose a new random-walk based algorithm that empirically finds approximate k-nearest neighbors accurately in an efficient manner. Numerical results for real-world protein-protein interaction networks are presented to illustrate the efficiency and robustness of the proposed algorithm. The set of approximate k-nearest neighbors performs well when used to predict proteins' functional labels.

Keywords: Diffusion State Distance, Random Walk, k-nearest neighbors

1 Introduction

A classical and well-studied problem in bioinformatics involves leveraging neighborhood information in protein-protein interaction (PPI) networks to predict protein functional labels. In a typical setting, a PPI network is provided, where vertices represent proteins and edges are placed between two proteins if there is experimental evidence that they interact in the cell. In addition, some vertices are partially labeled with one or more functional labels representing what is known about their functional role in the cell. These functional labels are derived from some biological ontology (most commonly GO, the Gene Ontology [7]).

[⋆] The work of Cowen, Hu, and Wu was partially supported by the National Science Foundation under grant DMS-1812503, CCF-1934553, and OAC-2018149

The goal is to use the network structure to impute functional labels for the nodes where they are missing. In 2013, Cao et al. [5] proposed a simple function prediction method based on a novel metric, Diffusion State Distance (DSD). In particular, they proposed having the k-nearest neighbors in the DSD distance vote for their functional label(s) and then assigning the functional label that got the most votes. They showed that the k-nearest neighbors in the DSD metric can achieve state-of-the-art results for function prediction. In [4], this approach was generalized in a straightforward way to incorporate confidence weights on the edges of the PPI network. The focus of this paper is on whether this algorithm can be sped up in practice.

Computationally, using the naive algorithm to compute all DSD distances and then, finding the k-nearest neighbors according to DSD in a brute-force way takes $O(n^3)$ time, where n is the number of nodes in the PPI network. In [15], computing DSD of a given undirected but possibly weighted graph is reformulated to solving a series of linear systems of graph Laplacians, which can be approximately solved by existing graph Laplacian solvers in an efficient manner. Furthermore, the authors use Johnson-Lindenstrauss Lemma [11] and random projections [1] to reduce the dimension and, thus, further reduce the computational cost to $\mathcal{O}(n^2 \log n)$.

The next step is to find k-nearest neighbors (NN) based on the DSD metric, which is a kNN search problem. In general, existing exact and approximate search methods are subject to an unfavorable trade-off: either they need to construct a complex search structure so that the subsequent query retrievals on it are inexpensive, or they build a simpler data structure for which accurate searches remain costly [16, 12, 10, 13]. In this paper, our goal is to develop an accurate approximate kNN algorithm, that downstream, allows us to perform functional label prediction. In other words, we develop a method that efficiently computes a set of neighbors that perform well in the kNN function prediction task, even if they are not exactly the k-nearest neighbors by the DSD metric. To achieve this, we use the special property that DSD is random-walk based, and naturally, adopt a random-walk based approach to approximate k-nearest neighbors in the DSD metric. Comparing with the generic K-dimensional tree (K-d tree) algorithm [2], our random-walk based approach is more efficient for the DSD metric. On the other hand, our random-walk based approximate kNN algorithm is capable of finding a set of neighbors that provide good (or occasionally even better) performance for function predictions in practice. Overall, with properly chosen parameters, our random-walk based algorithm's computational complexity is $\mathcal{O}(n \log n)$. Thus, coupling with the algorithm for computing DSD that is developed in [15], we have a function prediction method with competitive function prediction performance comparing with the original method developed in [5], but the running time of the entire procedure is reduced from $\mathcal{O}(n^3)$ to $\mathcal{O}(n \log n)$.

The structure of the paper is organized as follows: Section 2 introduces basic matrices related to the graphs and reviews algorithms to compute the exact and the approximate DSD. In Section 3, we develop random-walk based approximate kNN algorithms and discuss their computational complexity. Numerical experiments are presented in Section 4 to show the efficiency and accuracy of our proposed function prediction algorithm for real-life PPI networks. Finally, we give some conclusions in Section 5.

2 Background

In this section, we review the related background and algorithms of DSD. We follow the notation in [15] for definitions related to graphs and random walks. We also briefly summarize the definitions of DSD and algorithms for computing DSD in this section.

2.1 Notation

In general, PPI networks are presented as connected undirected graphs with positive edge weights (where the edge weights most typically represent the confidence in the experimental evidence that indicates the proteins are interacting). We use $G = (V, E, \mathcal{W})$ to denote a connected undirected graph, where V is the vertex set, E is the edge set and $\mathcal{W}$ is the weight set. If two vertices v_i and v_j are incident, we denote an edge $e_{ij} \in E$ with positive weight $w_{ij} > 0$. We denote the number of vertex by $n = |V|$ and the number of edges by $m = |E|$. The degree of a vertex $v_i \in V$ is denoted as d_i, which is the number of edges connected to v_i and the weighted degree is defined as $\delta_i = \sum_{v_j \in V, e_{ij} \in E} w_{ij}$.

Next, we introduce several matrices related to the graphs. The adjacency matrix $A \in \mathbb{R}^{n\times n}$ and the degree matrix $D \in \mathbb{R}^{n\times n}$ are defined as follows,

$$A_{ij} = \begin{cases} w_{ij}, & \text{if } e_{ij} \in E, \\ 0, & \text{otherwise.} \end{cases} \qquad D_{ij} = \begin{cases} \delta_i, & i = j, \\ 0, & i \neq j. \end{cases}$$

Then the graph Laplacian is defined as $L = D - A$ and the normalized graph Laplacian N is defined as $N = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$. N is positive semi-definite and 0 is an eigenvalue of N. The eigenvector $\boldsymbol{d}$ corresponding to the zero eigenvalue is $\boldsymbol{d} := \delta_{\text{total}}^{-1/2} D^{\frac{1}{2}} \mathbf{1}$ where $\delta_{\text{total}} = \sum_{v_i \in V} \delta_i$ is the total weighted degree and $\mathbf{1} \in \mathbb{R}^n$ is the all one vector. Note that $\|\boldsymbol{d}\| = 1$. Since the idea of DSD is based on random walks on the graph, we also introduce the transition matrix P which is defined as $P = D^{-1}A$. Since P is row stochastic and irreducible, the steady state distribution $\boldsymbol{\pi} \in \mathbb{R}^n$ is defined as $\boldsymbol{\pi} = \delta_{\text{total}}^{-1} D\mathbf{1}$, which is the left eigenvector of P, i.e. $\boldsymbol{\pi}^T P = \boldsymbol{\pi}^T$. In addition, $\boldsymbol{\pi}$ is normalized as $\boldsymbol{\pi}^T \mathbf{1} = 1$. Finally, we introduce $W = \mathbf{1}\boldsymbol{\pi}^T$ which is the so-called Perron projection.

2.2 Diffusion State Distance (DSD)

In this section, we briefly recall the diffusion state distance which was introduced in [5]. We associate each vertex i with a vector $\boldsymbol{h}_i^q \in \mathbb{R}^n$ such that $(\boldsymbol{h}_i^q)_j$ represents the expected number of times for a random walk to start from vertex i and visit vertex j within q steps. Based on the definition of the transition matrix P, $\boldsymbol{h}_i^q := (I + P^T + (P^T)^2 + \cdots + (P^T)^q)\boldsymbol{e}_i$, where $\boldsymbol{e}_i$ is the i-th column of the identity matrix. Then the q-th step DSD between the vertex i and the vertex j is defined as $\mathrm{DSD}^q(i,j) := \|\boldsymbol{h}_i^q - \boldsymbol{h}_j^q\|_p$, where $\|\cdot\|_p$ is the standard ℓ_p-norm. Then the diffusion state distance is defined by letting $q \to \infty$, i.e, $\mathrm{DSD}(i,j) := \lim_{q\to\infty} \mathrm{DSD}^q(i,j)$. In [5, 15], it has been shown that the above limit exists and DSD is well-defined. We only state the convergence theory here.

Theorem 1 (Convergence for DSD [5, 15]). *Assume that P is row stochastic and irreducible, then $DSD^q(i,j)$ converges as $q \to \infty$, i.e.,*

$$\mathrm{DSD}(i,j) := \lim_{q\to\infty} \mathrm{DSD}^q(i,j) = \|X(\boldsymbol{e}_i - \boldsymbol{e}_j)\|_p,$$

where $X = (I - P^T + W^T)^{-1} \in \mathbb{R}^{n\times n}$ is called the diffusion state.

In practice, to compute DSD between all the pairs of vertices, X is precomputed and stored. Furthermore, an alternative formulation, $X = D^{\frac{1}{2}}(N^\dagger + \boldsymbol{d}\boldsymbol{d}^T)D^{-\frac{1}{2}}$, is used in the implementation to improve overall efficiency.

Note that the i-th column of the diffusion state X can be viewed as a new coordinate presentation of the vertex v_i in $\mathbb{R}^n$. When n is large, i.e. the coordinates of the new representation are in high dimension, the computational cost is $\mathcal{O}(n^3)$ in general and can be improved to $\mathcal{O}(n^2)$ for sparse graphs. This is still expensive or even infeasible. Therefore, dimension reduction is needed in order to further reduce the computational cost. In [15], based on the well-known Johnson-Lindenstrauss Lemma [11, 1], we apply a random matrix $Q \in \mathbb{R}^{s\times n}$ that reduces the dimension of the data from n to $s = \mathcal{O}(\log n)$ in the l_2 norm with high probability. This approach yields the approximate diffusion state $\widetilde{X} :== QD^{\frac{1}{2}}N^\dagger D^{-\frac{1}{2}} \in \mathbb{R}^{s\times n}$. Each column of $\widetilde{X}$ represents the coordinates of each vertex in a vector space of dimension $s = \mathcal{O}(\log n) \ll n$, which naturally reduces the computational cost to $\mathcal{O}(m \log n)$ in general, or $\mathcal{O}(n \log n)$ for sparse graphs [15].

Based on the approximate diffusion state $\widetilde{X}$, the approximate DSD is defined as $\widetilde{\mathrm{DSD}}(i,j) = \|\tilde{X}(e_i - e_j)\|$. The following theorem guarantees the goodness of its approximation property.

Theorem 2 ([15]). *Given $\epsilon, \gamma > 0$, let $s \geq \frac{4+2\gamma}{\epsilon^2-\epsilon^3}\log n$. Then with probability at least $1 - n^{-\gamma}$, for all pair $0 \leq i, j \leq n$,*

$$\sqrt{1-\epsilon}\,\mathrm{DSD}(i,j) \leq \widetilde{\mathrm{DSD}}(i,j) \leq \sqrt{1+\epsilon}\,\mathrm{DSD}(i,j).$$

In [8], a spectral dimensional reduction approach was introduce to compute the approximate diffusion state $\widetilde{X}$ as well as $\widetilde{\mathrm{DSD}}$. We want to point out that the kNN construction algorithms developed in this paper can be applied to that case as well. In fact, our algorithm can be applied to any approximation of the diffusion state X.

2.3 Construction of k Nearest-Neighbor Set based on DSD

Once the diffusion state X or its approximation $\widetilde{X}$ has been computed and stored, we want to know, for each vertex, the local k-NN set in the DSD metric.

The simplest method to find the k closest vertices to each vertex is the brute-force method, which computes the distances between all pairs of vertices and finds the k smallest distances for each vertex. In our case, although this method gives the exact kNN neighborhood, the computational cost is $\mathcal{O}(n^3)$ if we use X and $\mathcal{O}(n^2 \log n)$ if we use $\widetilde{X}$. When n is large, the cost of this method is expensive in practice.

There are several methods that find the exact kNN more efficiently when the data lie in a low dimensional manifold. For example, the K-d trees [9], where K is the dimension of the data, has a computational cost of $\mathcal{O}(Kn \log n)$. However, when the data sets are high dimensional, these algorithms are still slow. In our case, if X is used ($K = n$), the cost of the K-d tree approach is $\mathcal{O}(n^2 \log n)$ and, if $\widetilde{X}$ is used ($K = \mathcal{O}(\log n)$), the cost is $\mathcal{O}(n \log^2 n)$. This is, of course, much better than the brute-force method, but still expensive for our biological application since, as suggested in [9], K-d trees works the best when $d = \mathcal{O}(1)$.

One natural idea is to restrict the search to a smaller set for each vertex instead of considering all the vertices. This subset should contain most of the target node's k-nearest neighbors. Then the k-nearest neighbors in this subset can be returned as the approximate kNN set. Motivated by the random walk interpretation of DSD, we form the neighborhood subset for each vertex via performing short random walks. We show that the resulting approximate kNN sets can be substituted for the exact kNN sets for function prediction, without loss of prediction percent accuracy.

3 Approximate kNN Set Construction

In this section, we develop fast algorithms for constructing approximate kNN set. We first present the general algorithm and then discuss how to apply them to the diffusion state X and approximate diffusion state $\widetilde{X}$, respectively.

3.1 Random-walk based kNN Set Construction for X

In [5, 15], it has been shown that $X = I + P^T + (P^T)^2 + \cdots = \sum_{q=1}^{\infty}(P^T)^q$, which reveals how the diffusion state X is closely related to the random walks on the graph. Based on the fact that the derived infinite series converges, one can approximate X by its partial sums, namely by doing several steps of random walks on the graphs to approximate the diffusion state X. Since the goal is to find k-nearest neighbors under the DSD metric, if the DSD can be approximated via several steps of random walks, it is natural to assume those k-nearest neighbors can also be reached via several steps of random walks with high probability.

To be more specific, from a vertex v, if we take one random walk step and get to u_1, we put u_1 into v's neighborhood set $R(v)$. Then from u_1, if we take one more random walk step and get to u_2 (which means we take two steps of random walk from v to get to u_2), then we add u_2 into the set $R(v)$. We continue until t_1 steps of a random walk are done and gather all vertices ever reached in $R(v)$. Then, we repeat this process t_2 times from vertex v and put all the vertices that are reached by t_1 steps of the random walks into the set $R(v)$. Finally, we find k-nearest neighbors from $R(v)$ for the vertex v. The overall algorithm is presented in Algorithm 1.

Algorithm 1 Random-walk based kNN set for the diffusion state X

```
1: for all $v \in V$ do
2:     $R(v) = \emptyset$.
3:     for $j = 1 : t_2$ do
4:         for $h = 1 : t_1$ do
5:             $R(v) = \{u$ : vertex reached by $j$ steps of random walk from $v\} \cup R(v)$.
6:         end for
7:     end for
8:     Use $X$ to compute the distances between $v$ and vertices in $R(v)$ and find $k$-nearest
       neighbors of $v$.
9: end for
```

The computational complexity of Algorithm 1 is discussed next.

Theorem 3. *The overall computational cost of Algorithm 1 is at most* $3t_1t_2n^2 + (k+1)t_1t_2n$.

Proof. If we take t_1 steps of a random walk and repeat this t_2 times to collect $R(v)$ for one vertex v, the complexity of performing the random walk is t_1t_2 and the size of $R(v)$ is at most t_1t_2. Therefore, the computational cost for performing all random walks over the n vertices is bounded by t_1t_2n.

Since we are using the diffusion state X here, each vertex is represented by one column of X whose dimension is n, computing DSD between one pair of vertex costs $3n$ and computing DSD between v and vertexes in $R(v)$ costs at most $3t_1t_2n$. Therefore, for all n vertices, the overall cost for computing DSD is $3t_1t_2n^2$.

For finding the k-nearest neighbors, since $|R(v)| \leq t_1t_2 \ll n$ in practice, we use naive approach to find k-nearest neighbors for each vertex and, the computational cost is at most kt_1t_2n.

To summarize, the overall computational cost of Algorithm 1 is at most $3t_1t_2n^2 + (k+1)t_1t_2n$, which completes the proof.

3.2 Random-walk based kNN Set Construction for $\widetilde{X}$

In Algorithm 1, using random walks, we avoid computing too many distances, which reduces the overall computational cost from $\mathcal{O}(n^3)$ to roughly $\mathcal{O}(n^2)$. This is the best we can do for the exact diffusion states, because the diffusion state

X represents each vertex in $\mathbb{R}^n$. Therefore, to further reduce the computational cost, the approximate diffusion state $\widetilde{X}$ should be used since it embeds each vertex in $\mathcal{R}^s$ where $s = \mathcal{O}(\log n) \ll n$. This can be done by replacing X with $\widetilde{X}$ in Algorithm 1. However, note that $\widetilde{X}$ is an approximation of X, simply using $\widetilde{X}$ might affect the accuracy of the resulting approximate kNN graphs. To mitigate the effect, since random projection is used to compute $\widetilde{X}$, we average over multiple random projections to further improve the quality of the approximate kNN graph.

The overall algorithm based on $\widetilde{X}$ is presented in Algorithm 2 and we analyze its computational cost in Theorem 4.

Algorithm 2 Random-walk based kNN set for $\widetilde{X}$

```
1: Given $t_3$ copies of different $\{\widetilde{X}^i\}_{i=1}^{t_3}$.
2: for all $v \in V(G)$ do
3:     $R(v) = \emptyset$.
4:     for $j = 1 : t_2$ do
5:         for $h = 1 : t_1$ do
6:             $R(v) = \{u$ : vertex reached by$j$ steps of random walk from $v\} \cup R(v)$.
7:         end for
8:     end for
9:     for $i = 1 : t_3$ do
10:        Compute DSD between $v$ and vertices in $R(v)$ by using $\widetilde{X}^i$.
11:    end for
12: end for
13: Compute the average DSD between $v$ and vertices in $R(v)$ and find $k$ nearest
    neighbors of $v$.
```

Theorem 4. *The overall computational cost of Algorithm 2 is at most* $3t_1t_2t_3sn + (k+1)t_1t_2n + t_3$.

Proof. Following the same argument as in the proof of Theorem 3, performing random walks to find the set $R(v)$ for all vertices costs t_1t_2n. For each approximate diffusion state $\widetilde{X}^i$, computing DSD costs at most $3t_1t_2sn$. Since we have t_3 copies of $\widetilde{X}$, the total cost for this part is $3t_1t_2t_3sn$. In the final step, computing the average DSD costs t_3 and finding the k nearest neighbors cost at most kt_1t_2n. Therefore, the overall cost of Algorithm 2 is at most $3t_1t_2t_3sn+(k+1)t_1t_2n+t_3$.

Remark 1. In our numerical experiments, we choose $t_1 = \mathcal{O}(1)$, $t_2 = \mathcal{O}(1)$, $t_3 = \mathcal{O}(1)$, $k = \mathcal{O}(1)$, and $s = \mathcal{O}(\log n)$. Therefore, the computational costs for Algortihm 1 and 2 are $\mathcal{O}(n^2)$ and $\mathcal{O}(n \log n)$, respectively. Comparing with K-d trees, Algorithm 2 has a slightly better computational complexity. In the numerical experiments, we focus on Algorithm 2 and demonstrate that it efficiently computes approximate kNN sets with good approximation qualities, and that preserve function prediction accuracy.

4 Numerical Experiments

In this section, we present several numerical experiments to show the efficiency and accuracy of the proposed random-walk based construction of approximate kNN graphs.

4.1 Information about the Data Sets

The Disease Module Identification DREAM Challenge [6] released a heterogeneous collection of six different human protein-protein association networks with different biological criteria for placing edges between different protein pairs, in order to benchmark different methods for unsupervised network clustering. Here, we use the first two of the DREAM networks, DREAM1 and DREAM2, to additionally benchmark our exact and approximate function prediction methods, where we briefly describe how they were constructed, below. DREAM1 was derived from the STRING database [17], which integrates known and predicted protein-protein interactions across multiple resources, including both direct (physical) and indirect (functional) associations. The edge weights are derived from the STRING association score [17]. DREAM2 was produced from the InWeb database [14], which aggregates evidence that pairs of proteins physically interact in the cell from different databases. Interaction edges supported by multiple sources are given higher confidence. Both DREAM1 and DREAM2 are low diameter and highly connected(see Table 1); note that we remove some small number of isolated nodes and node-pairs and restrict our study only to the largest connected component of each network (see Table 1) to guarantee the irreducible assumption for convergence. We further remark that the typical "small-world" properties of these networks make shortest-path-based distance metrics very uninformative in differentiating meaningful neighborhoods for these networks, and we expect that nearby nodes according to the DSD metric will be more relevant for understanding function (see discussion in [5]).

Table 1: Largest connected components of the DREAM networks

	Vertices	Edges	Type	Edge Weight
Network 1	17,388	2,232,398	PPI	Confidence score
Network 2	12,325	397,254	PPI	Confidence score

4.2 Numerical Results on Approximate *k*NN Set Construction

To test if the constructed approximate kNN graph is both efficient and accurate in preserving distance in different networks, we compare it with the standard K-d tree approach. As mentioned in Section 1, K-d trees require $n \gg 2^K$ to achieve the best performance, where K is the dimension of a data point. Therefore, we focus on the approximate diffusion state $\widetilde{X}$ in our tests. Note that, for $\widetilde{X}$, the computational complexity of the K-d tree is $\mathcal{O}(n \log^2 n)$ while our proposed Algorithm 2 costs $\mathcal{O}(n \log n)$.

The numerical tests are conducted on a 2.8 GHz Intel Core i7 CPU with 16 GB of RAM. The K-d tree approach is implemented based on the built-in MATLAB functions `KDTreeSearcher` and `knnsearch` based on the approximated DSD computed by $\widetilde{X}$. For comparison, Algorithm 2 is implemented in MATLAB as well. In our comparison, we fix $\epsilon = 0.5$ (tolerance for generating approximate diffusion state $\widetilde{X}$), $k = 10$ (the number of nearest neighbors for the kNN neighborhood – matching the recommended choice of this parameter in [5]), and fix $t_3 = 1$ (number of copies of $\widetilde{X}$ used in Algorithm 2). We first perform a parameter study to find suitable choices of t_1 and t_2 and compare the accuracy and efficiency of Algorithm 2 with the K-d tree approach. The accuracy is measured by $\frac{\sum_{i=1}^{n} |\{kNN_{exact}(i)\} \cap \{kNN_{app}(i)\}|}{k \cdot n}$, where $kNN_{exact}(i)$ is the set of k-nearest neighbors of node i using the exact DSD computed by the exact diffusion state X. $kNN_{app}(i)$ is the set of approximate k-nearest neighbors of node i found by either the K-d tree approach or our method (Algorithm 2) using the approximate DSD computed by the approximate diffusion state $\widetilde{X}$.

CPU time of K-d tree: **990.63**

t_2 \ t_1	1	5	10	20	30
40	36.81	44.07	57.48	86.01	109.76
50	34.25	48.85	66.45	101.53	130.07
60	35.81	50.71	70.65	110.53	146.92
70	38.84	54.02	76.30	122.21	165.19

(a) CPU time in seconds

Accuracy with K-d tree: **0.1873**

t_2 \ t_1	1	5	10	15	20
40	0.3258	0.2687	0.2399	0.2171	0.2064
50	0.3321	0.2678	0.2401	0.2170	0.2063
60	0.3367	0.2672	0.2403	0.2163	0.2051
70	0.3377	0.2664	0.2380	0.2155	0.2043

(b) kNN accuracy

Table 2: CPU time in seconds and kNN accuracy of applying Algorithm 2 on DREAM1.

CPU time of K-d tree: **442.07**

t_2 \ t_1	1	2	3	4	5
110	15.46	19.83	23.72	20.63	30.74
120	15.68	20.42	24.39	30.21	32.22
130	15.57	20.77	25.14	29.06	33.92
140	15.72	21.62	26.04	30.19	35.19

(a) CPU time in seconds

Accuracy of K-d tree: **0.4565**

t_2 \ t_1	1	2	3	4	5
110	0.4591	0.5643	0.5440	0.5201	0.5215
120	0.4607	0.5660	0.5455	0.5318	0.5224
130	0.4596	0.5671	0.5452	0.5332	0.5221
140	0.4608	0.5657	0.5446	0.5331	0.5217

(b) kNN accuracy

Table 3: CPU time in seconds and kNN accuracy of applying Algorithm 2 on DREAM2.

For DREAM1 (Table 2), as t_2 (times of repeating the random walk process) increases, the CPU time needed increases, and the accuracy gets better slightly as expected. On the other hand, the accuracy gradually decreases as t_1 (steps of random walk) gets larger, which means, in practice, we actually do not need to take long random walks to find k-NN, especially for clustered networks such as DREAM 1. We observe similar results for DREAM2, see Table 3. Basically, for high-density networks, relatively shorter random walks should be used and we do not need to repeat the random walks many times. In addition, for both DREAM1 and DREAM2, comparing with the K-d tree approach, our proposed Algorithm 2 is consistently faster (about $6\times$ to $29\times$ speedup for DREAM1 and $13\times$ to $29\times$ speedup for DREAM2) while achieves higher kNN accuracy. This demonstrates the efficiency and effectiveness of Algorithm 2.

4.3 Numerical Results on Function Prediction

Predicting proteins' functions based on proteins with known labels and the network structure is a classical and well-studied problem in computational biology. We can make inferences for the proteins with unknown labels because proteins that are close to each other (under some appropriate metric) tend to share similar functions. In this section, we want to show that the approximate kNN neighbors produced by Algorithm 2 can provide accurate function prediction results in biological applications.

We carry out the tests on the DREAM1 and DREAM2 networks. The biological function labels were collected from both the Biological Process (BP) and the Molecular Function (MF) hierarchies from the Gene Ontology database [7] via FuncAssociate3.0 [3] on 04/24/2020. On each network, we only consider the GO labels that represent neither too general nor too specific functions by restricting to labels that appear between 100 and 500 times in the largest connected component of that network. The experiments were performed separately for labels in the Biological Process hierarchy and the Molecular Function hierarchy. In our cross-validation experiments, we will mark a functional label prediction correct if it matches one of the functional labels that FuncAssociate assigns to that node. The percent accuracy of the function prediction method represents the percent of top predicted GO functional labels that are correct. The number of functional labels to be voted are different across network and functional hierarchies (BP and MF); the proteins in the largest connected component of DREAM1 have 1007 BP and 165 MF labels that appear between 100 and 500 times. For DREAM2, the numbers are 888 BP and 139 MF labels.

On each network, we carry out five-fold cross-validation on its largest connected component to obtain a percent accuracy score for the top predicted GO functional labels. For every protein i in the test set, we find its k-nearest neighbors $\{r_1, r_2, \ldots, r_k\}$ based on Algorithm 2 and use them to perform function prediction by majority voting with weights $\frac{1}{\widetilde{\text{DSD}}(i, r_j)}$, for $j = 1, 2, \ldots, k$, where each protein in the training set votes with this weight for all its functional labels. The label with the most (weighted) votes is then assigned as the top function prediction label for the protein.

The accuracy score of function prediction for the network is the average percentage of correct assignments in the cross-validation test. We compare the function prediction result based on the k-nearest neighbors obtained by using Algorithm 2 using approximate diffusion state $\widetilde{X}$ to the one based on the exact kNN measured in exact DSD computed by the exact diffusion state X.

In our test, we set $k = 10$ (number of the nearest neighbors, as recommended by [5]) and $t_3 = 1$ for Algorithm 2, i.e., only 1 copy of the approximate diffusion state $\widetilde{X}$ (computed by fixing the tolerance $\epsilon = 0.5$) was used. Different combinations of random walk steps t_1 and repetitions t_2 are tested for both GO Biological Process hierarchy and Molecular Function hierarchy of function labels. The results are averaged over 100 cross-validation trials.

Accuracy with exact kNN: **0.0245**

t_2 \ t_1	1	5	10	15	20
1	0.0178	0.0178	0.0225	0.0242	0.0232
5	0.0178	0.0238	0.0225	0.0238	0.0246
10	0.0191	0.0243	0.0244	0.0227	0.0238
15	0.0201	0.0238	0.0248	0.0237	0.0242
20	0.0206	0.0238	0.0255	0.0256	0.0229
25	0.0228	0.0263	0.0245	0.0255	0.0234
30	0.0216	0.0247	0.0251	0.0237	0.0244

(a) Biological Process

Accuracy with exact kNN: **0.0236**

t_2 \ t_1	1	5	10	15	20
1	0.0477	0.0488	0.0315	0.029	0.0278
5	0.0487	0.0296	0.0266	0.0269	0.0308
10	0.0387	0.028	0.0298	0.0275	0.0267
15	0.0338	0.0277	0.0279	0.0276	0.0270
20	0.0328	0.0289	0.0296	0.0268	0.0278
25	0.0325	0.0279	0.0291	0.0263	0.0272
30	0.0327	0.0274	0.0285	0.0268	0.0284

(b) Molecular Function

Table 4: Accuracy score of function prediction on DREAM1 for GO Biological Process and Molecular Function hierarchy labels. The cells are shaded when the function prediction performance is comparable or even better than the performance obtained by using exact k nearest neighbors measured in exact DSD.

In Table 4(a) and 5(a) and (b), the function prediction accuracy increases as t_2 increases. In Table 4(b), the trend is the opposite with performance peaks when $t_2 = 1$ or $t_2 = 5$. We observe improved function prediction performance by voting with the approximate k-nearest neighbors computed by appropriate choice of t_1 and t_2 values instead of voting with the exact kNN neighbors measured with the exact DSD distance, for both the Biological Process and Molecular Function labels, in both DREAM1 and DREAM2 networks. We remark that this similar or even improved prediction accuracy is obtained at a much lower computational cost than the exact kNN version.

The strong performance of approximate kNN on the function prediction task is curious. We hypothesize that in addition to producing a set of neighbors that largely overlaps the kNN neighbors, the approximate kNN neighbors produced by averaged local random walks are somehow denoising the signal as well to improve the function prediction accuracy.

Accuracy with exact kNN: **0.1047**

t_2 \ t_1	1	5	10	15	20
1	0.0308	0.027	0.0873	0.1157	0.1156
5	0.0303	0.2472	0.2401	0.2330	0.2324
10	0.1060	0.2821	0.2755	0.2733	0.2670
15	0.1887	0.2981	0.2885	0.2822	0.2798
20	0.2055	0.3051	0.2964	0.2893	0.2843
25	0.2175	0.3059	0.2990	0.2953	0.2941
30	0.2207	0.3124	0.3027	0.2992	0.2938

(a) Biological Process

Accuracy with exact kNN: **0.0991**

t_2 \ t_1	1	5	10	15	20
1	0.0612	0.0489	0.1112	0.1107	0.1102
5	0.0563	0.1977	0.1937	0.1889	0.1882
10	0.1533	0.2214	0.2136	0.2120	0.2077
15	0.1711	0.2326	0.2264	0.2170	0.2153
20	0.1791	0.2356	0.2293	0.2215	0.2237
25	0.1853	0.2407	0.232	0.2262	0.2261
30	0.1852	0.2395	0.2337	0.2315	0.2256

(b) Molecular Function

Table 5: Accuracy score of function prediction on DREAM2 for GO Biological Process and Molecular Function hierarchy labels.The cells are shaded when the function prediction performance is comparable or even better than the performance obtained by using exact k-nearest neighbors measured in exact DSD.

5 Conclusions

In this paper, we consider the k-nearest neighbors problem for the DSD metric. Since DSD is a diffusion-based distance and closely related to the random walks on the graph, we developed approximate kNN algorithms that use random walks to find a set of possible neighbors of each vertex and then identify k nearest ones from this set. Our approach provides a good approximation of the kNN while reducing the computational cost since the size of the set that is explored is kept small. More precisely, when combined with the approximate DSD computed by approximate diffusion state $\widetilde{X}$, the computational complexity of our approximate kNN algorithm (Algorithm 2) is $\mathcal{O}(n \log n)$. This is not only much better than the naive kNN approach which has complexity $\mathcal{O}(n^3)$ for DSD but also slightly better than the K-d tree approach which has complexity $\mathcal{O}(n \log^2 n)$ for approximate DSD in theory. In practice, for the DREAM networks, our random-walk based algorithm can achieve about 30 times speed up in time while maintaining or even achieving better kNN accuracy. In addition, when applying our algorithm for biological applications such as function predictions on PPI networks, our method provides competitive prediction performance while significantly reducing the computational cost. The focus in this paper was on the relative performance of exact and approximate measures for function prediction, not absolute performance. However, we note that in our setting, in an absolute sense, we are doing much better on the function prediction task using DREAM2 than using DREAM1. This is probably because DREAM1 is filled with lots of very low confidence edges, and even the way the confidence weights are incorporated from DREAM1 may be insufficient to denoise the signal, without further thresholding or tuning. DREAM2 is only including edges that pass a confidence filter.

References

1. Achlioptas, D.: Database-friendly random projections: Johnson-lindenstrauss with binary coins. Journal of computer and System Sciences **66**(4), 671–687 (2003)
2. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Communications of the ACM **18**(9), 509–517 (1975)
3. Berriz, G.F., Beaver, J.E., Cenik, C., Tasan, M., Roth, F.P.: Next generation software for functional trend analysis. Bioinformatics **25**(22), 3043–3044 (08 2009)
4. Cao, M., Pietras, C.M., Feng, X., Doroschak, K.J., Schaffner, T., Park, J., Zhang, H., Cowen, L.J., Hescott, B.J.: New directions for diffusion-based network prediction of protein function: incorporating pathways with confidence. Bioinformatics **30**(12), i219–i227 (2014)
5. Cao, M., Zhang, H., Park, J., Daniels, N.M., Crovella, M.E., Cowen, L.J., Hescott, B.: Going the distance for protein function prediction: A new distance metric for protein interaction networks. PLoS ONE **8**(10), 1–12 (10 2013)
6. Choobdar, S., Ahsen, M.E., Crawford, J., Tomasoni, M., Fang, T., Lamparter, D., Lin, J., Hescott, B., Hu, X., Mercer, J., et al.: Assessment of network module identification across complex diseases. Nature methods **16**(9), 843–852 (2019)
7. Consortium, T.G.O.: The Gene Ontology Resource: 20 years and still GOing strong. Nucleic Acids Research **47**(D1), D330–D338 (11 2018)
8. Cowen, L., Devkota, K., Hu, X., Murphy, J.M., Wu, K.: Diffusion state distances: Multitemporal analysis, fast algorithms, and applications to biological networks. SIAM Journal on Mathematics of Data Science **3**(1), 142–170 (2021)
9. Finkel, R., Friedman, J., Bentley, J.: An algorithm for finding best matches in logarithmic expected time. ACM Transactions on Mathematical Software pp. 200–226 (1977)
10. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing. pp. 604–613 (1998)
11. Johnson, W.B., Lindenstrauss, J.: Extensions of lipschitz mappings into a hilbert space. Contemporary mathematics **26**(189-206), 1 (1984)
12. Kleinberg, J.M.: Two algorithms for nearest-neighbor search in high dimensions. In: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing. pp. 599–608 (1997)
13. Kushilevitz, E., Ostrovsky, R., Rabani, Y.: Efficient search for approximate nearest neighbor in high dimensional spaces. SIAM Journal on Computing **30**(2), 457–474 (2000)
14. Li, T., Wernersson, R., Hansen, R.B., Horn, H., Mercer, J., Slodkowicz, G., Workman, C.T., Rigina, O., Rapacki, K., Stærfeldt, H.H., et al.: A scored human protein–protein interaction network to catalyze genomic interpretation. Nature methods **14**(1), 61 (2017)
15. Lin, J., Cowen, L.J., Hescott, B., Hu, X.: Computing the diffusion state distance on graphs via algebraic multigrid and random projections. Numerical Linear Algebra with Applications **25**(3), e2156 (2018)
16. Liu, T., Moore, A.W., Yang, K., Gray, A.G.: An investigation of practical approximate nearest neighbor algorithms. In: Advances in neural information processing systems. pp. 825–832 (2005)
17. Szklarczyk, D., Franceschini, A., Wyder, S., Forslund, K., Heller, D., Huerta-Cepas, J., Simonovic, M., Roth, A., Santos, A., Tsafou, K.P., Kuhn, M., Bork, P., Jensen, L.J., von Mering, C.: STRING v10: protein-protein interaction networks, integrated over the tree of life. Nucleic acids research **43**, D447–D452 (2015)