SARKAR & MOORE 335

**A Tractable Approach to Finding Closest Truncated-commute-time Neighbors in Large Graphs**

**Purnamrita Sarkar** Machine Learning Department Carnegie Mellon University Pittsburgh, PA 15213

## Andrew W. Moore

Google Inc.

Pittsburgh, PA 15213

# Abstract



Recently there has been much interest in graph-based learning, with applications in collaborative filtering for recommender net- works, link prediction for social networks and fraud detection. These networks can consist of millions of entities, and so it is very im- portant to develop highly efficient techniques. We are especially interested in accelerating random walk approaches to compute some very interesting proximity measures of these kinds of graphs. These measures have been shown to do well empirically (Liben-Nowell & Kleinberg, 2003; Brand, 2005). We intro- duce a truncated variation on a well-known measure, namely *commute times* arising from random walks on graphs. We present a very novel algorithm to compute all *interesting* pairs of approximate nearest neighbors in truncated commute times, without comput- ing it between all pairs. We show results on both simulated and real graphs of size up to 100*,* 000 entities, which indicate near-linear scaling in computation time.

# 1 Introduction

The main aim of link-prediction and collaborative- filtering is to answer the question, for any selected node: which other nodes in the graph are *nearest* to this node? It is important to have a useful proximity measure for this purpose and very desirable for near nodes to be computed quickly. Ideally we will want this similarity metric to capture the graph structure. For example if two nodes have many common neigh- bors, they are highly similar. On the other hand, imag- ine a graph with two distinct connected components, connected by a single link. The two nodes connecting the components are neighbors in the graph but belong to different clusters, and are expected to be less similar than their neighbors in the same component.

A rather intuitive way of capturing this is the expected path length from one node to another during a random walk. This measure is called the hitting time (Aldous & Fill, 2001), and it tells us how long it will take on an average to hit the destination node from a source node. Note however that this measure is not guaran- teed to be symmetric. Hence it’s easier to work with the round trip time, i.e. the *commute time* between two nodes. These measures are inherently robust to noise and exploit the information encoded in the graph structure. They are widely used for unsupervised and semi-supervised learning (Zhu et al., 2003).

In spite of being very successful for collaborative filter- ing (Brand, 2005), dimensionality reduction (Saerens et al., 2004) or image segmentation problems (Qiu & Hancock, 2005), the main drawback of these mea- sures is the computational load. In previous work au- thors have used approximate sparse matrix inversion techniques (Brand, 2005) or subspace approximations using principal eigenvectors (Saerens et al., 2004) to

avoid *O*(*n*3) computation. However these techniques do not scale to graphs with more than few thousands of

nodes. Brand (2005) uses clever iterative approaches to avoid this bottleneck, but some quantities derived from the random walk are loosely approximated.

In this paper we introduce a truncated variant of ran- dom walks in order to exploit local structure in graphs, and devise a novel algorithm to compute all interesting pairs of near-neighbors under this truncated version of commute times without having to compute the entire commute times matrix. Results on both simulated and real networks of size up-to 100*,* 000 (on a single proces- sor) indicate near-linear scaling in computation time.

# 2 Random walks on undirected graphs

In this section we introduce some basic concepts of random walks on undirected graphs. An undirected graph *tt* has a set of vertices *V* numbered 1 to *n*, and a set of edges *E*. An edge is a pair (*i, j*) where *i, j* ∈ *V* ,

336 SARKAR & MOORE

1

and (*i, j*) ∈ *E* ⇔ (*j, i*) ∈ *E*. The adjacency matrix where *xi* = (*L*+) 2 *ei*. Let us now look at the entries of *W* of *tt* is an *n* × *n* matrix of real values, where *Wij L*+ in terms of these position vectors. The *ij*-th entry denotes the weight on edge *i, j*, and is zero if the edge *l*+ = *xT xj*, denotes the dot-product between the po-

*ij i*

does not exist. For undirected graphs *W* is symmetric. sition vectors of vertex *i* and *j*. The diagonal element

*D* is an *n* × *n* diagonal matrix, where *Dii* = Σ*j Wij*. *l* denotes the squared length of the position vector of

+

*ii*

We will denote *Dii* by the degree *d*(*i*) of node *i* from *i*. The cosine similarity (Brand, 2005) between two

*ij /*.*lii ljj* . We discuss different

now on. The Laplacian *L* of *tt* is defined as *D* − *W* . nodes is defined as *l*+ + +

Consider a random walk (Lovasz, 1996) on *tt* such ways of computing the above quantities in Section 3. that, if at step *t* we are in vertex *vt* = *i*, then in

the next step we move to a neighbor of *i* with prob- **Robustness towards noise:** Hitting and commute ability proportional to the weight *wij* of the link, times are robust to noise. Doyle and Snell (1984) view

i.e. *wij* . Clearly the sequence of nodes *v*0*, v*1*, ..., vt* an undirected unweighted graph as an electrical net-

*Dii*

form a markov chain. We have *Pt*(*i*) = *Pr*(*vt* = *i*). work with every edge replaced by an unit weight resis- *P* = *pij , i, j* ∈ *V* denotes the transition probability tor. In Chandra et al. (1989) the authors show that matrix of this markov chain, so that *pij* = *wij /Dii* if the commute time between two nodes *s, t* in a graph is (*i, j*) ∈ *E* and zero otherwise. proportional to the effective resistance between *s* and

In a random walk, if node *v*0 is chosen from a distribu- *t*. Since addition or deletion of a few edges usually do tion *P*0, then the distributions *P*0*, P*1*, ..* are in general not change the electrical properties of a network sub- different from one another. However if *P*0 = *P*1, then stantially, the above relation makes it easy to see that we say that *P*0 is the stationary distribution for the commute times are also robust to minor perturbation graph. It can be shown that for a graph *tt* the sta- of the graph structure.

tionary distribution is given by *π*(*v*) = *d*(*v*) , where We give a very simple but intuitive example. Consider

*V* (*G*)

*V* (*tt*) = Σ*v d*(*v*), denotes the volume of the graph. a undirected unweighted graph *tt*. We select any node

We now introduce two main proximity measures de- *c* and add links from *c* to all other nodes in the graph. rived from random walks, namely the hitting and the Let *i*, and *j* be any two nodes, s.t. *i* ƒ= *j* ƒ= *c*. A commute time (Aldous & Fill, 2001). shortest path between *i* and *j* becomes 2 hops now.

However the hitting time *hij* hardly changes. If a ran-

**Hitting time** *H***:** The hitting time between nodes *i* dom walk starting at *i* hits *c*, then there is a very and *j* is defined as the expected number of steps in a small chance of hitting any other node from *c*, since random walk starting from *i* before node *j* is visited ∀*j pc,j* = 1*/*(*n* − 1). Hence the hitting times between

for the first time. *H* = *hij, i, j* ∈ *V* is an *n* × *n* matrix. the nodes in the graph stay almost the same. Recursively *hij* can be written as *hij* = 1 + Σ*k pikhkj* This property is very desirable since in most real-world if *i* ƒ= *j* and zero otherwise. data-sets there are spurious links resulting from erro-

Hitting times are unsymmetric, however they follow neous observations. We also demonstrate this using a

the triangle inequality (Lovasz, 1996). simulated dataset in the experimental section.

**Commute time** *C***:** Commute time between a pair

of nodes is defined as *cij* = *hij* + *hji* and is symmetric. **3 Related Work**

*cij* can also be defined as **3.1 Applications**

*cij* = *V* (*tt*)(*l*+ + *l*+ − 2*l*+)

*ii jj ij* The random walks approach has been highly successful

= *V* (*tt*)(*ei* − *ej* )*T L*+(*ei* − *ej* ) (1) in social network analysis (Katz, 1953) and computer

vision (Gorelick et al., 2004; Qiu & Hancock, 2005).

Since each row of the graph Laplacian sums to zero, Here we briefly describe some of these applications. it has an eigenvector of all ones, with eigen-value zero.

It is common practice (Saerens et al., 2004) to shift Saerens et al. (2004) exploit equation (1) to embed

the zero-eigenvalue by subtracting the all ones matrix, a graph and provide distances between any pair of inverting the resulting matrix, and then adding the all nodes. Yen et al. (2005) replace traditional shortest- ones matrix back again, i.e. *L*+ = (*L−* 1 **11***T* )*−*1 + 1 **11***T* . path distances between nodes in a graph by hitting and

*n n* commute times and show that standard clustering al-

The pseudo-inverse of *L*, i.e. *L*+, can be viewed as a gorithms (e.g. K-means) produce much better results kernel (Smola & Kondor, 2003) which maps each ver- when applied to these re-weighted graphs. These tech- tex of a graph to a Euclidian space *i* ›→ *xi*. Pairwise niques exploit the fact that commute times are very commute time can be expressed as the squared Euclid- robust to noise and provide a finer measure of cluster

ian distance in the transformed space (equation (1)), cohesion than simple use of edge weight.



SARKAR & MOORE 337

Gorelick et al. (2004) use the average hitting time of a random walk from an object boundary to characterize object shape from silhouettes. Grady and Schwartz (2006a) introduced a novel graph clustering algorithm which was shown to have an interpretation in terms of random walks. Hitting times from all nodes to a des- ignated node were thresholded to produce partitions with various beneficial theoretical properties. Grady and Schwartz (2006b) used the above approach for au- tomated image segmentation. Also Qiu et al have used commute times clustering for robust multibody motion tracking (Qiu & Hancock, 2006) and image segmenta- tion (Qiu & Hancock, 2005).

Commute times and hitting times have been success- fully used for collaborative filtering. In Brand (2005) the authors use different measures resulting from ran- dom walks to recommend products to users based on their purchase history . The authors give empirical ev- idence of the fact that hitting and commute times often are small if one of the nodes has a high degree.whereas the cosine similarity (defined in Section 2) does not have that problem since in some sense the effect of individual popularity is normalized out.

Given a snapshot of a social network, a very interesting question is: which new connections among entities are likely to occur in the future? In Liben-Nowell and Kleinberg (2003) different proximity measures between nodes in a graph are used for link prediction tasks on social networks. The authors showed that the hitting and commute times suffer from the fact that they take into account information from long paths. The most effective measure was shown to be the Katz measure (Katz, 1953) which directly sums over the collection of paths between two nodes with exponentially decaying weights so that small paths are given more weight.

However, computing this for all pairs of nodes needs at least *O*(*n*2) time and storage.

## Computation

Most clustering applications either require extensive computation to produce pairwise proximity measures, or employ heuristics for restricting the computation to

subgraphs. Saerens et al. (2004) compute the trailing eigenvectors of *L* to reconstruct *L*+. This still requires at least *O*(*n*2) computation and storage.

Cubic computation is avoided by using sparse matrix manipulation techniques. In Brand (2005) the authors try to compute sub-matrices of *H* and *C* by iterative sparse matrix multiplications. Note that computing

the *ith* row of *C*, i.e. *C*(*i,* ∗) or the cosine-similarities of all *j* with *i* requires *l*+ *,* ∀*j*. The exact value of *l*+ re-

*jj*

*jj*

tion of *j* is a close constant factor approximation to *l*+ , which is fairly loose. In short, it is only tractable to compute these measures on graphs with a few thou-

sand nodes for most purposes.

*jj*

The main problem with computing hitting times is that its easy to compute every node’s hitting time to a single node, i.e. one column of the hitting time matrix using dynamic programming approaches. However its hard to compute the hitting time from one node to every other node. For that one will have to effectively compute the entries of an entire column in order to get a single entry. This problem can be reduced to the problem of finding all diagonal elements of the pseudo- inverse of the graph Laplacian without computing the entire pseudo-inverse.

## Other graph-based proximity measures

Spectral clustering (Ng et al., 2001) is a body of al- gorithms that cluster datapoints *xi* using eigenvectors of a matrix derived from the affinity matrix *W* con- structed from the data. Often *wij* is obtained from the Euclidian distance between the datapoints, i.e.

*wij* = exp( *xi xj* 2*/σ*2), where *σ* is a free param- eter. Zhu et al. (2003) uses graph theoretic approach

ǁ − ǁ

for semi-supervised learning. Given labeled and unla- beled data-points as nodes in a graph, the main goal is to exploit the graph structure to label the unlabeled examples using the few labeled ones.

# The Algorithm

In this section we present a novel algorithm to compute approximate nearest neighbors in commute time for all nodes, without computing the entire matrix.

## Truncated Hitting times

Hitting and commute times decrease if there are many short paths between the nodes. On the other hand, as observed by Liben-Nowell and Kleinberg (2003), two major drawbacks are that they tend to be small when- ever one of the nodes has a high degree, and they are also sensitive to parts of the graph far away from the nodes, even when short paths between the nodes exist.

In order to overcome these problems we define a *T- truncated hitting time*, where we only consider paths of length less than *T* . In Section 5 we show that *hT* approaches the true hitting time as *T* . We use *h*, *ht* interchangeably to denote truncated hitting time.

→ ∞

The *T* truncated hitting time i.e. *hT* from *i* to *j*, mea- sures the expected number of steps taken by a random walk starting at node *i* to hit *j* for the first time. It can also be defined recursively as

*ij*

quires solving linear systems of equations for all other

rows of *L*+. The authors state that for large markov chains the square of the inverse stationary distribu-

*T* = 1 + *pik T* 1

*k*

*h*

Σ

*h −*

*ij*

*kj*

(2)

338 SARKAR & MOORE

commute times are used for ranking entities, e.g. rec- ommend the *k* best movies based on choices already

G made by an user; or, find the *k* most likely co-authors of an author in a co-authorship network.

B Suppose we want to estimate *hij* for pairs of nodes *i, j*

*T*

which have relatively low *hT* values, and suppose we

*ij*

want to avoid *O*(*n*2) time or space complexity. There- fore we cannot do anything that requires representing

or iterating over all pairs of nodes. In fact, we can- not even afford to iterate over all pairs of nodes that

Figure 1: Neighborhood of node *j*. A directed graph is are less than *T* hops apart, since in general the num- drawn for clarity. ber of such pairs is *O*(*n*2). Suppose we restrict com- putation to iterate only over some subset of pairs of

nodes, which we will call the Active-Pairs-set or the

where *hT* is defined to be zero if *i* = *j* or if *T* = 0. *AP* set. The interesting question is how good are the The above equation expresses *hT* in a one step look- pessimistic and optimistic bounds we can get on *hT* ,

*ij*

ahead fashion. The expected time to reach a desti- ∀*i, j* using only *O*(|*AP* |) work?

nation within *T* timesteps is equivalent to one step As mentioned before we will consider bounded neigh- plus the average over the hitting times of it’s neigh- borhoods of each node in the graph. For clarity, in

bors to the destination. Equation (2) can be easily case of a directed graph the neighborhood consists of computed by using Gauss Seidel iterations. Let the all nodes with short paths *to* the destination node. truncated hitting times matrix be *HT* . Note that the Here is our algorithm in a nutshell: for each node,

hitting time from all nodes to a fixed destination, i.e. we will start with the direct neighbors in its neighbor-

a column of *HT* can be computed in *O*(*T* ∆*n*) time, hood, and then compute the optimistic and pessimistic where ∆ is the average degree of a node. However if bounds on hitting times of the nodes within a neigh- we need to compute the hitting time from one node to borhood to that node. We will then expand the neigh- everyone else, i.e. a row of *HT* , *HT* (*i,* ∗) then ∀*j* we borhood and re-iterate. The bounds will be tighter will compute the corresponding column in order to get and tighter as we keep expanding the neighborhoods. one entry *HT* (*i, j*). Hence we shall end up computing

the entire matrix. Instead of examining all pairs of Define the set *AP* as a set of pairs of nodes such that

nodes we develop a very efficient framework for doing if *i* is in the neighborhood of *j*, then (*i, j*) ∈ *AP* . Note a range-search to retrieve the *k* approximate nearest that (*i, j*) ∈ *AP* does not imply that (*j, i*) ∈ *AP* . Each neighbors of *all* nodes in a graph. neighborhood of node *i* has a boundary, *δi*, such that

**4.2 GRANCH** a boundary node on the neighborhood of *j* has direct neighbors outside the neighborhood. Again in a di-

The main intuition is that we think about the graph rected graph a boundary node has outgoing edges to in terms of *n* overlapping subgraphs. Each subgraph nodes outside the neighborhood.

is a bounded neighborhood for one node (discussed in Let’s first assume that the neighborhood is given. It’s detail later in the section). Consider the hitting times clear from the expression for truncated hitting time arising from the random walk from any node in this that *hij* can be computed from the hitting time of *i*’s neighborhood to the destination. We provide upper neighbors to *j*. In Figure 1 the random walk from *i* (*pessimistic*) and lower (*optimistic*) bounds of these can hit either of it’s neighbors *tt* or *B*. Since *tt* is inside hitting times, and show how to combine information the neighborhood of *j* we already have an estimate from all these overlapping subgraphs to obtain *k* near- of *hGj* . However *B* is outside the neighborhood of est neighbors for all nodes. We will use the terms “op- *j*, and hence we find the optimistic and pessimistic timistic” and “lower”, and “pessimistic” and “upper”, values of *hBj* , resulting into lower bounds (*hoij* ) and interchangeably. In order to motivate the GRANCH upper bounds (*hpij* ) of *hij* . The optimistic bound is algorithm we present the following claim before for- obtained by allowing the random walk to jump to the mally describing the algorithm. boundary node with closest optimistic hitting time to

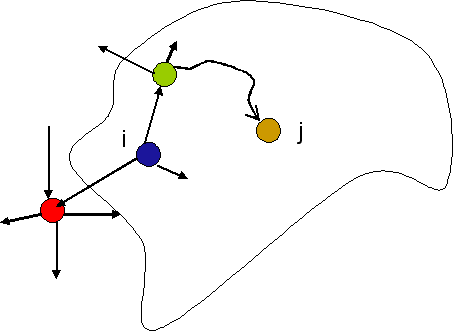
**Claim**: Given *k*, *s*, *T* and *T j* ≤ *T* , GRANCH returns *j*, and the pessimistic bound arises from the fact that

any *k* neighbors *j* of *i* ∀*i*, s.t. *cT* (*i, j*) ≤ *cT* (*i, ki,T r* )(1 + the walk might never come back to the neighborhood

*s*), where *ki,T r* is the true *kth* nearest neighbor of *i* after leaving it, i.e. takes *T* time.

within *T* -truncated hitting time *T j*. Now we revisit the data-structures in terms of their

This makes sense since in all applications hitting and notations. We denote the set of active pairs as *AP* .



SARKAR & MOORE 339

*AP* ( *, i*) is the set of nodes *k*, s.t. (*k, i*) *AP* . Hence the neighborhood of *i* can also be denoted by *AP* ( *, i*). Also *δ*(*i*) is the set of the boundary nodes of the neigh- borhood of *i*, i.e. *AP* (∗*, i*). Let *nbs*(*i*) denote the set

∗

∗ ∈

.

*hpij*

= *hpij* if *i* ∈ *AP* (∗*, j*) *T* else

of direct neighbors of *i*. Also denote by *ttnbs*(*i, j*) the

set *AP* (∗*, j*)∩*nbs*(*i*). The upper and lower bounds are:

Note that we are explicitly giving the expressions for all pairs of nodes in the graph for clarity. However the

*hpT* = 1 + Σ

*ij*

*pikhpT −*1

bounds need only *O*( *AP* ) space. Using the above we

also get optimistic and pessimistic bounds on commute

| |

 *k∈Gnbs*(*i,j*) 

*kj*

 Σ

times, namely *co* and *cp*.

+ 1 −

*k∈Gnbs*(*i,j*)

*pik* (*T* − 1) (3)

*coij*

= *hoij*

+ *hoji*

; *cpij*

= *hpij*

+ *hpji*

(5)

*T* = 1 + Σ

*ho*

*ij*

*pikhoT −*1

## 4.3 Expanding Neighborhood

 *k∈Gnbs*(*i,j*)  .

*kj*

*k∈Gnbs*(*i,j*)

of pairs in the APset, in order to avoid wasting com- putation on nodes that are very far apart. We present

Σ

Since time and space complexity of GRANCH is

*p∈δ*(*j*)

*pj*

+ 1 − Σ

*pik*

1 + min *hoT −*2

(4)

*O*(|*AP* |), we have to be clever in having the right set

We can also use a tighter bound than the above by us-

ing one step lookahead from the nodes on the bound- ary, which is omitted for space purposes. The *ho* and *hp* values of all nodes in *AP* ( *, j*) to *j* can be computed using the Gauss-Seidel technique, as in Algorithm 4.2. After obtaining the bounds on the hitting times be-

∗

Algorithm 1: compute-H(*dst*,*AP* ,*tt*,*T* )

1: *ho, holast, hp, hplast ones*(1 : *N* )1

←

2: *minvals*0:*T* 0; *minvals*1 = 1

←

3: **for** *t* = 2 to *T* **do**

4: *min* ← ∞

5: **for** *src* ∈ *AP* (∗*, dst*) **do**

6: *s*1*, s*2 ← 0, *prob* ← 0

7: **for** *i* ∈ *nbs*(*src*) **do**

a *neighborhood expansion scheme* which will guarantee

that we do not miss any potential nearest neighbors. We also note that adding nodes to the neighborhood of *i* can only tighten the bounds.

**Theorem 4.1** *The optimistic bounds on H, i.e. ho*

*values can only* ***increase****, and the pessimistic bounds,*

*i.e. hp values can only* ***decrease****, as a result of adding nodes to the AP set.*

**Proof** : The hitting time for a node outside the AP- set must be at least the minimum hitting time to the boundary of the AP-set, plus the minimum hitting time from the boundary to the destination. For a des-

tination node *j*, *hoT −*1 ≥ 1 + min*p∈δ*(*j*) *hoT −*2*,* ∀*m* ƒ∈

8: **if** *i* ∈ *AP* (∗*, dst*) **then**

*m,j pj*

*AP* (∗*, j*). Hence *hoT* in (4) is indeed a valid lower

*ij*

9: *s*1 ← *s*1 + *psrc,iholast*(*i*)

10: *s*2 ← *s*2 + *psrc,ihplast*(*i*)

bound on *hT* . Therefore it can never decrease.

The pessimistic bound in (3) is valid, since *ht* ≤

*ij*

11: *prob* ← *prob* + *psrc,i t ij*

∀

12: **end if**

13: **end for**

14: *ho*(*src*) 1 + *s*1 + (1 *prob*)(1 + *minvalst−*2)

← − −

← −

15: *hp*(*src*) 1 + *s*2 + (1 *prob*)(*t* 1)

16: **if** *ho*(*src*) *min* & *src δ*(*dst*) **then**

≤ ∈

17: *min ho*(*src*)

←

18: **end if**

19: **end for**

20: *minvalst* ← *min*

21: *holast* ← *ho*

22: *hplast hp*

←

23: **end for**

tween all pairs in the AP-set, we can obtain bounds on hitting times as follows:

.

*ho* = *hoij* if *i* ∈ *AP* (∗*, j*)

*ij*

1 + min

*hoT −*1

else

*t i, j*, by construction of *h* . Hence the pessimistic bounds can never increase.

**Deftnition**: Define the lower bound of the neighbor- hood of node *i* as *lb*(*i*) = 1 + min*p∈δ*(*i*) *hoT −*1.

*pi*

**Lemma 4.2** *The lower bound on hitting time of nodes outside the AP -set of node i, i.e. lb*(*i*)*, can only* ***in- crease*** *as a result of adding nodes to the AP set.*

**Proof** : This is a direct application of Theorem 4.1.

We start with all direct neighbors of the destination node *i*. Note that all nodes outside the boundary will

take at least 1+min*p δ*(*i*) *hoT −*1 time to reach *i*. Let us denote this by *lb*(*i*). Also remember that we are doing a range search of all neighbors within a range *T j < T* .

*pi*

*∈*

We want to increase this lower bound, so that we can

 *p∈δ*(*j*) *pj*

1Note that *∀i, j h*1(*i, j*) = 1.

guarantee that we shall never leave out a potential nearest neighbor. The optimal way to increase the

340 SARKAR & MOORE

Algorithm 2: Expand-AP(*tt, T, T j*) suffice to find all nodes *j*, s.t. *hpij* is smaller than the *h*

value of the (*k* + 1)*th* true nearest neighbor. Note that

1: Initialize2AP with all pairs *i, j* ∈ *EG* the (*k* + 1)*th* nearest neighbor’s hitting time is greater

2: **for** *i* = 1 to *N* **do** than the (*k* + 1)*th* largest *ho* value. Hence the upper-

3: **while** *lb*(*i*) ≤ *T j* **do** bound becomes *X* = (*k* + 1)*th largest hoi,∗*. Now we

4: compute-H(*i*,*AP* ,*tt*,*T* ) also allow a small error margin of *s* leading to a new

5: *lb*(*i*) ← 1 + min*p∈δ*(*i*) *hoT −*1

*pi* upper bound *X*(1 + *s*). All entities with upper bound

6: *q* ← arg min*p∈δ*(*i*) *hoT −*1 less than this are guaranteed to be *s*-approximate.

*pi*

7: *AP* ← *AP* ∪ {(*a, i*) : *a* ∈ *nbs*(*q*)} We also want the neighbors to be within *T j* hitting

8: **end while**

9: **end for** time of the source *i*. Hence we find the largest *kj* ≤

*k* + 1, s.t. *hoi r ,i* ≤ *T j* and *hpi r ,i* ≥ *T j*, where *ikr* is

*k k*

the node with the *kjth* largest *ho* value from *i*. Now

**bounds** the upper bound *X* becomes *X* = *hoi,ikr* . We return

all nodes *x*, s.t. *hpi,x < X*(1 + *s*). Therefore in Figure

**lower X(1+**ε**)** 2 we have *k* = 15, *kj* = 10. This means that at this

**times** point the bounds are not tight enough to determine

**increasinghitting X** the exact ranks of the *k* − *kj* neighbors. All the nodes

**by B** with lower bounds beyond *T j* are guaranteed not to be

**orderedtruncated B** within *T j*. The nodes labeled *B* are not guaranteed to

**iof** be *s* approximate and are not returned.

**of**

**Neighbors 4.4.2 Truncated commute time**

Now we look at the neighbors in commute times.

**Lower and Upper bounds of truncated hitting times** The new distance for range-search becomes 2*T j*. For

any node *i*, we now look at all nodes in the set Figure 2: Upper and lower bounds of *hT* (*i,* ∗) *AP* (*i,* ∗) ∪ *AP* (∗*, i*). *AP* (*i,* ∗) denotes the set of all

nodes that have *i* in their neighborhoods. After run-

ning Algorithm 2, if there exists some node *j*, s.t. lower bound is to add in the neighbors of *q*, where *j* ƒ∈ *AP* (∗*, i*) and *i* ƒ∈ *AP* (∗*, j*), then by Lemma 4.3, *q* = arg min*p∈δ*(*i*) *hoT −*1. *ho*(*i, j*)*, ho*(*j, i*) ≥ *T j* and hence *co*(*i, j*) ≥ 2*T j*, so *j*

*pi*

cannot be within a commute distance of 2*T j* of *i*.

**Lemma 4.3** *If i* ƒ∈ *AP* (∗*, j*) *after running Algorithm* We compute the nearest-neighbors for node *i, i* ∈ {1 :

*2, then hij* ≥ *T j. n*} using the APset resulting from Algorithm 2. Now

After running algorithm 2, *hoi,j* ≥ *lb*(*j*) ≥ *T j*. Also we find the nearest neighbors of *i* in commute times by construction we have *hi,j* ≥ *hoij* . using equation (5). Note that we only look at the

**4.4 Approximate Nearest Neighbors** nodes in set *Si* = *AP* (*i,* ∗)∪*AP* (∗*, i*). Sort all the nodes

in increasing order of optimistic bounds i.e. *co* values.

We shall first look at how to obtain the *k s*- Let {*i*1*, i*2*, ...*} denote the indexes of the sorted nodes. approximate nearest neighbors for each node in hitting Let *kj* the largest integer less than *k*, s.t. *co*(*i, ikr* ) ≥ time. Later we shall demonstrate how to extend that 2 ∗ *T j*. If such a *kj* exists define *X* as 2 ∗ *T j*. Otherwise to approximate nearest neighbors in commute times. clearly all *k* possible nearest neighbors might be within

Note that we only have upper and lower bounds on the range. Hence define *X* as *co*(*i, ik*). Return any hitting times from a node to other nodes. The key node *j* ∈ *Si* s.t. *cp*(*i, j*) ≤ *X*(1 + *s*). Also let *B* be the idea is that any value *x* will be smaller than *y* if the set of nodes *b* in *Si* s.t. *co*(*i, b*) ≤ *X*, but *cp*(*i, b*) ≥ upper bound of *x* is smaller that the lower bound of *y*. *X*(1 + *s*). Clearly these nodes cannot be guaranteed

to be among the *k* nearest neighbors. Now we expand

**4.4.1 Truncated hitting time** the AP-sets of {*i*} ∪ *B* to tighten the bounds more.

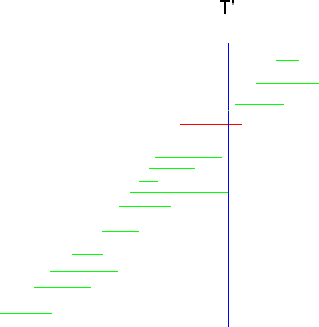
We illustrate with a toy example. Lets assume that we Because of the construction of the algorithm, the want to find the *k* nearest neighbors of node *i* within newly introduced nodes can never affect *kj*, and the *T j*. In Figure 2 we plot the lower and upper bounds other nearest neighbors. By Lemma 4.3, we will never

introduce a new possible nearest neighbor of a node

of *hi,j*, in the order of increasing lower bounds. It will *b* ∈ {*i*} ∪ *B*, since all nodes outside *Sb* are guaranteed

2The AP-set can be initialized to all pairs within *p* hops to have commute-distance worse than 2*T j*. Also an

too. We used *p* = 2 for some of our experiments. expansion of the AP-set can only increase the lower



SARKAR & MOORE 341

bounds, and decrease the upper bounds on the esti- mates on hitting times, which will never decrease the number of neighbors returned so far. Hence we will only improve the performance of the algorithm by ex- panding our AP-set beyond algorithm 2.

GRANCH uses Algorithm 2 to build the AP-set and then computes the *k s*-approximate nearest neighbors in truncated commute time.

# Truncated and True Hitting Times

In this section we examine the relation between the true and truncated hitting times. Let be the set of paths between *i, j*, such that *j* appears only once in the path. LSet P*t* be all such paths of length exactly

P

*t*. So P =

*t* P*t*. We can then express the truncated

The idea is to predict possible interactions between entities based on the structure of the network.

We present performances of GRANCH and a baseline algorithm, along with the number of pairs of nodes ex- amined by both. The baseline algorithm we use is the number of hops between two nodes, for which we do a simple Breadth First Search from the node upto 4 hops, and define everything beyond as 5 hops. Since social network datasets exhibit transitive social trends, this seemingly naive baseline is often hard to beat. It is used by many online friendship networks for rec- ommending friends. For the small simulated graphs we also computed the exact truncated commute time, which needed all pairs of nodes. Our results show that

the performance of GRANCH is comparable to the ex-

hitting time as an expected path length:

*T−*1

*hT* = *E*(*|P|*) = X *tPr{i j}* + (1 *−* X *Pr{i*

*t t*

*j}*)*T*

act version, even if it looks at small fraction of all pairs

of nodes. It also performs better than the baseline, and

*ij →−*

*t*=1

*t*

*→−* needs to examine much smaller number of pairs.

Where *Pr i →− j* denotes the probability that *i*

*t*

{ }

reaches *j* for the first time in exactly *t* steps.

Define *P*˜ as a matrix identical to *P* , except for the *jth* row. *m P*˜(*j, m*) = 0. Hence after the random walk hits *j* from *i*, there is a loss of probability mass from the system, which is why the sum is bounded. Thus *Pr*{*i →− j*} = *P*˜*t* . Substituting in the above equation:

*t*

*i,j*

∀

Here is a brief description of the experimental method- ology. We remove a random subset of links from the original graph. Now we compute the 2*T* -truncated- commute times on the training links. The ranking induced by this is tested against the held out links us- ing their AUC-scores. We only take nodes which has at least one link held out. For each such node we take the proximity measures from the algorithms of all nodes

*T −*1

*T −*1

within 4 hops of the source in the original graph(not

*hT* = Σ *tP*˜*t* + (1 − Σ *P*˜*t* )*T*

*ij*

*t*=1

*ij*

*t*=1

*ij*

all other nodes) and compute the AUC score of that

vector. We present the average AUC score.

Note that as *T* the probability of *i* not reaching *j* goes to zero if the graph is connected. Hence the above gets reduced to the true hitting time.

→ ∞

The rate at which the *T* -truncated hitting time ap- proaches the true hitting time depends on the mixing time of a random walk on the graph. The mixing time equals the number of steps before the random walk converges to the stationary distribution. It is the in- verse of the mixing rate (Lovasz, 1996) which measures how fast the distribution approaches the stationary distribution. This can be easily computed from the eigenvalues of the graph Laplacian. One way of set- ting the value of *T* dynamically will be by evaluating the mixing time. For a graph with a small mixing time, *T* should be small, since after a few steps the distribution becomes independent of the start node.

# Experiments

We present our experiments in two parts. First we present results on simulated data. Then we give re- sults from large real world data-sets extracted from the Citeseer co-authorship network. We use link- prediction tasks to evaluate the commute times result- ing from GRANCH. Link prediction is a very impor- tant and fairly hard task in social network analysis.

We denote the exact commute time by *C*. We also compare our algorithm for different values of *T* , 3*,* 6*,* 10. For most of the graphs we noticed that *T* = 6 was enough. *T* in all tables translates to a 2*T* -truncated commute time. We also saw that as we

increase our range of search, i.e. *T j* we get closer to the accuracies of the exact truncated hitting times. A typical choice of *T j* is 2*.*9 for *T* = 3, 5*.*95 for *T* = 6, and 9*.*75 for *T* = 10.

We present the total number of pairs in the *AP* -set for GRANCH in Tables 2 and 4. These tables also show the total number of pairs the baseline needs to look at while examining all nodes with at least one edge held out. This is because we compute AUC scores for only these nodes. We also put the total number of pairs within 4 hops for the Citeseer graphs in Table 5. The total number of pairs for the baseline can be as much as these numbers in the worst case.

## Simulated Data

A graph *tt* can be quantified in terms of its growth rate *ρG* (Krauthgamer & Lee, 2003). *ρG* is defined as the minimum *ρ*, s.t. every ball of radius *r >* 1 in *tt* contains at most *O*(*rρ*) nodes. By definition, this implies that the total number of nodes within *c* hops

342 SARKAR & MOORE

Table 1: AUC score on 30% links held out from graphs of size *n*, edges *e* , growth-rate *d* for algorithms *C*, *ttRANCH* with *T* = 10, *T* = 6, *T* = 3 and the baseline (number of hops between node-pairs within 4 hops).

*d n e* C,T=10 GRANCH,T=10 C,T=6 GRANCH,T=6 C,T=3 GRANCH,T=3 Baseline 2 1,000 2700 82.4 82.4 82.4 82 71 71 69

4 1,000 2700 85 83 85 83 68 68 74

2 10,000 27,000 - 82.2 - 83 - 73 70.6

4 10,000 27,000 - 80 - 80 - 66 73

Table 2: On training set of 70% links from simulated graphs of dimension *d*, size *n*, edges *e*, number of pairs in AP-set of GRANCH and number of pairs examined by algorithm *C* (with *T* = {10*,* 6*,* 3}) and the baseline.

*d n e* C,T=10 GRANCH,T=10 C,T=6 GRANCH,T=6 C,T=3 GRANCH,T=3 Baseline 2 1,000 2,700 1M 24,594 1M 29,656 1M 13,521 34,902

4 1,000 2,700 1M 47,846 1M 45,623 1M 14,839 72,646

2 10,000 27,000 - 257,110 - 311,944 - 141,136 381,564

4 10,000 27,000 - 503,998 - 478,409 - 170,144 781,515

of any node is at most *O*(*cρG* ). For example the 2D **6.2 Real world graphs**

grid graph has *ρ* = 2, and in general a *k* dimensional In this section we perform similar experiments as in hypergrid has *ρ* = *k*. the last section on real world graphs. We use sub-

We generate graphs of growth-rate *d*, by assigning *d* sets of the Citeseer network of sizes: 20,000 nodes dimensional Euclidian coordinates to the nodes, and with 55,000 edges, 50,000 nodes with 160,000 edges adding links between close neighbors plus some long and 100,000 nodes with 280,000 edges. These graphs distance links. This brings about the very famous are connected but sparse. We present the total number *small world phenomenon* (Milgram, 1967).We do the of pairs within 4 hops of one another in the training following experiments, graphs in table 5. These numbers reflect the small world phenomenon in real-world social networks. Ev-

1. AUC scores on held out data ery node can reach every other node within a few hops.

2. AUC scores on data with noisy links This also emphasizes the fact that its not even feasible

We only present the performance of the exact trun- to take all nodes within *T* hops to compute the trun- cated commute time for the 1*,* 000-node graph, since cated commute times exactly. For reasonable values of the computation becomes too expensive for the *T* , the above will be a huge fraction of all pairs.

10*,* 000-node graph. As in Table 1, our algorithm beats

the baseline algorithm, and does almost as well as Table 5: Number of pairs within 4 hops in training the exact truncated commute time(for the 1000 node graphs(90% links from Citeseer) of different sizes. graph) without looking at all pairs of nodes.

*n e* # of pairs within 4 hops

Also from Table 2 we can see how for a given number 19,477 54,407 3,401,812 of nodes and edges the dimensionality of the graph 52,033 152,518 20,478,196 affects the number of pairs needed by both GRANCH 103,500 279,435 88,418,488 and the baseline algorithm.

**Noisy Data**: We generate an example where we make

a dataset noisy by adding links from a randomly cho- We remove a randomly chosen 10% of the links to ob- sen node to 20% of the remaining nodes. Now we hold tain the training and test graphs, with the same set- out 30% links from this graph, and test the AUC score ting of *T j*. For these datasets, the exact truncated of our algorithm. We expect to see that the commute commute time is intractable. In Table 4 note that the times outperform the baseline, since it’s much more ro- baseline has to compute hops between number of pairs bust to noise. For a 100-node graph, GRANCH with that is more that one order of magnitude larger than *T* = 6 has an AUC score of 85*.*3%, whereas the number the number of pairs considered by GRANCH. Also the

of hops has an AUC score of 70*.*23%. numbers in Table 4 are only for the nodes which had at

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

SARKAR & MOORE 343

Table 3: AUC score of GRANCH and the baseline on 10% links held out from graphs of size *n*, edges *e* with

*T* = 10; *T* = 6; *T* = 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *n* | *e* | GRANCH,*T* = 10 | GRANCH,*T* = 6 | GRANCH,*T* = 3 | Baseline |
| 19477 | 54407 | 82 | 80 | 78 | 74 |
| 52033 | 152518 | 85 | 84 | 78.6 | 79 |
| 103500 | 279435 | 86.7 | 86 | 79 | 83 |

Table 4: On the training set of 90% links from graphs of size *n*, edges *e*, number of pairs in AP-set of GRANCH with *T* = 10; *T* = 6; *T* = 3; and number of node-pairs used by the baseline.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *n* | *e* | GRANCH,*T* = 10 | GRANCH,*T* = 6 | GRANCH,*T* = 3 | Baseline |
| 19,477 | 54,407 | 600,000 | 335,769 | 279,736 | 1,336,683 |
| 52,033 | 152,518 | 2,017,931 | 1,532,534 | 293,366 | 8,723,436 |
| 103,500 | 279,435 | 5,143,982 | 4,156,792 | 1,661,598 | 40,563,881 |



least one edge held out from the original graph. In the worst case these numbers could be as large as in Ta- ble 5. The baseline algorithm performs quite well for these datasets indicating existence of transitive trends in co-authorship networks.

# Conclusion and Future Work

This paper has addressed an important gap in graph proximity measures: that the important measures are also the hardest to compute. We designed an algo- rithm that guarantees an *s*-error and showed empiri-

cally that it avoids *O*(*n*2) computation or storage in practice while retaining the predictive performance of

the expensive commute time methods.

For future directions we will like to analyze the run- time of GRANCH. We would also like to do a detailed comparison of our algorithm with different link predic- tion algorithms. Currently we set the parameter *T* for truncation at different values to have different behav- ior of the algorithm. It would be interesting to choose *T* dynamically based on the mixing time of the graph.

# Acknowledgements

We are very grateful to Jeremy Kubica for his helpful com- ments and suggestions. This work was done while the first author was an intern at Google Pittsburgh.

# References

Aldous, D., & Fill, J. A. (2001). Reversible markov chains. Brand, M. (2005). A Random Walks Perspective on Max-

imizing Satisfaction and Profit. *SIAM ’05*.

Chandra, A. K., Raghavan, P., Ruzzo, W. L., & Smolensky,

R. (1989). The electrical resistance of a graph captures its commute and cover times. *STOC’89* (pp. 574–586).

Doyle, P. G., & Snell, J. L. (1984). *Random walks and electric networks.* The Mathematical Assoc. of America.

Gorelick, L., Galun, M., Sharon, E., Basri, R., & Brandt,

A. (2004). Shape representation and classification using the poisson equation. *CVPR*.

Grady, L., & Schwartz, E. L. (2006a). Isoperimetric graph partitioning for data clustering and image segmentation. *IEEE PAMI*.

Grady, L., & Schwartz, E. L. (2006b). Isoperimetric parti- tioning: A new algorithm for graph partitioning. *SIAM Journal on Scientific Computing*.

Katz, L. (1953). A new status index derived from socio- metric analysis. *Psychometrika*.

Krauthgamer, R., & Lee, J. R. (2003). The intrinsic di- mensionality of graphs. *Proceedings of the STOC’03*.

Liben-Nowell, D., & Kleinberg, J. (2003). The link predic- tion problem for social networks. *CIKM ’03*.

Lovasz, L. (1996). Random walks on graphs: a survey. Milgram, S. (1967). The small world problem. *Psychology*

*Today*.

Ng, A., Jordan, M., & Weiss, Y. (2001). On spectral clus- tering: Analysis and an algorithm.

Qiu, H., & Hancock, E. R. (2005). Image segmentation using commute times. *Proc. BMVC*.

Qiu, H., & Hancock, E. R. (2006). Robust multi-body mo- tion tracking using commute time clustering. *ECCV’06*.

Saerens, M., Fouss, F., Yen, L., & Dupont, P. (2004). The principal component analysis of a graph and its relation- ships to spectral clustering.

Smola, A., & Kondor, R. (2003). Kernels and regulariza- tion on graphs.

Yen, L., Vanvyve, L., Wouters, D., Fouss, F., Verleysen, F., & Saerens, M. (2005). Clustering using a random-walk based distance measure.

Zhu, X., Ghahramani, Z., & Lafferty, J. (2003). Semi- supervised learning using gaussian fields and harmonic functions.