Fast Direction-Aware Proximity for Graph Mining

Hanghang Tong
Carnegie Mellon University
htong@cs.cmu.edu

Yehuda Koren
AT&T Labs – Research
yehuda@research.att.com

Christos Faloutsos
Carnegie Mellon University
christos@cs.cmu.edu

ABSTRACT

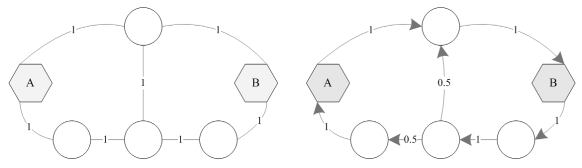
In this paper we study asymmetric proximity measures on directed graphs, which quantify the relationships between two nodes or two groups of nodes. The measures are useful in several graph mining tasks, including clustering, link prediction and connection subgraph discovery. Our proximity measure is based on the concept of *escape probability*. This way, we strive to summarize the multiple facets of nodes-proximity, while avoiding some of the pitfalls to which alternative proximity measures are susceptible. A unique feature of the measures is accounting for the underlying directional information. We put a special emphasis on computational efficiency, and develop fast solutions that are applicable in several settings. Our experimental study shows the usefulness of our proposed direction-aware proximity method for several applications, and that our algorithms achieve a significant speedup (up to 50,000x) over straightforward implementations.

1. INTRODUCTION

Measuring node proximity is a fundamental problem in many graph mining settings. While most of existing measurements are (implicitly or explicitly) designed for undirected graphs; edge directions in the graph provide a new perspective to proximity measurement: measuring the proximity *from* A *to* B; rather than *between* A *and* B (See Figure 1). Here, we study the role of direction in measuring proximity on graphs. To be specific, we will try to answer the following questions in this paper:

- **Q1 Problem definitions:** How to define a direction-aware proximity? How to generalize the definition to measure the proximity for two groups of nodes?
- **Q2 Computational issues:** How to compute the proximity score efficiently in all settings of interest?
- **Q3 Applications:** How can direction-aware proximity benefit graph mining?

We begin (Section 2) by proposing a novel direction-aware proximity definition, based on the notion of escape probability of random walks. It is carefully designed to deal with practical problems such as the inherent noise and uncertainties associated with real life networks. Moreover, the proposed definition is generalized to measure group proximities by defining group escape probability. Then, in Section 3, we address computational efficiency, by concentrating on two scenarios: (1) the computation of a single proximity on a large, disk resident graph (with possibly millions of nodes). (2) The computation of multiple pairwise proximities on a medium sized graph (with up to a few tens of thousand nodes). For the former scenario, we develop an iterative solution to avoid matrix inversion, with convergence guarantee. For the latter scenario, we develop an efficient solution, which requires only a *single* matrix inversion, making careful use of the so-called block-matrix inversion lemma. Finally (Sections 4-5), we apply our direction-aware proximity to some real life problems. We demonstrate some encouraging results of the proposed direction-aware proximity for predicting the existence of links together with their direction. Other applications include directed center-piece subgraphs, and attribute graphs.



(a) Undirected graph (b) Directed graph

Figure 1: On undirected graphs (a) we are interested in the proximity between A and B, whereas in a directed graph (b) we make the distinction between the proximity from A to B, or from B to A

2. DIRECTION-AWARE PROXIMITY

Here we give the main definitions behind our proposed node-to-node proximity measure, namely, the *escape probability*; then we give the justification for our modifications to it; and finally we generalize our definition to handle the group-to-group proximity.

2.1 Node-to-Node Proximity

Let us start with the node-to-node proximity score, or simply *node proximity*. (We will introduce a more general, group proximity score, in Subsection 2.3.) Following some recent works [6, 15, 19], our definition is based on properties of random walks associated with the graph. Random walks mesh naturally with the random nature of the self-organizing networks that we deal with here. Importantly, they allow us to characterize relationships based on multiple paths. Random walk notions are known to parallel properties of corresponding electric networks [5]. For example, this was the basis for the work of Faloutsos et al. [6] that measured nodes proximity by employing the notion of effective conductance. Since electric networks are inherently undirected, they cannot be used for our desired directed proximity measure. Nonetheless, the effective conductance can be adequately generalized to handle directional information by using the concept of *escape probability* [5]:

DEFINITION 1. *The* escape probability *from node i to node j, $ep_{i,j}$, is the probability that the random particle that starts from node i will visit node j before it returns to node i*

Thus we adopt the escape probability as the starting point for our direction-aware node-to-node proximity score. That is, for the moment, we define the proximity $\mathrm{Prox}(i, j)$ *from* nodes i *to* j as exactly $ep_{i,j}$.

An important quantity for the computation of $ep_{i,j}$ is the *generalized voltage* at each of the nodes, denoted by $v_k(i, j)$: this is defined as the probability that a random particle that starts from node k will visit node j before node i. This way, our proximity measure can be stated as:

$$\mathrm{Prox}(i, j) \triangleq ep_{i,j} = \sum_{k=1}^{n} p_{i,k} \cdot v_k(i, j) \tag{1}$$

where $p_{i,k}$ is the probability of a direct transition from node i to node k.

For example, in Figure 1(b), we have $\mathrm{Prox}(A, B) = 1$ and $\mathrm{Prox}(B, A) = 0.5$, which is consistent with our intuition that connections based of longer paths should be weaker. Table 1 gives a list of symbols used in this paper. Following standard notation, we use calligraphic font for sets (e.g., $\mathcal{V}$), bold capitals for matrices (e.g., $\mathbf{W}$, $\mathbf{P}$), and arrows for column vectors (e.g., $\vec{1}$).

Table 1: Symbols

Symbol	Definition
$\mathbf{W} = [w_{i,j}]$	the weighted graph, $1 \leq i, j \leq n$, $w_{i,j}$ is the weight of edge $i \longrightarrow j$
$\mathbf{A}^T$	the transpose of matrix $\mathbf{A}$
$\mathbf{D}$	$n \times n$ diagonal matrix of out-degrees: $D_{i,i} = \sum_j w_{i,j}$ and $D_{i,j} = 0$ for $i \neq j$
$\mathbf{P} = [p_{i,j}]$	the transition matrix associated with the graph
$\mathbf{G} = [g_{i,j}]$	the G-matrix associated with $\mathbf{P}$: $\mathbf{G} = (\mathbf{I} - c\mathbf{P})^{-1}$
$\mathcal{V}$	the whole set of the nodes $\mathcal{V} = \{1, .., n\}$
$\mathcal{A}, \mathcal{B}$	two groups of nodes $\mathcal{A} = \{i_1, ...i_{\|\mathcal{A}\|}\}$, $\mathcal{B} = \{j_1, ...j_{\|\mathcal{B}\|}\}$, $\mathcal{A}, \mathcal{B} \subseteq \mathcal{V}$
$\mathbf{P}(\mathcal{A}, \mathcal{B})$	a block of matrix $\mathbf{P}$: $\mathbf{P}(\mathcal{A}, \mathcal{B})= [p_{i,j}]$, $(i \in\mathcal{A}, j \in \mathcal{B})$
$\mathbf{P}(i, :)$	i^{th} row of matrix $\mathbf{P}$
$\mathbf{P}(:, j)$	j^{th} column of matrix $\mathbf{P}$
$\mathrm{Prox}(i, j)$	node proximity from node i to node j
$\mathrm{Prox}(\mathcal{A}, \mathcal{B})$	group proximity from group $\mathcal{A}$ to group $\mathcal{B}$
$\vec{1}$	a (column) vector whose all elements are 1's
$\vec{e}_i$	a column vector whose i^{th} element is 1 and the rest elements are 0's
c	$1 - c$ is the transition probability from each node to the sink
n	the total number of the nodes in the graph
m	the maximum number of iterations

2.2 Practical Modifications

Given a weighted directed graph $\mathbf{W}$, there is a natural random walk associated with it whose transition matrix $\mathbf{P}$ is the normalized version of $\mathbf{W}$, defined as $\mathbf{P} = \mathbf{D}^{-1}\mathbf{W}$. Recall that $\mathbf{D}$ is the diagonal matrix of the node out-degrees (specifically, sum of outgoing weights). However, for real problems, this matrix leads to escape probability scores that might not agree with human intuition. In this subsection, we discuss three necessary modifications, to improve the quality of the resulting escape probability scores.

2.2.1 *Augmenting the network with a universal sink*

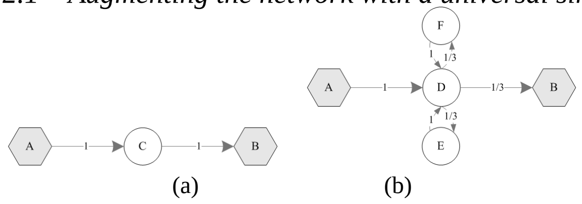


Figure 2: Nodes F and E have no influence on the A → B escape probability

When measuring the escape probability from i to j we assume that the random particle must eventually reach either i or j. This means that no matter how long it wanders around, it will make unlimited tries, till reaching i or j.

This ignores any noise or friction that practically exist in the system and causes the particle to disappear or decay over time. In particular, this problem is manifested in dead-end paths, or degree-1 nodes, which are very common in practical networks whose degree distribution follows a power law. We demonstrate this in Figure 2 where, intuitively, nodes E and F, distract the A-D-B connection, so the proximity from A to B in (a) should be larger than that in (b). However, in terms of the escape probability, they are the same (both equal 1). In other words, the influence of those degree-1 nodes (E and F) is not taken into account. To address this issue, we model the friction in the system by augmenting it with a new node with a zero out degree known as *the universal sink* (mimicking an absorbing boundary): Now, each node has some small transition probability, 1-c, to reach the sink; and whenever the random particle has reached the sink, it will stay there forever. For example, if we add a sink to Figure 2 with $c = 0.9$, Prox(A,B) is 0.81 in (a), but only 0.74 in (b), which is consistent with our intuition. Note that in this case, equation (1) becomes:

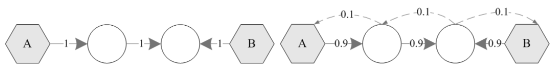
$$ep_{i,j} = \sum_{k=1}^{n} c \cdot p_{i,k} \cdot v_k(i, j) \tag{2}$$

2.2.2 *Partial symmetrization*

A weakly connected pair is two nodes that are not connected by any directed path, but become connected when considering undirected paths (see, e.g., Figure 3(a)). For such weakly connected pairs, the direction-aware proximity will be zero. However, in some situations, especially when there are missing links in the graph, this might not be desirable. In fact, while we strive to account for directionality, we also want to consider some portion of the directed link as an undirected one. For example, in phone-call networks, the fact that person A called person B, implies some symmetric relation between the two persons and thereby a greater probability that B will call A (or has already called A but this link is missing).

A random walk modeling of the problem gives us the flexibility to address this issue by introducing lower probability *backward* edges:

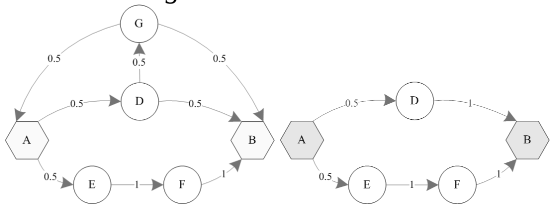
whenever we observe an edge $w_{i,j}$ in the original graph, we put another edge in the opposite direction $w_{j,i} \propto w_{i,j}$. In other words, we replace the original $\mathbf{W}$ with $(1-\beta)\mathbf{W} + \beta\mathbf{W}^T (0 < \beta < 1)$. For example, in Figure 3(b), by introducing backward edges with $\beta = 0.1$, we get $\text{Prox}(B, A) = 0.009$, which is much smaller than $\text{Prox}(A, B) = 0.081$, but nonetheless greater than zero. Note that $\beta = 0.5$ is equivalent to ignoring edge directions.



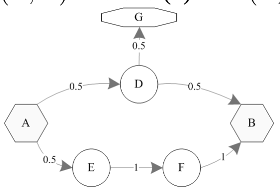
(a) (b)

Figure 3: Dealing with weak connections by partial symmetrization

2.2.3 Preventing size bias



(a) $\text{Prox}(A, B) = 0.875$ (b) $\text{Prox}(A, B) = 1$



(c) $\text{Prox}(A, B) = 0.75$

Figure 4: Dealing with size bias: (a) is the original graph; (b) and (c) are candidate graphs with and without out-degree preservation, respectively.

Many of the graphs we deal with are huge, so when computing the proximity, a frequently used technique is to limit its computation to a much smaller *candidate graph*, which will significantly speed-up the computation. Existing techniques [6, 15], are looking a for a moderately sized graph that contains the relevant portions of the network (relatively to a given proximity query), while still enabling quick computation. These techniques can be directly employed when computing our proximity measure.

As pointed out in [15], for a robust proximity measure, the score given by the candidate graph should not be greater than the score which is based on the full graph. The desired situation is that the proximity between two nodes will monotonically converge to a stable value as the candidate graph becomes larger. However, this is often not the case if we compute the escape probability directly on the candidate graph (see Figure 4(a) and (b) for an example)

To address this issue, we should work with degree preserving candidate graphs. That is, for every node in the candidate graph, its out degree is the same as in the original graph. Now in Figure 4(c), by preserving degrees, the proximity from A to B is smaller than that in (a). More precisely, we have the following lemma:

LEMMA 1. *Consider two nodes i and j, and let Prox^{Ori} be their proximity score computed on the original graph, and Prox^{Cand} be the one computed on the degree preserving candidate graph. Then we have $\text{Prox}^{Ori} \geq \text{Prox}^{Cand}$, for any pair of nodes i and j.*

Proof: Omitted for brevity

2.3 Group-to-Group Proximity

Now, let us generalize our directed proximity measure to measure the proximity between two groups of nodes. To this end, we define the group escape probability as:

- $gep_{\mathcal{A},\mathcal{B}}$: the probability that a random particle, starting from *any* node in group $\mathcal{A}$ (uniformly chosen), will visit *any* node in group $\mathcal{B}$, before it returns to *any* node in group $\mathcal{A}$.

Analogously, we define:

- $v_k(\mathcal{A},\mathcal{B})$: the probability that a random particle, starting from node k, will visit *any* node in group $\mathcal{B}$, before it visits *any* node in group $\mathcal{A}$

Therefore our group proximity is defined as follows:

$$\text{Prox}(\mathcal{A},\mathcal{B}) \triangleq gep_{\mathcal{A},\mathcal{B}} = \frac{1}{|\mathcal{A}|}\sum_{i\in\mathcal{A}}\sum_{k=1}^{n} c \cdot p_{i,k} \cdot v_k(\mathcal{A},\mathcal{B}) \quad (3)$$

From equations (2) and (3), we see that the node-to-node proximity is a special case of the group-to-group proximity, by setting $\mathcal{A} = \{i\}$ and $\mathcal{B} = \{j\}$.

3. FAST SOLUTIONS

Here we deal with the computational aspects of the directed proximity measures defined by equations (2) and (3). First, we will show that straight-forward ways to compute these proximities correspond to solving a specific linear system, which involves a matrix inversion. Then, we will propose fast solutions for computing a single proximity on a large graph or all pairwise proximities on a medium sized graph. Notice that these fast solutions will be also beneficial for measuring undirected proximity.

3.1 Straightforward Solvers

3.1.1 Node proximity computation

Node proximity is based on the linear system [5]:

$$\begin{aligned} v_k(i,j) &= \sum_{t=1}^{n} c \cdot p_{k,t} v_t(i,j) (k \neq i,j) \\ v_i(i,j) &= 0;\ \ and\ \ v_j(i,j) = 1 \end{aligned} \quad (4)$$

By solving the above linear system, we have [5]:

$$\text{Prox}(i,j) = c^2 \mathbf{P}(i,\mathcal{I})\mathbf{G}''\mathbf{P}(\mathcal{I},j) + c p_{i,j} \quad (5)$$

where $\mathcal{I} = \mathcal{V} - \{i,j\}$; $\mathbf{P}(i,\mathcal{I})$ is the i^{th} row of $\mathbf{P}$ without i^{th} and j^{th} elements; $\mathbf{P}(\mathcal{I},j)$ is the j^{th} column of $\mathbf{P}$ without i^{th} and j^{th} elements; and

$$\mathbf{G}'' = (\mathbf{I} - c\mathbf{P}(\mathcal{I},\mathcal{I}))^{-1} \quad (6)$$

3.1.2 Group proximity computation

Using a similar procedure, we reach the following lemma for computing group proximity:

LEMMA 2. *Let $\mathcal{C} = \mathcal{A} \cap \mathcal{B}$. If $\mathcal{C} = \Phi$, the group proximity – $\mathrm{Prox}(\mathcal{A},\mathcal{B})$ of equation* (3) *– is determined by equation* (7)*; otherwise, it is determined by equation* (8).

$$\mathrm{Prox}(\mathcal{A},\mathcal{B}) = \frac{\vec{1}^T(c^2\mathbf{P}(\mathcal{A},\mathcal{I})\mathbf{G}''\mathbf{P}(\mathcal{I},\mathcal{B}) + c\mathbf{P}(\mathcal{A},\mathcal{B}))\vec{1}}{|\mathcal{A}|} \quad (7)$$

$$\mathrm{Prox}(\mathcal{A},\mathcal{B}) = \frac{|\mathcal{A}-\mathcal{C}|\mathrm{Prox}(\mathcal{A}-\mathcal{C},\mathcal{B}-\mathcal{C}) + |\mathcal{C}|}{|\mathcal{A}|} \quad (8)$$

where $\mathcal{I} = \mathcal{V} - (\mathcal{A}\cup\mathcal{B})$, $\mathcal{A}-\mathcal{C} = \{i, i \in \mathcal{A},\ i \notin \mathcal{C}\}$, $\mathcal{B}-\mathcal{C} = \{i, i \in \mathcal{B},\ i \notin \mathcal{C}\}$, and $\mathbf{G}'' = (\mathbf{I} - c\mathbf{P}(\mathcal{I}))^{-1}$

Proof: Omitted for brevity. □

3.1.3 Analysis of computational bottlenecks

Both equations (5) and (7) involve $\mathbf{G}''$ – an inversion of a matrix. Suppose $|\mathcal{A}| \ll n$ and $|\mathcal{B}| \ll n$,[1] the major computational cost in both cases is the inversion of an $n \times n$ matrix, which brings up two computational efficiency challenges:

1. (**Setting 1**) For a large graph, say with hundreds of thousands of nodes, matrix inversion would be very slow if not impossible. In this case we would like to completely avoid matrix inversion when computing a single proximity.
2. (**Setting 2**) The above computation requires a *different* matrix inversion for each proximity value. Thus, computing k proximities, requires performing k separate matrix inversions. We will show how to eliminate all matrix inversions, but one, regardless of the number of needed proximities.

3.2 Setting 1: Computing a Single Proximity

3.2.1 Fast direction-aware node proximity

We propose *FastOneDAP* (Table 2), a fast iterative solution for computing one node proximity.

Table 2: *FastOneDAP*

```
Input: The transition matrix $\mathbf{P}$, $c$, the starting node $i$
and the ending node $j$
Output: The proximity from $i$ to $j$ : $\widehat{\mathrm{Prox}}(i,j)$
1. Initialize:
    1.1 If $i > j, i_0 = i$; else, $i_0 = i - 1$
    1.2 $\vec{v}^T = \vec{y}^T = \vec{e}^T_{i_0}$
2. Iterate until convergence:
    2.1 $\vec{y}^T \leftarrow c\vec{y}^T\mathbf{P}(\mathcal{V}-\{j\},\mathcal{V}-\{j\})$
    2.2 $\vec{v}^T \leftarrow \vec{v}^T + \vec{y}^T$
3. Normalize: $\vec{v}^T \leftarrow \frac{\vec{v}^T}{\vec{v}^T(i_0)}$;
4. Return: $\widehat{\mathrm{Prox}}(i,j) = c\vec{v}^T\mathbf{P}(\mathcal{V}-\{j\},j)$
```

In *FastOneDAP* , the major computational cost lies in the iterative step. Let m be the maximum number of iterations and E the number of total edges in the graph, The complexity of *FastOneDAP* is $O(mE)$. In other words, *FastOneDAP* is linear in the number of edges in the graph. Often, real graphs are very sparse, which means that E is much less than n^2 (e.g., for typical graphs that obey power law [7]: $E \propto n^{1+\epsilon}$). Thus, *FastOneDAP* is significantly more efficient than the straightforward solver, whose running time is $O(n^3)$ due to matrix inversion.

[1]On the other hand, if this is not true, we can always directly get $\mathrm{Prox}(\mathcal{A},\mathcal{B})$ by equation (7) efficiently. Thus, throughout this paper, we suppose that the size of the group is always much smaller than that of the whole graph.

Before discussing the correctness of *FastOneDAP* , we need the following block matrix inversion lemma, which plays an essential role in our fast solutions:

LEMMA 3. ***Block Matrix Inversion Lemma****: Partition the matrix $\mathbf{M}$ into four blocks:*

$$\mathbf{M} = \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} \quad (9)$$

Then,

$$\mathbf{M}^{-1} = \begin{pmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}\mathbf{S}^{-1}\mathbf{C}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}\mathbf{S}^{-1} \\ -\mathbf{S}^{-1}\mathbf{C}\mathbf{A}^{-1} & \mathbf{S}^{-1} \end{pmatrix} \quad (10)$$

where $\mathbf{S} = \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}$

Proof: See, e.g., [10] □

Based on the block matrix inversion lemma, the correctness of *FastOneDAP* is guaranteed by the following lemma.

LEMMA 4. *The value $\widehat{\mathrm{Prox}}(i,j)$, as computed by FastOneDAP converges to the proximity $\mathrm{Prox}(i,j)$ defined in equation* (5).

Proof: Without loss of generality, we only need to prove the case when $i = n-1$ and $j = n$. Let $\mathcal{V}_{n-1} = \mathcal{V} - \{j\} = \{1, ..., n-1\}$ and $\mathcal{V}_{n-2} = \mathcal{V} - \{i,j\} = \{1, ..., n-2\}$. (for uniformity, from now on, we refer $\mathcal{V}$ as $\mathcal{V}_n$.) With this notation, we have $i_0 = n-1$ and $\mathbf{G}'' = [g''_{i,j}] = (\mathbf{I} - c\mathbf{P}(\mathcal{V}_{n-2},\mathcal{V}_{n-2}))^{-1}$.

$$\text{Define: } \mathbf{G}' = [g'_{i,j}] = (\mathbf{I} - c\mathbf{P}(\mathcal{V}_{n-1},\mathcal{V}_{n-1}))^{-1} \quad (11)$$

First, Note that as the iteration k of step 2 goes to infinite:

$$\begin{aligned} \|\vec{y}^T\| &= \|\vec{e}_i^T c^k \mathbf{P}(\mathcal{V}_{n-1},\mathcal{V}_{n-1})^k\| \\ &\le c^k\|\vec{e}_i^T\|\|\mathbf{P}(\mathcal{V}_{n-1},\mathcal{V}_{n-1})^k\| \\ &\le c^k\|\mathbf{P}\| \le c^k \to 0\,, \end{aligned} \quad (12)$$

which proves the convergence of step 2 in *FastOneDAP* .

Next, by Taylor expansion, we have the following equation:

$$\begin{aligned} \vec{v}^T &= \sum_{k=0}^{\infty} \vec{e}_i^T c^k \mathbf{P}(\mathcal{V}_{n-1},\mathcal{V}_{n-1})^k \\ &= \vec{e}_i \sum_{k=0}^{\infty} c^k \mathbf{P}(\mathcal{V}_{n-1},\mathcal{V}_{n-1})^k \\ &\to \vec{e}_i \cdot (\mathbf{I} - c\mathbf{P}(\mathcal{V}_{n-1},\mathcal{V}_{n-1}))^{-1} \\ &= \mathbf{G}'(i,:) \end{aligned} \quad (13)$$

which proves that at the end of step 2, $\vec{v}^T$ will converge to the i^{th} row of $\mathbf{G}'$.

Furthermore, by the lemma for block matrix inversion, we have:

$$\begin{aligned} s &= c^2\mathbf{P}(i,\mathcal{V}_{n-1})\mathbf{G}''\mathbf{P}(\mathcal{V}_{n-1},i) \\ g'_{i,i} &= \frac{1}{1-cp_{i,i}-s} \\ \mathbf{G}'(i,\mathcal{V}_{n-2}) &= c\mathbf{P}(i,\mathcal{V}_{n-2})\mathbf{G}''g'_{i,i} \end{aligned} \quad (14)$$

Finally, we have:

$$\begin{aligned} \widehat{\mathrm{Prox}}(i,j) &= \vec{v}^T c\mathbf{P}(\mathcal{V}_{n-1},j) \\ &\to \frac{\mathbf{G}'(i,:)}{g'_{i,i}} c\mathbf{P}(\mathcal{V}_{n-1},j) \\ &= c\mathbf{P}(i,\mathcal{V}_{n-2})\mathbf{G}''\mathbf{P}(\mathcal{V}_{n-2},j)+cp_{i,j} \\ &= \mathrm{Prox}(i,j) \end{aligned} \quad (15)$$

which completes the proof of lemma 4. □

3.2.2 *Fast direction-aware group proximity*

Similarly, we can develop the following fast algorithm (*FastOneGDAP* ; Table 3) for computing one group proximity on a large graph.

$$\mathbf{P}' = \begin{pmatrix} \mathbf{P}(\mathcal{I},\mathcal{I}) & \mathbf{P}(\mathcal{I},\mathcal{A})\vec{1} \\ \vec{1}^T\mathbf{P}(\mathcal{A},\mathcal{I}) & \vec{1}^T\mathbf{P}(\mathcal{A},\mathcal{A})\vec{1} \end{pmatrix} \quad (16)$$

where $\mathcal{I} = \mathcal{V} - (\mathcal{A} \cup \mathcal{B})$

Table 3: *FastOneGDAP*

```
Input: The transition matrix $\mathbf{P}$, $c$, the starting group $\mathcal{A}$
and the ending group $\mathcal{B}(\mathcal{A} \cap \mathcal{B} = \Phi)$
Output: The group proximity from $\mathcal{A}$ to $\mathcal{B}$ : $\widehat{\mathrm{Prox}}(\mathcal{A},\mathcal{B})$
1. Initialize:
   1.1 Define matrix $\mathbf{P}'$ as in equation (16)
   1.2 $\vec{v}^T = \vec{y}^T = \vec{e}^T_{i_0}$, $(i_0 = |\mathcal{V} - \mathcal{A} - \mathcal{B}| + 1)$
2. Iterate until convergence:
   2.1 $\vec{y}^T \leftarrow c\vec{y}^T\mathbf{P}'$
   2.2 $\vec{v}^T \leftarrow \vec{v}^T + \vec{y}^T$
3. Normalize: $\vec{v}^T \leftarrow \frac{\vec{v}^T}{\vec{v}^T(i_0)}$;
4. Return: $\widehat{\mathrm{Prox}}(\mathcal{A},\mathcal{B}) = \frac{1}{|\mathcal{A}|}c\vec{v}^T\mathbf{P}(\mathcal{V}-\mathcal{B},\mathcal{B})\vec{1}$
```

As in *FastOneDAP* , let m be the maximum number of iterations and E' the total number of edges in $\mathbf{P}'$, the complexity of *FastOneGDAP* is $O(mE')$. Suppose $|\mathcal{A}| \ll n$ and $|\mathcal{B}| \ll n$, we have $E' \approx E$. Thus, the complexity of *FastOneGDAP* is $O(mE)$. Similarly, we can prove the following lemma for *FastOneGDAP* .

LEMMA 5. *The group proximity* $\widehat{\mathrm{Prox}}(\mathcal{A},\mathcal{B})$ *by FastOneGDAP converges to the proximity* $\mathrm{Prox}(\mathcal{A},\mathcal{B})$ *by equation* (7).

Proof: Omitted for brevity. □

3.3 Setting 2: Computing Multiple Proximities

Earlier we showed how to make the computations efficient, when we have only one pair of nodes (or one pair of groups) for which we want the proximity score. Here we show how to estimate multiple such pairs of scores very efficiently.

3.3.1 *Fast direction-aware all-pairs proximity*

Suppose that we want to compute all $n(n-1)$ pairwise node proximities. There are various situations where one might want to compute all (or many) proximities. First, collecting all proximities and studying their distribution can reveal interesting features of the network and tell us about its global structure. In addition, some algorithms – such as distance based clustering – require the knowledge of all (or at least many) pairwise proximities.

Computation of many pairwise proximities in the same network involves solving many linear systems (in fact, one matrix inversion for each proximity). However, there is a lot of redundancy among different linear systems. In fact, we propose a much more efficient method, that need only solve one linear system (or, invert a single matrix) and leverage its result to quickly solve all the others. Consequently, we suggest the *FastAllDAP* algortihm (Table 4).

Table 4: *FastAllDAP*

```
Input: The transition matrix $\mathbf{P}$, $c$
Output: All proximities $\mathbf{Pr}' = [\widehat{\mathrm{Prox}}(i,j)]_{1 \leqslant i \neq j \leqslant n}$
1. Compute $\mathbf{G} = (\mathbf{I} - c\mathbf{P})^{-1}$
2. For $i = 1 : n$
      For $j = 1 : n$ $(j \neq i)$
           Compute: $\widehat{\mathrm{Prox}}(i,j) = \frac{g_{i,j}}{g_{i,i}g_{j,j} - g_{i,j}g_{j,i}}$
      EndFor
   EndFor
```

The major benefit of *FastAllDAP* is the dramatic reduction of matrix inversion operations from $n(n-1)$ to a single one. Its correctness is guaranteed by the following lemma:

LEMMA 6. *FastAllDAP gives exactly the same result as equation* (5) .

Proof: Without loss of generality, we only need to prove that when $i = n-1$ and $j = n$, $\widehat{\mathrm{Prox}}(n-1,n) = \mathrm{Prox}(n-1,n)$ holds. Let $\mathbf{G}'$ be defined as in equation (11).

According to the proof of Lemma 4 (Eq. 15), we have

$$\mathrm{Prox}(n-1,n) = \frac{\mathbf{G}'(n-1,:)c\mathbf{P}(\mathcal{V}_{n-1},n)}{g'_{n-1,n-1}} \quad (17)$$

Furthermore, by applying the lemma of block matrix inversion [10] on $\mathbf{G}$ and $\mathbf{G}'$, we have:

$$\begin{aligned} \mathbf{G}'(n-1,:)c\mathbf{P}(\mathcal{V}_{n-1},n) &= \frac{g_{n-1,n}}{g_{n,n}} \\ g'_{n-1,n-1} &= g_{n-1,n-1} - \frac{g_{n-1,n}g_{n,n-1}}{g_{n,n}} \end{aligned} \quad (18)$$

Combining equations (17) and (18) we have the proof. □

3.3.2 *Fast direction-aware all-pairs group proximity*

Let $\{\mathcal{A}_1,...,\mathcal{A}_{n_1}\}$, $\{\mathcal{B}_1,...,\mathcal{B}_{n_2}\}$ be two sets of groups. To compute all $n_1 \times n_2$ induced group proximities, we employ the *FastManyGDAP* algorithm (Table 5).

Suppose that $|\mathcal{A}_i| \ll n$ and $|\mathcal{B}_j| \ll n$. Then, compared with step 1, the computational cost of Step 2.1.1 and 2.1.3 can be ignored. Thus, if we take matrix inversion as the basic operation,

Table 5: *FastManyGDAP*

```
Input: The transition matrix $\mathbf{P}$, $c$, two sets of groups:
        $\{\mathcal{A}_1, ..., \mathcal{A}_{n_1}\}$,$\{\mathcal{B}_1, ..., \mathcal{B}_{n_2}\}$
Output: $\mathbf{Pr}' = [\text{Prox}(\mathcal{A}_i, \mathcal{B}_j)]_{1 \leqslant i \leqslant n_1, 1 \leqslant j \leqslant n_2}$
1. Compute $\mathbf{G} = (\mathbf{I} - c\mathbf{P})^{-1}$
2. For $i = 1 : n_1$
     For $j = 1 : n_2$
       2.1 If $\mathcal{A}_i \neq \mathcal{B}_j$:
         2.1.1 $\mathbf{M}_0 = \mathbf{G}(\mathcal{B}_j, \mathcal{B}_j)^{-1}$
         2.1.2 $\mathbf{M} \leftarrow \mathbf{G}(\mathcal{A}_i, \mathcal{B}_j)\mathbf{M}_0\mathbf{G}(\mathcal{B}_j, \mathcal{A}_i)$
         2.1.3 $\mathbf{M} \leftarrow (\mathbf{G}(\mathcal{A}_i, \mathcal{A}_i) - \mathbf{M})^{-1}$
         2.1.4 $\mathbf{M} \leftarrow \mathbf{M}\mathbf{G}(\mathcal{A}_i, \mathcal{B}_j)\mathbf{M}_0$
         2.1.5 $\widehat{\text{Prox}}(\mathcal{A}_i, \mathcal{B}_j) = \frac{1}{|\mathcal{A}_i|}\vec{1}^T\mathbf{M}\vec{1}$
       2.2 Else: $\widehat{\text{Prox}}(\mathcal{A}_i, \mathcal{B}_j) = 1$
     EndFor
   EndFor
```

FastManyGDAP reduces the computational cost from $O(n_1 \times n_2)$ to $O(1)$. By extending the proof of Lemma 6, we have the following lemma:

LEMMA 7. *FastManyGDAP gives exactly the same result as equation* (5).

Proof: Omitted for brevity. □

4. APPLICATIONS

We focus on three graph mining applications, to illustrate the effectiveness of our proposed proximity functions. The applications are (a) link prediction (b) "Center Piece Subgraphs" for the directed case and (c) the creation of "Attribute Graphs".

4.1 Link Prediction

As a proximity measurement, our direction-aware proximity can be directly used for link prediction. More specifically, it can be used for the following two tasks:

T1: (**Existence**) Given two nodes, predict the existence of a link between them

T2: (**Direction**) Given two adjacent (linked) nodes, predict the direction of their link

For *T1*, we use the simple rule:

A1: Predict a link between i and j iff $\text{Prox}(i, j) + \text{Prox}(j, i) > th$ (th is a given threshold).

Alternatively, we can use group proximity for *T1*:

A1': (Node Expansion) Predict a link between i and j iff $\text{Prox}(\mathcal{N}(i), \mathcal{N}(j)) + \text{Prox}(\mathcal{N}(j), \mathcal{N}(i)) > th$, where $\mathcal{N}(i)$ and $\mathcal{N}(j)$ are the neighborhoods of i and j, respectively.

As for directionality prediction, *T2*, we use the rule:

A2: Predict a link from i to j if $\text{Prox}(i, j) > \text{Prox}(j, i)$, otherwise predict the opposite direction.[2]

Related experimental results will be given in Section 5.

4.2 Directed Center-Piece Subgraph

The concept connection subgraphs, or center-piece subgraphs, was proposed in [6, 19]: Given Q query nodes, it creates a subgraph $\mathcal{H}$ that shows the relationships between the query nodes. The resulting subgraph should contain the nodes that have strong connection to all or most of the query nodes. Moreover, since this subgraph $\mathcal{H}$ is used for visually demonstrating node relations, its visual complexity is capped by setting an upper limit, or a *budget* on its size. These so-called connection subgraphs (or center-piece subgraphs) were proved useful in various applications, but currently only handle undirected relationships.

With our direction-aware proximity, the algorithm for constructing center-piece subgraphs (CePS) can be naturally generalized to handle directed graphs. A central operation in the original CePS algorithm was to compute an importance score, $r(i, j)$ for a single node j w.r.t. a single query node q_i. Subsequently, these per-query importance scores are combined to importance scores w.r.t. the whole query set, thereby measuring how important each node is relatively to the given group of query nodes. This combination is done through a so-called K_softAND integration that produces $r(\mathcal{Q}, j)$ – the importance score for a single node j w.r.t. the whole query set $\mathcal{Q}$. For more details please refer to [19].

The main modification that we introduce to the original CePS algorithm is the use of directed proximity for calculating importance scores. The resulting algorithm is named *Dir-CePS* and is given in Table 6.

Table 6: *Dir-CePS*

```
Input: digraph $\mathbf{W}$, query set $\mathcal{Q} = \{q_1, ..., q_Q\}$,
       budget $b$, token vector $\vec{f} = [f_1, ...f_Q], (f_i = \pm 1)$
Output: the resulting subgraph $\mathcal{H}$
1. For each query node $q_i \in \mathcal{Q}$
     For each node $j$ in the graph
       If $f_{q_i} = +1$, $r(i, j) = \text{Prox}(q_i, j)$
       Else, $r(i, j) = \text{Prox}(j, q_i)$
2. Combine $r(i, j)$ to get $r(\mathcal{Q}, j)$ by K_softAND
3. While $\mathcal{H}$ is not big enough
   3.1 Pick up the node $pd = argmax_{j \notin \mathcal{H}} r(\mathcal{Q}, j)$
   3.2 For each active source $q_i$ wrt $pd$
     3.2.1 If $f_{q_i} = +1$, find a key path from $q_i$ to $pd$
           Else, find a key path from $pd$ to $q_i$
     3.2.2 Add the key path to $\mathcal{H}$
```

Directional information must also involve the input to *Dir-CePS* , through the token vector $\vec{f} = [f_1, ...f_Q], (f_i = \pm 1)$, which splits the query set into "sources" and "targets", such that each proximity or path are computed from some source to some target. The remaining parts of *Dir-CePS* are exactly the same as in [19]; details are skipped here due to space limitations.

Figure 5 presents an example of our *Dir-CePS* on a citation network (see Subsection 5.1.1 for a description of the data). Given two pa-

[2]On the other hand, from practical point of view, if $\text{Prox}(i, j)$ and $\text{Prox}(j, i)$ is closed with each other(say, by some threshold), we can always predict a bi-directional edge.

pers as the query nodes, by setting different token vectors $\vec{f}$, we can visually explore different perspectives on how these two papers relate: (1) let $\vec{f} = [+1, -1]$, we can examine how the former paper influences the latter one (Figure 5(a)); (2) let $\vec{f} = [+1, +1]$, we can visually show how these papers influence other papers (Figure 5(b)); and (3) let $\vec{f} = [-1, -1]$, we can examine how these papers are influenced by other papers (Figure 5(c)).

4.3 Attribute Graph

For graphs whose nodes are associated with discrete attributes, we would like to explore the relationships among these attributes, and find, e.g., which attributes are close and whether we can find symmetric attributes. For example, consider an who-mails-whom network, where nodes/people are labeled by their job title - we want to find whether and how "managers" are related to "sales-persons", and conversely. We propose to use our group proximity scores, to construct an *Attribute Graph (AG)*:

DEFINITION 2. *Let* $\mathbf{W}$ *be a directed graph, where the nodes have an attributed with* n' *categorical values* $(a_1, ..., a_{n'})$*. Let* $\mathcal{A}_i$ *be the group of nodes whose attribute value is* a_i*. The associated* attribute graph AG *is an* $n' \times n'$ *weighted digraph where every node corresponds to an attribute and the edge weights show the group-to-group proximity scores:* $AG(a_i, a_j) = \text{Prox}(\mathcal{A}_i, \mathcal{A}_j)$.

The Web Link dataset (WL) is described in Subsection 5.1.1. It consists of web pages pointing to each other, and every web page has one of the labels: 'faculty', 'project', 'staff' etc. We show its attribute graph in Figure 6. All edges of weight less than 0.05 were removed for visual clarity. This graph reveals interesting relationships among the attributes. For example, we can observe the roles of different attributes: 'department' as the "core", 'staff' as the "sent-out guy", and 'student' as the "sink". Closely related attributes include 'faculty'–'course', and 'staff'–'project'. However, we can see that while 'faculty' and 'course' are symmetrically close, the 'staff'–'project' closeness is asymmetric.

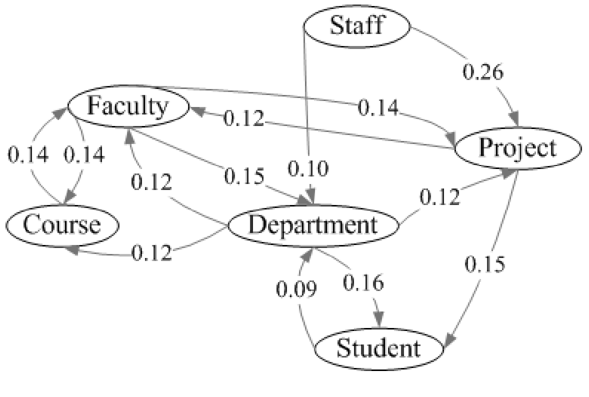


Figure 6: Attribute graph for the Web Link dataset

5. EXPERIMENTAL RESULTS

We present experiments on large, real datasets. Our goals are to do an evaluation of our design decisions, to illustrate the effectiveness of our approach on real tasks like link prediction, and to study the scalability of our methods. We start with the description of our datasets.

5.1 Experimental Setup

5.1.1 Datasets

We experimented with the following directed networks:

WL (Web Link Graph)[3]**.** In this unweighted graph, nodes denote web pages, and edges correspond to web links. The nodes are associated with one of the seven attributes: 'Department', 'Staff', 'Student', 'Faculty', 'Course', 'Project' and 'Other'. Totally, there are $\approx 4K$ nodes and $\approx 10K$ edges.

PC Personal Contact Network This dataset is a subset of an anonymized who-contacts-whom network. Nodes correspond to users, and edges are weighted by the corresponding average daily contact time. Totally, there are $\approx 36K$ nodes and $\approx 64K$ edges.

CN (Citation Network)[4]**.** This is an unweighted citation network, containing a subset of the arXiv dataset. Nodes are papers and edges are citations. It has $\approx 28K$ nodes and $\approx 353K$ edges.

EP (Epinions Who-Trusts-Whom Network)[5]**.** Unweighted network, used in [4]. Nodes denote users, and edge (i, j) means that user i trusts user j. It has $\approx 76K$ nodes and $\approx 509K$ edges.

AE (Anonymous Email Network). This network describes unweighted relations among email accounts from a large research organization. Nodes are (anonymized) accounts, and edge $(i,\ j)$ indicates that account i has sent one or more e-mail messages to account j. There are $\approx 38K$ nodes and $\approx 115K$ edges.

5.1.2 Parameter settings

After a parametric study (details skipped for space limitation), we set low values to the parameters c (strength of sink connections) and β (the symmetrization factor), and specifically $c = 0.99$ and $\beta = 0.001$ for all the experiments. We stop the iterations in both *FastOneDAP* and *FastOneGDAP* when we reach $m = 80$ iterations, or when the L_2 difference between successive values of $\vec{v}$ is below some threshold ($\xi = 10^{-9}$). All experiments were performed on the same machine with a 3.2GHz Pentium CPU and 2GB memory.

5.2 Link Prediction Effectiveness

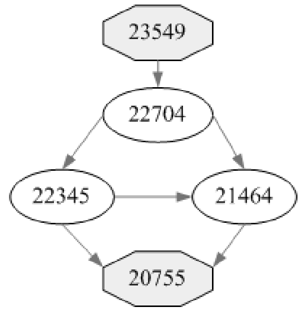
First, we evaluate the ability to predict the existence of a link. We compare four methods: *A1* (plain node-to-node comparisons, see Subsection 4.1) utilizing directionality (**ND**), and ignoring directionality (**NU**); *A1'*(Node Expansion - see Subsection 4.1) utilizing directionality (**GD**), and ignoring it (**GU**). For all the methods, the threshold (th) is determined by leave-one-out cross validation [12]. For all the experiments, the number of test pairs with link between them roughly equals the number of unlinked test pairs. In addition to the prediction accuracy, we also use *average margin* (AM):[6]. Suppose we have n' test pairs: $(i_1, j_1), \ldots, (i_{n'}, j_{n'})$. Let $Sc(k) = \text{Prox}(i_k, j_k) + \text{Prox}(j_k, i_k)$ when using *A1* or $Sc(k) = \text{Prox}(\mathcal{N}(i_k), \mathcal{N}(j_k)) + \text{Prox}(\mathcal{N}(j_k), \mathcal{N}(i_k))$ when using *A1'*. The

[3]http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/

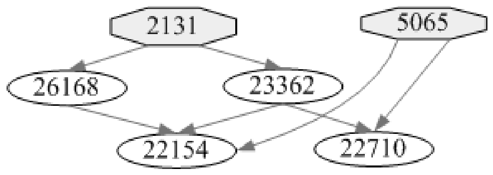
[4]http://www.cs.cornell.edu/projects/kddcup/datasets.html

[5]http://www.epinions.com/

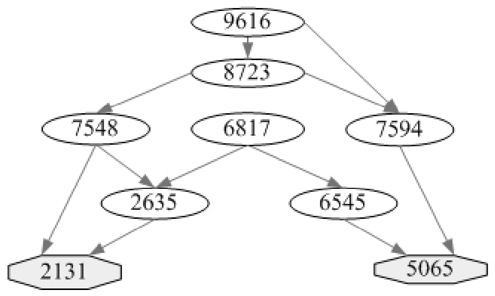
[6]The motivation behind it is based on the optimization criteria in logistic regression and boosting [3, 8]



(a) $\vec{f} = [+1, -1]$



(b) $\vec{f} = [+1, +1]$



(c) $\vec{f} = [-1, -1]$

Figure 5: By employing directional information, *Dir-CePS* can explore several relationships among the same query nodes (the two octagonal nodes): (a) A query-node to query-node relations; (b) common descendants of the query nodes (paper number 22154 apparently merged the two areas of papers 2131 and 5036); (c) common ancestors of the query nodes: paper 9616 seems to be the paper that initiated the research areas of the query papers/nodes.

average margin (AM) is defined as:

$$AM = \frac{1}{n'} \sum_{k=1}^{n'} e^{(Sc(k)-th)y(k)} \qquad (19)$$

where $y(k) = 1$ if there exists a link between i_k and j_k, whereas $y(k) = -1$ otherwise. Basically, AM estimates the confidence of the prediction by measuring how far the test pair is from the threshold (further is better).

The results are shown in Tables 7 and 8. It can be seen that while all four methods are effective for link prediction, the direction-aware proximity (both node and group versions) is usually better than the undirected proximity. E.g., in terms of prediction accuracy, the winning method is always based on direction-aware proximity. Another interesting observation is that the methods based on group proximity (GD and GU) usually outperform those based on node proximity (PD and PU). In particular, GD and GU tend to lead to much higher margins (AM) than PD and PU.

Table 7: Existence of link (Accuracy); percents indicates fraction of successful predictions. Directionality-aware methods (GD, ND) outperform the rest

Dataset	GD	GU	ND	NU
WL	**65.49%**	65.42%	65.42%	65.42%
PC	**81.20%**	**81.20**%	79.60%	80.78%
AE	**82.51%**	81.81%	81.51%	80.60%
CN	85.10%	**86.71**%	**86.71**%	84.00%
EP	88.07%	87.31%	**92.21%**	92.09%

Table 8: Existence of link (Average Margin); higher values indicate greater confidence. Again, directionality-aware methods win (GD,ND); Node expansion also helps (GD method).

Dataset	GD	GU	ND	NU
WL	**1.58**	1.58	1.16	1.16
PC	**1.36**	**1.36**	1.12	1.12
AE	**1.33**	**1.33**	1.03	1.03
CN	1.44	**1.48**	**1.48**	1.44
EP	**1.28**	1.25	1.02	1.02

We also evaluated the performance on predicting the direction of a link; see Table 9. Please notice that here comparison with an undirected version is inherently impossible. Our method seems to be effective on all datasets, though its degree of success varies a lot among the datasets . Additionally, we construct a test set $(i_1, j_1), \ldots, (i_{n'}, j_{n'})$, such that for each pair (i_k, j_k) in the test set, there exists a link from i_k to j_k and there is no link in the opposite direction. Figure 7 plots the histogram of $\text{Prox}(i_k, j_k) - \text{Prox}(j_k, i_k)$ for WL . It can be seen that as desired, the histogram is biased w.r.t. the origin: there are many more pairs in the positive zone.

Table 9: Predicting the direction of a link

Dataset	Accuracy
WL	62.4%
AE	81.4%
EP	75.2%
PC	56.7%
CN	92.4%

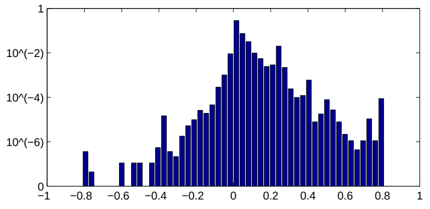


Figure 7: Density histogram of $\text{Prox}(i_k, j_k) - \text{Prox}(j_k, i_k)$ on the WL dataset. Successful predictions correspond to the positive values of the x axis, and they clearly outnumber the wrong predictions (the negative x-values).

5.3 Computational Efficiency

We measured the efficiency of the proposed fast solutions. A subset of the EP network is adopted to verify how the fast solutions scale with the size of the graph. Let N' be the number of nodes in the subset. We fix the group size to be $0.01N'$. To test *Fast-ManyGDAP* , we generate two sets of groups, each of them contains $0.25N'$ groups; and we compute totally $0.25N' \times 0.25N'$ group proximities. Figure 8 plots the mean running times vs. the number of the nodes. Notice that the y axis is *logarithmic*, to accommodate the huge performance savings of our methods. In all cases, our solutions are *1-4 orders of magnitude faster* than the straight-forward ones. For example, for a graph with $61K$ nodes and $484K$ edges, *FastOneDAP* is 59,000 times faster than the straight-forward methods (2.2 sec vs. 114K sec), and *FastOneGDAP* is 18,000 times faster(6.2 sec vs. 111K sec); for a graph with $5K$ nodes and $25K$ edges, *FastAllDAP* is $3,000$ times faster (57 sec vs. 167K sec), and *FastManyGDAP* is 57 faster (2.8K Sec vs. 163K Sec). We stress that, in all cases, our proposed methods do not lose accuracy since they either converge (*FastOneDAP* and *FastOneGDAP*) or are exactly equal (*FastAllDAP* and *FastManyGDAP*) to the true proximity value.

6. RELATED WORK

In the literature, there are several measures of node proximity. Most standard measures are based on basic graph theoretical concepts - the shortest path length and the maximum flow. However the dependency of these measures on a single element of the graph – the shortest path or the minimum cut – makes them more suitable to managed networks, but inappropriate for measuring the random nature of relationships within social networks or other self organizing networks. Consequently, some works suggested more involved measures such as the sink-augmented delivered current [6], cycle free effective conductance [15], survivable network [11], random walks with restart [14, 17] and more. Notice that none of the existing methods meets all the three desirable properties that our approach meets: (a) dealing with directionality, (b) quality of the proximity score and (c) scalability.

Graph proximity is an important building block in many graph mining settings. Representative work includes connection subgraph [6, 15, 19], personalized PageRank [13], neighborhood formulation in bipartite graphs [18], content-based image retrieval [14], cross modal correlation discovery [17], the BANKS system [1], link prediction [16], detecting anomalous nodes and links in the graph [18], ObjectRank [2] and RelationalRank [9].

7. CONCLUSIONS

In this work, we study the role of directionality in measuring proximity on graphs. We define a direction-aware proximity measure based on the random walk notion of escape probability. This measure naturally weights and quantifies the multiple relationships which are reflected through the many paths connecting node pairs. Moreover, the proposed proximity measure is carefully designed to deal with practical situations such as accounting for noise and facing partial information. A useful generalization of the measure deals with proximity between groups of nodes.

Given the growing size of networked data, a good proximity measure should be accompanied with a fast algorithm. Consequently we offer fast solutions, addressing two settings. First, an iterative algorithm, with convergence guarantee, to compute a single node-to-node proximity value on a large graph. Second, an accurate algorithm that computes all (or many) pairwise proximities on a medium sized graph. These proposed algorithms achieve orders of magnitude speedup compared to straightforward approaches, without quality loss.

We have studied the applications of the proposed proximity measure to real datasets. Encouraging results demonstrate that the measure is effective for link prediction. Importantly, being direction-aware, it enables predicting not only the existence of the link but also its direction. Another application is the so-called directed center-piece subgraph, where we employ the proposed proximity measure to carefully select presentable subgraphs that capture relations among a set of query nodes. In addition, our proposed group proximity is useful to explore the relationship among attributes by building Attribute Graph.

8. REFERENCES

[1] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, and S. S. Parag. Banks: Browsing and keyword searching in relational databases. In *VLDB*, pages 1083–1086, 2002.

[2] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *VLDB*, pages 564–575, 2004.

[3] M. Collins, R. Schapire, and Y. Singer. Logistic regression, adaboost and bregman distance. In *Proceedings of Thirteenth Annual Conference on Computational Learning Theory*, 2000.

[4] P. Domingos and M. Richardson. Mining the network value of customers. *KDD*, pages 57–66, 2001.

[5] P. Doyle and J. Snell. *Random walks and electric networks*, volume 22. Mathematical Association America, New York, 1984.

[6] C. Faloutsos, K. S. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *KDD*, pages 118–127, 2004.

[7] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. *SIGCOMM*, pages 251–262, Aug-Sept. 1999.

[8] J. Friedman. Greedy function approximation: A gradient boosting machine. In *Annual of Statistics*, 2001.

[9] F. Geerts, H. Mannila, and E. Terzi. Relational link-based ranking. In *VLDB*, pages 552–563, 2004.

[10] G. Golub and C. Loan. *Matrix Computation*. Johns Hopkins, 1996.

[11] M. Grötschel, C. L. Monma, and M. Stoer. Design of survivable networks. In *Handbooks in Operations Research and Management Science 7: Network Models*. North Holland, 1993.

[12] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer.

[13] T. H. Haveliwala. Topic-sensitive pagerank. *WWW*, pages 517–526, 2002.

[14] J. He, M. Li, H. Zhang, H. Tong, and C. Zhang. Manifold-ranking based image retrieval. In *ACM Multimedia*, pages 9–16, 2004.

[15] Y. Koren, S. C. North, and C. Volinsky. Measuring and extracting proximity in networks. In *KDD*, pages 245–255, 2006.

[16] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *Proc. CIKM*, pages 556–559, 2003.

[17] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In *KDD*, pages 653–658, 2004.

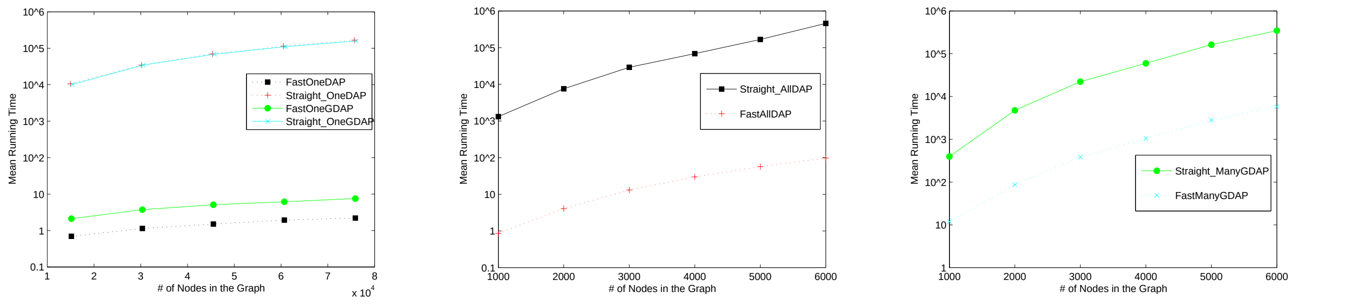


Figure 8: Evaluation on the efficiency of the proposed fast solutions(The running time is in Log scale)

Computation of one proximity **Computation of all proximities** **Computation of many proximities**

[18] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *ICDM*, pages 418–425, 2005.

[19] H. Tong and C. Faloutsos. Center-piece subgraphs: Problem definition and fast solutions. In *KDD*, pages 404–413, 2006.

APPENDIX

A. APPENDIX

A.1 Variants

There are some plausible variants of the proposed direction-aware proximity. One possibility is to multiply the escape probability by the out-degree of the origin node:

$$\mathrm{Prox}(i,j) \triangleq D_{i,i} \cdot ep_{i,j} \tag{20}$$

By multiplying by the degree, we measure absolute connectivity instead of relative connectivity. This way, proximities related to high degree nodes become more prominent. Interestingly, for undirected graphs multiplying the escape probability by the origin's degree is equivalent to effective conductance [5].

Some recent works [17, 19] relied on another random work notion for measuring node proximity on undirected graphs - *random walk with restart*. This notion considers a random walk that starts wandering in the graph beginning at an origin node i, and at each step has some probability $1-c$ to retract back to i. Then, $\mathrm{Prox}(i,j)$ could be defined as the steady state probability $r_{i,j}$ that the particle will finally stay at node j.

Based on the proof of Lemma 6, the relationship between random walk with restart and escape probability (ep) is made explicit by the following lemma:

LEMMA 8. *Let $r_{i,j}$ be the steady-state probability for node j w.r.t. node i, then:*

$$ep_{i,j} = \frac{(1-c)r_{i,j}}{r_{i,i}r_{j,j} - r_{i,j}r_{j,i}} \tag{21}$$

Acknowledgement

This material is based upon work supported by the National Science Foundation under Grants No. IIS-0326322 IIS-0534205 and under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No.W-7405-ENG-48. This work is also partially supported by the Pennsylvania Infrastructure Technology Alliance (PITA), an IBM Faculty Award, a Yahoo Research Alliance Gift, with additional funding from Intel, NTT and Hewlett-Packard. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or other funding parties. The authors give their sincere gratitude to Chris Volinsky, Jure Leskovec, Pedro Domingos and Matt Richardson for helping us collect some of the datasets.