Chapter 3

RANDOM WALKS IN SOCIAL NETWORKS AND THEIR APPLICATIONS: A SURVEY

Purnamrita Sarkar
Machine Learning Department
Carnegie Mellon University
psarkar@cs.cmu.edu

Andrew W. Moore
Google, Inc.
awm@google.com

Abstract A wide variety of interesting real world applications, e.g. friend suggestion in social networks, keyword search in databases, web-spam detection etc. can be framed as ranking entities in a graph. In order to obtain ranking we need a graph-theoretic measure of similarity. Ideally this should capture the information hidden in the graph structure. For example, two entities are similar, if there are lots of short paths between them. Random walks have proven to be a simple, yet powerful mathematical tool for extracting information from the ensemble of paths between entities in a graph. Since real world graphs are enormous and complex, ranking using random walks is still an active area of research. The research in this area spans from new applications to novel algorithms and mathematical analysis, bringing together ideas from different branches of statistics, mathematics and computer science. In this book chapter, we describe different random walk based proximity measures, their applications, and existing algorithms for computing them.

Keywords: random walks, proximity measures, hitting times, personalized pagerank, link prediction

1. Introduction

Link prediction in social networks, personalized graph search techniques, spam detection in the World Wide Web and collaborative filtering in recommender networks are important practical problems that greatly rely on graph theoretic measures of similarity. All these problems can be framed as ranking entities in a graph.

Take for example online social networks, where an important application is to suggest friends to a newcomer. Another example is movie and music recommendation systems (*Netflix, Last.fm*), where an user is suggested new movies and music based on his or her ratings so far. The graph in this case is a bipartite network between users and items, and an edge can be weighted by the rating given by the user to the item. Keyword search in large publication databases leads to another interesting problem setting, where the database can be seen as an entity-relation graph between papers, authors and words. The goal is to find papers which are "contextually" similar to the query submitted by the user.

The underlying question in all these problems is: given a node in a graph, which other nodes are most similar to this node. Ideally we would like this proximity measure to capture the graph structure such as having many common neighbors or having several short paths between two nodes.

Real world graphs are huge, complex and continuously evolving over time. Therefore it is important to find meaningful proximity measures, and design fast space-efficient algorithms to compute them.

A widely popular approach in graph-mining and machine learning literature is to compute proximity between nodes by using random walks on graphs: diffusion of information from one node to another. This chapter will be focused on these measures: the intuition behind their usefulness and effective algorithms to compute them, and important real-world problems where they have been applied.

Random walks provide a simple framework for unifying the information from ensembles of paths between two nodes. The ensemble of paths between two nodes is an essential ingredient of many popular measures such as personalized pagerank [37], hitting and commute times [4], Katz measure [46], harmonic functions [84]. Personalized page-rank vectors (PPV) have been used for keyword search in databases [9] and entity-relation graphs [18]. Hitting and commute times have been shown to be empirically effective for query suggestion [55], ranking in recommender networks [14] and image segmentation problems [64]. In [39] the authors show how to use hitting times for designing provably manipulation resistant reputation systems. Harmonic functions have been used for automated image-segmentation and coloring [52, 33], and document classification [84].

Naive computation of these measures usually requires $O(n^3)$ time, which is prohibitive for real world large graphs. Fast approximation algorithms for computing these bring together ideas from scientific computing, databases and graph theory. These algorithms span from clever offline preprocessing [40, 28, 74], fast online computation of sparse support of these measures [10, 70, 66] to hybrid algorithms which use best of both worlds [18]. In section 3 we would discuss these algorithms, study their interconnections, and see how they fit into the big picture.

Here is the organization: we will discuss two main aspects of random walk-based graph mining problems. First we will discuss different proximity measures arising from a random walk in section 2. Section 3 consists of some popular algorithms for computing different proximity measures. We show real-world applications of these measures in section 4, and conclude with experimental evaluation techniques and data-sources in section 5.

2. Random Walks on Graphs: Background

In this section we introduce some basic concepts of random walks on graphs. A graph $G = (V, E)$ is defined as a set of vertices V and edges E. The ij^{th} entry of the adjacency matrix A is nonnegative and denotes the weight on edge i, j, and is zero if the edge does not exist. In unweighted graphs the weight of any edge is 1, whereas in a weighted graph it can be any positive number. D is an $n \times n$ diagonal matrix, where $D_{ii} = \sum_j A_{ij}$. We will denote D_{ii} by the degree $d(i)$ of node i from now on. The Laplacian L of G is defined as $D - A$. This is used widely in machine learning algorithms as a regularizer ([73, 86]). For undirected[1] graphs, L is positive semi-definite, since for any vector $\boldsymbol{g}$, $\boldsymbol{g}^T L \boldsymbol{g}$ equals $\sum_{i,j \in E}(g(i) - g(j))^2$.

$P = p_{ij}, i, j \in V$ denotes the transition probability matrix, so that $P_{ij} = A_{ij}/D_{ii}$ if $(i, j) \in E$ and zero otherwise. A random walk on this graph is a Markov chain ([4]) with transition probabilities specified by this matrix. For an undirected graph, A is symmetric and L is symmetric positive semi-definite. We will denote the set of neighbors of a node i by $\mathcal{N}(i)$. For an unweighted graph, $d(i)$ is simply the size of this neighborhood. For a directed graph, the set of nodes having an edge to node i is denoted by $I(i)$, and the indegree is denoted by $d^-(i)$. Similarly the set of out-neighbors is denoted by $O(i)$, and the outdegree is denoted by $d^+(i)$. Both in and outdegree are weighted for weighted graphs.

In a random walk, if node v_0 is chosen from a distribution x_0, then the distributions $x_0, x_1, ..$ are in general different from one another. However if $x_t = x_{t+1}$, then we say that x_t is the stationary distribution for the graph. According to the Perron-Frobenius theorem, there exists a unique stationary distribution, if P is irreducible and aperiodic. For graphs that are undirected, non-bipartite and connected (both irreducible and aperiodic) the stationary distribution is proportional to the degree distribution.

[1]Laplacian matrices for directed graphs have been proposed in [21].

2.1 Random Walk based Proximity Measures

Now we will briefly describe popular random walk based similarity measures, e.g. personalized pagerank, hitting and commute times, simrank, etc. We will first discuss a few query-independent and broad topic based search algorithms, more specifically pagerank, HITS and its variants including SALSA. Note that the primary focus of this chapter is to give a comprehensive survey of random walk based proximity measures, and hence we will not discuss fast algorithms for computing pagerank, HITS or SALSA in section 3. We will briefly mention some algorithms for making SALSA and HITS faster after introducing them. An excellent study of pagerank can be found in [50].

Query independent search and broad topic search. Pagerank [15] is a widely known tool for ranking web-pages. If the world-wide-web is considered as a graph then the pagerank is defined as the distribution that satisfies the following linear equation (3.1):

$$\mathbf{v} = (1-\alpha)P^T\mathbf{v} + \frac{\alpha}{n}\mathbf{1} \tag{3.1}$$

α is the probability of restarting the random walk at a given step. Note that the α restart factor plays a very important role. The Perron-Frobenius theorem indicates that a stochastic matrix P has a unique principal eigenvector iff it is irreducible and aperiodic. The random restart assures that, since under this model 1) all nodes are reachable from all other nodes, and 2) the Markov chain is aperiodic. Also the restart probability makes the second largest eigenvalue to be upper bounded by α [45].

Pagerank does not provide topic-specific search results. In [47] the authors introduced the idea of Hubs and Authorities (HITS) for distilling search results for a broad topic. The key intuition behind the HITS algorithm is that the useful web-pages for searching a broad topic are generally of two types: the authorities and the hubs. The authorities are pages which are good sources of information about a given topic, whereas a hub is one which provides pointers to many authorities. Given a topic, first the authors find a subgraph of the web A. This subgraph ideally should be small, should consist of pages relevant to the given query and should contain most of the authoritative pages. The authors obtain this subgraph by expanding a root set of nodes “along the links that enter and leave it”. The details can be found in the paper. The root set consists of the top k search results for the given query from a standard text-based search engine. Given this subgraph, the idea is to assign two numbers to a node: a hub-score and an authority score. A node is a good hub if it points to many good authorities, whereas a node is a good authority if many good hubs point to it. This leads to iterating over two vectors $\boldsymbol{h}$ and $\boldsymbol{a}$, namely:

$$\boldsymbol{a}(i) \leftarrow \sum_{j:j\in I(i)} \boldsymbol{h}(j) \tag{3.2}$$

$$\boldsymbol{h}(i) \leftarrow \sum_{j:j\in O(i)} \boldsymbol{a}(j) \tag{3.3}$$

Both these vectors are normalized to have unit length (the squares of the entries sum to 1). The last two equations can also be written as follows. Let A be the un-weighted adjacency matrix of the subgraph.

$$\boldsymbol{a} = A^T\boldsymbol{h} \qquad\qquad \boldsymbol{h} = A\boldsymbol{a}$$

The above equation simply means that $\boldsymbol{h}$ converges to the principal eigenvector of AA^T, whereas $\boldsymbol{a}$ converges to the principal eigenvector of A^TA. As mentioned in [51], the matrix AA^T can be described as the bibliographic coupling matrix. This is because, the $\{i,j\}^{th}$ entry of the matrix AA^T is given by $\sum_k A(i,k)A(j,k)$, which is simply the number of nodes both i and j point to. On the other hand, A^TA can be thought of as the co-citation matrix, i.e. the $\{i,j\}^{th}$ entry is given by $\sum_k A(k,i)A(k,j)$, i.e. the number of nodes which point to both i and j.

There has been algorithms which use the same idea as in HITS but alter the recurrence relation. The main idea behind these is to emphasize the higher authority-scores more while computing the hub scores. These algorithms can be divided in roughly two kinds: the authority threshold ($AT(k)$) and the $Norm(p)$ families[2]. In [13] the authors restrict the sum in eq (3.3) to the k largest authority scores, while keeping eq (3.2) intact. The underlying intuition is that a good hub should point to at least k good authorities. When k is the maximum out-degree, this algorithm is identical to HITS. The $Norm(p)$ family uses a smoother way to scale the authority weights. The p-norm of the authority weights vector is used in eq (3.3), while keeping eq 3.2 intact. A similar approach was used in [30]. When p is 1 this is the HITS algorithm. The MAX algorithm is a special case of the $AT(k)$ family (for $k = 1$), and the $Norm(p)$ family (when $p = \infty$). An in-depth study of this special case (MAX) of these two families is provided in [80].

SALSA [51] combines the idea of a random surfer from pagerank with hubs and authorities from HITS. The idea is to conceptually build an undirected bipartite graph, with two sides containing hubs and authorities. A node i can be both a hub ($h(i)$) and an authority ($a(i)$). A link is added between nodes

[2] We use the same names as mentioned in [80].

$h(i)$ and $a(j)$, if there is link $i \rightarrow j$ in the original subgraph. The nodes and links are constrained to a part of the web-graph relevant to the query, built in a manner similar to HITS. Two Markov chains are considered, s.t. one step of each consists of traversing two links in a row. This way each walk visits nodes only on one side of the bipartite graph. The stationary distributions of these walks are used as the hub and authority scores for the original subgraph.

It can be shown that these vectors are simply principal eigenvectors of $A_r A_c^T$ and $A_c^T A_r$ (instead of the matrices AA^T and $A^T A$ as in HITS). A_r is the row normalized version of A whereas A_c is the column normalized version of A. Let D_+ be the diagonal matrix of out-degrees, and D_- be the diagonal in-degree matrix. A_r is simply $D_+^{-1}A$, and A_c is AD_-^{-1}. Matrix $A_r A_c^T$ is equal to $D_+^{-1}AD_-^{-1}A^T$. Denote this as $D_+^{-1}\mathcal{H}$, where[3] $\mathcal{H}$ equals $AD_-^{-1}A^T$. The $\{i,j\}^{th}$ entry of this matrix is given by $\sum_k A(i,k)A(j,k)/d^-(k)$. Note that this is the number of nodes both i and j point to, as in AA^T, except each *common* neighbor is inversely weighted by its indegree. Here is an intuitive reason why this kind of weighting is useful. Consider two pairs of papers in a citation network. The first pair, i and j, both point to a very highly cited paper (high indegree), whereas the second pair, i' and j', both point to a rather obscure paper (low indegree). In this case, i' and j' are much more likely to be contextually similar than i and j.

Also, since the i^{th} row of matrix $\mathcal{H}$ sums to $d^+(i)$, the matrix $A_r A_c^T$ is a stochastic matrix (each row sums to 1). As $\mathcal{H}$ is undirected, when it is connected, the principal eigenvector of the probability transition matrix resulting from it is proportional to the degree distribution of this modified adjacency matrix, in this case, the out-degrees. The case of a number of connected components is described in [51]. Similarly, it can be shown that the principal eigenvector of $A_c^T A_r$, also a stochastic matrix, is the in-degree distribution.

Thus, the construction of SALSA simply means that high indegree nodes *in the topic-specific subgraph* are good authorities, whereas high out-degree nodes *in the topic-specific subgraph* are good hubs. This is surprising, since earlier it has been pointed out that ranking simply with respect to (w.r.t) in-degrees on the entire WWW graph is not adequate [47].In [51] the authors point out that this conflict results from better techniques for assembling a topic-specific WWW subgraph. These techniques augment the original subgraph-expansion [47] with link-filtering schemes to remove noisy links, and use simple heuristics to assign weights to links, so that weighted indegree or out-degrees can be computed.

One problem with the HITS ranking is the sensitivity to the tightly knit communities, coined as the TKC effect. This happens when a small tightly-

[3]The matrix AA^T is denoted by H in [51].

knit community of nodes rank highly, although they are not most authoritative. Using a combinatorial construction and examples of search results from the web graph, it has been demonstrated that SALSA is less vulnerable to the TKC effect than HITS [51].

Note that neighborhood construction is key to both HITS, MAX and SALSA, and is a major computational bottleneck at query time. In [58] the authors show how to precompute score maps for web-pages in order to drastically lower query latency. Later consistent unbiased sampling [17] and bloom filters [11] have been used for approximating neighborhood graphs [59].

Personalized Pagerank. PageRank gives a democratic view of the respective importance of the webpages. However by using a non-uniform restart probability density we can create a personalized view of the relative importance of the nodes. The basic idea is to start a random walk from a node i; at any step the walk moves randomly to a neighbor of the current node with probability $1-\alpha$, and is reset to the start node i with probability α. The stationary distribution of this process is the personalized pagerank w.r.t node i. As a result, the stationary distribution is localized around the start node. This was also called "rooted pagerank" in [53].

If we generalize the start node i to a start distribution $\boldsymbol{r}$, the personalization vector will be given by:

$$\mathbf{v} = (1-\alpha)P^T\mathbf{v} + \alpha\boldsymbol{r} \tag{3.4}$$

In order to obtain personalization w.r.t node i, the restart distribution $\boldsymbol{r}$ is set to a vector where the i^{th} entry is set to 1, and all other entries are set to 0. $P^T\boldsymbol{v}$ is the distribution after one step of random walk from $\boldsymbol{v}$. The above definition implies that personalized pagerank can be computed by solving a large linear system involving the transition matrix P.

While most algorithms use personalized pagerank as a measure of similarity between two nodes, in [81] the authors presented a setting where the restart distribution is learnt using quadratic programming from ordinal preferences provided by an administrator. The authors presented a scalable optimization problem by reducing the number of parameters by clustering the WWW graph.

Simrank. Simrank is a proximity measure defined in [41], which is based on the intuition that two nodes are similar if they share many neighbors. The recursive definition of simrank is given by

$$s(a,b) = \frac{\gamma}{|I(a)||I(b)|} \sum_{i\in I(a), j\in I(b)} s(i,j) \tag{3.5}$$

If two random surfers start two simultaneous backward walks from nodes a and b, then this quantity can be interpreted as the expectation of γ^{ℓ}, where ℓ equals the time when those two walks meet.

Hitting and Commute time. The hitting time ([4]) between nodes i and j is defined as the expected number of steps in a random walk starting from node i before node j is visited for the first time. H is an $n \times n$ matrix. Recursively h_{ij} can be written as

$$h_{ij} = \begin{cases} 1 + \sum_k p_{ik} h_{kj} & \text{If } i \neq j \\ 0 & \text{If } i = j \end{cases} \tag{3.6}$$

The hitting time can also be interpreted as weighted path-length from node i to node j, i.e. $h_{ij} = \sum_{path(i,j)} |path(i,j)| P(path(i,j))$. Hitting times are not symmetric even for undirected graphs. This can be seen by considering an undirected star graph with a central node with large degree, which is connected to many nodes with degree one. The central node has high hitting time to all of its neighbors, since there are many different destinations of a random walk. On the other hand, the degree-1 neighbors have a small hitting time ($= 1$) to the central node. Hitting times also follow the triangle inequality ([54]).

Commute time between a pair of nodes is defined as $c_{ij} = h_{ij} + h_{ji}$. We will now show some surprising properties of commute times for undirected graphs. For undirected graphs[4] there is a direct analogy of hitting and commute times with electrical networks ([25]). Consider an undirected graph. Now think of each edge as a resistor with conductance A_{ij}. In this setting, the commute time can be shown ([20]) to be equal to the effective resistance between two nodes up to a multiplicative constant.

Let $vol(G)$ denote the volume of the graph. This is defined as the sum of degrees of all nodes, which also equals twice the number of edges in an undirected graph. If $d(i)$ amount of current is injected at node $i, \forall i$, and $vol(G)$ amount of current is withdrawn from node j, let the voltage at of node i be $\psi(i)$. Now, the voltage at node i w.r.t node j is denoted by $\psi(i,j)$. This is essentially $\psi(i) - \psi(j)$. Using Kirchhoff's law we can write:

$$d(i) = \sum_{k \in \mathcal{N}(i)} (\psi(i,j) - \psi(k,j))$$

Using algebraic manipulation we have:

$$\psi(i,j) = \begin{cases} 1 + \sum_{k \in \mathcal{N}(i)} \dfrac{\psi(k,j)}{d(i)} & \text{If } i \neq j \\ 0 & \text{Otherwise} \end{cases} \tag{3.7}$$

[4]A detailed discussion of hitting and commute times in directed graphs can be found in [12].

Note that this linear system is identical to the definition of hitting time from i to j (eq. 3.6) and hence hitting time from i to j is identical to the voltage at node i w.r.t j. The above can be written as a solution to the following linear system:

$$L\boldsymbol{\psi} = \boldsymbol{y} \tag{3.8}$$

$\boldsymbol{y}$ is the vector of currents injected in the system, in particular, $y(x) = d(x)$, $\forall x \neq j$ and for node j, $y(j) = d(j) - vol(G)$. Note that, if $\boldsymbol{\psi}$ is a solution to eq. 3.8 then any vector obtained by shifting each entry of it by a fixed constant is also a solution. The intuition behind this is that, the currents in the system only depend on the difference in the voltages between two nodes, not the respective values. Hence, if all voltages are shifted by a constant, the currents will stay the same. More concretely, this happens because L is not full rank: one of its eigenvectors, the all ones vector, has eigenvalue zero. Hence, it is not invertible. This is why the Moore-Penrose pseudo-inverse, denoted by L^+, is used ([29]). This matrix, i.e. L^+, can also be written as $(L - \frac{1}{n}\mathbf{1}\mathbf{1}^T)^{-1} + \frac{1}{n}\mathbf{1}\mathbf{1}^T$ ([29, 88]).

Hence from eq. 3.8, we have:

$$\boldsymbol{\psi} - \mathbf{1}\frac{\mathbf{1}^T\boldsymbol{\psi}}{n} = L^+\boldsymbol{y} \tag{3.9}$$

The second term on the L.H.S is simply the constant vector, where each entry equals $\frac{\mathbf{1}^T\boldsymbol{\psi}}{n}$, i.e. the mean of $\boldsymbol{\psi}$. Since the hitting time measures the voltage difference, this mean shifting does not matter, i.e. $H(i,j) = \psi(i,j) = \psi(i) - \psi(j) = (\boldsymbol{e_i} - \boldsymbol{e_j})^T L^+\boldsymbol{y}$.

Now consider the system where $vol(G)$ current is withdrawn from i and $d(x)$ amount is injected at all nodes x. Now the voltage drop at j w.r.t i will be $h_{ji} = (\boldsymbol{e_j} - \boldsymbol{e_i})^T L^+\boldsymbol{y'}$, where $\boldsymbol{y'}$ is the new current vector with $\boldsymbol{y}(x) = d(x), \forall x \neq i$ and for node i, $y(i) = d(i) - vol(G)$. Note that $\boldsymbol{y} - \boldsymbol{y'} = vol(G)(\boldsymbol{e_i} - \boldsymbol{e_j})$. Hence, $c_{ij} = h_{ij} + h_{ji}$ is given by,

$$c_{ij} = vol(G)(\boldsymbol{e_i} - \boldsymbol{e_j})^T L^+(\boldsymbol{e_i} - \boldsymbol{e_j}) \tag{3.10}$$

This also gives us some intuition about the effectiveness of commute time as a proximity measure. First, if the effective resistance between two nodes is small, then it is easier for information to diffuse from one node to the other. Second, since electrical properties of networks do not change much by adding or removing a few edges, hitting and commute times should be robust to small perturbation in datasets ([25]).

From eq. (3.10) we see that L^+ maps each vertex of a graph to a Euclidian space $i \mapsto \boldsymbol{x_i}$, where $\boldsymbol{x_i} = (L^+)^{\frac{1}{2}}\boldsymbol{e_i}$. This is how in undirected graphs pairwise commute time can be expressed as the squared Euclidian distance in the transformed space (eq. 3.11).

$$\begin{aligned} c_{ij} &= vol(G)(l^+_{ii} + l^+_{jj} - 2l^+_{ij}) \\ &= vol(G)(\boldsymbol{e_i} - \boldsymbol{e_j})^T L^+(\boldsymbol{e_i} - \boldsymbol{e_j}) \end{aligned} \tag{3.11}$$

Let us now look at the entries of L^+ in terms of these position vectors. The ij^{th} entry $l^+_{ij} = \boldsymbol{x_i}^T \boldsymbol{x_j}$, denotes the dot-product between the position vectors of vertex i and j. The diagonal element l^+_{ii} denotes the squared length of the position vector of i. The cosine similarity ([14]) between two nodes is defined as $l^+_{ij}/\sqrt{l^+_{ii} l^+_{jj}}$. We discuss different ways of computing the above quantities in section 3.

More random-walk based proximity measures. A discounted version of escape probability from node i to node j (probability to hit node j before coming back to node i in a random walk started from node i) has been used for computing connection subgraphs in [26]. A connection subgraph is a small subgraph of the original network which best captures the relationship between two query nodes. Escape probabilities have also been used to compute *direction aware proximity* in graphs [78]. Koren et al. use cycle-free escape probability from i to j (probability that a random walk started at i will reach j without visiting any node more than once) for computing connection subgraphs for a group of nodes in [48].

2.2 Other Graph-based Proximity Measures

In their detailed empirical study of different graph-based proximity measures for link-prediction tasks [53] have used the Katz score, number of common neighbors and Jaccard score. The authors also presented higher-level *meta* approaches for link prediction which use the above measures as a basic component. These approaches included low rank approximations of the adjacency matrix, unseen bigrams from language modeling, and clustering.

Katz Score. [46] had designed a similarity measure based on ensemble of paths between two nodes.

$$Katz(i,j) = \sum_{\ell=1}^{\infty} \beta^\ell . A^\ell(i,j)$$

Note that $A^\ell(i,j)$ is simply the number of paths of length ℓ from i to j. Hence the matrix of scores can be written as $(I - \beta A)^{-1} - I$. In order to compute the Katz score from a given query node, one needs to solve a linear system over the adjacency matrix. For very small β the Katz score mostly returns nodes with many common neighbors with the query node, whereas larger values of beta allows one to examine longer paths. Also, the weighted version (which takes into account weights of edges in the graph) of the Katz score was introduced in [53].

Common Neighbors, Adamic/Adar and Jaccard Coefficient. Recall that in an undirected graph, we define the set of neighbors of node i by $\mathcal{N}(i)$. Then the number of common neighbors of nodes i and j is given by $|\mathcal{N}(i) \cap \mathcal{N}(j)|$. If two nodes have a large number of common neighbors they are more likely to share a link in the future. This leads to the use of number of common neighbors as a pairwise proximity measure.

The Adamic/Adar score [1] is similar to this, except the high degree common neighbors are weighed less. The Adamic/Adar score is defined as:

$$\texttt{Adamic/Adar}(i,j) = \sum_{k \in \mathcal{N}(i) \cap \mathcal{N}(j)} \frac{1}{\log |\mathcal{N}(k)|} \tag{3.12}$$

The intuition behind this score is that, a popular common interest, gives less evidence of a strong tie between two people. For example, many people subscribe to the New York Times. But a few subscribe to the Journal of Neurosurgery. Hence, it is much more likely that two people share similar interests if they both subscribe to the second journal in place of the first.

The Jaccard coefficient computes the probability that two nodes i and j will have a common neighbor k, given k is a neighbor of either i or j.

$$J(i,j) = \frac{|\mathcal{N}(i) \cap \mathcal{N}(j)|}{|\mathcal{N}(i) \cup \mathcal{N}(j)|} \tag{3.13}$$

The matrix of Jaccard coefficients can be shown to be positive definite [32].

Often the Jaccard coefficient is used to compute document similarity in information retrieval [65]. Note that for the bag of words model, computing this score between two documents can be expensive. An efficient randomized algorithm (shingles) for computing this was introduced by [16]. The idea is to represent a document by a set of subsequences of words using a moving window of length ℓ. Now these sequences are hashed to integers, and then the smallest k of these are used to form a sketch of each document. Now the Jaccard score is computed based on this new sketch of a document. The authors show that an unbiased estimate of the Jaccard score can be computed by comparing these sketches.

2.3 Graph-theoretic Measures for Semi-supervised Learning

While random-walks have been used to compute proximity measures between two nodes in a graph or global or topic-specific importance of nodes, they also provide a natural framework to include negative information. The question we ask for obtaining nearest neighbors is: which nodes are close or relevant to this query node? In semi-supervised learning we ask, given a graph where a few nodes are labeled relevant (positive) and irrelevant (negative) which nodes are more similar to the relevant ones than the irrelevant ones. We will now formalize this idea.

Consider a partially labeled dataset with l labeled data-points denoted by $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_l, y_l), \boldsymbol{x}_{l+1}, \ldots, \boldsymbol{x}_n\}$. The goal is to classify the unlabeled points, from the labeled examples. Often labels are available for a small fraction of dataset, since labeling might require human intervention. Szummer et al. [76] represent a data-point using random walks on the graph built from this dataset. For node k the authors compute the probability that a random walk started at node i, conditioned on the fact that it ended at k after t steps $(P_{0|t}(i|k))$. This distribution in terms of the start distribution over all nodes is the new representation of node k. This probability can be computed from the standard t-step probability of reaching k from i using Bayes rule. The intuition is that two datapoints are similar under this notion if they have similar start distributions. Now the classification rule is to assign to an unlabeled node k the class c, which maximizes $\sum_i P_{0|t}(i|k)P(y = c|i)$. The parameters $P(y|i)$ are the hidden class distribution over the nodes, and are estimated by maximizing likelihood via Expectation Maximization, or maximizing margin of classification. The parameter t is crucial for this algorithm and is chosen so that the average margin per class is maximized. The authors also propose heuristics for estimating *adaptive time scales* for different nodes.

Zhu et al. [84] use another graph theoretic approach for semi-supervised learning. Given labeled and unlabeled 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 authors introduce the harmonic function, which is based on the intuition that nodes *close* in the graph have similar labels. A harmonic function is defined as $f : V \to \mathcal{R}$ which has fixed values at the given labeled points and is smooth over the graph topology. More specifically the harmonic property means that the function value at an unlabeled node is the average of function values at the neighbors. Let U denote the set of unlabeled nodes in the graph. Now the harmonic property can be mathematically expressed as follows:

$$f(i) = \frac{\sum_j A_{ij} f(j)}{d(i)}, i \in U$$

Let's assume that for the labeled nodes we assign $f(i) = 1$, if i has label 1, and 0 otherwise. This can also be expressed as $f = Pf$. We now divide P into four blocks by grouping the labeled and unlabeled points together. f is grouped into

f_u (unlabeled) and f_l (labeled nodes with fixed values). This gives:

$$f_u = (I - P_{uu})^{-1} P_{ul} f_l \tag{3.14}$$

The above function represents the probability of hitting a label '1' before a label '0'. This can be easily generalized to multi-class classification setting, using a one vs. all encoding of labels. Harmonic functions can also be thought of as the voltage at all nodes in a electrical network, where the positive nodes are connected to a unit voltage external source and the negative nodes are grounded (zero voltage).

Relationship between harmonic functions, escape probabilities and commute times. We have shown that commute times, escape probabilities and harmonic functions are widely used for proximity computation and semi-supervised learning. The harmonic function f with boundary conditions $f(s) = 1$ and $f(t) = 0$ at node i, computes the probability that a random walk from node i will hit s before t. The escape probability from s to t [25] ($P_{esc}(s,t)$) is defined as the probability of reaching t before returning to s in a random walk stated at s. Hence $P_{esc}(s,t) = 1 - \sum_i P(s,i)f(i)$. This can be seen as the escape probability equals the probability of visiting a neighbor i, and from there reaching t before returning to s. However the latter has probability $1 - f(i)$, by definition of harmonic functions.

As mentioned earlier, harmonic functions can also be thought of as the voltage at node i resulting from connecting node s to a unit voltage source, and node t to a zero voltage source. Now we will see the relationship of escape probabilities and harmonic functions with commute time. Let the total current drawn by node s from the outside source be I_s. Using Kirchhoff's law this quantity is $\sum_i A_{s,i}(1 - f(i))$, where $C_{s,i}$ is the conductance or weight of the edge $\{s,i\}$, since $f(s) = 1$. This is simply $d(s)\sum_i P(s,i)(1 - f(i))$, which is simply the degree of node s times the escape probability from s to t. Hence the effective conductance between nodes s and t for a unit voltage drop is $d(s)P_{esc}(s,t)$. Recall that the effective conductance between nodes s and t is the reciprocal of the effective resistance between nodes s and t, which is proportional to the commute time between nodes s and t.

Using the Laplacian for Graph Regularization. Most semi-supervised algorithms are variations of learning a function over a graph. The goal is to optimize the function over the labeled nodes, and enforce smoothness over the graph topology via regularization. Zhu et al. [86] provide an excellent survey which unifies different semi-supervised learning algorithms under this framework. The authors discuss a number of algorithms and show the different choice of the loss function and the regularizer. A number of these algorithms use the graph Laplacian L, or the normalized graph Laplacian $D^{-1/2}LD^{-1/2}$

for regularizing the function. This, (e.g. in [84]) sometimes leads to a function closely related to a random walk on the underlying graph. Agarwal et al. [2] connect the Laplacian regularization framework with random walk based measures (e.g. personalized pagerank) and provide generalization guarantees for the latter in directed graphs.

2.4 Clustering with random walk based measures

Random walks provide a natural way of examining the graph structure. It is popular for clustering applications as well. Spectral clustering [60] is a body of algorithms that clusters datapoints $\boldsymbol{x_i}$ using eigenvectors of a matrix derived from the affinity matrix A constructed from the data. Each datapoint is associated to a node in a graph. The weight on a link between two nodes i and j, i.e. A_{ij}, is obtained from a measure of similarity between the two datapoints. One widely used edge weighting scheme transforms the Euclidian distance between the datapoints, i.e. $A_{ij} = \exp(\|\boldsymbol{x_i} - \boldsymbol{x_j}\|^2/\sigma^2)$, where σ is a free parameter.

Meila et al. [56] provide a random-walk based probabilistic interpretation of spectral clustering algorithms including Normalized Cut [72]. Building the affinity matrix poses an important algorithmic question, which has not yet been studied extensively. The choice of σ is crucial. Zhu et al. [86] describe different techniques for building graphs and estimating σ.

Global graph clustering using random walks. Saerens et al. [64] exploit equation (3.11) to embed a graph and provide distances between any pair of nodes. In [83] the authors replace traditional shortest-path distances between nodes in a graph by hitting and commute times and show that standard clustering algorithms (e.g. K-means) produce much better results when applied to these re-weighted graphs. These techniques exploit the fact that commute times are robust to noise and provide a finer measure of cluster cohesion than simple use of edge weight. Harel et al. [36] present a general framework for using random walk based measures as *separating operators* which can be repeatedly applied to reveal cluster structure at different scales of granularity. The authors propose using t step probabilities, escape probabilities and other variants to obtain edge separation. The authors show how to use these operators as a primitive for other high level clustering algorithms like multi-level and agglomerative clustering. In a similar sprit, Azran et al. [8] use different powers of the transition matrix P to obtain clustering of the data. The authors estimate the number of steps and the number of clusters by optimizing spectral properties of P.

Local Graph Clustering. Random walks provide a natural way of clustering a graph. If a cluster has relatively fewer number of cross-edges compared to number of edges inside, then a random walk will tend to stay inside that cluster. Recently there has been interesting theoretical work [75, 5] for using random walk based approaches for computing good quality local graph partitions (cluster) near a given seed node. The main intuition is that a random walk started inside a good cluster will mostly stay inside the cluster. Cluster-quality is measured by its conductance, which is defined as follows: For a subset of S of all nodes V, let $\Phi_V(S)$ denote conductance of S, and $vol(S) = \sum_{i \in S} d(i)$. As in [75], conductance is defined as:

$$\Phi_V(S) = \frac{E(S, V \setminus S)}{min(vol(S), vol(V \setminus S))} \tag{3.15}$$

A good-quality cluster has small conductance, resulting from a small number of cross-edges compared to the total number of edges. The smaller the conductance, the better the cluster quality. Hence 0 is perfect score, for a disconnected partition, whereas 1 is the worst score for having a cluster with no intra-cluster edges. Conductance of a graph is defined as the minimum conductance of all subsets S of the set of nodes V.

The formal algorithm to compute a low conductance local partition near a seed node is given in [75]. The algorithm propagates probability mass from the seed node and at any step rounds the small probabilities, leading to a sparse representation of the probability distribution. Now a local cluster is obtained by making a sweep over this probability distribution. The running time is nearly linear in the size of the cluster it outputs. [5] improve upon the above result by computing local cuts from personalized pagerank vectors from the predefined seed nodes. The authors also generalized their result to strongly connected directed graphs in a subsequent paper [6].

So far we have introduced different random-walk based proximity measures, and looked at how they are related to each other. The rest of this chapter will be divided into three parts. First we will discuss the existing algorithms for fast computation of different random walk based proximity measures introduced in the last section. Then we will describe different applications of these measures. Finally we will conclude with experimental evaluation of these measures, and some sources of publicly available networks.

3. Related Work: Algorithms

In this section, we will describe the popular algorithms designed for computing the proximity measures described in section 2. Katz, personalized pagerank, hitting times, simrank etc. can each be computed as a fixed point of a recursive definition. This can be computed either by computing the solution to a linear system, by clever dynamic programming approaches, or by efficient sampling algorithms. We will briefly describe some of these approaches. Some of the dynamic programming approaches are local in nature, i.e. given a query node, the algorithm only examines a local neighborhood of the graph. Some of these approaches are well-suited for external memory graphs, i.e. for graphs which are too big to fit into main memory.

3.1 Algorithms for Hitting and Commute Times

Most clustering applications based on commute times either require extensive computation to produce pairwise proximity measures, or employ heuristics for restricting the computation to subgraphs. Some authors avoid cubic computation by using sparse matrix manipulation techniques. Saerens et al. [64] compute the trailing eigenvectors of L to reconstruct L^+. This requires solving for the eigenvectors corresponding to the smallest eigenvalues of L, which requires significant pre-computation and can be expensive for very large graphs.

Brand et al. [14] compute sub-matrices of the hitting time and commute time matrices H and C by iterative sparse matrix multiplications. Note that computing the i^{th} row of C, i.e. $C(i,*)$ or the cosine-similarities of all j with i requires $L^+_{jj}, \forall j$. The exact value of L^+_{jj} requires 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 distribution of j is a close constant factor approximation to L^+_{jj}; however no approximation guarantee is provided. In short, it is only tractable to compute these measures on graphs with a few thousand nodes for most purposes. Also these techniques are not applicable to a directed graph.

Spielman at al. [74] have designed a fast algorithm for computing pairwise commute times within a constant factor approximation. The idea is to use the Johnson-Lindenstrauss lemma [43] in conjunction with a nearly linear time solver [75] to compute a $n \times \log n/\epsilon^2$ matrix of random projections in $\tilde{O}(m)$ time, ϵ being the approximation error and m the number of edges. At query time the commute time can be computed by computing the Euclidian distance between two points in $O(\log n)$ time.

Mei et al. [55] compute hitting time by iterating over the dynamic programming step for a fixed number of times. This leads to the *truncated* hitting time. In order to scale the search, the authors do a depth-first search from a query node and stop expansion when the subgraph exceeds a predefined number. Now the iterative algorithm is applied to this subgraph, and the query nodes are ranked based on hitting time. The problem with this approach is that there is no formal guarantee that the algorithm will not miss a potential nearest neighbor.

A formal approach of computing nearest neighbors in truncated hitting time to a node by doing neighborhood expansion can be found in [66]. The goal is to compute k ϵ-approximate nearest neighbors of query nodes s within T-

truncated hitting time τ. The main idea is to expand a neighborhood around the given query node, and stop when all nodes outside the neighborhood can be proven to be "un-interesting". The key is to obtain provable lower and upper bounds on the truncated hitting time from the nodes inside the current neighborhood to the query. These bounds enable the authors to expand the neighborhood in an adaptive fashion, i.e. include "potential" nearest neighbors. As the neighborhood is expanded these bounds become tighter. For all nodes outside the neighborhood the authors maintain a global lower bound on hitting time to the query node. The expansion is stopped when this bound exceeds τ; this guarantees that the hitting times of nodes outside the current neighborhood is larger than τ, and hence uninteresting.

Note that although it is easy to compute hitting time *to a node* by using dynamic programming, it is hard to use dynamic programming for computing hitting time *from a node*. The reason is: by definition of hitting times, i.e. expected time to hit a node *for the first time*, it is hard to define a problem based on smaller subproblems. We will show this using two examples:

$$P^t(i,j) = \sum_k P(i,k)P^{t-1}(k,j) \tag{3.16}$$

$$P^t(i,j) = \sum_k P^{t-1}(i,k)P(k,j) \tag{3.17}$$

Consider the t step probability of reaching node j from node i. Using eq. 3.16 we can compute t step probabilities from all nodes to node j, whereas eq. 3.17 can be iterated to compute t-step probabilities from node i to all other nodes. If we could write hitting time from node i using

$$h^t(i,j) \overset{?}{=} \sum_k h^{t-1}(i,k)P(k,j)$$

we would be able to use dynamic programming to compute hitting time from i to all other nodes. However this is not true, since $h^{t-1}(i,k)$ consists of paths through j, whereas $h^t(i,j)$ is defined to look at paths that stop at j. Sampling provides a useful approach for computing *truncated* hitting time from a node. The idea is simple: sample M random walks from the query node. Now look at the first occurrence time $X^t(i,k)$ of the nodes k which have been encountered during the random walk. Hitting time is given by $\sum_t min\{X^t(i,k),T\}/M$. Sarkar et al. [69] provide sample complexity bounds for this and use this in conjunction with a neighborhood expansion scheme to compute nearest neighbors in commute time from a node with high probability.

3.2 Algorithms for Computing Personalized Pagerank and Simrank

Personalized pagerank vectors (PPV) are often used to measure the proximity between two nodes in a network. Although it was first introduced in early 2000, fast algorithms for computing it is still an active area of research. It has been proven [40] that the PPV from a set of webpages can be computed by linearly combining those for each individual webpage in the set. The problem is that it is hard to store all possible personalization vectors or compute the personalized pagerank vector at query time because of the sheer size of the internet graph. We will briefly sketch a few well-known algorithms here.

Haveliwala et al. [37] divide the web-corpora into k (16) topics. The personalized pagerank for each topic (a collection of pages from the Open Directory) is pre-computed offline and stored. At runtime a user-query is transformed into a probability distribution on these basis topics. The resulting personalized pagerank for the query is now a linear combination of the topics, weighted by the probability distribution of the query over the topics.

Jeh et al. [40] assume that the preferences of personalized search, i.e. restart distributions are restricted to a set of important pages, denoted by the "hubs". Hence the size of this set determines the degree of personalization. The authors pick the 1000 to $100,000$ top pagerank pages as the "hub" set. If the hub vectors can be pre-computed and stored, it would be easy to linearly combine these vectors at query time to obtain different personalizations. If this set is too large, it becomes computationally expensive to compute the personalization vectors for each hub-node. In order to achieve higher degrees of personalization, the authors make the key observation that these vectors share common components, which can be pre-computed, stored and combined at query time. The two major components are the partial vectors and the hub skeleton. A hub-skeleton vector corresponding to a hub-node i measures the influence of node i on all other nodes via paths through the set of hub nodes, whereas the partial vector for node i is a result of the paths which do not go via H. If the hub nodes are chosen such that most paths in the graph go via them, then for many pairs i, j, the partial vector will be very sparse. The authors show how to compute these hub-skeletons and partial vectors via dynamic programming, so that redundant computation can be avoided.

Fogaras et al. [28] achieve full personalization by simulating random walks. Personalized pagerank from i to j can also be defined as the probability that a random walk of length L started at node i will stop at node j, where ℓ is chosen from the geometric distribution with parameter α, i.e. $P(\ell = t) = \alpha(1-\alpha)^{t-1}$. This leads to a simple estimation procedure: simulate M random walks from i, such that after each step the walk continues with probability $1 - \alpha$ and stops with probability α. The fraction of times the walk ends in j gives an estimate of $PPV(i, j)$. The authors also provide lower bounds on the size of a database needed for achieving approximate answers to different queries with high probability, e.g. query if a personalized pagerank value is positive, compare values at two nodes, compute an approximation of a PPV value, etc. The interesting observation is that if one wants to obtain full personalization the lower bound on database size is $O(n^2)$; However allowing a small probability of error and approximation can let one have database size linear in the number of vertices. The sampling algorithm in fact achieves this lower bound. The authors show how to compute “fingerprints” of random walks from each node efficiently for external memory graphs. A fingerprint is a compact version of a random walk: it just stores the start and end vertices of the walk so far. By sequentially passing through two files, one with the edges (sorted lexicographically by source), and the other with the current fingerprints (sorted lexicographically by the end-node), one can generate the fingerprint for the next time-step efficiently.

Fogaras et al. [27] use the sampling algorithm to give the first scalable algorithm for computing simrank. The authors first show how to estimate simrank values by generating independent backward random walks of a fixed length from every node. It is simple to estimate the simrank between two nodes from the first meeting time of pairwise walks from two nodes. However storing a number of fixed length random walks from each node in a graph can be prohibitive for a web-scale graph. The authors use coalesced random walks, which save both time and space-complexity. The authors also show how to represent a set of *coalesced* reverse random walks compactly. The key is to store the minimal information such that just the first meeting times for a pair of walks can be reconstructed without reconstructing the entire path. This reduces the storage complexity from fixed-length walks to one integer per node in the graph.

Sarlós et al. [70] improve upon the sampling algorithm in [28] by introducing rounding and sketching based deterministic algorithms which reduces the approximation error significantly. Instead of using a power method approach to compute personalized pagerank *from a node*, Sarlós et al. use dynamic programming to compute personalized pagerank *to a node* (similar to [40]). At any iteration of the dynamic programming, the authors round the small pagerank values to zero, and show that this rounding scheme leads to sparse supports for personalized pagerank while leading to a small approximation error. The authors also point out that in a power method approach, at a high in-degree node the errors from the in-neighbors add up and lead to large approximation error (proportional to the indegree). However, a dynamic programming approach computes an *average* over the pagerank values of the out-neighbors of a node and hence does not amplify error. The space-requirement of the deterministic rounding algorithm is improved further by using Count-Min sketches [23] to represent the intermediate sparse distributions. The authors also show that similar rounding ideas can be used to compute simrank, by reducing it to personalized pagerank.

The previously mentioned algorithms compute partial personalized pagerank vectors offline and combine them at query time. Berkhin et al. [10] propose a simple but novel "Bookmark Coloring Algorithm" (BCA) for computing sparse personalized pagerank from any given node. Personalized pagerank can be thought of as the stationary distribution of a random process where every node retains α fraction of its color (probability) and distributes the rest along its out-neighbors. The process starts by injecting unit color (probability) to the node, w.r.t which we want to personalize the measure. A matrix multiplication step in personalized pagerank, as in eq. (3.4) achieves exactly this, where at any node the contribution of its in-neighbors are added. BCA can be thought of as an adaptive version of the matrix multiplication, where only the large probability masses are propagated. The authors achieve this by maintaining a queue with the personalized pagerank values. At any step the queue is popped to find the largest $\{i,w\}$ pair, where i is the node with w amount of residual probability. The personalized pagerank vector is updated by adding α fraction of w to entry i, whereas the entries $\{j, (1-\alpha)/d^+(i)\}$ for all outgoing neighbors of j are enqueued in Q. The propagating of probability stops when the amount falls below a pre-specified small threshold ϵ.

This algorithm is formally analyzed in [5]. The authors improve the time complexity of BCA algorithm by eliminating the queue, and propagating any probability that is above ϵ. Theoretical guarantees are provided for both approximation error and runtime. The key intuition is to express the personalized pagerank as the sum of the approximate pagerank and the personalized pagerank with the start distribution of the residual probability vector (values stored in the queue in case of BCA). With ϵ as the threshold the approximation error simply involves the personalized pagerank w.r.t a residual vector whose largest entry is ϵ. The authors use this approximate pagerank vector to compute a local graph partition around the start node, and provide theoretical guarantees about the quality of the partition. Note that personalized pagerank computed using these approaches may have large accumulated approximation error at high-indegree nodes.

Chakrabarti et al. [18] show how to compute approximate personalized pagerank vectors using clever neighborhood expansion schemes which would drastically reduce the amount of off-line storage and computation. The motivation of the HubRank algorithm comes from using PPV for context-sensitive keyword search in entity-relation graphs. Since the vocabulary is huge, it is often prohibitive to cache PPV for every word. At query time the vectors for different query words are combined to compute ranking for a multi-word query. For using personalized pagerank in this setting Balmin et al. [9] rounded the personalized pagerank values in order to save space. However this can adversely affect the ranking accuracy of multi-word queries. In order to reduce this storage requirement, HubRank only stores Monte-Carlo fingerprints (defined in [28]) of a set of words and other entity nodes (e.g. papers or authors), by examining query logs. Given a query, the algorithm identifies a small *active* subgraph whose personalized pagerank values must be computed. Then the pagerank values are computed only on this subgraph by loading in bordering nodes of this subgraph for whom random walk fingerprints were already stored.

3.3 Algorithms for Computing Harmonic Functions

Zhu et al. [86] present a thorough survey of algorithms for semi-supervised learning algorithms. We will briefly describe a few of them, which use random walk based measures. The key to computing a harmonic function is to compute the solution to the linear system in eq. (3.14). A detailed comparison among the label propagation, loopy belief propagation, and conjugate gradient approaches is provided in [85]. Label propagation simply uses the basic definition of harmonic functions, i.e. the function value at an unlabeled node is the mean of the function value at its neighbors. The harmonic values can be viewed as the mean values of the marginal probabilities of the unlabeled nodes; this is why loopy belief propagation can be used for computing these. It was shown that loopy belief propagation and preconditioned conjugate gradient often converge faster than the other algorithms. Label propagation is the simplest among all these approaches, and also the slowest.

In [87] the authors combine the idea of learning mixture models with regularizing the function over the graph topology. The solution to the resulting *Harmonic Mixture Model* involves a much smaller "backbone" graph, where the nodes are merged into super-nodes using the mixture components. For example, if one obtains hard clustering, then all points in one mixture component are merged into one super-node representing that component. This considerably reduces the cost of solving the linear system. Sarkar et al. [68] use a truncated version of harmonic functions to rerank search results after incorporating user feedback. The authors give a neighborhood expansion scheme to quickly identify the top ranking nodes (w.r.t harmonic function values), where the labels on a few nodes are provided by the users through an interactive setting. Azran et al. [7] compute harmonic functions by computing the eigen-decomposition of a *Rendezvous* graph, where the labeled nodes are converted into sink nodes.

4. Related Work: Applications

The random walks approach has been highly successful in social network analysis [46] and computer vision [31, 62], personalized graph search [40, 37, 28], keyword search in database graphs [9, 18], detecting spam [35, 82]. Here we briefly describe some of these applications.

4.1 Application in Computer Vision

A common technique in computer vision is to use a graph-representation of an image frame, where two neighboring pixels share a strong connection if they have similar color, intensity or texture.

Gorelick et al. [31] use the average hitting time of a random walk from an object boundary to characterize object shape from silhouettes. Grady et al. [34] 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 designated node were thresholded to produce partitions with various beneficial theoretical properties. Qiu et al have used commute times clustering for robust multibody motion tracking in [63] and image segmentation [62].

Harmonic functions have been used for colorizing images [52], and for automated image-segmentation [33]. The colorization application involves adding color to a monochrome image or movie. An artist annotates the image with a few colored scribbles and the indicated color is propagated to produce a fully colored image. This can be viewed as a multi-class classification problem when the class (color) information is available for only a few pixels. The segmentation example uses user-defined labels for different segments and quickly propagates the information to produce high-quality segmentation of the image. All of the above examples rely on the same intuition: neighboring nodes in a graph should have similar labels.

4.2 Text Analysis

A collection of documents can be represented in graph in many different ways based on the available information. We will describe a few popular approaches. Zhu et al. [84] build a sparse graph using feature similarity between pairs of documents, and then a subset of labels are used to compute the harmonic function for document classification.

Another way to build a graph from documents in a publication database, is to build an entity-relation graph from authors and papers, where papers are connected via citations and co-authors. The *ObjectRank* algorithm in [9] computes personalized pagerank for keyword-specific ranking in such a graph built from a publication database. For keyword search surfers start random walks from different entities containing that word. Any surfer either moves randomly to a neighboring node or jumps back to a node containing the keyword. The final ranking is done based on the resulting probability distribution on the objects in the database. In essence the personalized pagerank for each word is computed and stored offline, and at query time combined linearly to generate keyword-specific ranking. Chakrabarti et al. [18] also use the same graph representation.

Mei et al. [55] use hitting times for query suggestion. The authors build a bipartite graph of query words and URLs, such that a query is connected to a URL if the user clicked on the URL when the query was submitted to the URL. Note that one can consider the Markov chain over this bipartite graph or construct a Markov chain between the query nodes, by using the shared URLs between two query nodes. For a given query q the top k query nodes in hitting time q are suggested as alternate queries. Lafferty et al. [49] build a bipartite graph out of words and documents, and use the top ranked words in personalized pagerank from a query for query expansion. This model is augmented with external information in [22]. Besides the linkage information inherent in co-occurring terms, the authors formed the links based on external information, like synonymy, stemming, word-association and morphology relations etc. Also their random walk consists of three stages: early, middle and final to emphasize the relationships differently at different stages of the walk, e.g. the co-occurrence relation could be weighted more in the early stage, whereas links from synonymy are weighed more highly in later steps.

Minkov et al. [57] build an entity relation graph from email data, where the entities and relations are extracted from the inherent *who-sent-whom* social network, content of the emails and the time-stamps. The authors use random walks with restart for contextual search and name disambiguation in this graph. The probabilities obtained from the random walk are augmented with global features to obtain better predictive performance. In a Markov chain where the states correspond to words, Toutanova et al. [79] focus on learning the transition probabilities of the chain to obtain better word dependency distributions. The authors define a number of different link types as the basic transition distribution. Each link type has a corresponding probability transition matrix. Learning these basic parameters reduces computational complexity, and also allows the authors to learn complex random walk models. In [3] Agarwal et al. show how to compute the parameters of a random walk on a entity relation graph by including relevance feedback. The authors bring together the idea of generalizing the Markov chain model to network flow models [77] and maximum margin optimization to learn rankings [42].

4.3 Collaborative Filtering

Hitting and commute times have been successfully used for collaborative filtering. Brand et al. [14] use different measures resulting from random walks on undirected graphs to recommend products to users based on their purchase history. The authors give empirical evidence of the fact that these measures are often small if one of the nodes has a high degree.It is also shown empirically that the cosine similarity (defined in section 2) does not have this problem since the effect of individual popularity is normalized out. Fouss et al. [29] use hitting and commute times for recommending movies in the MovieLens dataset and show that these perform much better than shortest path on link prediction tasks.

4.4 Combating Webspam

There have been a number of learning algorithms to aid spam detection in the web, which is an extremely important task. These algorithms can be roughly divided into two classes: content-based methods, and graph based approaches. The content-based approaches [61] focus on the content of the webpage, e.g. number of words in the page and page title, amount of anchor text, fraction of visible content etc. to separate a spam-page from a non-spam page.

Graph-based algorithms look at the link structure of the web to classify web-spam. We will briefly discuss two of these. TrustRank [35] uses a similar idea as personalized pagerank computation . The restart distribution contains the web-pages which are known to be not spammy.

Harmonic ranking has been successfully used for spam detection by [44]. The authors build an anchor set of nodes, and compute a harmonic function with restart. For the good anchor nodes the authors use the harmonic ranking where as for the bad anchors they use a forward-propagation from the anchor set to identify other nodes with similar labels. Note that the authors only used the harmonic rank with a homogeneous anchor set.

5. Related Work: Evaluation and datasets

In this section, we will briefly describe a widely used metric for evaluating proximity measures, namely, link prediction. Then we will give a few pointers to publicly available datasets which have been used by some of the publications mentioned in this chapter.

5.1 Evaluation: Link Prediction

While there has been a large body of work for modeling pairwise proximity in network data, evaluating the different measures is often hard. Although anecdotal evidence can be used to present nice properties of a measure, they hardly provide a solid ground for believing in the performance of a given measure. One option is to conduct a user study, which is time consuming and might suffer from selection bias. The other option which has been adopted by many authors is to obtain quantitative scores via link-prediction. Given a snapshot of a social network, the link prediction problem seeks this answer: which new connections among entities are likely to occur in the future?

In general there are a few ways of conducting link prediction experiments. These are:

1 Randomly hold out a links of the graph. *Rank all non-existing links* based on a given proximity measure, and see what fraction of the top k links were originally held out.

2 Randomly hold out a links of the graph. *For every node* compute k nearest neighbors based on a given proximity measure, and see what fraction of these nodes were originally held out.

3 For every node, hold out a few links randomly and compute nearest neighbors for prediction. Add these links back, and repeat the process.

4 If time information for the data is available, divide the time-steps into training and test parts, and compute the ranking based on the older training data, which is used to predict the future links in the test set. For example, one might want to predict, which two actors who had never worked together prior to 2005 will work together in 2010?

Obviously, it is important to rank only the candidate nodes/edges. For example if a node has only one neighbor, its link is not a candidate for removal. In fact one can pick a parameter to decide if a link is candidate for being held out. For example, if the source and destination nodes have degree higher than κ, we consider it for deletion. Liben-Nowell et al. [53] use two parameters κ_{test} and $\kappa_{training}$. They divide the time period into $< t_0, t'_0, t_1, t'_1 >$. Now the data in the period $< t_0, t'_0 >$ is used as training data, whereas $< t_1, t'_1 >$ is used as the test data. A node which has at least $\kappa_{training}$ edges in the training data and κ_{test} edges in the test data is a candidate node for link prediction (i.e. some edges from it can be held out).

Liben-Nowell et al. [53] also investigate different proximity measures between nodes in a graph for link prediction. While this gives a comprehensive and quantitative platform for evaluating the goodness of different proximity measures, this also can give analytical intuitions about the evolution of real world graphs. Now we will present a few empirical observations from ([53], [14] and [66]).

A surprising result by [53] is that simple measures like number of common neighbors, and Adamic Adar often perform as accurately as more complicated measures which take into account longer paths. Recall that the Adamic Adar score (eq. 3.12) weighs the contribution of high degree common neighbors less than low degree ones. It outperforms number of common neighbors in a number of datasets.

The authors showed that the hitting and commute times suffer from their sensitivity to long paths. The most effective measure was shown to be the Katz measure [46], with a small discount factor. Also, [14] shows that hitting and commute times tend to give high rank to high degree nodes, which hurt their predictive performance. This, along with the fact that Katz score performs well with a small discount factor provide strong evidence of the fact that focusing on shorter paths, as in personalized pagerank, simrank, truncated/discounted hitting and commute times can increase predictive performance of a measure.

Another observation from [53], [14] and [66] is that shortest path performs very poorly compared to measures which look at ensemble of paths (note that number of common neighbors and Adamic/Adar also look at ensemble of length 2 paths).

5.2 Publicly Available Data Sources

In section 4 we have given a few ways of building a graph from publication databases, email data and images. Here is a brief description of the sources of these publicly available datasets.

1 MovieLens data: bipartite graph of movies and users. (`http://www.grouplens.org/node/73`). This was used in [14, 29] etc.

2 Netflix data: bipartite graph of movies and users, available from the Netflix challenge.

3 Citeseer data: can be used to build co-authorship networks and Entity-relation graphs. Maintained by Prof. C. Lee. Giles (`http://clgiles.ist.psu.edu/`). This was used in [18, 66, 69] etc.

4 The DBLP dataset: can be used to build co-authorship networks and Entity-relation graphs. (`http://dblp.uni-trier.de/xml/`). Different versions of the dataset was used in [9, 68].

5 A variety of social networks are available at `http://snap.stanford.edu/data/`. Collected and maintained by Dr. Jure Leskovec. These include online social networks like LiveJournal, email communication datasets, citation and co-authorship networks, web-graphs, peer to peer datasets, blog data, the Epinions dataset etc. Smaller versions of Arxiv publication networks were used in [53].

6 A large collection of Web spam datasets are posted in `http://barcelona.research.yahoo.net/webspam/`.

7 A large collection of web sites can be found under the Stanford WebBase project [38] `http://diglib.stanford.edu:8091/~testbed/doc2/WebBase/`. Different snapshots of the web available from this project were used in [37, 40, 28, 70].

8 A large web-corpus was released for searching for related entities and properties of a given entity in the Entity Track in the Text Retrieval Conference (TREC) 2009.
`http://ilps.science.uva.nl/trec-entity/`.

9 Manually hand-labeled segmentations of images are available in:
`http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/`.

10 Document and hand written digit data-sets used for graph-based semi-supervised learning are available in Prof. Xiaojin Xhu's webpage:
`http://pages.cs.wisc.edu/~jerryzhu/publications.html`.

11 More datasets on handwritten digits, 20 Newsgroups, and publications in the Neural Information Processing System (NIPS) conference are available in Prof. Sam Roweis's webpage:
(`http://cs.nyu.edu/~roweis/data.html`).

6. Conclusion and Future Work

In this chapter, we have built the necessary background for understanding random walk based measures of proximity; studied popular proximity measures, fast algorithms for computing them, and real world applications. The last few sections lead to a challenging and interesting array of questions which involve machine learning with very large graphs. For example: how to design infrastructure to enable end-users with off-the-shelf computing units use the existing algorithms? How does one identify the right measure for a given learning task? In settings where human intervention is needed to make an informed decision, how can we minimize it as much as possible? Is there a relationship between random walks on graphs and regularized approaches to graph-based semi-supervised learning? We can answer these questions by unifying our knowledge in diverse areas like graph theory, data mining, information theory and database systems.

Often fast search algorithms make an inherent assumption: the graph can be fit into main memory. Consider the setting where the graph is too large to be memory-resident. Searching personal information networks [19], which requires integrating the users personal information with information from the Web, often needs to be performed on the user's own machine for preserving privacy [24]. For general purpose user-end machines we need memory-efficient fast algorithms. There has been some work to design algorithms on external memory graphs: Fogaras et al. [28] show how to precompute random walks from all nodes in external memory graphs. Das Sarma et al. [71] design streaming algorithms for sampling short random walks from a given query node using a few passes. Sarkar et al. [67] show how to use a clustered representation of disk-resident graphs for computing nearest neighbors using random walk-based proximity measures. One of the important future directions is to characterize the tradeoffs and limitations of algorithms in this external-memory setting.

Personalized search being the holy grail, algorithms for re-ranking and ranking refinement which incorporate user-feedback are gaining popularity. Under this setting the user is asked to label a set of results. Ideally we would want to exploit this feedback loop the most. This brings us to graph-based active and semi-supervised learning problems.

Zhu et al. [86] showed that there is a relationship between the popular regularization framework of semi-supervised learning on graphs with random walk based measures. Agarwal et al. [2] explore this connection for obtaining generalization guarantees for popular random walk based measures. It would be interesting to explore this direction more.

As mentioned in section 5, it is common practice to empirically evaluate different proximity measures using link prediction tasks. Previous empirical studies indicate that different proximity measures perform differently on different tasks on different graphs. However there has not been any theoretical work on *why some measures work better than others*. This is discussed in detail in [53], where the properties of different graphs are studied to better explain the behavior of different measures on different tasks. The answer to this would have a tremendous impact on our intuition about real world graphs, and would enable automatic identification of the right measure for a task at hand on a given graph. It would also save end-users the trouble to understand and compute every metric separately for a task.

Identifying nearest neighbors in graphs is a key ingredient in a diverse set of applications, starting from finding friends on a social network like Face-book to suggesting movies or music in recommender systems; from viral marketing to image-segmentation; from intelligence analysis to context-based keyword search in databases. Random walks provide a popular, intuitive and mathematically principled framework for computing the underlying measure of "nearness" or proximity. A deep understanding of the behavior of these measures holds the key to efficiently utilizing the massive source of networked data that is being generated from corporate and public sources every day.

Acknowledgements

The authors will like to thank Geoffrey Gordon for his valuable comments and suggestions on this chapter. Also, the material relating graph Laplacians to hitting and commute times have been largely inspired by Gary Miller's class on "Spectral Graph Theory and Scientific Computing" at Carnegie Mellon.

References

[1] Lada A. Adamic and Eytan Adar. Friends and neighbors on the web. *Social Networks*, 25, 2003.

[2] Alekh Agarwal and Soumen Chakrabarti. Learning random walks to rank nodes in graphs. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, 2007.

[3] Alekh Agarwal, Soumen Chakrabarti, and Sunny Aggarwal. Learning to rank networked entities. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006.

[4] David Aldous and James Allen Fill. *Reversible Markov Chains*. 2001.

[5] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *FOCS*, 2006. ISBN 0-7695-2720-5.

[6] Reid Andersen, Fan Chung, and Kevin Lang. Local Partitioning for Directed Graphs Using PageRank. *Algorithms and Models for the Web-Graph*, 2006.

[7] Arik Azran. The rendezvous algorithm: multiclass semi-supervised learning with markov random walks. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, 2007.

[8] Arik Azran and Zoubin Ghahramani. A new approach to data driven clustering. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, 2006.

[9] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-based keyword search in databases. In *VLDB*, 2004.

[10] P. Berkhin. Bookmark-Coloring Algorithm for Personalized PageRank Computing. *Internet Mathematics*, 2006.

[11] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7), 1970.

[12] D. Boley, G. Ranjan, and Z. Zhang. Commute times for a directed graph using an asymmetric Laplacian. Technical report:10-005, University of Minnesota, 2010.

[13] Allan Borodin, Gareth O. Roberts, Jeffrey S. Rosenthal, and Panayiotis Tsaparas. Finding authorities and hubs from link structures on the world wide web. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 415–429, New York, NY, USA, 2001. ACM. ISBN 1-58113-348-0.

[14] M. Brand. A Random Walks Perspective on Maximizing Satisfaction and Profit. In *SIAM '05*, 2005.

[15] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proc. WWW*, 1998.

[16] Andrei Z. Broder. On the resemblance and containment of documents. In *In Compression and Complexity of Sequences (SEQUENCES'97)*, pages 21–29. IEEE Computer Society, 1997.

[17] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations (extended abstract). In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, 1998.

[18] Soumen Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *Proc. WWW*, New York, NY, USA, 2007.

[19] Soumen Chakrabarti, Jeetendra Mirchandani, and Arnab Nandi. Spin: searching personal information networks. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, 2005.

[20] A. K. Chandra, P. Raghavan, W. L. Ruzzo, and R. Smolensky. The electrical resistance of a graph captures its commute and cover times. In *STOC*, pages 574–586, 1989.

[21] Fan Chung. Laplacians and the cheeger inequality for directed graphs. *Annals of Combinatorics*, 2005.

[22] K. Collins-Thompson and J. Callan. Query expansion using random walk models. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, 2005.

[23] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1): 58–75, 2005.

[24] Bhavana Bharat Dalvi, Meghana Kshirsagar, and S. Sudarshan. Keyword search on external memory data graphs. *Proc. VLDB Endow.*, 1(1):1189–1204, 2008.

[25] P. G. Doyle and J. L. Snell. *Random Walks and Electric Networks.* The Mathematical Assoc. of America., 1984.

[26] Christos Faloutsos, Kevin S. McCurley, and Andrew Tomkins. Fast discovery of connection subgraphs. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004.

[27] D. Fogaras and B. Racz. Scaling link-based similarity search. In *Proceedings of the 14th Int'l World Wide Web Conference*, 2005.

[28] D. Fogaras, B. Rcz, K. Csalogány, and Tamás Sarlós. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics*, 2004.

[29] Francois Fouss, Alain Pirotte, Jean michel Renders, and Marco Saerens. A novel way of computing dissimilarities between nodes of a graph, with application to collaborative filtering. In *ECML workshop on Statistical Approaches for Web Mining*, 2004.

[30] David Gibson, Jon Kleinberg, and Prabhakar Raghavan. Clustering categorical data: an approach based on dynamical systems. *The VLDB Journal*, 8(3-4), 2000.

[31] L. Gorelick, M. Galun, E. Sharon, R. Basri, and A. Brandt. Shape representation and classification using the poisson equation. In *CVPR*, 2004.

[32] J.C. Gower. A general coefficient of similarity and some of its properties. *Biometrics*, 27:857–871, 1971.

[33] Leo Grady. Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1768–1783, 2006. ISSN 0162-8828.

[34] Leo Grady and Eric L. Schwartz. Isoperimetric graph partitioning for data clustering and image segmentation. In *IEEE PAMI*, 2006.

[35] Zoltan Gyongyi, Hector Garcia-Molina, and Jan Pedersen. Combating web spam with trustrank. In *VLDB '2004: Proceedings of the Thirtieth international conference on Very large data bases*. VLDB Endowment, 2004.

[36] David Harel and Yehuda Koren. On clustering using random walks. In *Foundations of Software Technology and Theoretical Computer Science 2245*. Springer-Verlag, 2001.

[37] T. Haveliwala. Topic-sensitive pagerank. In *Proceedings of the Eleventh International World Wide Web Conference*, 2002.

[38] Jun Hirai, Sriram Raghavan, Hector Garcia-molina, and Andreas Paepcke. Webbase : A repository of web pages. In *In Proceedings of the Ninth International World Wide Web Conference*, 2000.

[39] John Hopcroft and Daniel Sheldon. Manipulation-resistant reputations using hitting time. Technical report, Cornell University, 2007.

[40] G. Jeh and J. Widom. Scaling personalized web search. In *Stanford University Technical Report*, 2002.

[41] Glen Jeh and Jennifer Widom. Simrank: A measure if structural-context similarity. In *ACM SIGKDD*, 2002.

[42] Thorsten Joachims. Optimizing search engines using clickthrough data. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002.

[43] W. Johnson and J. Lindenstrauss. Extensions of lipschitz maps into a hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[44] Amruta Joshi, Ravi Kumar, Benjamin Reed, and Andrew Tomkins. Anchor-based proximity measures. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, 2007.

[45] Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub. Extrapolation methods for accelerating pagerank computations. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, 2003.

[46] L. Katz. A new status index derived from sociometric analysis. In *Psychometrika*, 1953.

[47] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5), 1999.

[48] Yehuda Koren, Stephen C. North, and Chris Volinsky. Measuring and extracting proximity in networks. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006.

[49] John Lafferty and Chengxiang Zhai. Document language models, query models, and risk minimization for information retrieval. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, 2001.

[50] Amy N. Langville and Carl D. Meyer. Deeper inside pagerank. 2003.

[51] R. Lempel and S. Moran. Salsa: the stochastic approach for link-structure analysis. *ACM Trans. Inf. Syst.*, 19(2), 2001.

[52] Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. *ACM Transactions on Graphics*, 23:689–694, 2004.

[53] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *CIKM '03*, 2003.

[54] Laszlo Lovasz. Random walks on graphs: a survey. 1996.

[55] Qiaozhu Mei, Dengyong Zhou, and Kenneth Church. Query suggestion using hitting time. In *CIKM '08*, pages 469–478, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-991-3.

[56] Marina Meila and Jianbo Shi. A random walks view of spectral segmentation. In *AISTATS*, 2001.

[57] Einat Minkov, William W. Cohen, and Andrew Y. Ng. Contextual search and name disambiguation in email using graphs. In *SIGIR '06*, 2006.

[58] Marc Najork and Nick Craswell. Efficient and effective link analysis with precomputed SALSA maps. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, 2008.

[59] Marc Najork, Sreenivas Gollapudi, and Rina Panigrahy. Less is more: sampling the neighborhood graph makes SALSA better and faster. In

WSDM '09: Proceedings of the Second ACM International Conference on Web Search and Data Mining, 2009.

[60] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, 2001.

[61] Alexandros Ntoulas, Marc Najork, Mark Manasse, and Dennis Fetterly. Detecting spam web pages through content analysis. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, 2006.

[62] Huaijun Qiu and Edwin R. Hancock. Image segmentation using commute times. In *Proc. BMVC*, 2005.

[63] Huaijun Qiu and Edwin R. Hancock. Robust multi-body motion tracking using commute time clustering. In *ECCV'06*, 2006.

[64] M. Saerens, F. Fouss, L. Yen, and P. Dupont. The principal component analysis of a graph and its relationships to spectral clustering, 2004.

[65] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986. ISBN 0070544840.

[66] Purnamrita Sarkar and Andrew Moore. A tractable approach to finding closest truncated-commute-time neighbors in large graphs. In *Proc. UAI*, 2007.

[67] Purnamrita Sarkar and Andrew Moore. Fast nearest-neighbor search in disk-resident graphs. Technical report, Machine Learning Department, Carnegie Mellon University, 2010.

[68] Purnamrita Sarkar and Andrew W. Moore. Fast dynamic reranking in large graphs. In *International World Wide Web Conference*, 2009.

[69] Purnamrita Sarkar, Andrew W. Moore, and Amit Prakash. Fast incremental proximity search in large graphs. In *Proceedings of the 25th International Conference on Machine Learning*, 2008.

[70] Tamas Sarlos, Andras A. Benczur, Karoly Csalogany, Daniel Fogaras, and Balazs Racz. To randomize or not to randomize: space optimal summaries for hyperlink analysis. In *WWW*, 2006.

[71] Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. Estimating pagerank on graph streams. In *PODS*, 2008.

[72] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. In *CVPR '97: Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, 1997.

[73] A. Smola and R. Kondor. Kernels and regularization on graphs. In Springer Verlag, editor, *Learning Theory and Kernel Machines*, 2003.

[74] D. Spielman and N. Srivastava. Graph sparsification by effective resistances. In *Proceedings of the STOC'08*, 2008.

[75] D. Spielman and S. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the STOC'04*, 2004.

[76] Martin Szummer and Tommi Jaakkola. Partially labeled classification with markov random walks. In *Advances in Neural Information Processing Systems*, volume 14, 2001.

[77] John A. Tomlin. A new paradigm for ranking pages on the world wide web. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, 2003.

[78] Hanghang Tong, Yehuda Koren, and Christos Faloutsos. Fast direction-aware proximity for graph mining. In *ACM SIGKDD, Proceeding of the Internation Conference on Knowledge Discovery and Data Mining*, 2007.

[79] Kristina Toutanova, Christopher D. Manning, and Andrew Y. Ng. Learning random walk models for inducing word dependency distributions. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, 2004.

[80] Panayiotis Tsaparas. Using non-linear dynamical systems for web searching and ranking. In *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2004.

[81] Ah Chung Tsoi, Gianni Morini, Franco Scarselli, Markus Hagenbuchner, and Marco Maggini. Adaptive ranking of web pages. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, 2003.

[82] Baoning Wu and Kumar Chellapilla. Extracting link spam using biased random walks from spam seed sets. In *WWW*, 2007.

[83] L. Yen, L. Vanvyve, D. Wouters, F. Fouss, F. Verleysen, and M. Saerens. Clustering using a random-walk based distance measure. In *ESANN 2005*, pages 317–324, 2005.

[84] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML, volume 20*, 2003.

[85] Xiaojin Zhu. *Semi-supervised learning with graphs*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2005. Chair-Lafferty, John and Chair-Rosenfeld, Ronald.

[86] Xiaojin Zhu. Semi-supervised learning literature survey, 2006.

[87] Xiaojin Zhu and John Lafferty. Harmonic mixtures: combining mixture models and graph-based methods for inductive and scalable semi-supervised learning. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, 2005.

[88] C. Rao and S. Mitra. *Generalized inverse of matrices and its applications*. John Wiley and Sons, 1971.