Tractable Algorithms for Proximity Search on Large Graphs

Purnamrita Sarkar

July 2010
CMU-ML-10-107

Carnegie Mellon

Tractable Algorithms for Proximity Search on Large Graphs

Purnamrita Sarkar

July 2010
CMU-ML-10-107

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Andrew W. Moore, Chair
Geoffrey J. Gordon
Anupam Gupta
Jon Kleinberg, Cornell University

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation under grant numbers AST-0312498, IIS-0325581, and IIS-0911032, the Defense Advanced Research Projects Agency under grant number S518700 and EELD grant number F30602-01-2-0569, the U.S. Department of Agriculture under grant numbers 53-3A94-03-11 and TIRNO-06-D-00020, the Department of Energy under grant numbers W7405ENG48 and DE-AC52-07NA27344, the Air Force Research Laboratory under grant numbers FA865005C7264 and FA875006C0216, and the National Institute of Health under grant number 1R01PH000028. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: random walks, proximity measures, graphs, nearest neighbors, link prediction, algorithms

To,

Ramaprasad Das, my grandfather,
who is behind everything I am, or ever will be.

Abstract

Identifying the nearest neighbors of a node in a graph is a key ingredient in a diverse set of ranking problems, e.g. friend suggestion in social networks, keyword search in databases, web-spam detection etc. For finding these "near" neighbors, we need graph theoretic measures of similarity or proximity. Most popular graph-based similarity measures, e.g. length of shortest path, the number of common neighbors etc., look at the paths between two nodes in a graph. One such class of similarity measures arise from random walks.

In the context of using these measures, we identify and address two important problems. First, we note that, while random walk based measures are useful, they are often hard to compute. Hence we focus on designing tractable algorithms for faster and better ranking using random walk based proximity measures in large graphs. Second, we theoretically justify why path-based similarity measures work so well in practice.

For the first problem, we focus on improving the quality and speed of nearest neighbor search in real-world graphs. This work consists of three main components: first we present an algorithmic framework for computing nearest neighbors in truncated hitting and commute times, which are proximity measures based on short term random walks. Second, we improve upon this ranking by incorporating user feedback, which can counteract ambiguities in queries and data. Third, we address the problem of nearest neighbor search when the underlying graph is too large to fit in main memory. We also prove a number of interesting theoretical properties of these measures, which have been key to designing most of the algorithms in this thesis.

We address the second problem by bringing together a well known generative model for link formation, and geometric intuitions. As a measure of the quality of ranking, we examine link prediction, which has been the primary tool for evaluating the algorithms in this thesis. Link prediction has been extensively studied in prior empirical surveys. Our work helps us better understand some common trends in the predictive performance of different measures seen across these empirical results.

Acknowledgements

It is almost impossible to express my gratitude to Andrew Moore, my advisor, in a couple of words. To him, I owe many things. Through his brilliant insights, infectious enthusiasm, and gentle guidance he has helped me formulate the problems of this thesis, and solve them. He has been like a father figure for me through all these years. Besides having time for brainstorming on new ideas and exciting algorithms, he has also lent a patient and affectionate ear to my personal problems. I have learnt many things about research and life from him. I have learnt that, when it comes to dealing with issues with people, the best way is to be honest, direct and sincere. And above all, I have learnt that, it is not the success and fame, but "loving the science" that brings us to work every morning.

This thesis was shaped by the insightful suggestions, and perceptive comments of my thesis committee members. In fact, the last chapter of this thesis was mostly written because of the great questions they had brought up during my thesis proposal. Anupam Gupta and Geoffrey Gordon have always had open doors for all my questions, doubts, and half-formed ideas; I am indeed very thankful to them. I am grateful to Jon Kleinberg, for always finding time to talk to me about my research at conferences, or during his brief visits to Carnegie Mellon. He has also been extremely supportive and helpful during the stressful times of job search.

I am very grateful to many of the faculty members at Carnegie Mellon. Gary Miller's lectures and discussions on Spectral Graph Theory and Scientific Computing have been a key to my understanding of random walks on graphs. Drew Bagnell and Jeff Schneider have given valuable suggestions related to my research and pointed me to relevant literature. Artur Dubrawski has been very supportive of my research, and has always inspired me to apply my ideas to completely new areas. I will like to thank Carlos Guestrin, who has helped me immensely in times of need. Tom Mitchell and John Lafferty have helped me through some difficult times at CMU; I am grateful to them. I would like to acknowledge Christos Faloutsos, Steve Fienberg and Danny Sleator for their help and guidance during my years at CMU. I will also like to thank Charles Isbell at Georgia Tech; interning with him as an undergraduate gave me my first taste of ideas from machine learning and statistics, and so far it has been the most enjoyable and fulfilling experience.

During my stay at Carnegie Mellon, I have made some great friends and life would not

have been the same without them. I have had the pleasure to know Shobha Venkataraman, Sajid Siddiqi, Sanjeev Koppal, Stano Funiak, Andreas Zollmann, Viji Manoharan, Jeremy Kubica, Jon Huang, Lucia Castellanos, Khalid El-Arini, Ajit Singh, Anna Goldenberg, Andy Schlaikjer, Alice Zheng, Arthur Gretton, Andreas Krause, Hetunandan and Ram Ravichandran. They have looked out for me in difficult times; brainstormed about new exciting ideas; read my papers at odd hours before pressing deadlines; and last, but not least, helped me during job search. To my officemates Indrayana Rustandi, and Shing-Hon Lau, it was a pleasure to share an office, which led to the sharing of small wins and losses from everyday. Thanks to the current and past members of the Auton Lab for their friendship, and good advice. I am grateful to Diane Stidle, Cathy Serventi and Michele Kiriloff for helping me out with administrative titbits, and above all being so kind, friendly and patient.

Finally, I would like to thank my family, my parents Somenath and Sandipana Sarkar, and my little brother Anjishnu for their unconditional love, endless support and unwavering faith in me. To Deepayan, my husband, thanks for being a wonderful friend and a great colleague, and for being supportive of my inherent stubbornness.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Begin at the beginning," the King said, very gravely, "and go on till you come to the end: then stop."
–Lewis Carroll

A broad range of graph-based real world applications, e.g. collaborative filtering in recommender networks, link prediction in social networks, fraud detection and personalized graph search techniques require a deep understanding of the underlying graph structure. These networks can consist of millions of entities, and hence scalability is a critical concern for designing algorithms in these settings. The focus of my thesis is design and analysis of scalable algorithms for proximity search in graphs, which is a core primitive at the heart of many graph-based learning problems.

Graphs come up in many real-world problems. Any dataset, which is a collection of co-occurrences, or events relating a pair of entities, can be represented as a graph. The nodes are the entities, whereas the edges denote the pairwise relationships. A graph can be directed or undirected: in a directed graph, an edge from node i to j does not imply the existence of an edge from node j to i. On the other hand, in an undirected graph an edge between two nodes, can be thought of as two directed edges from i to j, and j to i. Some graphs, as we will show are inherently undirected in nature, while others are inherently directed.

Many graph-related machine learning applications can be framed as ranking problems. Let us take online social networks, e.g. Orkut or Facebook. Here the users are the entities, and the edges are formed via who-knows-whom relationships. An important application is to suggest friends to a newcomer. The idea is to predict who the newcomer is most likely to be friends with, based on the few friends he/she has already made.

Another example is movie recommendation systems (*Netflix, MovieLens*), where the underlying network is a bipartite graph of users and movies. The undirected connections are formed between an user, and the movies (s)he has rated, and they can be weighted by

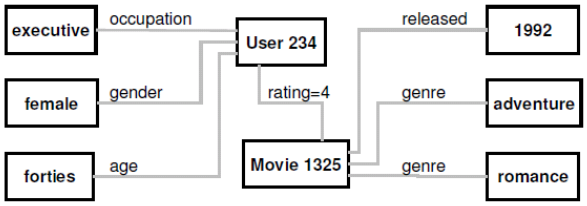


Figure 1.1: A snippet of a movie-recommendation network from Brand [2005]

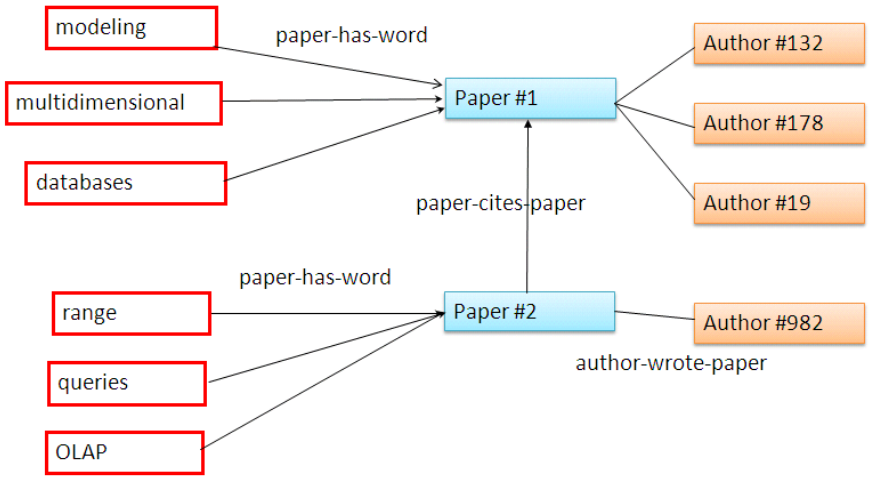


Figure 1.2: An example of a graph-representation of a publication network, inspired by an example in Balmin et al. [2004]

the rating. The question here is to recommend to the user movies (s)he is most likely to watch, based on the ratings so far. In fact this graph can be made even richer by using additional information about the movies and users. An example (courtesy Brand [2005]) of an association graph can be found in figure 1.1.

Keyword search in large publication databases is another graph-based problem setting, where the database can be seen as an entity-relation graph between papers, authors and words. Words are connected to a paper, if they appeared in its title; papers are connected via citations and cross-references, and an author is connected to a paper if (s)he was a co-author of the paper (figure 1.2). The goal is to find papers which are "contextually" similar to the query submitted by the user. Note that it is not enough to just return the documents whose titles contain the query word. This is true because a document can have the same context as the query, in-spite of not having the query word in its title. In fact figure 1.2 shows this effect. Note that this graph is inherently directed, because of the citations.

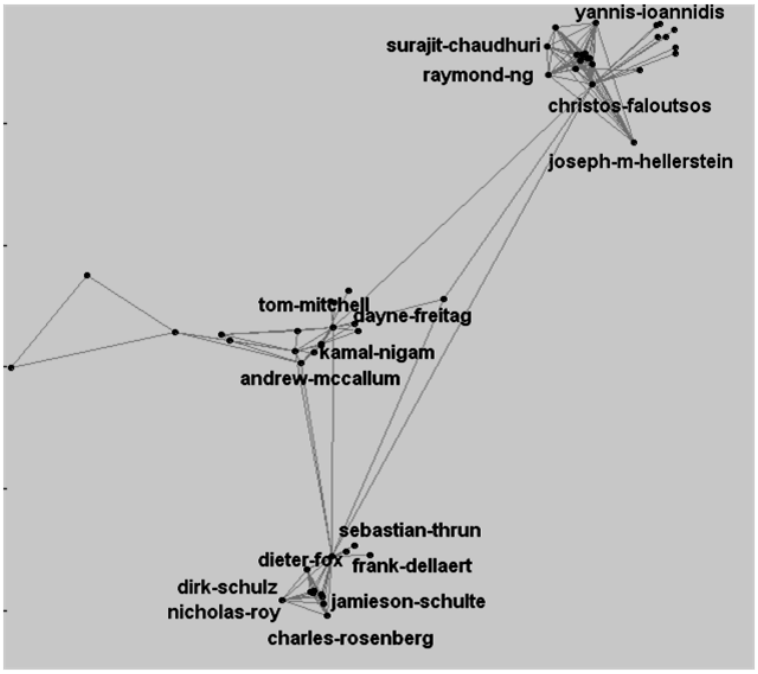


Figure 1.3: An example of a graph-representation of a co-authorship network

Another way of representing a publication graph is the co-authorship network (figure 1.3). A co-authorship network is undirected, and an edge can be weighted by the number of papers the two authors have written together. In this setting, one interesting question is: given the graph structure at this time-step, who is the most likely co-author of author "X" in the next time step. This naturally brings us to the link prediction problem. In fact most ranking problems can also be thought of as link prediction problems. And this is an excellent tool to evaluate the effectiveness of a proximity measure in a quantifiable way. We will describe link-prediction, and discuss previous empirical results in relation to it in section 2.5 of chapter 2.

The world wide web is a massive graph, where webpages are nodes, which are linked via hyperlinks. With the fast growth of the WWW, spam and malicious content is becoming a serious issue. Often an analyst hand-labels some webpages as spammy or malicious, and the spam detection algorithm aims to find the pages which are most likely to be spammy. This is again a ranking problem, where the intuition is, if a node is "similar" to a spammy node, then it is more likely to be spammy. Figure 1.4 shows a snapshot of a web-graph, where some nodes are labeled as spam.

Throughout this thesis, we focus on large publicly available publication networks like Citeseer and DBLP. In chapter 5, we also use the LiveJournal dataset, which is an online community. These graphs have some notable properties, which are often observed in many

other real world networks. They are large but sparse, i.e. there are many entities, but the average number of edges per entity is small. Let us define by degree the number of edges an entity or node has. For example, in a publication graph (see figure 1.2), the degree of an author will be the number of papers written by that author. We will define this more formally in the next chapter. These graphs have a power law degree distribution, i.e. there are a few nodes with high degree, and most nodes have small degree. They also exhibit the small world phenomenon (Kleinberg [2000]). Essentially, in spite of being so sparse, a large fraction of the graph can be reached in a few steps from a node. We will use these properties for designing faster and better algorithms in the chapters to come.

So far, we have given examples of interesting real world problems that involve graphs. The underlying question in all these problems is: given a node in a graph we would like to ask which other nodes are most similar to this node. Ideally we would like this proximity measure to capture the graph structure. There are a number of popular heuristics derived from the graph structure which are used as proximity measures between pairs of nodes in a graph. For example, one common measure is the length of the shortest path between two nodes. One expects that if this is small then the pair of nodes are similar. Another useful measure is the number of common neighbors. In fact, when the online social network Facebook suggests new friends to its users it also reports the number of common neighbors. The hope is that two people are probably similar if they have many common friends. Note that, common neighbors is based on the number of length 2 paths. One

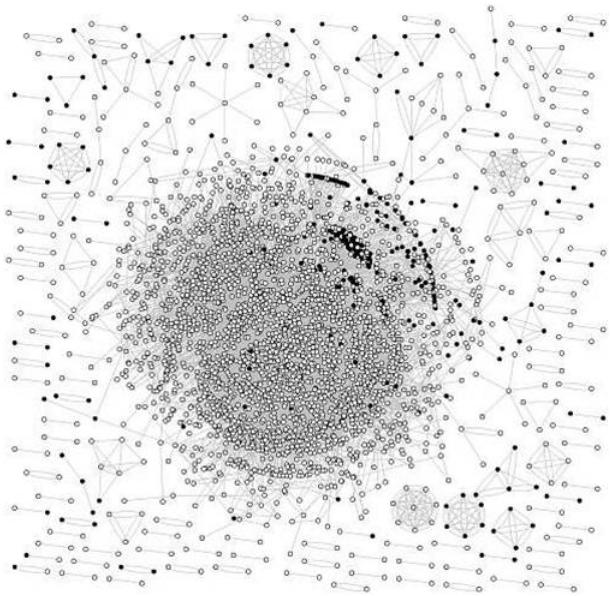


Figure 1.4: Partial corpus of webpages from webspam-UK2007. Black nodes have been labeled spam

could then also ask for the number of length 3 paths, and so on. An elegant and intuitive way to take into account the information from the ensemble of paths between two nodes is provided by random walks in graphs. Random walk based measures have been widely applied to recommender networks, contextual search in databases, web-spam detection, image segmentation, and graph-clustering, to name a few. Since the underlying graphs in real world problems are huge, and continuously evolving over time, it is important to design fast, space-efficient algorithms for computing nearest neighbors of the query node.

In chapter 2, we discuss a number of popular random walk-based measures, existing algorithms for computing them, and their applications to real world problems. Most of these measures use short term random walks, either in the form of restarting the random walk with some probability, or using a discount factor, i.e. stopping the random walk with some probability. Some measures also only look at paths of length upto a given constant. It has been shown empirically that focussing on short paths for computing proximity measures is not only useful for computational efficiency, but it has properties which yield meaningful ranking. Also, real world graphs exhibit the small world property, i.e. from a given node, it is often possible to reach most of the nodes of the graph in a few number of steps (4 or 5). This is why in most of our experimental work we saw that a path length of 10 suffices for extracting useful information from the graph structure. We also show the efficacy of these measures via qualitative link prediction tasks. Link prediction is a quite well known problem, where we answer the question: "which node that is not currently connected to query node q, should be connected to it?".

As we show in the next chapter, existing algorithms often need to iterate over the entire graph, or restrict the computation on smaller subgraphs. The first approach can be expensive if we want to compute nearest neighbors on the fly. The second approach does not have completeness guarantees, i.e. it does not guarantee that no nearest neighbor will be missed. In this thesis, we make an effort towards designing an efficient algorithmic framework for relevance search with formal guarantees. This framework is quite general in the sense that, it can be easily extended to a number of other random walk based measures, and also an external memory setting.

Our main contributions are as follows. We first design an algorithm for local nearest neighbor search. This algorithm adaptively finds a small neighborhood around a query node and prunes away the rest of the graph. We guarantee that no nearest neighbor will be missed in this process. Since our final goal is to improve relevance ranking, we also look at interactive settings where the user can provide feedback on the search results, and the system quickly generates a re-ranked list. We show how to extend our local algorithm to this setting as well.

Local algorithms often suffer from the presence of high degree nodes. For example, the size of the neighborhood explodes on encountering a high degree node, and this considerably slows down the algorithm. We propose a simple solution, and theoretically show

that this does not hurt the ranking very much. Empirically, our results indicate that this solution, in fact, improves link prediction accuracy.

Since real world graphs are very large, it is natural to consider graphs that are too huge to fit into main memory. We extend our algorithmic framework to this external memory setting as well.

We conclude our results on a more theoretical note. The popular heuristics for ranking and link prediction including random walks mainly look at paths of different length between two nodes in a graph. Using a well known generative model, we theoretically justify the common trends in the performance of these measures across a number of existing empirical studies.

In a nutshell,

1. We present efficient local algorithms with formal guarantees for nearest neighbor search in truncated hitting and commute times.

2. We design fast reranking algorithms suited for interactive applications where users can give feedback.

3. We study properties of graphs and proximity measures for improving these algorithms.

4. We investigate the problem of nearest-neighbor search in large disk-resident graphs.

5. Finally we propose a theoretical framework to justify popular link prediction heuristics.

We will now briefly describe these points.

In chapter 3, we focus on fast computation of proximity measures arising from short-term random walks, in particular truncated hitting and commute times. Hitting and commute times suffer from sensitivity to long paths (Liben-Nowell and Kleinberg [2003]), which results in non-local nature. It has also been empirically demonstrated that these are prone to be small if one of the nodes is of large degree (Brand [2005]), which renders them ineffective for personalization purposes. This is why we consider a truncated variant of random walks. In this chapter we show how to compute k ϵ-approximate nearest neighbors of query nodes s within T-truncated hitting time T'.

First we theoretically examine the compactness properties of truncated hitting and commute times, thereby proving the feasibility of local algorithms. Then we present a local algorithm, the main idea of which is to expand a neighborhood around the given query node, and stop when all nodes outside the neighborhood can be proven to be "uninteresting". 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 us 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 algorithm maintains a global lower bound on hitting time to the query node. The expansion is stopped when this bound exceeds the threshold T'; this guarantees that the hitting times of nodes outside the current neighborhood is larger than T', and hence uninteresting. Compact versions of the algorithms and results of these chapters can be found in Sarkar and Moore [2007], and Sarkar et al. [2008].

In chapter 4, we examine the problem of reranking search results by incorporating user feedback. We use a truncated version of harmonic functions (Zhu et al. [2003]), a popular random-walk based measure used widely for graph-based semi-supervised learning tasks. Surprisingly our former algorithmic framework can be generalized to this setting as well. The content of this chapter has also been published in Sarkar and Moore [2009].

In chapter 5, we mathematically show the connection between some useful measures (PPV, and discounted hitting times), which helps us to extend the characteristics of one measure to the other. In particular, we analyze the effect of high degree nodes on personalized pagerank, and subsequently discounted hitting times. We also show how to compute ranking in a disk-resident graph. We propose a simple clustered representation of a large graph on disk, and present an algorithm (RWDISK) which uses only sequential sweeps through the data to generate this clustering. We show that the local algorithms we have developed can be extended for computing other measures like degree normalized personalized pagerank in a clustered representation of the graph on disk. We have published these results in Sarkar and Moore [2010].

One main component of our experimental evaluation is various quantifiable link-prediction tasks on publicly available large networks. In chapter 6, we develop (Sarkar et al. [2010]) a theoretical basis for explaining the usefulness of various graph-based heuristics for link prediction. This allows us to understand what triggers different behaviors of different heuristics, as noted by previous empirical studies (Liben-Nowell and Kleinberg [2003], Brand [2005]) to some extent.

Chapter 2

Background and Related Work

Education never ends, Watson. It is a series of lessons with the greatest for the last.

— Sir Arthur Conan Doyle's Sherlock Holmes.

2.1 Introduction

A widely popular approach in graph-mining and machine learning literature is to compute proximity between nodes in a graph by using random walks: diffusion of information from one node to another. This chapter will be focused on these measures: the intuition behind their usefulness, the common issues with them as have been pointed out in earlier empirical studies, 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 (Haveliwala [2002]), hitting and commute times (Aldous and Fill [200X]), Katz measure (Katz [1953]), harmonic functions (Zhu et al. [2003]). Personalized page-rank vectors (PPV) have been used for keyword search in databases (Balmin et al. [2004]) and entity-relation graphs (Chakrabarti [2007]). Hitting and commute times have been shown to be empirically effective for query suggestion (Mei et al. [2008]), ranking in recommender networks (Brand [2005]) and image segmentation problems (Saerens et al. [2004]). Hopcroft and Sheldon [2007] show how to use hitting times for designing provably manipulation resistant reputation systems. Harmonic functions have been used for automated image-segmentation and coloring (Levin et al. [2004], Grady [2006]), and document classification (Zhu et al. [2003]).

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 measures bring together ideas from scientific computing, databases and graph theory. These algorithms range from clever offline preprocessing (Jeh and Widom [2002a], Fogaras et al. [2004]), fast online computation of sparse probability distributions (Berkhin [2006], Sarlós et al. [2006]) to hybrid algorithms which use best of both worlds (Chakrabarti [2007]). In section 2.3 we will 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 random walk-based graph mining problems. First we will discuss the different proximity measures arising from a random walk in 2.2. We will present the popular algorithms (2.3), applications (2.4), and finally experimental evaluation techniques (2.5) for different proximity measures.

2.2 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 i, j^{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. D is an $n \times n$ diagonal matrix, where $D_{ii} = \sum_j A_{ij}$. We will denote D_{ii} by the degree $d(i)$ (or 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 (Smola and Kondor [2003], Zhu [2006]). 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 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 (Aldous and Fill [200X]) with transition probabilities specified by this matrix. For an undirected graph, A is symmetric and L is symmetric positive semi-definite. This can be easily verified by observing that for any n dimensional vector $\boldsymbol{z}$ with real values entries, $\boldsymbol{z}^T L \boldsymbol{z} = \sum_{\{i,j\} \in E} (z(i) - z(j))^2$. We will denote the set of neighbors of a node i by $\mathcal{N}(i)$. For unweighted graphs (where all edges are of unit weight), $d(i)$ 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 $d^+(i)$. For a weighted graph, we denote by weighted degree the sum of edge weights of a node, i.e. $\sum_j A_{ij}$. The only difference is that, A_{ij} is a non-negative real number; not a binary number (zero or one) as in the case of unweighted graphs.

In a random walk, if the initial node is chosen from a distribution $\boldsymbol{x_0}$, then the distributions $\boldsymbol{x_0}, \boldsymbol{x_1}, ..$ are in general different from one another. However if $\boldsymbol{x_t} = \boldsymbol{x_{t+1}}$, then we say that $\boldsymbol{x_t}$ is the stationary distribution for the graph. According to the Perron-Frobenius

[1] Laplacian matrices for directed graphs have been proposed by Chung [2005].

theorem, there exists a unique stationary distribution, if P is irreducible and aperiodic. For graphs that are undirected, non-bipartite and connected (and therefore both irreducible and aperiodic) the stationary distribution at any node is proportional to its degree (Aldous and Fill [200X]).

2.2.1 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 2.3. We will briefly mention some algorithms for making SALSA and HITS faster after introducing them. An excellent study of pagerank can be found in Langville and Meyer [2003].

Query independent search and broad topic search Pagerank (Brin and Page [1998]) is a widely known tool for ranking web-pages. If the world-wide-web is considered as a graph with transition matrix P, then the pagerank is defined as the distribution that satisfies the following linear equation (2.1):

$$\boldsymbol{v} = (1-\alpha)P^T\boldsymbol{v} + \frac{\alpha}{n}\mathbf{1} \tag{2.1}$$

The probability of restarting the random walk at a given step is α. 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 this, since under this model 1) all nodes are reachable from all other nodes, and 2) the Markov chain is aperiodic. Also the restart probability α upper bounds the second largest eigenvalue (Kamvar et al. [2003]).

Pagerank does not provide topic-specific search results. Kleinberg [1999] 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:

$$\boldsymbol{a}(i) \leftarrow \sum_{j:j\in I(i)} \boldsymbol{h}(j) \tag{2.2}$$

$$\boldsymbol{h}(i) \leftarrow \sum_{j:j\in O(i)} \boldsymbol{a}(j) \tag{2.3}$$

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.

$$\boldsymbol{a} = A^T\boldsymbol{h} \qquad\qquad \boldsymbol{h} = A\boldsymbol{a} \tag{2.4}$$

Equation (2.4) 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 Lempel and Moran [2001], the matrix AA^T can be described as the bibliographic coupling matrix. This is because, the $\{i,j\}^{th}$ entry of 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]. Borodin et al. [2001] restrict the sum in eq. (2.3) to the k largest authority scores, while keeping eq. (2.2) 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. (2.3), while keeping eq. (2.2) intact. A similar approach was used by Gibson et al. [2000]. 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 $p = \infty$). Tsaparas [2004] provides an in-depth study of this special case (MAX) of these two families.

SALSA, introduced by Lempel and Moran [2001] 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 $h(i)$ and $a(j)$ if there is link $i \rightarrow j$ in the original subgraph. The nodes and links are

[2]We use the same names as mentioned in Tsaparas [2004].

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). Here, 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 $D_+^{-1}AD_-^{-1}A^T$. Let us denote this by $D_+^{-1}\mathcal{H}$, 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 $d^+(i)$, $A_r A_c^T$ 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 Lempel and Moran [2001]. 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 Kleinberg [1999] point out that ranking simply w.r.t in-degrees on the entire WWW graph is not adequate. Lempel and Moran [2001] point out that this conflict results from better techniques for assembling a topic-specific WWW subgraph. These techniques augment the original subgraph-expansion (Kleinberg [1999]) 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-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, Lempel and Moran [2001] demonstrate that SALSA is less vulnerable to the TKC effect than HITS.

[3]Lempel and Moran [2001] denote the matrix AA^T by H.

Note that neighborhood construction is key to both HITS, MAX and SALSA, and is a major computational bottleneck at query time. Najork and Craswell [2008] show how to precompute score maps for web-pages in order to drastically lower query latency. Najork et al. [2009] show the use of consistent sampling (Broder et al. [1998]) and bloom filters (Bloom [1970]) for approximating neighborhood graphs.

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. Liben-Nowell and Kleinberg [2003] also call this the "rooted pagerank".

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

$$\boldsymbol{v} = (1-\alpha)P^T\boldsymbol{v} + \alpha\boldsymbol{r} \tag{2.5}$$

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, Tsoi et al. [2003] 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 by Jeh and Widom [2002b], which is based on the intuition that two nodes are similar if they share many neighbors. The recursive definition of simrank is given by eq. (2.6)

$$s(a,b) = \frac{\gamma}{|I(a)||I(b)|}\sum_{i\in I(a), j\in I(b)} s(i,j) \tag{2.6}$$

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 (Aldous and Fill [200X]) 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

$$h_{ij} = \begin{cases} 1 + \sum_k p_{ik} h_{kj} & \text{If } i \neq j \\ 0 & \text{If } i = j \end{cases} \tag{2.7}$$

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 (Lovász [1996]).

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[4]. For undirected graphs[5] there is a direct analogy of hitting and commute times with electrical networks (Doyle and Snell [1984]). 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 (Chandra et al. [1989]) 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:

$$\psi(i,j) = \begin{cases} 1 + \sum_{k \in \mathcal{N}(i)} \dfrac{\psi(k,j)}{d(i)} & \text{If } i \neq j \\ 0 & \text{Otherwise} \end{cases} \tag{2.8}$$

[4]The proofs of these properties are largely inspired by Prof. Gary Miller's class on "Spectral Graph Theory and Scientific Computing" at Carnegie Mellon.

[5]A detailed discussion of hitting and commute times in directed graphs can be found in Boley et al. [2010].

Note that this linear system is identical to the definition of hitting time from i to j (eq. 2.7) 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:

$$L\boldsymbol{\psi} = \boldsymbol{y} \tag{2.9}$$

$\boldsymbol{y}$ 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. 2.9 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 (Fouss et al. [2004]). This matrix, i.e. L^+, can also be written as $(L - \frac{1}{n}\mathbf{1}\mathbf{1}^T)^{-1} + \frac{1}{n}\mathbf{1}\mathbf{1}^T$ (Fouss et al. [2004], Rao and S.Mitra [1971]).

Hence from eq. 2.9, we have:

$$\boldsymbol{\psi} - \mathbf{1}\frac{\mathbf{1}^T\boldsymbol{\psi}}{n} = L^+\boldsymbol{y} \tag{2.10}$$

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,

$$c_{ij} = vol(G)(\boldsymbol{e_i} - \boldsymbol{e_j})^T L^+ (\boldsymbol{e_i} - \boldsymbol{e_j}) \tag{2.11}$$

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 (Doyle and Snell [1984]).

From eq. (2.11) 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. 2.12).

$$\begin{aligned} c_{ij} &= vol(G)(l^+_{ii} + l^+_{jj} - 2l^+_{ij}) \\ &= vol(G)(\boldsymbol{e_i} - \boldsymbol{e_j})^T L^+ (\boldsymbol{e_i} - \boldsymbol{e_j}) \end{aligned} \tag{2.12}$$

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 (Brand [2005]) between two nodes is defined as $l^+_{ij}/\sqrt{l^+_{ii} l^+_{jj}}$. We discuss different ways of computing the above quantities in section 2.3.

More random-walk based proximity measures Faloutsos et al. [2004] use 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) for computing connection subgraphs, which is a small subgraph of the original network which best captures the relationship between two query nodes. Tong et al. [2007] use escape probabilities to compute *direction aware proximity* in graphs. Koren et al. [2006] 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.

2.2.2 Other Graph-based Proximity Measures

In their detailed empirical study of different graph-based proximity measures for link-prediction tasks Liben-Nowell and Kleinberg [2003] use the Katz score, number of common neighbors and Jaccard score. The authors also present higher-level *meta* approaches for link prediction which use the above measures as a basic component. These approaches include low rank approximations of the adjacency matrix, unseen bigrams from language modeling, and clustering.

Katz score Katz [1953] designed a similarity measure based on ensemble of paths between two nodes.

$$Katz(i,j) = \sum_{\ell=1}^{\infty} \beta^\ell . A^\ell(i,j) \tag{2.13}$$

It is to be noted 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, Liben-Nowell and Kleinberg [2003] introduce the weighted version of the Katz score, where edges of the graph are weighted.

Common neighbors, Adamic/Adar and Jaccard coefficient We 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 (Adamic and Adar [2003]) is similar to this, except the high degree common neighbors are weighed less. The Adamic/Adar score is defined as:

$$\texttt{Adamic/Adar}(i,j) = \sum_{k \in \mathcal{N}(i) \cap \mathcal{N}(j)} \frac{1}{\log |\mathcal{N}(k)|} \tag{2.14}$$

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.

$$J(i,j) = \frac{|\mathcal{N}(i) \cap \mathcal{N}(j)|}{|\mathcal{N}(i) \cup \mathcal{N}(j)|} \tag{2.15}$$

The matrix of Jaccard coefficients can be shown to be positive definite (Gower [1971]).

Often the Jaccard coefficient is used to compute document similarity in information retrieval (Salton and McGill [1986]). Note that for the bag of words model computing this score between two documents can be expensive. Broder [1997] introduced the idea of shingles to compute this using a efficient randomized algorithm. 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 the expectation of this estimate is in fact the Jaccard score.

2.2.3 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/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 nodes than the irrelevant ones. We will now formalize this idea.

Consider a partially labeled dataset with l labeled points, $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_l, y_l), \boldsymbol{x}_{l+1}, \ldots, \boldsymbol{x}_n\}$. 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 and Jaakkola [2001] 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. [2003] 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 harmonic functions based on the intuition that nodes *close* in the graph have similar labels. A harmonic function is defined as $f : V \rightarrow \mathcal{R}$ 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. Let's assume that for the labeled nodes we assign $f(i) = 1$, if i has label 1, and 0 otherwise. For the unlabeled nodes we enforce the following relation for smoothness,

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

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:

$$f_u = (I - P_{uu})^{-1} P_{ul} f_l \tag{2.16}$$

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 (Doyle and Snell [1984]) ($P_{esc}(s,t)$) 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 regularize it to enforce smoothness over the graph topology. Zhu [2006] provides 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. for Zhu et al. [2003]) sometimes leads to a function closely related to a random walk on the underlying graph. Agarwal and Chakrabarti [2007] connect the Laplacian regularization framework with random walk based measures (e.g. personalized pagerank) and provide generalization guarantees for the latter in directed graphs.

2.2.4 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 (Ng et al. [2001]) 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 and Shi [2001] provide a random-walk based probabilistic interpretation of spectral clustering algorithms including Normalized Cut (Shi and Malik [1997]). Building the affinity matrix poses an important algorithmic question, which has not yet been studied extensively. E.g. for the scheme above, the choice of σ is crucial. Zhu [2006] describes different techniques for building graphs and estimating σ.

Global graph clustering using random walks Saerens et al. [2004] exploit equation (2.12) to embed a graph and provide distances between any pair of nodes. Yen et al. [2005] transform a dataset to a graph, where every node represents a datapoint, and is connected to its k nearest neighbors in Euclidian distance. Now the authors replace traditional Euclidian distances between datapoints, by commute times between the corresponding nodes in the graph. They show that standard clustering algorithms (e.g. K-means) produce much better results when applied to commute times. These techniques exploit the fact that commute times are robust to noise and provide a finer measure of cluster cohesion than simple use of Euclidian distances.

Harel and Koren [2001] 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 and Ghahramani [2006] 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. These papers mostly demonstrate their results on Euclidian datasets transformed to graphs.

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 (Spielman and Teng [2004], Andersen et al. [2006a]) for using random walk based approaches for computing good quality local graph partitions (cluster) near a given node. The main intuition is that a random walk started inside a good quality 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 Spielman and Teng [2004], conductance is defined as:

$$\Phi_V(S) = \frac{E(S, V \setminus S)}{min(vol(S), vol(V \setminus S))} \tag{2.17}$$

A good 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.

A formal algorithm to compute a low conductance local partition near a seed node has been proposed by Spielman and Teng [2004]. This algorithm propagates the probability mass from this seed node to the other nodes, and at any step rounds the small probabilities to zero. This leads 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. Andersen et al. [2006a] improve upon the above result by computing local cuts from personalized pagerank vectors from the predefined seed nodes. This result is generalized by the authors to strongly connected directed graphs in their subsequent paper (Andersen et al. [2006b]).

We have now introduced the major random-walk based proximity measures. We will next discuss the existing algorithms for fast computation of these proximity measures. Then we will describe different applications of these measures. Finally we will conclude with a brief description of techniques for evaluating these measures, and pointers to some publicly available graph datasets.

2.3 Related Work: Algorithms

In this section, we will describe the popular algorithms designed for computing the proximity measures described in section 2.2. Katz score, 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.

2.3.1 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. [2004] 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 [2005] computes 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 the above techniques are not applicable to a directed graph.

Spielman and Srivastava [2008] have designed a fast algorithm for computing pairwise commute times within a constant factor approximation. The idea is to use the Johnson-Lindenstrauss lemma (Johnson and Lindenstrauss [1984]) in conjunction with a nearly linear time solver (Spielman and Teng [2004]) 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. We will discuss the practical issues with this algorithm in chapter 3.

Mei et al. [2008] compute hitting time by iterating over the dynamic programming step for a fixed number of times. This leads to the *truncated* hitting time (later we will argue that this is better than ordinary 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. However in this approach, there is no formal guarantee that the algorithm will not miss a potential nearest neighbor.

Truncated hitting time is a short-term variant of the original hitting time. It has been shown by several authors (Liben-Nowell and Kleinberg [2003], Brand [2005]) that original hitting time is sensitive to long range paths and tends to favor high degree nodes. These properties are not desirable for ranking purposes; first, we do not want the proximity measure to be dependent on parts of the graph far away from the query. Second, we want a personalized measure, e.g. in the context of movie suggestion in Netflix, we do not want to recommend the most popular movies (movies with highest degree in a bipartite graph

representation of movies and users connected via ratings). Liben-Nowell and Kleinberg [2003] also showed the effectiveness of using walks of smaller length in link-prediction. This is why we believe that truncated hitting time will be more effective for ranking problems. We discuss this in more detail in chapter 3, where we present a formal approach of computing nearest neighbors in truncated hitting time to a node by doing neighborhood expansion. Our goal is to compute k ϵ-approximate nearest neighbors of query nodes s within T-truncated hitting time T'.

2.3.2 Algorithms for Computing Personalized Pagerank and Simrank

Personalized pagerank (PPV) is 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 (Jeh and Widom [2002a]) 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 PPV at query time because of the sheer size of the internet graph. We will briefly sketch a few well-known algorithms here.

Haveliwala [2002] divide the web-corpus into k (16) topics. The PPV 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 PPV for the query is now a linear combination of the topics, weighted by the probability distribution of the query over the topics.

Jeh and Widom [2002a] 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 the hub nodes. If this set is chosen such that most paths in the graph go via nodes in it, 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. The intuition behind this algorithm appears in Figure 2.1. Here, a triangle with root h represents the paths starting at h. A notch in the

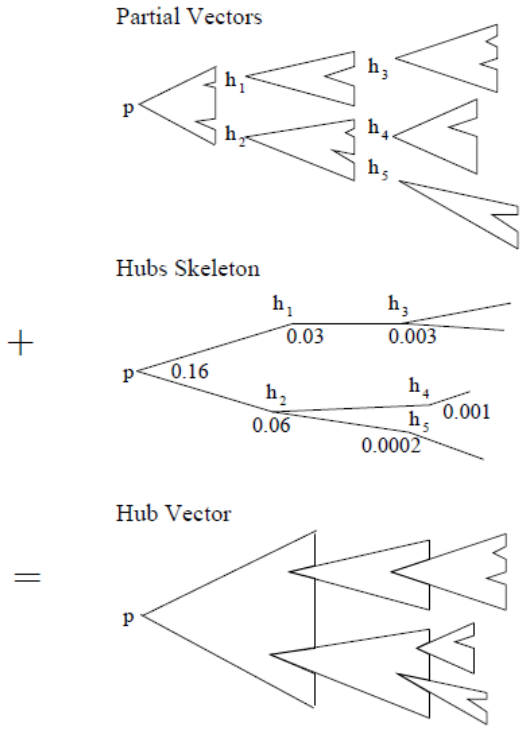


Figure 2.1: Construction of hub vectors from partial vectors and the hubs skeleton, provided by Jeh and Widom [2002a].

triangle shows the hub where the computation of a partial vector stopped. The hub vector is constructed by combining the partial vectors with the corresponding hub weights from the hub skeleton.

Fogaras et al. [2004] 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 and Rácz [2005] 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 saves 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. [2006] improve upon the sampling algorithm in Fogaras et al. [2004] by introducing rounding and sketching based deterministic algorithms which reduce 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 Jeh and Widom [2002a]). 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 further improved by using Count-Min (Cormode and Muthukrishnan [2005]) sketches 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 [2006] 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 (eq. 2.5) 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 by Andersen et al. [2006a]. 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 [2007] shows 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 personalized pagerank for context-sensitive keyword search in entity-relation graphs. Since the vocabulary is huge, it is often prohibitive to cache personalized pagerank vectors 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. [2004] 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 Fogaras et al. [2004]) 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.

2.3.3 Algorithms for Computing Harmonic Functions

Zhu [2006] presents a detailed 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. (2.16). Zhu [2005] presents a detailed comparison among the label propagation, loopy belief propagation, and conjugate gradient approaches. 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.

Zhu and Lafferty [2005] 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. Azran [2007] computes harmonic functions by computing the eigen-decomposition of a *Rendezvous* graph, where the labeled nodes are converted into sink nodes. In chapter 4, we use a truncated version of harmonic functions to rerank search results after incorporating user feedback. We present 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.

2.4 Related Work: Applications

The random walks approach has been highly successful in social network analysis (Katz [1953]) and computer vision (Gorelick et al. [2004], Qiu and Hancock [2005]), personalized graph search (Jeh and Widom [2002a], Haveliwala [2002], Fogaras et al. [2004]), keyword search in database graphs (Balmin et al. [2004], Chakrabarti [2007]), detecting spam (Gyongyi et al. [2004], Wu and Chellapilla [2007]). Here we briefly describe some of these applications.

2.4.1 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. [2004] use the average hitting time of a random walk from an object boundary to characterize object shape from silhouettes. Grady and Schwartz [2006] 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 image segmentation (Qiu and Hancock [2005]) and robust multibody motion tracking (Qiu and Hancock [2006]).

Harmonic functions have been used for colorizing images (Levin et al. [2004]), and for automated image-segmentation (Grady [2006]). 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.

2.4.2 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. [2003] 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.

In the context of text analysis, where the objects are stored in a database, we will often talk about *entity relation (ER)* graphs. The *entity relationship* model (Ullman and Widom [1997]) is a commonly used language for an abstract representation of the database structure. For example, an ER graph can contain as entities people, companies and emails. The relations connecting these entities can be "sent" or "received" between a person and an email, or "employed by" between a person and a company, etc. There has been a body of work which models relational data, e.g. plate models (Buntine [1994]), Probabilistic Relational Models (Friedman et al. [1999]), and Probabilistic Entity-Relationship Models (Heckerman et al. [2004]) etc. These models have attributes associated with each entity, and are often aimed at predicting the values of these attributes or the relations between entities. In some cases, the objective is to learn the structure and the parameters

of the probabilistic relationships. In contrast, the goal of our examples is to use the ER graph derived from data, and extract useful information from its link structure. The way in which these examples capture node attributes, is to actually build the graph based on attribute similarity, e.g. add weighted or unweighted edges based on how similar two documents are. Also, in this chapter, we will mention some approaches which aim to learn the weights or transition probabilities on the edges which already exist in the graph.

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 of Balmin et al. [2004] 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 [2007] also uses the same graph representation.

Mei et al. [2008] 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 to q are suggested as alternate queries. Lafferty and Zhai [2001] 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 Collins-Thompson and Callan [2005]. 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 weighted more in later steps.

Minkov et al. [2006] 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. [2004] 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. Agarwal et al. [2006] 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 (Tomlin [2003]) and maximum margin optimization to learn rankings (Joachims [2002]).

2.4.3 Collaborative Filtering

Hitting and commute times have been successfully used for collaborative filtering. Fouss et al. [2004] use hitting and commute times for recommending movies in the MovieLens dataset and show that these measures perform much better than shortest path on link prediction tasks. Brand [2005] uses 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 2.2) does not have this problem since the effect of individual popularity is normalized out. This observation points to the fact that short term random walks are more useful for prediction tasks.

2.4.4 Combating Webspam

There have been a number of learning algorithms to aid spam detection in the web, which is an extremely important task. Some of these algorithms are content-based (Ntoulas et al. [2006]), i.e. they primarily use the content of an webpage, whereas others extract information from the hyper-link structure of the web to classify a page as spam. We will focus on some examples of the second approach.

Graph-based algorithms look at the link structure of the web to classify web-spam. We will briefly discuss two of these. TrustRank (Gyöngyi et al. [2004]) 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 Joshi et al. [2007b]. 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.

2.5 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.

2.5.1 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 network, the link prediction problem seeks this answer: which new connections among the entities are likely to occur in the future.

In general, there are a few ways of conducting a link prediction experiment as follows:

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, it is considered for deletion. Liben-Nowell and Kleinberg [2003] 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 and Kleinberg [2003] 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 the works of Liben-Nowell and Kleinberg [2003], and Brand [2005].

A surprising result obtained by Liben-Nowell and Kleinberg [2003] 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. Let us recall that the Adamic/Adar score 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 conclude that the hitting and commute times suffer from their sensitivity to long paths. The most effective measure was shown to be the Katz measure (Katz [1953]), with a small discount factor (close to zero). Also, Brand [2005] 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. In the next chapter we will describe truncated hitting times and show how to compute approximate nearest neighbors in this measure efficiently.

Another observation from Liben-Nowell and Kleinberg [2003] and Brand [2005] 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 look at the ensemble of length 2 paths). We also notice this trend in our experiments.

2.5.2 Publicly Available Data Sources

In section (2.4) we have shown 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 by Brand [2005], Fouss et al. [2004] 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 by Chakrabarti [2007] etc. We have used this data for experiments in chapter 3.

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 by Balmin et al. [2004]. We use this data for evaluating our reranking algorithm in chapter 4.

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 by Liben-Nowell and Kleinberg [2003].

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 (Hirai et al. [2000]) http://diglib.stanford.edu:8091/~testbed/doc2/WebBase/. Different snapshots of the web available from this project were used by Haveliwala [2002], Jeh and Widom [2002a], Fogaras et al. [2004], Sarlós et al. [2006].

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).

2.6 Discussion

In this chapter, we have built 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. We will identify a interconnected set of these questions.

The first question is related to computational efficiency: how do we obtain ranking using these measures on the fly? Surprisingly enough, this is still an active area of research. This will be discussed more in detail in chapter 3. We also present an interactive setting where the user can give feedback to improve the quality of the search results. This is described in chapter 4.

Second, what are the properties of these measures, and how do they relate to the properties of the graph? This question, is not only of pure theoretical interest; it can also help us design better algorithms. Our analysis of the measures, and their connection to the properties of the graph is key to some of the techniques and algorithms described in the different chapters.

Third, how do we deal with graphs that are too big to fit into main memory? This will help us transfer our fast technology to end users with modest computing resources. We show how to cleverly arrange the graph on the disk, such that "similar" nodes are placed on the same disk page. This is the content of chapter 5.

Finally, there is a myriad of different similarity measures, which have been used for ranking. These measures exhibit a consistent trend in terms of which yields better ranking. Is there a way to justify these common trends in the performances of the popular heuristics? In chapter 6, we connect some useful heuristics with a well studied generative model, and theoretically show why graph-based heuristics capture the "similarity" between pairs of nodes so well.

In the next chapter we describe the core component of this thesis. We mainly focus on hitting and commute times. Previous work has pointed out the negative influence of long range paths on hitting and commute times. Hence we examine a short term variant, namely the truncated hitting and commute times. We describe a local algorithm which helps us find nearest neighbors in these measures. We will exploit the main idea of this algorithm for obtaining ranking in other random walk based measures in the next chapters.

Chapter 3

Fast Local Algorithm for Nearest-neighbor Search in Truncated Hitting and Commute Times

The shortest distance is not always the path of least resistance.

3.1 Introduction

In the last chapter we have described a number of real-world applications of graph-based proximity measures. All these applications require ranking, i.e. computing nearest neighbors under a given measure. While random walk based proximity measures like hitting and commute times have been used as proximity measures before, these measures are hard to compute. They also suffer from some issues which root from their dependence on long range paths. In this chapter, first we propose a truncated variant of hitting and commute times. This variant looks at paths of length smaller than a given truncation parameter. Next, we present an algorithm (GRANCH) which looks at local neighborhoods in order to compute all pairs of nearest neighbors in hitting and commute times. We also augment this with a simple sampling scheme to give a local algorithm to compute nearest neighbors of a query node on the fly.

Our work has two main parts. First, we study the properties of these measures, which show that there is a small local neighborhood of potential near neighbors around the query node. Next, we design an algorithm to actually find this neighborhood without computing the measures exactly. Our algorithm has provable completeness guarantees and have been empirically shown to be efficient. Also the main idea behind the algorithm is generalizable to many other random walk based measures, as we will show in the chapters to come.

These also include the discounted variant of hitting time, where there is a small probability of stopping the random walk at any step.

Let us start with the computational aspect of nearest neighbor search under hitting and commute times. We will then talk about truncated hitting and commute times. As we have discussed in detail in chapter 2, hitting and commute times are popular measures of similarity, and they are widely used for clustering and ranking. Often authors cluster the graph, and restrict the computation to heuristically chosen subgraphs. Naive computation of all pairs of commute times involve computing the pseudo-inverse of the graph Laplacian. Sometimes cubic computation is avoided by using sparse matrix manipulation techniques (Brand [2005]). Low rank approximation (Saerens et al. [2004]) of the inverse of the graph Laplacian matrix L is another way to avoid prohibitive computation. However, these techniques are better suited for offline settings, and are too expensive for ranking at query-time.

One inherent problem with computing hitting/commute time is their asymmetric nature. 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: from the 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:

$$P^t(i,j) = \sum_k P(i,k)P^{t-1}(k,j) \tag{3.1}$$

$$P^t(i,j) = \sum_k P^{t-1}(i,k)P(k,j) \tag{3.2}$$

Consider the t step probability of reaching node j from node i. Using eq. 3.1 we can compute t step probabilities from all nodes to node j, whereas eq. 3.2 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

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

we would be able to use dynamic programming to compute hitting time from i to all other nodes. However this is not true, since $h^{t-1}(i,k)$ includes paths through j, whereas $h^t(i,j)$ is defined to look at paths that stop at j.

We will show this issue with several other representations of hitting times. Consider commute time from eq. (2.12). Say we want to compute nearest neighbors of node i in commute time from node i. This will require computing $l^+_{ii} + l^+_{jj} - 2l^+_{ij}$, for all j. Computing the i^{th} row of L^+ gives us the $(l^+_{ii} - 2l^+_{ij})$ part of the commute time for all j; however we also need the l^+_{jj} part. This will require computing the diagonal of the inverse Laplacian. However to our knowledge, there is no algorithm to compute this using one linear

solver. Brand [2005] states that for large markov chains the square of the inverse stationary distribution of j is a close constant factor approximation to l^+_{jj}; however the authors do not provide any approximation guarantee. In short, it is only tractable to compute hitting and commute times on graphs with a few thousand nodes for most purposes. Also these techniques are not applicable to a directed graph.

Recall the relationship of pairwise commute times with squared Euclidian distance in a transformed space. A straightforward application of the transformation would require computing the square root of the inverse of the Laplacian matrix. Instead Spielman and Srivastava [2008] propose to use a random projection of the nodes, which directly uses the Johnson-Lindenstrauss lemma (Johnson and Lindenstrauss [1984]) to compute the random projections by solving $O(\log n/\epsilon^2)$ linear systems over the graph Laplacian. This approach is only applicable to undirected graphs. For experimental purposes the factor of $1/\epsilon^2$ masks the asymptotic behavior of $\log n$, for a reasonably small value of ϵ. Another problem is that this technique uses a complicated nearly-linear time algorithm for solving sparse diagonally dominant linear system of equations (Spielman and Teng [2004]). To our knowledge this has not yet been realized in practice.

Mei et al. [2008] use truncated hitting times to obtain ranking for the purpose of query suggestion. The authors first find a small sub-graph via depth-first search from a query node, and stop when the size exceeds a given threshold. Then the computation of hitting time is restricted to this subgraph. The problem with this approach is that there is no formal guarantee the algorithm will not miss a potential nearest neighbor.

In this chapter, we provide formal approach of computing nearest neighbors in truncated hitting time to a node by doing neighborhood expansion. We show how to compute approximate nearest neighbors in truncated hitting time to a query node. Using upper and lower bounds on the hitting times within an active neighborhood, we can quickly prune away the "uninteresting" part of the graph.

Our key contributions are: 1) we design a tractable algorithm to compute all "interesting" pairs of nearest neighbors in truncated commute time; 2) we show how to combine sampling with deterministic pruning to retrieve top k neighbors of a query in truncated commute time incrementally without caching information about all nodes in the graph; 3) we investigate locality properties of truncated hitting and commute times; 4) we show that on several link prediction tasks these measures outperform personalized pagerank in terms of predictive power, while on others they do comparably; 5) our algorithm can process queries at around 4 seconds on average on graphs of the order of $600,000$ nodes on a single CPU machine.

3.2 Short Term Random Walks on Graphs

In this section, we motivate and then define truncated hitting and commute times, show their relation to the un-truncated versions, and then present some interesting locality properties of these measures in directed and undirected graphs.

3.2.1 Truncated Hitting Times

Consider the undirected graph such that two clusters C_s and C_t are connected via a single edge between s, t. Node s and t are direct neighbors. However the hitting time from s to t is very large, because the random walk spends a lot of time in parts of the graph which are far away from s. In fact for the above case where the edge (s, t) divides the graph in two clusters, C_s and C_t, such that s belongs to C_s and t to C_t, we can prove that $h(s, t) = 2 * E(C_s) + 1$, where $E(C_s)$ is the number of edges in cluster C_s. This implies that the rank of t in the list of nodes sorted according to the hitting time from s can be pushed down arbitrarily if we increase number of edges in C_s. This shows the non-local nature of hitting times.

Hitting and commute times decrease if there are many short paths between the pair of nodes. On the other hand, as observed by Liben-Nowell and Kleinberg [2003], two major drawbacks are that they favor high degree nodes. In fact Lovász [1996] show that the pairwise commute time between nodes s and t can be upper and lower bounded as

$$m \leq \left(\frac{1}{d(s)} + \frac{1}{d(t)}\right) C(s, t) \leq \frac{2m}{1 - \lambda_2} \left(\frac{1}{d(s)} + \frac{1}{d(t)}\right)$$

Here, λ_2 is the second largest eigenvalue of the matrix $D^{-1/2} A D^{-1/2}$. Note that $1 - \lambda_2$ is the spectral gap Lovász [1996], which determines how fast a markov chain mixes, i.e. approaches the stationary distribution. Hence this expression says that the commute time is small if s or t has large degree, and this effect is more pronounced in graphs with a large spectral gap, i.e. where the chain mixes fast.

This renders them ineffective for personalization purposes. They are also sensitive to parts of the graph far away from the nodes, even when short paths between the nodes exist. This results in non-local nature.

In order to overcome these problems we define a *T-truncated hitting time*, where we only consider paths of length less than T. In Section 3.2.2 we show that h^T approaches the true hitting time as $T \to \infty$. We use h, h^T interchangeably to denote truncated hitting time.

The T truncated hitting time, i.e. h_{ij}^T from i to j, measures the expected number of steps taken by a random walk starting at node i to hit j for the first time. If i and j are

more than T hops away then this definition sets the hitting time to be T. Formally we can define this as follows:

$$h_{ij}^T = \begin{cases} 1 + \sum_k p_{ik} h_{kj}^{T-1} & \text{If } i \neq j \text{ and } T > 0 \\ 0 & \text{Otherwise} \end{cases} \tag{3.3}$$

where h^T is defined to be zero if $i = j$ or if $T = 0$. The above equation expresses h^T in a one step look-ahead fashion. The expected time to reach a destination within T timesteps is equivalent to one step plus the average over the hitting times of its neighbors to the destination. Equation (3.3) can be easily computed by using Gauss Seidel iterations. Let the truncated hitting times matrix be H^T. Note that the hitting time from all nodes to a fixed destination, i.e. a column of H^T, can be computed in $O(T\Delta n)$ time, where Δ is the average degree of a node. However if we need to naively compute the hitting time from one node to everyone else, i.e. a row of H^T, $H^T(i, *)$ then for all j we will compute the corresponding column in order to get one entry $H^T(i, j)$. Hence we will end up computing the entire matrix. To avoid this expensive computation, which examines all pairs of nodes, we develop a very efficient framework for doing a range-search to retrieve the k approximate nearest neighbors of *all* nodes in a graph.

3.2.2 Truncated and True Hitting Times

In this section we examine the relation between the true and truncated hitting times. We can express the truncated hitting time as follows:

$$h_{ij}^T = \sum_{t=1}^{T-1} t Pr\{i \xrightarrow{t} j\} + (1 - \sum_t Pr\{i \xrightarrow{t} j\})T$$

where $Pr\{i \xrightarrow{t} j\}$ denotes the probability that i reaches j for the first time in exactly t steps.

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

3.2.3 Locality Properties

It is easy to compute hitting time to a node by dynamic programming. The question however is, how do we compute the hitting time from the potential nearest neighbors to a node, without exploring the entire graph. Note that we can take all nodes within T hops of a node to compute the truncated hitting times exactly. If n is the number of nodes in

the graph, then for a small value of T (e.g. 4 or 5) the number of nodes within T hops of a node can be $O(n)$, which may lead to computing all pairs of hitting times if we are interested in computing nearest neighbors of all nodes at one time. This is because of the small world phenomenon (described by Milgram [1967]), which tells us that real graphs have small diameter (the longest shortest path between any pair of vertices). Instead, now we will focus on the locality properties of truncated hitting and commute time. Ideally we want to find a small neighborhood around the query node which consists of all the close neighbors in hitting or commute time from the query. Hence in this section, we would examine the locality properties by investigating how many nodes are within T-truncated hitting time of T', and $2T$-truncated commute time of $2T'$ from the query node.

A Bound on Nodes within T' Hitting Time from i

Using this part we get a very simple bound for all nodes within T' truncated hitting time from a node, where T the truncation parameter. Mathematically, we want to bound the size of the set $\{j|h^T(i,j) \leq T'\}$.

Let $P_{ij}^{<T}$ denote the probability of hitting node j starting at i within T steps. Let us recall that the probability of hitting j in exactly t steps for the first time from i is denoted by $Pr\{i \xrightarrow{t} j\}$. Also note that this quantity is smaller than the t step transition probability from node i to node j, i.e. $P^t(i,j)$. Then we have,

$$\begin{aligned} & T' \geq h_{ij}^T \geq T(1 - P_{ij}^{<T}) \\ \implies \quad & P_{ij}^{<T} \geq \frac{T - T'}{T} \end{aligned} \tag{3.4}$$

Define S_i^+ as the neighborhood of i which consists of only the nodes within hitting time T' **from** i.

$$\begin{aligned} \sum_{j \in S_i^+} P_{ij}^{<T} &= \sum_{j \in S_i^+} \sum_{t=1}^{T-1} Pr\{i \xrightarrow{t} j\} \\ &= \sum_{t=1}^{T-1} \sum_{j \in S_i^+} Pr\{i \xrightarrow{t} j\} \\ &\leq T - 1 \end{aligned}$$

However the left hand side is lower bounded by $|S_i^+|\frac{T-T'}{T}$ using eq. (3.4), which leads us to the upper bound:

$$|S_i^+| \leq \frac{T^2}{T - T'} \tag{3.5}$$

A Bound on Nodes within T' Hitting Time to j

Define S_j^- as the neighborhood of j which consists of only the nodes within hitting time T' **to** j.

All total there are only $N\frac{T^2}{T-T'}$ pairs within T' hitting distance. Without any distributional assumption, we can only claim that there can be at most $K\sqrt{n}$ columns with $\sqrt{n}$ elements in it. Note that other bounds are possible, but for $\sqrt{n}$ we can say that there are not *too many* columns with *too many* nonzero elements in them. For general graphs this is all we can claim so far. However if the graph is undirected, i.e. the markov chain is reversible, then we can make stronger claims, as shown below.

Similarly as before we have

$$P_{ij}^{<T} \geq \frac{T-T'}{T} \tag{3.6}$$

We use the property of a reversible chain, since the underlying graph in undirected. Let π be the stationary distribution of the random walk. Then we have, (Aldous and Fill [200X], chapter 3)

$$\pi(i)P^t(i,j) = \pi(j)P^t(j,i) \tag{3.7}$$

$$\begin{aligned}
\sum_{i\in S_j^-} P_{ij}^{<T} &= \sum_{i\in S_j^-}\sum_{t=1}^{T-1} Pr\{i \overset{t}{\to} j\} \\
&\leq \sum_{t=1}^{T-1}\sum_{i\in S_j^-} P_{ij}^t \\
&= \sum_{t=1}^{T-1}\sum_{i\in S_j^-} P_{ji}^t \frac{\pi(j)}{\pi(i)} \\
&\leq T \times \frac{d(j)}{d_{min}}
\end{aligned}$$

We use the fact that $\pi(j)/\pi(i) = d(j)/d(i) \leq d(j)/d_{min}$, where d_{min} is the minimum degree.

However the left hand side is lower bounded by $|S_j^-|\frac{T-T'}{T}$ using eq. (3.6), which leads us to the upper bound:

$$\boxed{|S_j^-| \leq \frac{d(j)}{d_{min}} \times \frac{T^2}{T-T'}} \tag{3.8}$$

A bound on nodes within $2T'$ commute time to j

We already have a bound of $T^2/(T-T')$ on the number of nodes with hitting time smaller than T' **from** a node i, and a bound of $d(i)T^2/(T-T')$ for undirected graphs. We want to bound the number of nodes with commute time $2T'$. Here we will give a bound for general graphs, and a stronger one for undirected graphs.

In other words we have proven that $\{j|h_{ij}^T < T'\}$ and $\{j|h_{ji}^T < T'\}$ are small. Now we need to prove that $\{j|h_{ij}^T + h_{ji}^T < 2T'\}$ is also small. Note that the above set consists of

1. $S_1 = \{j|h_{ij}^T \leq T' \bigcap h_{ij}^T + h_{ji}^T < 2T'\}$
2. $S_2 = \{j|h_{ij}^T \geq T' \bigcap h_{ij}^T + h_{ji}^T < 2T'\}$

S_1 can have at most $T^2/(T-T')$ nodes. Now consider S_2. $|S_2|$ will have size smaller than $\{j|h_{ji}^T < T'\}$. Using our result from before we can say the following:

Lemma 3.2.1. *Let T be constant w.r.t n. If T' is bounded away from T by a **constant** w.r.t n, i.e. $T^2/(T-T')$ is constant w.r.t n, then in any (directed or undirected) graph not more than $O(\sqrt{n})$ nodes will have more than $O(\sqrt{n})$ neighbors with truncated commute time smaller than $2T'$.*

Lemma 3.2.2. *For an undirected graph, the total number of neighbors within $2T'$ T-truncated commute time can be upper bounded by $(1 + d(i)/d_{min})\frac{T^2}{T-T'}$.*

$$\{j|c_{ij}^T < 2T'\} \leq |S_i^+| + |S_i^-| \leq \left(1 + \frac{d(i)}{d_{min}}\right)\frac{T^2}{T-T'} \tag{3.9}$$

3.3 GRANCH: Algorithm to Compute All Pairs of Nearest Neighbors

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

The main intuition is that we think about the graph in terms of n overlapping subgraphs. Each subgraph is a bounded neighborhood for one node (discussed in detail later in the section). Consider the hitting times arising from the random walk from any node in this neighborhood to the destination. We provide upper (*pessimistic*) and lower (*optimistic*) bounds of these hitting times, and show how to combine information from all these overlapping subgraphs to obtain k nearest neighbors for all nodes. We will use the terms "optimistic" and "lower", and "pessimistic" and "upper", interchangeably. In order to motivate the GRANCH algorithm we present the following claim before formally describing the algorithm.

Claim: Given k, ϵ, T and $T' \leq T$, GRANCH returns any k neighbors j of i $\forall i$, s.t. $c^T(i,j) \leq c^T(i, k_{i,T'})(1+\epsilon)$, where $k_{i,T'}$ is the true k^{th} nearest neighbor of i within T-truncated hitting time T'.

This makes sense since in all applications hitting and commute times are used for ranking entities, e.g. recommend the k best movies based on choices already made by an user; or, find the k most likely co-authors of an author in a co-authorship network.

Suppose we want to estimate h_{ij}^T for pairs of nodes i, j which have relatively low h_{ij}^T values, and suppose we want to avoid $O(n^2)$ time or space complexity. In this case we cannot do anything that requires representing or iterating over all pairs of nodes. In fact, we cannot even afford to iterate over all pairs of nodes that are less than T hops apart, since in general the number of such pairs is $O(n^2)$. Suppose we restrict computation to iterate only over some subset of pairs of nodes, which we will call the Active-Pairs-set or the *AP* set. The interesting question is how good are the pessimistic and optimistic bounds we can get on h_{ij}^T, $\forall i, j$ using only $O(|AP|)$ work.

As mentioned before we will consider bounded neighborhoods of each node in the graph. For clarity, in case of a directed graph, the neighborhood consists of all nodes with short paths *to* the destination node. Here is our algorithm in a nutshell: for each node, we will start with the direct neighbors *to* it in its neighborhood, and then compute the optimistic and pessimistic bounds on hitting times of the nodes within a neighborhood to that node. We will then expand the neighborhood and re-iterate. The bounds will be tighter and tighter as we keep expanding the neighborhoods.

Define the set AP as a set of pairs of nodes such that if i is in the neighborhood of j, then $(i,j) \in AP$. Note that $(i,j) \in AP$ does not imply that $(j,i) \in AP$. Each neighborhood of node i has a boundary, δ_i, such that a boundary node on the neighborhood of j has at least one direct neighbor (to it in the directed case) outside the neighborhood. Intuitively a node is a boundary node if there is at least one path (from some node outside the neighborhood) that enters the neighborhood through that node.

Let us first assume that the neighborhood is given. It's clear from the expression for truncated hitting time that h_{ij} can be computed from the hitting time of i's neighbors to j. In Figure 3.1 the random walk from i can hit either of it's neighbors G or B. Since G is inside the neighborhood of j, we already have an estimate of h_{Gj}. However B is outside the neighborhood of j, and hence we find the optimistic and pessimistic values of h_{Bj}, resulting into lower bounds (ho_{ij}) and upper bounds (hp_{ij}) of h_{ij}. The optimistic bound is obtained by allowing the random walk to jump to the boundary node with closest optimistic hitting time to j, and the pessimistic bound arises from the fact that the walk might never come back to the neighborhood after leaving it, i.e. takes T time.

Now we revisit the data-structures in terms of their notations. We denote the set of active pairs as AP. $AP(*,i)$ is the set of nodes k, s.t. $(k,i) \in AP$. Hence the neigh-

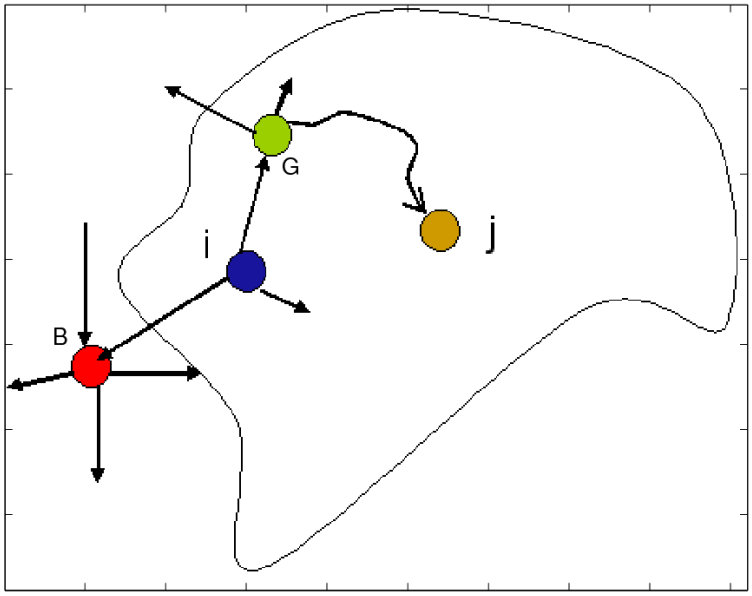


Figure 3.1: Neighborhood of node j. A directed graph is drawn for clarity.

borhood of i can also be denoted by $AP(*, i)$. Also $\delta(i)$ is the set of the boundary nodes of the neighborhood of i, i.e. $AP(*, i)$. Let $nbs(i)$ denote the set of all neighbors to and from i. Let $O(i)$ denote the direct neighbors from i. Also denote by $Gnbs(i, j)$ the set-$AP(*, j) \cap O(i)$. The upper and lower bounds are:

$$hp_{ij}^T = 1 + \sum_{k \in Gnbs(i,j)} p_{ik} hp_{kj}^{T-1} + (1 - \sum_{k \in Gnbs(i,j)} p_{ik})(T-1) \tag{3.10}$$

$$ho_{ij}^T = 1 + \sum_{k \in Gnbs(i,j)} p_{ik} ho_{kj}^{T-1} + (1 - \sum_{k \in Gnbs(i,j)} p_{ik})(1 + \min_{p \in \delta(j)} ho_{pj}^{T-2}) \tag{3.11}$$

We can also use a tighter bound than the above by using one step lookahead from the nodes on the boundary, which is omitted for space purposes. The ho and hp values of all nodes in $AP(*, j)$ to j can be computed using the Gauss-Seidel technique, as in Algorithm 1.

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

Algorithm 1 compute-H(dst,AP,G,T)

$ho, ho_{last}, hp, hp_{last} \leftarrow ones(1:N)$ (Note that $\forall i,j,\ \ h^1(i,j)=1$.)
$minvals_{0:T} \leftarrow 0$; $minvals_1 = 1$
for $t = 2$ to T **do**
  $min \leftarrow \infty$
  for $src \in AP(*, dst)$ **do**
    $s_1, s_2 \leftarrow 0$, $prob \leftarrow 0$
    for $i \in O(src)$ **do**
      if $i \ \in AP(*, dst)$ **then**
        $s_1 \leftarrow s_1 + p_{src,i} ho_{last}(i)$
        $s_2 \leftarrow s_2 + p_{src,i} hp_{last}(i)$
        $prob \leftarrow prob \ + \ p_{src,i}$
      end if
    end for
    $ho(src) \leftarrow 1 + s_1 + (1 - prob)(1 + minvals_{t-2})$
    $hp(src) \leftarrow 1 + s_2 + (1 - prob)(t - 1)$
    if $ho(src) \leq min$ & $src \in \delta(dst)$ **then**
      $min \leftarrow ho(src)$
    end if
  end for
  $minvals_t \leftarrow min$
  $ho_{last} \leftarrow ho$
  $hp_{last} \leftarrow hp$
end for

$$ho_{ij} = \begin{cases} ho_{ij} & \text{if } i \in AP(*, j) \\ 1 + \min_{p \in \delta(j)} ho_{pj}^{T-1} & \text{else} \end{cases} \tag{3.12}$$

$$hp_{ij} = \begin{cases} hp_{ij} & \text{if } i \in AP(*, j) \\ T & \text{else} \end{cases} \tag{3.13}$$

Note that we are explicitly giving the expressions for all pairs of nodes in the graph for clarity. However the bounds need only $O(|AP|)$ space. Using the above we also get optimistic and pessimistic bounds on commute times, namely co and cp.

$$co_{ij} = ho_{ij} + ho_{ji}; \;\; cp_{ij} = hp_{ij} + hp_{ji} \tag{3.14}$$

3.3.1 Expanding Neighborhood

Since time and space complexity of GRANCH is $O(|AP|)$, we have to be clever in having the right set of pairs in the AP-set, in order to avoid wasting computation on nodes that are very far apart. We present a *neighborhood expansion scheme* which will guarantee that we do not miss any potential nearest neighbors. We also note that adding nodes to the neighborhood of i can only tighten the bounds.

Lemma 3.3.1. *The optimistic bounds, i.e. ho values are valid lower bounds on truncated hitting times. The pessimistic bounds are valid upper bounds on truncated hitting times.*

Proof: The hitting time for a node outside the AP-set must be at least the minimum hitting time to the boundary of the AP-set, plus the minimum hitting time from the boundary to the destination. For a destination node j, $ho_{m,j}^{T-1} \geq 1 + \min_{p \in \delta(j)} ho_{pj}^{T-2}$, $\forall m \notin AP(*, j)$. Hence ho_{ij}^{T} in eq. (3.11) is indeed a valid lower bound on h_{ij}^{T}.

The pessimistic bound in eq. (3.10) is valid, since $h_{ij}^{t} \leq t, \;\; \forall \, i, j$, by construction of h^t.

Theorem 3.3.2. *The optimistic bounds on H, i.e. ho values can never **decrease**, and the pessimistic bounds, i.e. hp values can never **increase**, as a result of adding nodes to the AP set.*

Proof: The proof involves an inductive argument. We provide this in more detail in appendix A.A.

Definition: Define the lower bound of the neighborhood of node i as

$$lb(i) = 1 + \min_{p \in \delta(i)} ho_{pi}^{T-1} \tag{3.15}$$

Lemma 3.3.3. *The lower bound on hitting time of nodes outside the AP-set of node i, i.e. $lb(i)$, can never **decrease** as a result of adding nodes to the AP set.*

Proof: This is a direct application of Theorem 3.3.2.

Neighborhood expansion scheme: We start with all direct neighbors to the destination node i. Note that all nodes outside the boundary will take at least $1 + \min_{p \in \delta(i)} ho_{pi}^{T-1}$ time to reach i. Let us denote this by $lb(i)$. Also remember that we are doing a range search of all neighbors within a range $T' < T$. We want to increase this lower bound, so that we can guarantee that we shall never leave out a potential nearest neighbor. One heuristic to increase the lower bound is to add in the incoming and outgoing neighbors of q, where $q = \text{argmin}_{p \in \delta(i)} ho_{pi}^{T-1}$.

Algorithm 2 Expand-AP(G, T, T')

1: Initialize AP with all pairs $i, j \in E_G$. The AP-set can be initialized to all pairs within p hops too. We use $p = 1$ for all of of our experiments.
2: **for** $i = 1$ to N **do**
3: **while** $lb(i) \leq T'$ or $|AP(*, i)| \leq N$ **do**
4: compute-H(i,AP,G,T)
5: $lb(i) \leftarrow 1 + \min_{p \in \delta(i)} ho_{pi}^{T-1}$
6: $q \leftarrow \text{argmin}_{p \in \delta(i)} ho_{pi}^{T-1}$
7: $AP \leftarrow AP \cup \{(a, i) : a \in nbs(q)\}$
8: **end while**
9: **end for**

Lemma 3.3.4. *If $i \notin AP(*, j)$ after running Algorithm 2, then $h_{ij} \geq T'$.*

After running algorithm 2, $ho_{i,j} \geq lb(j) \geq T'$. Also by construction we have $h_{i,j} \geq ho_{ij}$.

3.3.2 Approximate Nearest Neighbors

We shall first look at how to obtain the k ϵ-approximate nearest neighbors for each node in hitting time. Later we shall demonstrate how to extend that to approximate nearest neighbors in commute times.

Note that we only have upper and lower bounds on hitting times from a node to other nodes. The key idea is that any value x will be smaller than y if the upper bound of x is smaller that the lower bound of y.

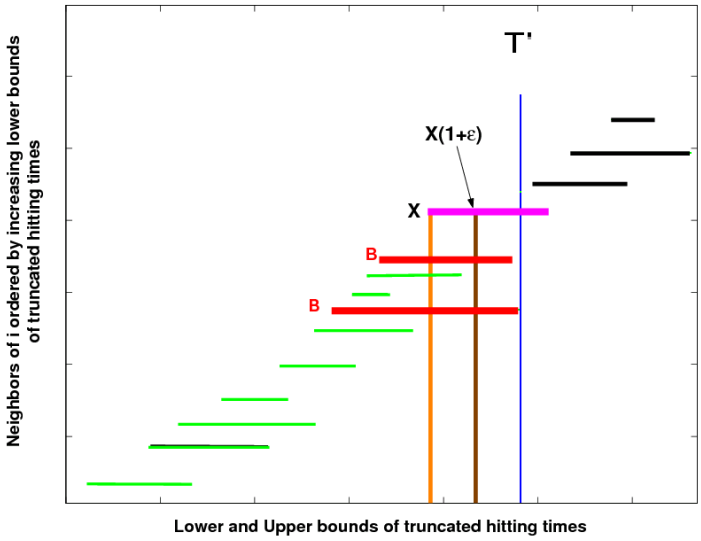


Figure 3.2: Upper and lower bounds of $h^T(i, *)$

Truncated Hitting Time

We illustrate with a toy example. Let us assume that we want to find the k nearest neighbors of node i within T'. In Figure 3.2 we plot the lower and upper bounds of $h_{i,j}$, in the order of increasing lower bounds. It will suffice to find all nodes j, s.t. hp_{ij} is smaller than the h value of the $(k+1)^{th}$ true nearest neighbor. Note that the $(k+1)^{th}$ nearest neighbor's hitting time is greater than the $(k+1)^{th}$ largest ho value. Let us denote by X the $(k+1)^{th}$ largest ho value from i. We also allow a small error of ϵ. Now, all entities with upper bound less than $X(1+\epsilon)$ are guaranteed to be ϵ-approximate.

We also want the neighbors to be within T' hitting time of the source i. Hence we find the largest $k' \leq k+1$, s.t. $ho_{i_{k'},i} \leq T'$ and $hp_{i_{k'},i} \geq T'$, where $i_{k'}$ is the node with the k'^{th} largest ho value from i. Now the upper bound X becomes $X = ho_{i,i_{k'}}$. We return all nodes x, s.t. $hp_{i,x} < X(1+\epsilon)$. Therefore in Figure 3.2 we have $k = 15$, $k' = 10$. This means that at this point the bounds are not tight enough to determine the exact ranks of the $k - k'$ neighbors. All the nodes with lower bounds beyond T' are guaranteed not to be within T'. The nodes labeled B are not guaranteed to be ϵ approximate and are not returned.

Truncated Commute Time

Now we look at the neighbors in commute times. The new distance for range-search becomes $2T'$. For any node i, we now look at all nodes in the set $AP(i, *) \cup AP(*, i)$.

$AP(i,*)$ denotes the set of all nodes that have i in their neighborhoods. After running Algorithm 2, if there exists some node j, s.t. $j \notin AP(*,i)$ and $i \notin AP(*,j)$, then by Lemma 3.3.4, $ho(i,j), ho(j,i) \geq T'$ and hence $co(i,j) \geq 2T'$, so j cannot be within a commute distance of $2T'$ of i.

Ranking using Upper and Lower Bounds

We compute the nearest-neighbors for node $i, i \in \{1:n\}$ using the AP-set resulting from Algorithm 2. Now we find the nearest neighbors of i in commute times using eq. (3.14). Note that we only look at the nodes in set $S_i = AP(i,*) \cup AP(*,i)$. Sort all the nodes in increasing order of optimistic bounds i.e. co values. Let $\{i_1, i_2, ...\}$ denote the indexes of the sorted nodes. Let k' the largest integer less than k, s.t. $co(i, i_{k'}) \geq 2 * T'$. If such a k' exists, define X as $2 * T'$. Otherwise clearly all k possible nearest neighbors might be within the range. Hence define X as $co(i, i_k)$. Return any node $j \in S_i$ s.t. $cp(i,j) \leq X(1+\epsilon)$. Also let B be the set of nodes b in S_i s.t. $co(i,b) \leq X$, but $cp(i,b) \geq X(1+\epsilon)$. Clearly these nodes cannot be guaranteed to be among the k nearest neighbors. Now we expand the AP-sets of $\{i\} \cup B$ to tighten the bounds more.

Because of the construction of the algorithm, the newly introduced nodes can never affect k', and the other nearest neighbors. By Lemma 3.3.4, we will never introduce a new possible nearest neighbor of a node $b \in \{i\} \cup B$, since all nodes outside S_b are guaranteed to have commute-distance worse than $2T'$. Also an expansion of the AP-set can only increase the lower bounds, and decrease the upper bounds on the estimates on hitting times, which will never decrease the number of neighbors returned so far. Hence we will only improve the performance of the algorithm by expanding our AP-set beyond algorithm 2.

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

3.4 Hybrid Algorithm

We present an algorithm to compute approximate nearest neighbors in commute times, without iterating over the entire graph. We combine random sampling with the branch and bound pruning scheme mentioned before, in order to obtain upper and lower bounds on *commute times* from a node. This enables us to compute the k nearest neighbors from a query node *on the fly*.

For any query node we compute hitting time from it using sampling. We maintain a bounded neighborhood for the query node at a given time-step. We compute estimated bounds on the commute time from the nodes within the neighborhood to the query. Com-

mute time of nodes outside the neighborhood to the query are characterized by a single upper and lower bound. We expand this neighborhood until this lower bound exceeds $2T'$, which guarantees that with high probability we are excluding nodes which are more than $2T'$ commute distance away. These bounds are then used for ranking the nodes inside the neighborhood.

We will first describe a simple sampling scheme to obtain ϵ-approximate truncated hitting times **from** a query node with high probability.

3.4.1 Sampling Scheme

We propose a sampling scheme to estimate the truncated hitting time from a given query node i in a graph. We run M independent T-length random walks from i. Let us say out of these M runs m random walks hit j for the first time at $\{t_{k_1}, ...t_{k_m}\}$ time-steps. From these we can estimate the following

1. The probability of hitting any node j *for the first time* from the given source node within T steps can be estimated by $\frac{m}{M}$.
2. The first hitting time can be estimated by
$$\hat{h}^T(i,j) = \frac{\sum_r t_{k_r}}{M} + (1 - \frac{m}{M})T$$

We provide bounds (details in appendix A.A) similar to Fogaras et al. [2004]

1. The number of samples M required in order to give an ϵ- correct answer with probability $1 - \delta$.
2. The number of samples M required in order to get the top k neighbors correct.

Theorem 3.4.1. *For a given pair of nodes i and u, in order to obtain $P(|\hat{h}^T(i,u) - h^T(i,u)| \geq \epsilon T) \leq \delta$, number of samples M should be at least $\frac{1}{2\epsilon^2}\log(\frac{2}{\delta})$.*

Corollary 3.4.2. *For a given node i, in order to obtain $P(\exists u \in \{1,\ldots,n\}, |\hat{h}^T(i,u) - h^T(i,u)| \geq \epsilon T) \leq \delta$, number of samples M should be at least $\frac{1}{2\epsilon^2}\log(\frac{2n}{\delta})$.*

Theorem 3.4.3. *Let $v_j, j = 1 : k$ be the top k neighbors of i in exact T-truncated hitting time. Let $\alpha = h^T(i,v_{k+1}) - h^T(i,v_k)$. Then number of samples M should be at least $\frac{2T^2}{\alpha^2}\log(nk/\delta)$ in order to have $Pr(\exists j \leq k, q > k, \hat{h}^T(i,v_j) > \hat{h}^T(i,v_q)) \leq \delta$.*

The details are provided in appendix A.A. The above theorem says nothing about the order of the top k neighbors, only that if the gap between the hitting times from i to the k^{th} and $k+1^{th}$ nearest neighbors is large, then it is easy to retrieve the top k nearest neighbors. We could change the statement slightly to obtain a sample complexity bound to guarantee the exact order of the top k neighbors with high probability. The main difference will be that it will depend on $\min_{j \leq k} h^T(i,v_{j+1}) - h^T(i,v_j)$.

3.4.2 Lower and Upper Bounds on Truncated Commute Times

In this section we will denote $AP(*, j)$ in section 3.3 by $NBS(j)$. We have the expressions for the lower and upper bounds for the hitting times of the nodes in $NBS(j)$ to j (ho and hp values) from equations (3.11) and (3.10). The hitting time from j to nodes within $NBS(j)$ can be estimated using the sampling scheme described in section 3.4.1. Combining the two leads to the following.

Theorem 3.4.4. *The truncated commute time between nodes $i \in NBS(j)$ and j will be lower and upper bounded by co_{ij}^T and cp_{ij}^T with probability $1 - \delta$ if the number of samples for estimating $\hat{h}_{ij}^T$ exceeds the lower bound in theorem 3.4.1, where*

$$co_{ij}^T = \hat{h}_{ij}^T + ho_{ij}^T \tag{3.16}$$

$$cp_{ij}^T = \hat{h}_{ij}^T + hp_{ij}^T \tag{3.17}$$

In order to prune away nodes which are not potential nearest neighbors we also need to obtain a lower bound on the commute time between j and any node outside $NBS(j)$. The incoming lower bound is given by equation 3.15. Now note that for the outgoing lower bound we need the minimum of $h_{jk}^T, \forall k \notin NBS(j)$.

Lemma 3.4.5. *The number of samples M should be at least $\frac{1}{2\epsilon^2}\log(\frac{2}{\delta})$ in order to obtain $Pr(|\min_{k \notin NBS(j)} \hat{h}_{jk}^T - \min_{k \notin NBS(j)} h_{jk}^T| \geq \epsilon T) \leq 2\delta$.*

Thus an estimate of the outgoing lower bound can be computed from the hitting times obtained from sampling. Combining the two we obtain an estimate on the lower bound on $2T$-truncated commute time $\widehat{lb\text{-}ct}(j)$ from j to any node outside $NBS(j)$.

$$\widehat{lb\text{-}ct}(j) = 1 + \min_{p \in \delta(j)} ho_{pj}^{T-1} + \min_{k \notin NBS(j)} \hat{h}_{jk}^T \tag{3.18}$$

For our implementation, we always used estimated, not the exact bounds. This introduces an additive error in our results (proof excluded for lack of space).

3.4.3 Expanding Neighborhood

Now we need to find a heuristic to expand the neighborhood such that both the outgoing and incoming components of the lower bound increase quickly so that the threshold of $2T'$ is reached soon.

For the incoming lower bound we just find the x closest nodes on the boundary which have small optimistic hitting time to the query. We add the neighbors of these nodes to the neighborhood of j. For the outgoing lower bound, we compute the nodes outside the boundary which a random walk is most probable to hit. We do this by maintaining a set of

paths from j which stop at the boundary. These paths are augmented one step at a time. This enables one step look-ahead in order to figure out which nodes outside the boundary are the most probable nodes to be hit. We add y of these nodes to the current boundary.

Ranking

The ranking scheme is similar to GRANCH and is rather intuitive. So far we only have lower and upper bounds for commute times from node j to the nodes in $NBS(j)$. Let us denote this set as S. The commute time from j to any node outside S is guaranteed to be bigger than $2T'$. The true k^{th} nearest neighbor will have commute time larger than the k^{th} smallest lower bound i.e. co value. Let us denote the k^{th} smallest co value by X. Now consider the nodes which have upper bounds (cp values) smaller than X. These are guaranteed to have commute time smaller than the true k^{th} nearest neighbor. Adding a multiplicative slack of α to X allows one to return the α-approximate k nearest neighbors which have commute time within $2T'$. Note that the fact that no node outside set S has hitting time smaller than $2T'$ is crucial for ranking, since that guarantees the *true* k^{th} *nearest neighbor* within $2T'$ commute distance to be within S. Since all our bounds are probabilistic, i.e. are true with high probability (because of the sampling), we return α-approximate k nearest neighbors with high probability. Also the use of estimated bounds ($\widehat{co}$,$\widehat{cp}$) will introduce an additive error of $2\epsilon T$ (ignoring a small factor of $\epsilon\alpha T$).

3.4.4 The Algorithm at a Glance

In this section we describe how to use the results in the last subsections to compute nearest neighbors in truncated commute time from a node. *Given T, α, k our goal is to return the top k α-approximate nearest neighbors (within $2\epsilon T$ additive error) with high probability (w.h.p).*

First we compute the outgoing hitting times from a node using sampling. We initialize the neighborhood with the direct neighbors of the query node (We have set up our graph so that there are links in both directions of an edge, only the weights are different). At any stage of the algorithm we maintain a bounded neighborhood for the query node. For each node inside the neighborhood the hitting times *to* the query can be bounded using dynamic programming. Combining these with the sampled hitting times gives us the estimated $\widehat{co}$, and $\widehat{cp}$ values. We also keep track of the lower bound $\widehat{lb\text{-}ct}$ of the commute time from any node outside the neighborhood to the query node. At each step we expand the neighborhood using the heuristic in section 3.4.3. Similar to GRANCH we recompute the bounds again, and keep expanding until $\widehat{lb\text{-}ct}$ exceeds $2T'$. With high probability, this guarantees that all nodes outside the neighborhood have commute time larger than $2T' - \epsilon T$.

Then we use the ranking as in section 3.4.3 to obtain k α-approximate nearest neighbors (with an additive slack of $2\epsilon T$) in commute time. We start with a small value of T' and increase it until all k neighbors can be returned. As in section 3.3 it is easy to observe that the lower bound can only increase, and hence at some point it will exceed $2T'$ and the algorithm will stop. The question is how many nodes can be within $2T'$ commute distance from the query node. In section 3.2.3 we prove that this quantity is not too large for most query nodes.

For directed graphs, one can construct examples such that the bounds on commute times can be loose leading to exploration of a lot of nodes. Also, in section 3.2.3, we show that for directed graphs we can only give a loose bound on number of nodes within $2T'$ commute time, using counting arguments. However even for the directed entity-relation graphs which we use in section 3.5.2, our algorithm is very fast in practice. In all our experiments with directed graphs, all edges had links in both directions, but with different weights.

As in section 3.2.3, for undirected graphs, the number of nodes within $2T'$ commute distance of node q, can be shown to be not more than $d(q)/d_{min} \times T^2/(T - T')$, where d_{min} is the minimum degree of the nodes. Note that, even then we cannot guarantee that the neighborhood explored has only the nodes within $2T'$ commute time.

3.5 Empirical Results

We will evaluate our algorithm on link prediction tasks on undirected co-authorship graphs, and directed Entity Relation (ER) datasets extracted from the Citeseer corpus[1].

3.5.1 Co-authorship Networks

Here is a brief description of the experimental methodology. We remove a random subset of the links from the original graph. Now we compute the $2T$-truncated-commute times on the training links. We only take nodes which have at least one link held out. For each such node we take the proximity measures from the algorithms of all nodes within 5 hops of the source in the original graph(not all other nodes) and count how many of the held-out neighbors appear in top 10 neighbors. We present the average score.

We present the performances of the following algorithms in Table 3.1.

1. Sampled hitting times (SH)
2. Hybrid commute times (HC)

[1]We thank Soumen Chakrabarti and Lee Giles for sharing the dataset.

3. Number of hops (truncated at 5) (B1)
4. Number of common neighbors (B2)
5. Jaccard score (B3)
6. Adamic/Adar score (B4)
7. Random (B5)

We present the average time in seconds per query in table 3.2. T in all tables translates to a $2T$-truncated commute time. It was shown by Liben-Nowell and Kleinberg [2003]

Table 3.1: Link prediction accuracy on 30% links held out from graphs of size n, edges e, for sampled hitting times (SH) and hybrid commute times (HC), with $T \in \{10, 6\}$. As described before, **B1** denotes the number of hops between node-pairs within 5 hops, **B2** denotes the number of common neighbors, **B3** denotes the Jaccard score, **B4** denotes the Adamic/Adar score, and **B5** denotes the random predictor.

n	e	SH, HC, T=10	SH, HC, T=6	B1	B2	B3	B4	B5
19,477	54,407	81,85	83,86	47	71	80	77	.07
52,033	152,518	76,80	80,82	41	66	76	73	.04
103,500	279,435	77,81	77,80	38.5	66.5	73.4	71	.02

Table 3.2: Time in seconds on link prediction tasks on graphs of size n, edges e , for sampled hitting times (SH) and hybrid commute times (HC), with $T \in \{10, 6\}$

n	e	SH, HC, T=10	SH, HC, T=6
19,477	54,407	.01,.05	.01,.03
52,033	152,518	.01,.11	.01,.05
103,500	279,435	.01,.19	.01,.09

that simple measures like number of common neighbors, Adamic/Adar or Jaccard do surprisingly well on link prediction tasks. The Adamic/Adar score between two nodes i and j (eq. 2.14, chapter 2) down-weights the high degree common neighbors. The Jaccard score (eq. 2.15, chapter 2) measures the probability that a node is a common neighbor of i and j, conditioned on the fact that it is a neighbor of i or j. Note that common neighbors, Adamic/Adar and Jaccard would not be applicable if we try to predict links between two

layers of nodes in a bipartite graph. However these do perform quite well in our experiments, which shows that these co-authorship networks have a lot of transitive links, i.e. a node is very likely to have a link with the neighbor of a neighbor. In all these cases commute times perform the best, hitting times are the second best, and Jaccard is in the third place. Adamic/Adar performs almost as well as Jaccard. Both Jaccard and Adamic/Adar outperform number of common neighbors by a large margin. Number of hops performs quite poorly, although it is significantly better than the random predictor.

3.5.2 Entity Relation Graphs

We extracted Entity Relation (ER) datasets from the Citeseer corpus, as in Chakrabarti [2007]. This is a graph of authors, papers and title-words (shown in figure 3.3).

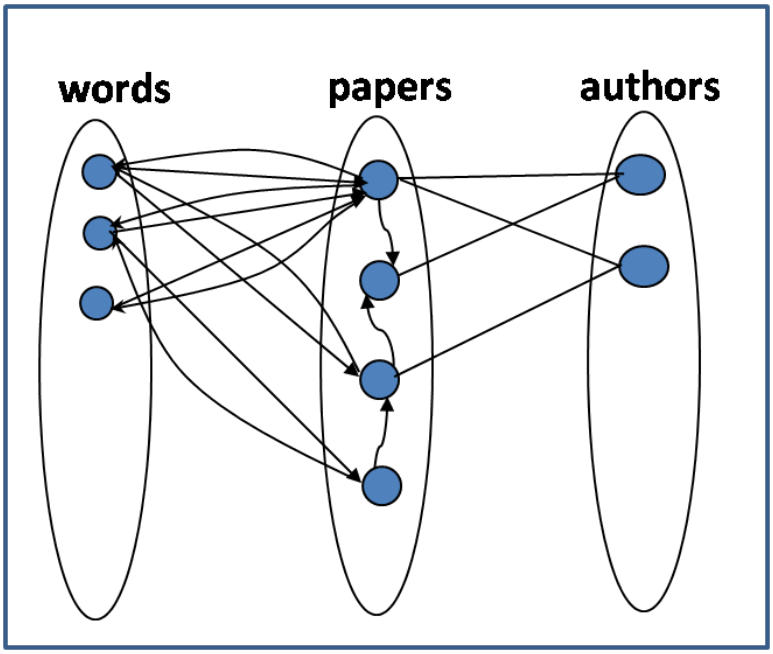


Figure 3.3: The Citeseer entity-relation graph.

Dataset and Link Structure

The link structure is as follows:

1. Between a paper and a word appearing in its title.
2. From a paper to the paper it cites, and one with one-tenth the strength the other way.

3. Between a paper and each of its authors.

As observed by Chakrabarti [2007], the weights on these links are of crucial importance. Unlike Balmin et al. [2004] and Chakrabarti [2007], we also put links *from* the paper layer *to* the word layer. This allows flow of information from one paper to another via the common words they use. The links between an author and a paper are undirected. The links within the paper layer are directed. We use the convention in Chakrabarti [2007] to put a directed edge from the cited paper to the citing paper with one-tenth the strength.

For any paper we assign a total weight of W to the words in its title, a total weight of P to the papers it cites and A to the authors on it. We use an inverse frequency scheme for the paper-to-word link weight, i.e. the weight on link from paper p to word w is $W \times 1/f_w/(\sum_{p \to u} 1/f_u)$, where f_w is the number of times word w has appeared in the dataset. We set $W = 1, A = 10, P = 10$ so that the word layer to paper layer connection is almost directed. We add a self loop to the leaf nodes, with a weight of 10 times the weight of its single edge, so that the hitting time from these leaf nodes are not very small.

We use two subgraphs of Citeseer. The small one has around $75,000$ nodes and $260,000$ edges: $16,445$ words, $28,719$ papers and $29,713$ authors. The big one has around $600,000$ nodes with 3 million edges : $81,664$ words, $372,495$ papers and $173,971$ authors.

Preprocessing

We remove the stopwords and all words which appear in more than 1000 papers from both the datasets. The number of such words was around 30 in the smaller dataset and 360 in the larger one. Excluding these words *did not change* the predictive performance of any of the algorithms.

3.5.3 Experiments

The tasks we consider are as follows,

1. Paper prediction for words (Word task): We pick a paper X at random, remove the links between it and its title words. Given a query of exactly those words we rank the papers in the training graph. For different values of y the algorithm has a score of 1 if X appears in the closest y papers. *For any search engine, it is most desirable that the paper appears in the top k results,* $k \leq 10$.
2. Paper prediction for authors (Author task): Exactly the above, only the link between the paper and its authors are removed.

The hybrid algorithm is compared with: 1) exact truncated hitting time from the query; 2) sampled truncated hitting time from the query; 3) exact truncated commute time from

the query; 4) exact truncated hitting time to the query; 5) personalized pagerank vector (PPV) and 6) random predictor. Note that we can compute a high accuracy estimate of the *exact* hitting time *from* a query node by using a huge number of samples. We can also compute the exact hitting time *to* a node by using dynamic programming by iterating over all nodes. Both of these will be slower than the sampling or the hybrid algorithm as in Table 3.3.

Distance from a set of nodes Hitting and commute times are classic measures of proximity from a single node. We extend these definitions in a very simple fashion in order to find near-neighbors of a set of nodes. The necessity of this is clear, since a query often consists of more than one word. We define the hitting time from a set of nodes as a weighted average of the hitting times from the single nodes. For hitting time to a set, we can change the stopping condition of a random walk to "stop when it hits any of the nodes in the set". We achieve this via a simple scheme: for any query q, we *merge* the nodes in the query in a new mega node Q as follows. For any node $v \notin Q$ $P(Q, v) = \sum_{q \in Q} w(q) P(q, v)$, where $P(q, v)$ is the probability of transitioning to node v from node q in the original graph. $\sum_{q \in Q} w(q) = 1$. We use a uniform weighing function, i.e. $w(q) = 1/|Q|$. The hitting/commute time is computed on this modified graph from Q. These modifications to the graph are local and can be done at query time, and then we undo the changes for the next query.

Our **average query size** is the average number of words (authors) per paper for the word (author) task. These numbers are around 3 (2) for the big subgraph, and 5 (2) for the small subgraph. Figure 3.4 has the performance of all the algorithms for the author task

Table 3.3: Run-time in seconds for Exact hitting time from query, Sampled hitting time from query, Exact commute time, Hybrid commute time, Exact hitting time to query, PPV

# nodes	# edges	Task	Exact Ht-from	Sampled Ht-from	Exact Ct	Hybrid Ct	Exact Ht-to	PPV
74,877	259,320	Author	1.8	**.02**	9.2	**.28**	6.7	18
		Word	3.1	**0.3**	10.4	**1.2**	6.56	50
628, 130	2, 865, 660	Author	6.9	**.07**	79.07	**1.8**	67.2	337.5
		Word	12.3	**0.7**	88.0	**4.3**	70	486

on the (A) smaller, (B) larger dataset and the word task on the (C) smaller and (D) larger dataset, and Table 3.3 has the average runtime. As mentioned before for any paper in the testset, we remove all the edges between it and the words (authors) and then use the different algorithms to rank all papers within 3 hops of the words (5 for authors, since authors have smaller degree and we want to have a large enough candidate set) in the new

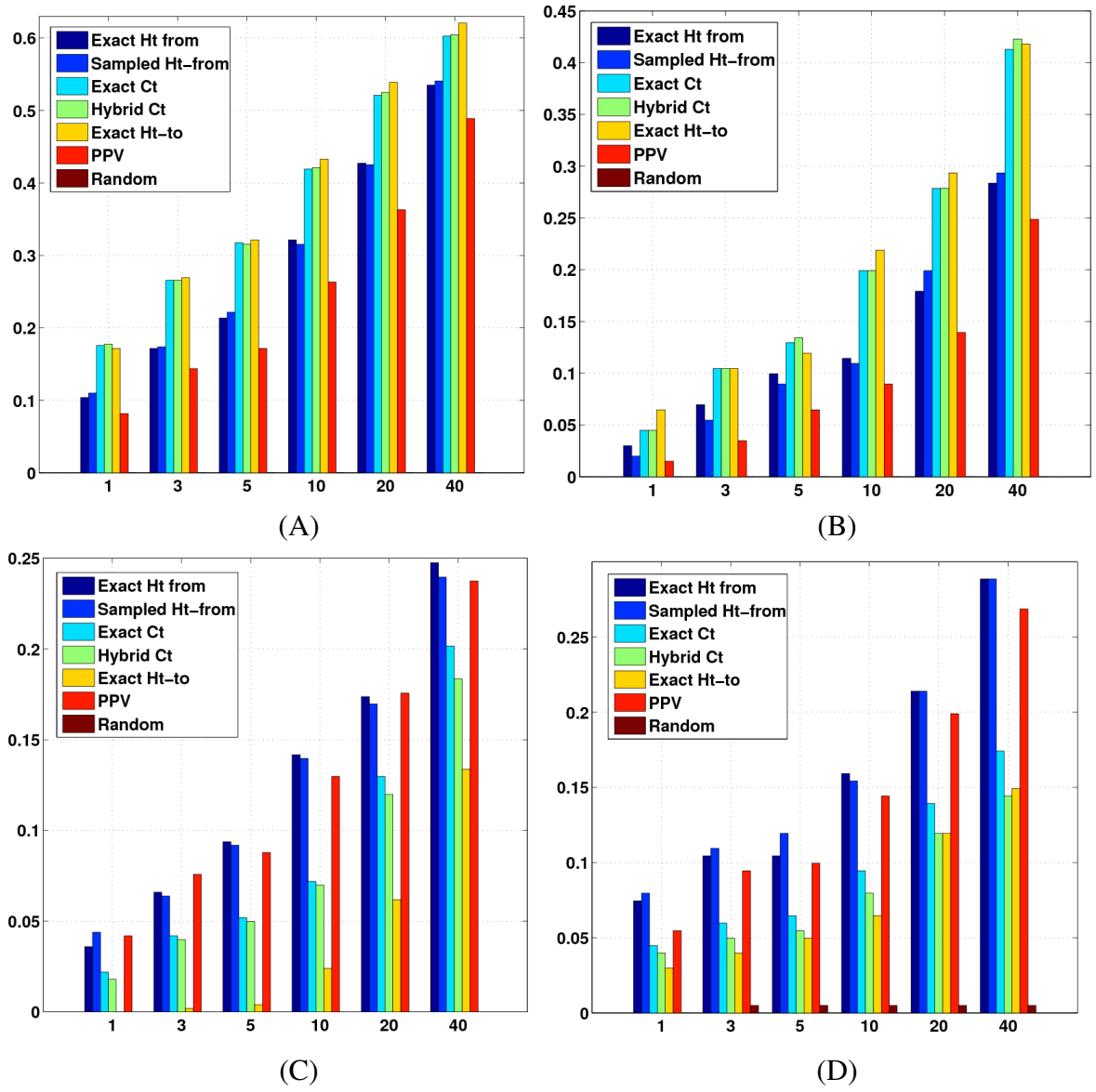


Figure 3.4: Author task for (A) Small and (B) Large datasets. Word task for (C) Small and (D) Large datasets. x-axis denotes the number of neighbors and y-axis denotes accuracy.

graph and the removed paper. For any algorithm the percentage of papers in the testset which get ranked within the top k neighbors of the query nodes is plotted on the y axis vs. k on the x axis. We plot the performances for six values of k: $1, 3, 5, 10, 20$ and 40.

The results are extremely interesting. Before going into much detail let us examine the performance of the exact algorithms. Note that for the author task the *exact hitting time* ***to a node*** and the *exact commute time from a node* consistently beats the *exact hitting time*

from a node, and PPV. However for the word task the outcome of our experiments are **the opposite**. This can be explained in terms of the inherent directionality of a task. The distance *from* the word-layer to the paper-layer gives more information than the distance from the paper layer to the word layer, whereas both directions are important for the author task, which is why commute times, and hitting time to a node outperform all other tasks.

We only compare the predictive power of PPV with our measures, not the *runtime*. Hence we use the exact version of it. We used $c = 0.1$, so that the average path-length for PPV is around 10, since we use $T = 10$ for all our algorithms (however, much longer paths can be used for the exact version of PPV). PPV and hitting time from a node essentially rely on the probability of reaching the destination from the source. Even though hitting time uses information only from a truncated path, in all of our experiments it performs better than PPV, save one, where it behaves comparably.

Word task: The sampling based hitting time beats PPV consistently by a small margin on the bigger dataset, whereas it performs comparably on the smaller one. Hitting times and PPV beat the hitting time to nodes for the word task. In fact for $k = 1, 3, 5$, for the smaller dataset the hitting time *to* a query node isn't much of an improvement over the random predictor (which is zero). This emphasizes our claim that the hitting time to the word layer does not provide significant information for the word task. As a result the performance of the exact and hybrid commute times deteriorates.

Author task: The hitting time *to the query* and the exact commute time *from a query* have the best performance by a large margin. The hybrid algorithm has almost similar performance. Hitting time from the query is beaten by these. PPV does worse than all the algorithms except of course the random predictor.

Number of samples: For the small graph, we use $100,000$ samples for computing the high accuracy approximation of hitting time from a node; 5000 samples for the word task and 1000 samples for the author task. We use 1.5 times each for the larger graph. We will like to point out that our derived sample complexity bounds are interesting for their asymptotic behavior. In practice we expect much fewer samples to achieve low probability of error. In Figure 3.4, sometimes the exact hitting (commute) time performs worse than the sampled hitting time (hybrid algorithm). This might happen by chance, with a small probability.

3.6 Summary and Discussion

In this chapter, we investigate two aspects of ranking problems on large graphs. First, we present a deterministic neighborhood expansion to compute all pairs of interesting nearest neighbors in a graph. Then we augment it with sampling techniques to compute approximately correct rankings with high probability under random walk based proximity measures *at query time*. Second, we prove some surprising locality properties of these proximity measures by examining the short term behavior of random walks. The proposed algorithm can answer queries *on the fly without caching any information about the entire graph*. We present promising empirical results on graphs derived from the Citeseer domain.

Now, we will present some weaknesses of the branch and bound algorithm, and present some preliminary ideas about overcoming them. As we will show in the later chapters, this algorithm is extremely generalizable and can be used to compute other random walk based measures, including discounted hitting times, and personalized pagerank to a query node. However, one weakness of it is that every time new nodes are added to the neighborhood, the values have to be recomputed. This can be problematic when the neighborhood becomes bigger. One solution will be to estimate or approximate the changes, and actually recompute them only a few times. This will make this algorithm more suitable for discounted hitting times or personalized pagerank, which have no upper bound on the number of steps.

In section 3.2.3 we bound the size of the neighborhood containing nodes within a small truncated hitting or commute time from a query node. It should be noted that, this is not a bound on the neighborhoods returned by GRANCH or the other algorithms. This is because, our algorithms are guaranteed to return a set of nodes which is a *superset* of the ideal neighborhood. The empirical results do verify that this set is small in size. However, it would indeed be interesting to bound the size of the sets returned by the algorithms.

Another interesting observation from our analysis from section 3.2.3 is that the neighborhood size of a node depends on its degree. This is important, because we can use this to obtain a cross between caching and query-time computation. In particular, one might identify the nodes with a large neighborhood and compute them and cache them online. In undirected graphs these are the nodes with high degree. Although our analysis holds only for undirected graphs, The intuition holds for directed graphs as well; i.e. high indegree nodes will have a lot other nodes with small hitting time *to them*. Real world graphs exhibit power law degree distribution, and hence there are only a few high degree nodes. Hence we would not need to catch a lot of information either.

The other issue with GRANCH and the Hybrid algorithm is that they suffer from high degree nodes. This is because, whenever a high degree node is encountered, the neighborhood size becomes very large, slowing down the algorithms considerably. How much

does these high degree nodes contribute to a random walk based measure? Intuitively, a node with high out-degree (degree in undirected graphs), only passes on a very small probability mass to its direct neighbors. Hence they should not have a large contribution to measures like truncated hitting or commute time. In chapter 5 we give a simple solution to deal with high out degree nodes. For ease of analysis, we examine discounted hitting times in undirected graphs. But the intuition is true in both truncated hitting times and directed graphs. This, in addition to caching the neighborhoods of high indegree nodes should lead to considerable improvement in runtime.

There is one detail that we have not discussed so far. How do we generate the graph underlying the Citeseer data? Clearly, the weights on different links are quite crucial for obtaining good link prediction accuracy. Most graph based ranking systems like OBJECTRANK use manually tuned weights on the edges of the graph. Recently there has been some work on learning these weights automatically (Chakrabarti and Agarwal [2006]). While we used the convention of Chakrabarti [2007] and some common heuristics like the inverse frequency scheme to weight the links in the graph, we can also use the output of existing weight learning algorithms as the input to our algorithm.

However one question remains. Can we improve upon this ranking? Often collected data is noisy leading to ambiguity in the graph structure. Also, users might not know the exact keyword to search for. Under such settings we need to include extra information to obtain better ranking. In the next chapter, we will show how to incorporate user-feedback and quickly rerank the base ranking generated by hitting or commute times or any other proximity measure. Surprisingly, the algorithmic framework from this chapter can be generalized to this interactive setting.

Chapter 4

Fast Dynamic Reranking in Large Graphs

The firmest friendship is based on an identity of likes and dislikes.
–Sallust.

4.1 Introduction

In the last chapter we described a local algorithm for nearest neighbor search in truncated hitting and commute time. These measures extract the information from the graph structure to compute the similarity between nodes in a graph. For example, if the query node is connected to another node via many short paths, then that node will be ranked high under these measures.

However, because of noise in the collected data, the top ranked nodes might not be relevant to the user's query. For example, in a DBLP graph different authors with the same name are often treated as one entity, which completely confuses standard search algorithms. Another issue is that, often the user does not know the right query terms to search for. This is why query completion or query suggestion is gaining importance in search technology.

In order to get around these problems and improve quality of relevance search, we propose a simple solution. We let the end user give feedback in the form of which of the top k search results are relevant, and which are not. Our goal is to quickly produce a reranked list of results by incorporating this feedback. This can be thought of as a graph-based semi-supervised learning problem. The only significant difference between traditional semi-supervised learning problems is that here, the labeled set of nodes, i.e. the user's feedback is quickly changing. This, in addition to the large size of real world

networks, calls for extremely fast and memory efficient algorithms.

One solution would be to compute a proximity measure from the relevant nodes. TrustRank (Gyöngyi et al. [2004]) uses personalized pagerank from the trusted nodes to discriminate between good and spammy nodes in the web. Recent work (Jin et al. [2008]) on ranking refinement uses boosting to learn the new ranking function simultaneously from the base ranking function and the user feedback. The problem with the first approach is that it does not take the negative information into account, and query-time computation of personalized pagerank is still an active area of research. The second approach is quadratic in the number of entities to be ranked and is not appropriate for quick reranking.

Harmonic functions for graph-based semi-supervised learning were introduced by Zhu et al. [2003]. Given a few positive and negative labels the harmonic function value at any node is simply the probability of hitting a positive label before a negative label. To be more specific, this denotes the probability that a random walk started at an unlabeled node will hit a positive label without hitting a negative label first. Variations of the same measure have also been successfully used for web-spam detection (Joshi et al. [2007b]), automated image colorization (Levin et al. [2004]) and image segmentation with user feedback (Grady [2006]). Similar to much spreading activation work, the main intuition is that nodes close in the graph topology will have similar labels.

The standard tool for computing harmonic functions either involves sparse solvers for graph Laplacians or iterative matrix operations. There are algorithms which solve undirected graph Laplacians in near-linear time (Spielman and Teng [2004]). However these results do not apply to directed graphs. They are also not appropriate for dynamic reranking. The main reason behind this is that they are designed for one time computation over the entire graph with static positive and negative labels. We want fast local computation for labels generated by users in real time. Note that, this labeled set is quickly changing over time.

In this chapter we propose a short term variation of harmonic functions. In Entity-Relation graphs, or social networks, we are interested in information flow in shorter range. Hence we compute the probability of hitting a positive node before a negative node in T steps, where T is set to be around 10. This is similar to (Joshi et al. [2007b]), where the harmonic rank is computed with respect to a random walk with a restart probability. We present two different algorithms for computing the above function. One is a simple sampling algorithm, and the other is a dynamic neighborhood expansion technique which provably returns the approximate top k nodes under this measure. Both of these are useful for two different and complementary scenarios. If one had a candidate set of nodes to be ranked based on the feedback, the sampling technique is appropriate to compute function values at the specific nodes. On the other hand, the second algorithm is more appropriate for computing top k nodes in this measure for a given set of labels.

We use quantifiable entity disambiguation tasks in the DBLP corpus to evaluate the

proposed measure and its conditional version (probability of hitting a positive node before a negative node given the random walk hits some label in T steps). We compare our results with two standard diffusion based proximity measures from the positive nodes in the labeled set: a) expected time to hit a node from the positive labels in T steps, and b) personalized pagerank from the positive labels. Our results indicate that the measure which combines information from both the positive and negative feedback performs much better than proximity measures which use only positive relevance information. We also show timing results on state of the art paper-author-citation graph (not containing explicit word nodes) built from DBLP. The branch and bound algorithm takes about 1.5 seconds to compute the top 10 nodes for 10 labeled nodes on average.

The chapter is organized as follows: we briefly describe relation to previous work in section 4.2, and motivate the reranking problem in section 4.3. In section 4.4 we define the discriminative measures (conditional and unconditional probabilities), and illustrate their behavior on toy examples. Section 4.5 contains the proposed algorithms. In section 4.6 we present our experimental results.

4.2 Relation to Previous Work

In this chapter we present a short term variation of harmonic functions on graphs. We show the effectiveness of these functions for semi-supervised classification and also present efficient algorithms to compute them. First we will point out the relationship of our work with previous work. We will remind the reader of the previous applications and algorithms, some of which has been discussed in greater detail in chapter 2. In the related work section, we have sketched on applications like classification tasks (the newsgroup and digits datasets), vision problems including image coloring and image segmentation, and web-spam. Here we will focus on how our work fits in the big picture. Since our objective is to improve upon keyword search in databases by using user feedback, we will describe related work in this area.

One main component of graph-based semi-supervised learning is to build a graph from the data. For example document datasets can be turned into a graph where each document is an entity, and two entities are connected if they share many common words. Similarly, the similarity between two handwritten digits can be computed from their features (see figure 4.1). In a graph representation of an image, two neighboring pixels share a strong connection if they have similar color, intensity or texture. The colorization application involves coloring a grey-scale image, where the user provides a few colored scribbles to some portions (see figure 4.2). Similarly the segmentation example produces high-quality segmentation of the image by incorporating user-generated labels. The common component of the above examples is same: nearby nodes in a graph have similar properties,

and hence should have similar labels. The label can be 1 or 0, if a digit is 6 or 9, if a news-article is about hockey or baseball, or if a segment of an image is red or blue etc.

A harmonic function (eq. 2.16) can be computed using linear solvers for graph Laplacians. There has been much work on near-linear-time solvers (Spielman and Teng [2004]) for undirected graph Laplacians. Recently a linear-work parallel algorithm has been proposed for solving planar Laplacians (Koutis and Miller [2007]). Unfortunately these algorithms do not apply to our directed graph setting. Zhu [2006] discusses a number of different algorithms including loopy belief propagation and preconditioned gradient descent. However in our application, the labeled nodes change from user to user. Hence these approaches will not be suitable for interactive purposes.

Spam detection in the web is an extremely important application. Spam-detection algorithms can be roughly divided into two classes: content-based and graph-based approaches. The content-based approaches (Ntoulas et al. [2006]) 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.

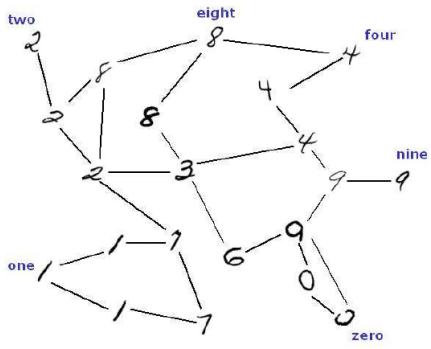


Figure 4.1: A graph built from handwritten digits, as in Zhu et al. [2003]

Figure 4.2: The colorization example, as in Levin et al. [2004]. Left panel has the grey-scale image with the color scribbles; Middle panel has the colored image, and the right panel has the original image.

TrustRank (Gyöngyi et al. [2004]) is similar to personalized pagerank, where the restart distribution is biased towards a set of "trusted" nodes. Joshi et al. [2007b] use harmonic ranking for finding spam. The authors build an anchor set of nodes, which can be made of all trusted or all spammy nodes. 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 use the harmonic rank with a homogeneous anchor set (all good or all bad nodes). When the anchor set consists of both good and bad anchor nodes, the harmonic rank is very similar to our formulation. Other approaches include algorithms which combine both of the above, i.e. classify the spam pages using content and graph-based features. Abernethy et al. [2008] optimize an objective function which minimizes the error on the labeled examples with an additional constraint of regularizing the function over the graph Laplacian. The idea is to penalize unsmooth functions over the graph-topology.

All these algorithms are used to classify a *static* set of spammy vs. good webpages. However in order to rerank nodes where the user-feedback is changing quickly these will have to be recomputed over the entire graph.

Ranking refinement using boosting has been used by Jin et al. [2008], where the main idea is to boost a base ranking function using the set of labeled examples. Every instance (a movie or a document) is defined by a set of features, and a base ranking algorithm outputs a ranking function from these objects. There is also additional label information obtained from user feedback which can be encoded as ranking preference. The goal is to combine these two and learn a ranking function which is consistent with both the base ranker and the user labels. The authors show that user-feedback considerably improves the performance of the base-ranking algorithm. A straight-forward use of the RankBoost algorithm (Freund et al. [2003]) on the weak learners' ranking will give rise to an $O(n^4)$ algorithm. However the authors reduce the complexity of the algorithm to $O(n^2)$, i.e. quadratic in the number of instances to rank. Note that this is still expensive when the candidate set of instances to rank is large or the labeled set is changing over time and hence is not appropriate for producing quick reranking of results.

4.3 Motivation

The graph in Figure 4.3 is an example Entity-Relation graph constructed from the papers of *awm*, an author in DBLP. There are three types of nodes: words (leftmost), papers (middle), and authors (rightmost). Words are connected to a paper if they appear on the title of the paper. Papers are connected via citation and authors are connected to a paper if they wrote the paper. We only show words which appear at least twice in this figure. For this example we also stem the word nodes, but for the experiments on the DBLP corpus

we do not do so.

Careful examination of this graph will show that author *awm* has written papers on detecting disease outbreaks, and other topics including bayesian network structure learning, and link mining. Papers 4-10 contain the words significant, disease, outbreak, food, safety, scan, statistic, monitoring etc. Papers $0, 1, 2, 3,$ and 12 contain the words asteroid, tractable, link, completion, group, structure, learning. Both of these groups contain common words, e.g. algorithm, detection, bayesian, network, pattern, spatial.

Let us assume a user searches for "awm", "disease" and "bayesian". This will return papers on disease outbreaks, and also other irrelevant papers from the other group of papers containing the word "bayesian" by author awm. Lets assume that the user is shown four papers. The user is unsure about the relevance of first result, and hence chooses not to label it, marks the second two (about disease outbreaks) as relevant and the fourth (about bayesian network structure learning) as irrelevant. A node is ranked high if in a 10-step random walk starting from it the probability of hitting a relevant node before an irrelevant one is large. Table 4.1 contains the results before and after incorporating user feedback. Note that the papers on spatial scan or spatial cluster detection are related to disease outbreaks, although the titles do not contain either of the words. They also come up to the top of the list. Also after incorporating the user feedback the list in the bottom panel shows a clear division between the relevant and irrelevant documents.

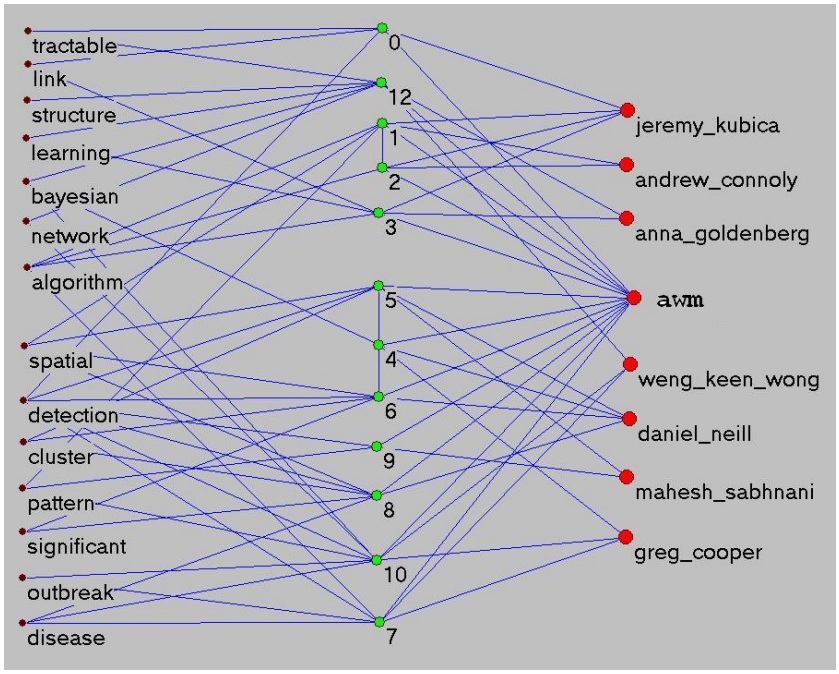


Figure 4.3: ER-graph for co-authors and papers of author *awm*

Table 4.1: Paper nodes from figure 4.3 ranked before and after incorporating user feedback (the papers annotated with "⇒" were labeled by an user. A "✓" implies that the paper is relevant to the query, whereas a "×" implies that it was irrelevant.)

Search results prior to user feedback	**relevance**
A Bayesian Scan Statistic	✓
⇒**Bayesian Network Anomaly Pattern Detection for Disease Outbreaks.**	✓
⇒**Algorithm for Early Disease Outbreak Detection**	✓
⇒**Optimal Reinsertion: Bayesian Network Structure Learning.**	×
Tractable Learning of Large Bayesian Network Structures	×
Detecting Significant Multidimensional Spatial Clusters.	✓
A Fast Multi-Resolution Method for Detection of Significant Spatial Disease Clusters.	✓
Detection of Emerging Spatial Cluster	✓
A Multiple Tree Algorithm for the Efficient Association of Asteroid Observations.	×
Variable KD-Tree Algorithms for Spatial Pattern Search and Discovery	×
Comparison of Statistical Machine Learning Algorithm on Link Completion Tasks	×
Monitoring Food Safety by Detecting Patterns in Consumer Complaints	✓
Tractable Algorithm for Group Detection and Link Mining in Large Datasets	×

Search results after incorporating user feedback	**relevance**
Bayesian Network Anomaly Pattern Detection for Disease Outbreaks.	✓
A Fast Multi-Resolution Method for Detection of Significant Spatial Disease Clusters.	✓
Algorithm for Early Disease Outbreak Detection	✓
A Bayesian Scan Statistic	✓
Detecting Significant Multidimensional Spatial Clusters.	✓
Detection of Emerging Spatial Cluster	✓
Monitoring Food Safety by Detecting Patterns in Consumer Complaints	✓
Tractable Algorithm for Group Detection and Link Mining in Large Datasets	×
Comparison of Statistical Machine Learning Algorithm on Link Completion Tasks	×
Variable KD-Tree Algorithms for Spatial Pattern Search and Discovery	×
A Multiple Tree Algorithm for the Efficient Association of Asteroid Observations.	×
Tractable Learning of Large Bayesian Network Structures	×
Optimal Reinsertion: Bayesian Network Structure Learning.	×

4.4 Harmonic Functions on Graphs, and T-step Variations

For the sake of clarity we will only describe a two class semi-supervised problem. However this can be easily generalized to a multiclass classification setting. Consider a graph with vertices V, among which U, and L are respectively the set of unlabeled and labeled nodes. A labeled node i has label $y_i \in \{0, 1\}$. Also from now on, we will interchangeably

use 'positive' for label '1' and 'negative' for label '0'.

Recall the definition of harmonic function (2.16). A harmonic function has fixed values at the given labeled points and at any other node it is an average of its neighbors. This also represents the probability of hitting a label '1' before a label '0'. We can easily generalize this to compute the probability of hitting '0' before '1', using a one vs. all encoding of labels. In order to compute the probability of hitting label $y \in \{0, 1\}$ before label $1-y$ we set $f_l(i) = 1$, if node i has label y and 0 otherwise. By using a 2 column representation of f, where the y^{th} column ($f(:, y)$) encodes the label y nodes, we can compute $f_u(i, y)$ for $y \in \{0, 1\}$ simultaneously from equation (2.16). In other words $f(i, y)$ is the probability of hitting a node with label y node before a node with label $1 - y$.

In this chapter we will use a T-step random walk. $f^T(i, 1)$ denotes the probability of hitting a label 1 node before a label 0 node in a T step random walk starting at node i. As T approaches infinity f^T approaches f. Note that in a strongly connected graph $f(i, 1) + f(i, 0) = 1$, since in infinite number of steps a random walk will hit some label. However in a T step random walk that is not the case.

Effectively f is the result of an infinite length Markov chain, and is sensitive to long range paths. In this section, we will briefly demonstrate how that could influence the classification obtained from f. We will use a simple way to classify a node: assign label 1 to it, if $f(i, 1) > f(i, 0)$, and similarly for f^T. Figure 4.4 has an undirected random graph of 100 nodes, where each node is assigned random x and y coordinates, and nodes close in the Euclidean space share a link with high probability. We use $T = 10$. The relatively larger nodes in the graph are labeled. We classify the rest of the nodes based on their f values (see Figure 4.4A.) and their f^T values (see Figure 4.4B.). The resulting classifications are color-coded (red squares:label 1) and (green circles: label 0).

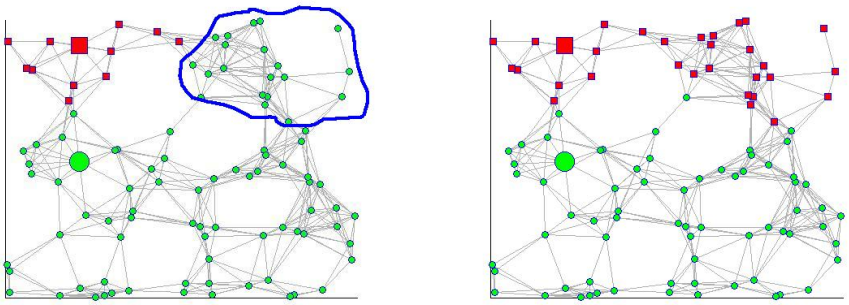
In Figure 4.4A note that the nodes within the blue handmarked area are labeled green by f, whereas they are marked red by the short term random walk f^T. This shows the difference in behavior of the measures arising from long and short term random walks.

4.4.1 Unconditional vs. Conditional Measures

We have mentioned that $f^T(i, 1) + f^T(i, 0)$ can be strictly less than 1 for small T. This is because of the fact that the random walk might not hit any label in T steps. This observation leads to a new measure g^T.

$$g^T(i, 1) = \frac{f^T(i, 1)}{f^T(i, 0) + f^T(i, 1)} \tag{4.1}$$

g can also be viewed as the probability of hitting a label 1 node before a label 0 node in T steps, conditioned on the fact that the random walk hits some label in T steps. This measure clearly has more information than unconditioned f^T. However it can be misleading



(A) (B)

Figure 4.4: A. Classification obtained from f, B. Classification obtained from f^T. A circle implies label '0', and a square label '1'. The two nodes with larger size are the two labeled nodes. The hand-marked area in (A) shows the difference in classification obtained from the infinite and short term random walks.

for the nodes where the probability of hitting any label in T steps, i.e. $f^T(i,0) + f^T(i,1)$ is very small. In order to alleviate this, we smooth the above with a small factor λ, which works like a prior belief on hitting a positive or negative label from i.

$$g^T_\lambda(i,1) = \frac{f^T(i,1) + \lambda}{f^T(i,0) + f^T(i,1) + 2\lambda} \tag{4.2}$$

When $f^T(i,0) + f^T(i,1) = 0$, this small change makes $g^T_\lambda = 0.5$. Note that this formulation puts a uniform prior over the positive and negative classes. It is also possible to put a non-uniform prior.

In the rest of the section, we will demonstrate how the unconditional, conditional and smoothed-conditional probabilities perform in terms of AUC scores in a toy example. We construct a toy example with 200 nodes, 2 clusters, 260 edges, out of which 30 are inter-cluster links. We will denote one cluster as the positive cluster and the other as the negative cluster. We made intra-cluster connections sparse, in order to emphasize the effect of T (not all nodes are reachable from others for small T). 20 randomly picked nodes are assigned labels. We vary the number of positive labels n_p from 1 to 19. For each number, the AUC score is averaged over 10 random runs. In each random run n_p positive nodes are chosen from the positive cluster and $n_n = 20 - n_p$ negative nodes are chosen from negative cluster uniformly at random. For each random run, the $f^T(i,1), g^T(i,1), g^T_\lambda(i,1)$ values of the unlabeled nodes i are computed and AUC scores for each function are recorded and averaged over the runs. The results are shown in Figure 4.5. The upper panel of the figure has the results for $T = 5$, whereas the lower panel has results for $T = 10$. Here are the interesting observations:

1. The three functions perform comparably for $T = 10$.

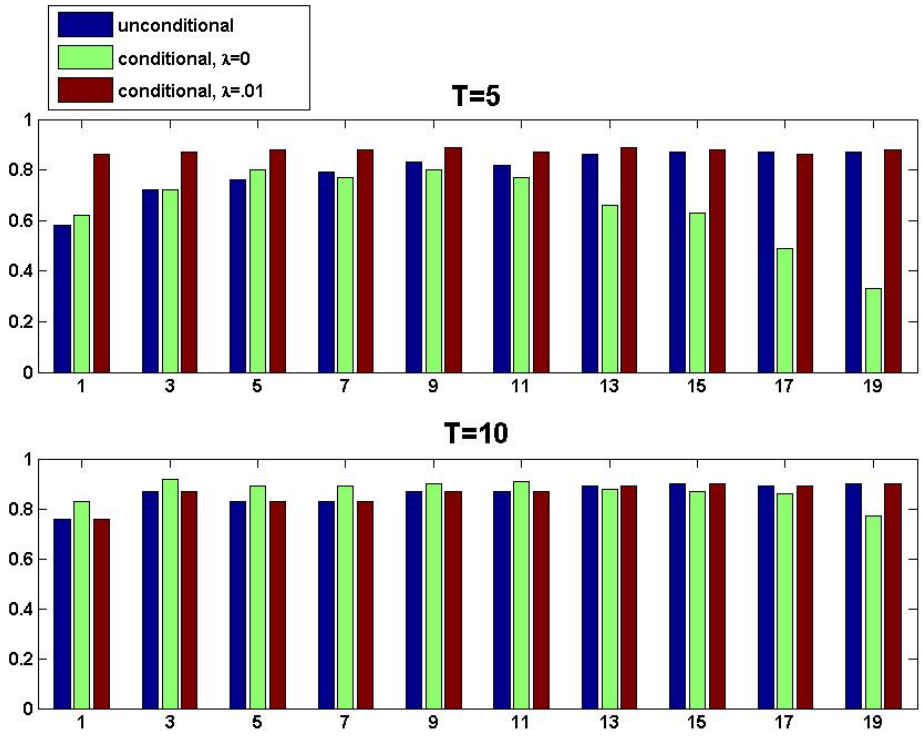


Figure 4.5: From left to right the bars show AUC scores for unconditional, conditional with $\lambda = 0$ and conditional with $\lambda = 0.01$, respectively. x-axis shows number of positive labels, and y-axis shows AUC scores.

2. When $T = 5$, for small n_p the unconditional probabilities (f^T) and the unsmoothed conditional probabilities (g^T) perform poorly. As n_p is increased both of these show an improved performance. However the performance of g^T becomes poor as n_p increases beyond 10.
3. The performance of g^T_λ is good (around and above 80%) for different values of T and different number of positive labels.

Let us justify the three observations. For $T = 5$, some nodes might not reach any labels, leading to the different behavior of the three different algorithms, whereas for $T = 10$ all of them perform reasonably well. Since the number of labels is fixed at 20, small n_p values indicate large n_n (number of negatives). For small values of n_p, length 5 random walks from many nodes hit some negative node before hitting any positive node, and hence $f^T(i, 1)$ values are zero. For all these nodes $g^T(i, 1)$ is also zero, leading to poor AUC score. The parameter λ enables $g^T_\lambda(i, 1)$ to successfully use the $f^T(i, 0)$. As n_p increases and n_n decreases, we observe a symmetric situation w.r.t the negative nodes. Even if ranking w.r.t. $f^T(i, 1)$ gives good AUC scores (now that we have a lot of positive

labels), the unsmoothed conditional function $g^T(i,1)$ evaluates to 1 for many nodes, since length 5 random walks from those now do not hit a negative node before a positive node. g^T_λ however uses best of both worlds.

4.5 Algorithms

In this section we will present two different algorithms to compute the probability of hitting a positive node before a negative node within T steps. We will consider two scenarios which justify the use of both algorithms. 1) The user wants to rerank only the top 100 search results shown before based on his/her user feedback. 2) The user wants to rerank the entire graph based on the user feedback.

How do we justify these two needs? The first one is useful when we believe that the top 100 results from the search engine prior to any relevance feedback are most relevant to the query. Hence reranking will simply reorder these to show the most relevant results on top of the ranklist. Now consider the following case: let us say the user searches for the author Ting Liu in DBLP. There are two authors in DBLP with the same name, and they are treated as the same record. One of these authors is much more prolific compared to the other. The top 100 results will mostly contain results contextually similar to the prolific author. If however the user is interested in the less prolific author, then the better approach is to rank-order all nodes in the graph.

We would also like to clarify one point. Although we are not using the query node directly for ranking, it is present indirectly in both these scenarios. This is because the user enters the feedback on the top few relevant search results *w.r.t the query*. This feedback is used as the labeled set. Now, the only difference between the two schemes is that, in the first one we rank only the top 100 search results using the labeled nodes, whereas in the second scenario the entire graph is ranked using the labels. In some settings it might be possible to use the query directly by incorporating it in the positively labeled nodes. However in the above example of DBLP, where two authors of the same name are often merged into one record, labeling this merged node by a positive label only confuses the algorithm further.

For the first scenario we present a simple sampling algorithm, whereas for the second we propose a branch and bound algorithm which finds the potential top-ranked nodes by provably pruning away the un-interesting nodes. For notational simplicity, we will ignore the superscript T and subscript λ for now. Unless pointed out f represents f^T and g represents g^T_λ. Also we will use positive label for label 1 and negative label for label 0.

4.5.1 The Sampling Scheme

For every node i in the *candidate set*, simulate M independent length T random walks. A walk stops if it hits any labeled node. Let us say M_p of these hit a positive node, and M_n of these hit a negative node. Hence the estimates of $f(i,1)$ and $f(i,0)$ are respectively given by $\hat{f}(i,1) = \frac{M_p}{M}$ and $\hat{f}(i,0) = \frac{M_n}{M}$. By a straightforward application of the Hoeffding bound, we can say that one needs $O(1/\epsilon^2 \log(1/\delta))$ random runs to achieve $|\hat{f}(i,1) - f(i,1)| \leq \epsilon$ with probability at least $1-\delta$. Let us say we also want to obtain estimates $\hat{g}$ of the conditional probability g.

$$\hat{g}(i,1) = \frac{\hat{f}(i,1) + \lambda}{\hat{f}(i,1) + \hat{f}(i,0) + 2\lambda}$$

If $\hat{f}(i,y) \in [f(i,y) - \epsilon, f(i,y) + \epsilon]$, for $y \in \{0,1\}$ with high probability then a simple algebraic manipulation gives: $\hat{g}(i,y) \in [g(i,y) - \epsilon', g(i,y) + \epsilon']$, w.h.p. where $\epsilon' = \epsilon/(f(i,0) + f(i,1) + 2\lambda)$. In order to use this for getting an intuition about how big λ should be, we notice that λ needs to be at least $0.5 * (1 - f(i,0) - f(i,1)))$ in order to obtain $\epsilon' = \epsilon$. Note that $1 - f(i,0) - f(i,1)$ is simply the probability that a random walk from i will not hit any label in T steps.

4.5.2 The Branch and Bound Algorithm

In chapter 3, we have described how to compute functions over a graph using neighborhood expansion. In particular, we use this for computing nearest neighbors in hitting time to a node. We will show how to generalize this technique to compute top k nodes in f values.

Here is the main intuition behind the algorithm. We maintain a neighborhood around the labeled points. At any time we compute a lower ($fl^T(i,1)$) and upper ($fu^T(i,1)$) bound of $f(i,1)$ for all nodes i in this neighborhood. We characterize $f(i,1)$ for all i outside the neighborhood by a single lower and upper bound. The global lower bound is zero. Let the global upper bound be x. As we expand the neighborhood, these bounds keep getting tighter, and x keeps decreasing. We stop expansion when x falls below a small constant α. We will first describe the lower and upper bounds and then present the algorithm and the neighborhood expansion scheme. We have:

$$f^T(i,1) = \begin{cases} 1 & \text{If } i = + \\ 0 & \text{If } i = - \\ 0 & \text{If } T = 0 \\ \sum_j P(i,j) f^{T-1}(j,1) & \text{Otherwise} \end{cases} \tag{4.3}$$

The fastest way of hitting a positive node before a negative node from outside the neighborhood is achieved by magically jumping to the boundary node which has the maximum probability to hit a positive node before a negative node. Also from equation (4.3) note that the only case where we need to use this is when any of the direct outgoing neighbors of i is outside the neighborhood.

Since we are computing the neighborhood for $f^T(i, y)$, let us denote the neighborhood as S_y, for $y \in \{0, 1\}$. Let $O(i)$ be the set of direct outgoing neighbors of node i. Let $I(i)$ be the direct incoming neighbors of i. Let $\delta(S_y)$ be the boundary of the neighborhood S_y, i.e. it contains all nodes in S_y which has at least one incoming neighbor outside S_y. Hence the T-step upper bound is defined as

$$\begin{aligned} fu^T(i, y) = &\sum_{j \in O(i) \cap S_y} P(i, j) fu^{T-1}(j, y) \\ &+ (1 - \sum_{j \in O(i) \cap S_y} P(i, j)) \max_{m \in \delta(S_y)} fu^{T-2}(m, y) \end{aligned} \tag{4.4}$$

For the lower bound, we just use the fact that the lowest probability to hit a positive node before a negative one from outside the neighborhood is simply zero. This gives rise to

$$fl^T(i, y) = \sum_{j \in O(i) \cap S_y} P(i, j) fl^{T-1}(j, y) \tag{4.5}$$

The global upper bound $ub^T(S_y)$ is simply defined as the maximum probability to reach label y before any other label from a node on the boundary of the neighborhood.

$$ub^T(S_y) = \max_{m \in \delta(S_y)} fu^{T-1}(m, y) \tag{4.6}$$

Here is the pseudo-code of the algorithm:

1. Initialize $S_y = \bigcup_{i \in L} I(i)$.
2. $\forall i \in S_y$, compute $fl^T(i, y)$ (see eq. 4.5), $fu^T(i, y)$ (see eq. 4.4)
3. Compute $ub^T(S_y)$ (see eq. 4.6).
4. Expand S_y. (Details are provided below)
5. Stop if $ub^T(S_y) \leq \alpha$. Else reiterate from step 2.

By definition all nodes i outside S_y have $f^T(i, y) \leq ub^T(S_y) \leq \alpha$. Hence we have pruned away all nodes with very small probability of hitting a positive before a negative.

Now how do we expand a neighborhood in step 4? We use the heuristic of picking the k nodes on the boundary with k largest fu^{T-1} values, and augment the boundary by adding their incoming and outgoing neighbors to S_y. The intuition is that since all bounds use the maximum probability of hitting label y before any other label in $T - 1$ steps from

a boundary node, augmenting the neighborhood around these points will probably tighten the bounds more.

The ranking step to obtain top k nodes using upper and lower bounds is simple: we return all nodes which have lower bound greater than the $k+1^{th}$ largest upper bound (when $k=1$, the k^{th} largest is the largest probability). We denote this as fu_{k+1}. Since all nodes which have probability greater than α to hit a positive node before a negative node are guaranteed to be in the neighborhood expanded, we know that the true $(k+1)^{th}$ largest probability will be smaller than fu_{k+1}. Hence any node with lower bound greater than fu_{k+1} is guaranteed to be greater than the $(k+1)^{th}$ largest probability. We use a multiplicative slack, e.g. $(fu_{k+1}(1-\epsilon))$ in order to return the top k *approximately* large probability nodes. In our algorithm we initialize α with a large value and keep decreasing it until the bounds are tight enough to return k largest nodes. Note that one could rank the probabilities using the lower bounds, and return top k of those after expanding the neighborhood a fixed number of times. This will only mean a larger approximation slack.

In this section, we presented a branch and bound algorithm to compute top k nodes in the unconditional probability to hit a positive label before a negative label. How do we compute the conditional probability?

Given the upper and lower bounds $\{fu(i,0), fl(i,0)\}$ and $\{fu(i,1), fl(i,1)\}$, the upper and lower bounds on the conditional probabilities can be given by:

$$gu(i,1) = \frac{fu(i,1)+\lambda}{fu(i,1)+fl(i,0)+2\lambda}$$
$$gl(i,1) = \frac{fl(i,1)+\lambda}{fl(i,1)+fu(i,0)+2\lambda} \tag{4.7}$$

Currently we obtain bounds on $f(i,y)$, for $y \in \{0,1\}$, and use equation (4.7) for obtaining the conditional probabilities. Doing neighborhood expansion for the conditional probabilities is part of ongoing work.

4.5.3 Number of Nodes with High Harmonic Function Value

How many nodes in the neighborhood will have probability greater than α? It is hard to answer this question for a general directed graph. So we will present a result for an undirected graph. Let S be the set of nodes with $f \geq \alpha$. Also let L_p be the set of nodes with a positive label. We prove (appendix A.B) the following for an undirected graph.

Theorem 4.5.1. *In an undirected graph, the size of set $S = \{i | f^T(i) \geq \alpha\}$ is upper bounded as: $|S| \leq \frac{\sum_{p \in L_p} d(p)}{min_{i \in S} d(i)} \frac{T}{\alpha}$, where L_p is the set of nodes with a positive label.*

This implication of the above theorem is interesting. The upper bound on S is proportional to the total degree of all the positive labels and inversely proportional to the quantity

$min_{i \in S} d(i)$, which is the minimum degree of nodes in S. This means, if the positive nodes have a higher degree relative to the neighboring nodes then it is easy to reach a positive node. As we will point out in section 4.6.6 this brings us directly to an active learning question: how to select a good set of nodes to be labeled?

4.6 Results

In this section, we present results on the DBLP graph[1]. The graph has a similar schema as in Figure 4.3. The original DBLP graph has about $200,000$ words, $900,000$ papers and $500,000$ authors. There are about 10 million edges. We will mention the total number of undirected edges. As we will point out later, for each edge we have links in both directions with different weights. We present our results on two graphs built from this corpus:

1. **Two-layered graph**: We exclude words from the graph representation. Most citation-database related work (Balmin et al. [2004], Chakrabarti [2007]) uses this representation, where the words are used only in an inverted index pointing to the document originally containing those. This graph has around $1.4M$ nodes and $2.2M$ edges.

2. **Three-layered graph**:We include all the words which occur more than 20 times and less than $5,000$ times in our graph representation. There are $15,000$ such words. The resulting graph has around $6M$ edges and 1.4 M nodes. Note that the extra number of edges from $15,000$ word nodes is about $3.7M$.

The links between the paper and author layer is undirected. For citation links within the paper layer, a link from a paper x to paper y citing x, is assigned one-tenth of the weight on the link from y to x. This is very similar to the weighing convention of Chakrabarti [2007]. In the three layer graph-representation, for any paper we assign a total weight of W to the words in its title, a total weight of P to the papers it cites and A to the authors on it. We use an inverse frequency scheme for the paper-to-word link weight. The details can be found in the experimental section in section 3.5.2. We set $W = 1$; $A = 10$ and $P = 10$ so that the word layer to paper layer connection is almost directed. The leaf nodes are augmented with a self loop, with the same weight as its single edge.

4.6.1 Experimental Settings

In this chapter, we have proposed a scheme to rerank search results based on user feedback. However it is hard to do fair evaluation of user-feedback based ranking. Hence we have designed an automated way to mimic user-feedback. In DBLP two authors with the same

[1]We thank Andy Schlaikjer for sharing this dataset.

name are often listed as the same record. We use entity disambiguation as a task to evaluate the predictive performance of a measure. Here are the steps:

(A) Pick 4 authors having a common surname.

(B) Merge these authors in the same node.

(C) Rank other nodes using proximity measure $\mathfrak{h}$ from the merged node.

(D) Label the top L papers on this list using ground truth.

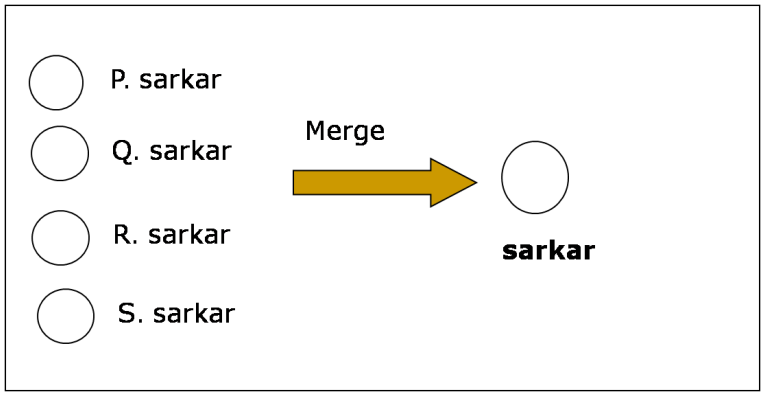
(E) The testset consists of all papers written by these authors modulo the ones already labeled.

(F) Compute different measures (e.g. conditional probability to hit a positive label before a negative label, etc) for each testnode and then compute AUC score against the ground truth.

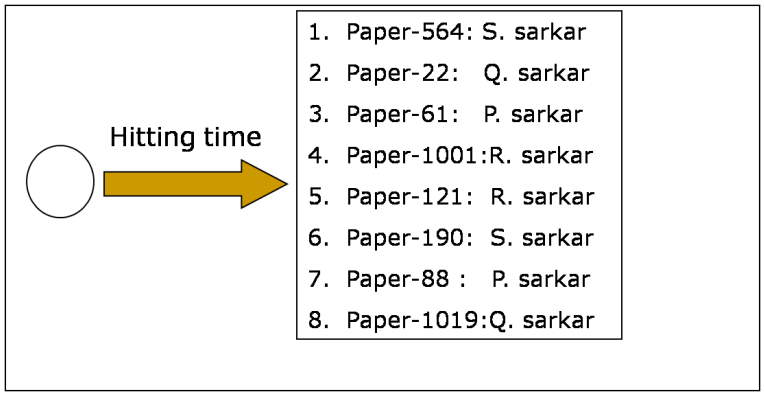
We pick about 20 common surnames. Note that for each surname there are 4 different authors and hence 4 different one-vs-all classification scenarios. However sometimes the top L papers might belong to the same author if he is relatively more prolific than the others. Removing all those cases, we had about $30 - 40$ points in our testset, which give about 2 classifications per surname. For each surname, we compute the mean AUC score, and then report the mean over all the surnames. We also apriori fixed the testset so that we can compare different experiments over the same graph, and same proximity measure $\mathfrak{h}$. The different experiments were conducted by varying values of the truncation parameter T and number of labels L. We use truncated hitting time (eq. 3.3) as the proximity measure ($\mathfrak{h}$) from the merged node, since its easy to compute and also is very similar to Personalized Pagerank (PPV). The hitting time from a node x to node y is the expected time to hit y for the first time in a random walk starting from node x. For the truncated version only paths of length less than T are considered. We hand-tuned the smoothing parameter λ to be $1e - 4$ for all our experiments. We also use $2,500$ samples for the sampling algorithm.

We will describe the results on the two and three-layered graphs. Note that although experiments with varying number of labels and varying values of T are conducted on the same testset across a graph, the testsets are slightly different for two different graphs. The purpose of showing results on both is to examine the general predictive behavior of the different measures on the two different graph topologies. In section 4.6.6 we show how the choice of the original proximity function $\mathfrak{h}$ affects the performance of the different measures. Also in this case the testsets are different, since the ranklists generated from different choices of $\mathfrak{h}$ are different.

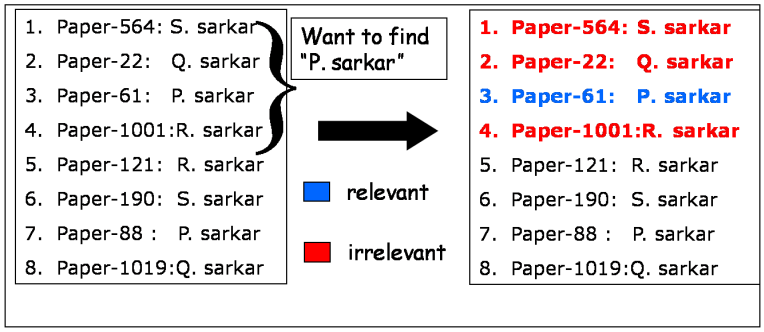
We compare with two alternative measures of proximity from only the positive labels. Note that this is a similar idea to TrustRank. We use 10-truncated hitting time (section



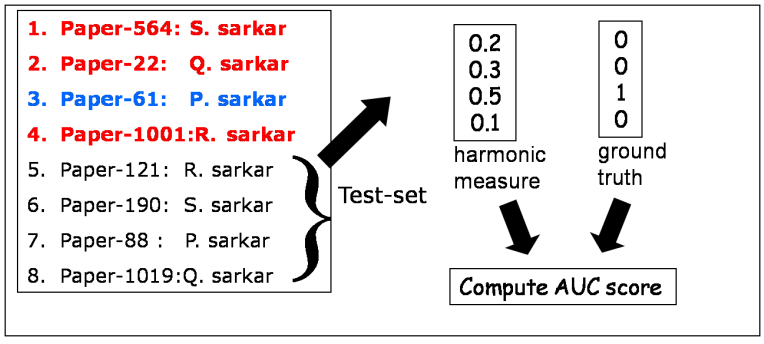
(A,B)



(C)



(D)



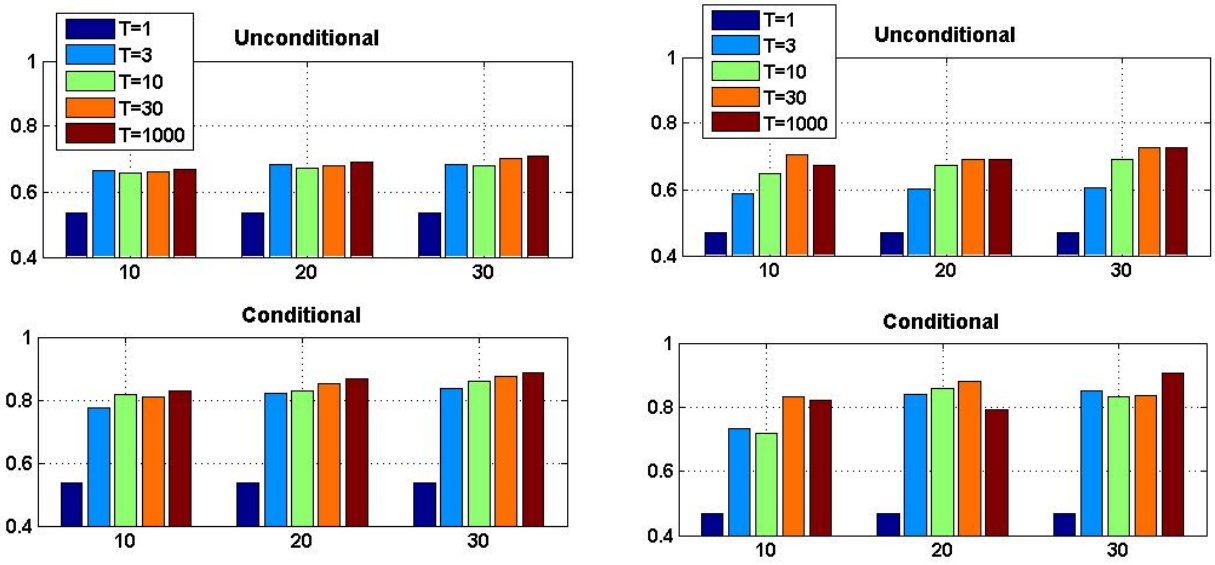
(E,F)

Figure 4.6: Steps of the entity-disambiguation task

4.6.3) and PPV (section 4.6.4) with teleportation value of 0.1 for our experiments. For all experiments other than section 4.6.2, we use $T = 10$ for computing conditional or unconditional probabilities to hit a positive node before a negative.

4.6.2 Effect of T

In this section we describe the effect of parameter T on the AUC scores. The results are described in figure 4.7. Figure 4.7A. contains the results on the two-layered graph, whereas figure 4.7B. contains the results on three-layered graph which contains the words as well, i.e. information can pass through common words in the graph. We consider the values of



(A) (B)

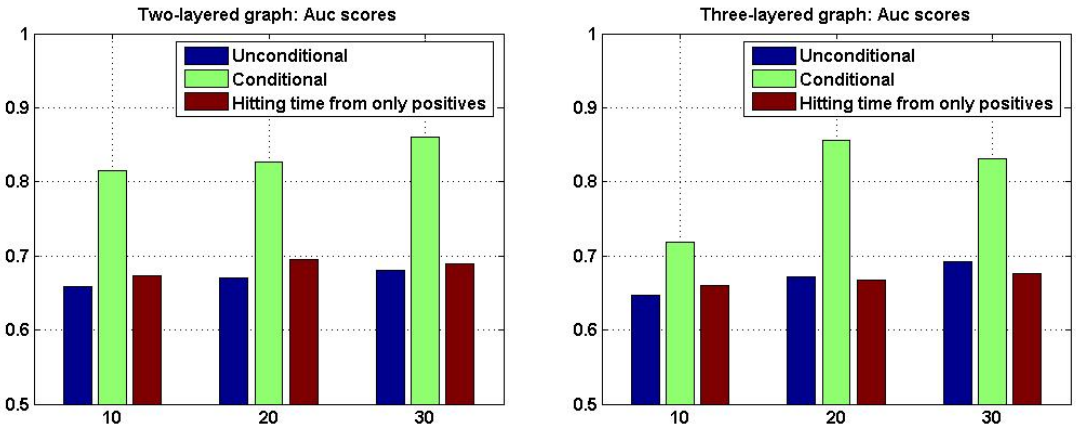
Figure 4.7: Experiments on A) The two-layered graph and B) The three-layered graph. The y axis has the AUC scores and the x axis has different number of labels. The different bars from left to right are for different values of T: 1,3,10,30 and 1000, respectively.

T in $\{1, 3, 10, 30, 1000\}$. For $T = 1$, the information can only come from direct neighbors. Since both our testset and labeled node-set consists of papers there is a very slight chance of hitting a labeled node via citation. Still in both cases, the AUC scores are close to random. Another interesting behavior is that $T = 3$ and $T = 10$ perform comparably well and much better than random in both graphs. However in the two-layered graph, with 10 labeled nodes, varying T from 3 to 1000 does not give much boost in performance. However in the graph with the word layer, from $T = 10$ to $T = 30$ there is about a 10% increase in AUC score. This is expected since in a graph with a word layer, it takes longer

time to hit a small set of labeled nodes. Also, note that the AUC scores increase only slightly as T is increased to 1000.

4.6.3 Comparison with Different Algorithms

In Figure 4.8, we compare the unconditional and conditional probabilities with an alternative diffusion based proximity measure, namely the hitting time from the positive nodes. We use $T = 10$ for these experiments. From figure 4.7 it can be observed that other values of T have better AUC score than $T = 10$. However we fix $T = 10$ as a reasonable truncation parameter. Note that in both the two and three layered graphs the conditional measure outperforms the hitting time. However the unconditional probability performs comparably. This may be attributed to the fact that the conditional measure successfully uses the information from both the positive and the negative labels. Also note that for the three-layered graph for 10 labels, the difference between the conditional and the hitting time is the smallest. This is because of the fact that in presence of the word nodes it takes more time to hit a relatively smaller set of labeled nodes. For $T = 20$ and $T = 30$ the conditional probability outperforms T-truncated hitting time by a large margin.

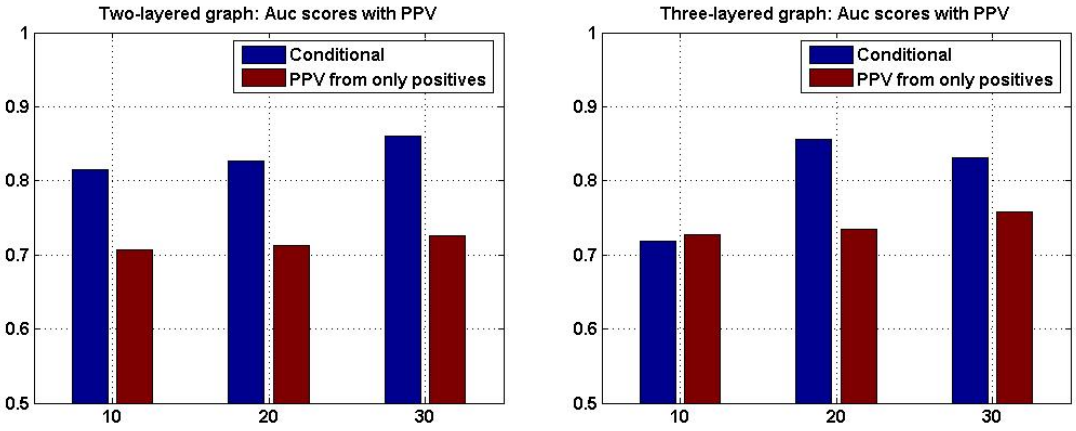


(A) (B)

Figure 4.8: Experiments on A) The two-layered graph and B) The three-layered graph. The y axis has the AUC scores and the x axis has different number of labels. The bars from left to right are for different measures, *unconditional*, *conditional*, and 10-*truncated hitting time from only positives.*, respectively.

4.6.4 Comparison with PPV

So far we have compared our algorithms only against proximity measures like hitting time. In this section we will also present the performance of PPV in Figures 4.9(A) and 4.9(B). We used PPV with its start distribution uniform over the positive nodes. We only compare the conditional probability ($T = 10$) with PPV (restart probability 0.1). The results are similar to those of the last section. For the three-layered graph, with 10 labels, PPV and conditional probabilities perform comparatively well. However for all other cases, PPV is outperformed by the former.



(A) (B)

Figure 4.9: Experiments on A. The two-layered graph and B. The three-layered graph. The y axis has the AUC score and the x axis has different number of labels. From left to right the bars are for two measures, respectively *conditional*, and *Personalized Pagerank from the Positive Labels*.

4.6.5 Sampling vs Branch and Bound

In all our experiments, we have a small set of candidate nodes to be ranked, and hence we have used sampling to compute the conditional and unconditional probabilities. In this section, we demonstrate the scenario where we want to rank a larger set of nodes. In Table 4.2 we present the average timing results for the branch and bound algorithm (BB) vs. the sampling algorithm (Samp) on around 15 disambiguation tasks with 10 labels each. The experimental setting is exactly the same as before. For each task we carry out sampling for $1,000$ candidate nodes. The timing provided for branch and bound is the average time in seconds to compute k nearest neighbors in $f(i, 0)$ and $f(i, 1)$. We also present the average number of neighbors (3-Hops) within 3 hops of the labeled nodes. This is to

Table 4.2: Timing results for Branch and Bound (BB) and Sampling (Samp) for 1000 candidate nodes. We also report the average number of nodes within 3 hops of the labeled nodes.

Two-layered				Three-layered		
k	BB(s)	Samp(s)	3-Hops(#)	BB(s)	Samp(s)	3-Hops(#)
10	1.6	90	2,000	37	283	160,000
30	3			62		

illustrate the growth properties of the two graphs. Note that the average time for branch and bound increases by a factor of 20 whereas that of sampling increases by a factor of 3. The number of edges in the three-layered graph is about 3 times as large as the two-layered one, whereas the number of nodes stay roughly the same. This implies that the average degree becomes about 3 times as large. Time for sampling is directly proportional to the average degree, which explains the increase in the time required for sampling.

The time increase for branch and bound can be explained by the numbers in the fourth and seventh columns (3-Hops(#)). The size of the neighborhood expanded depends on the growth rate of the graph. The total number of nodes within 3 hops of all the labeled nodes grow by a factor of around 80 from the two-layered to the three-layered representation. Although the neighborhood computed via branch and bound does not contain all nodes within 3 hops, its size still grows faster in the three layered graph than in the two-layered one.

4.6.6 Effect of the Original Ranking Function

In this section, we show the effect of the original ranking function from the merged node. So far we have only considered the nodes ranked by the hitting time from the merged author node. In a web-search setting, the user will be asked to label these nodes. Note that this is an active learning heuristic. Xu et al. [2007] point out three desirable properties of an effective active learning strategy for relevance feedback in information retrieval. Let us denote the set of feedback nodes as F. F should be:

1. Relevant to the query.
2. Diverse, in order to avoid redundancy.
3. Selected from a high density region of the data, so that the query-feedback algorithm has more information.

By picking the labeled nodes from the top ranked documents, we definitely fulfil the first criterion. In this section, we will show that picking hitting time as the original retrieval algorithm automatically satisfies the third criterion.

In order to illustrate this, we will pick truncated commute time as the ranking function. The commute time between two nodes x, y is defined as the expected time to hit node y from node x for the first time and come back. The truncated version is simply the sum of the truncated hitting time from x to y and y to x. We only present the results on the two-layered graph for simplicity. Also the comparison is not on the exact same testset, since the ranklist for the authors change from one measure to another. Note that the performance

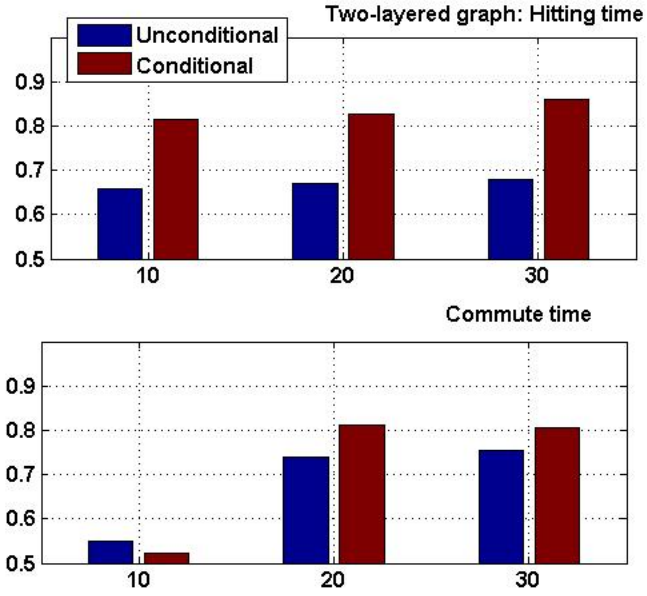


Figure 4.10: The top panel consists of the results when the original ranklist is based on Hitting time from the merged node. The bottom panel shows the results where the ranklist is created based on the Commute time from the merged node. The y axis has the AUC scores for *unconditional* and *conditional* probabilities (bars from left to right) and the x axis has different number of labels.

of the ranklist based on commute times for 10 labels is close to random. Recall the result in theorem 4.5.1. We have shown that in an undirected graph, if the positive nodes have higher relative degree compared to the neighboring nodes, then the number of nodes which can reach a positive label before a negative label increases. This intuition also helps to pick an appropriate active learning strategy for a directed graph. The hitting times from a merged node to high degree nodes are generally small, since a random walk is more probable to hit a high degree node. Hence the labeled set picked from the top-ranked

nodes will also be easily reachable from other nodes. On the other hand, commute time between nodes x and y is small if degree of y is small, since that reduces the expected return time to x. As a result, the top 10 labels in the commute-time based rank list can be harder to reach from other nodes. This is reflected in the results presented in Figure 4.10. Note that for 20 or 30 labels, this difference is not so noticeable as for 10 labels. This is because as we increase the labeled set size, probability of reaching some label increases.

4.7 Summary and Discussion

In this chapter, we consider the problem of re-ranking search results by incorporating user feedback. We use a short term variation of a well known graph theoretic measure, e.g. harmonic functions, for discriminating irrelevant results from relevant results using a few labeled examples provided by the user. The key intuition is that nodes relatively closer (in graph topology) to the relevant nodes than the irrelevant nodes are more likely to be relevant.

In the last chapter, we have presented a deterministic branch and bound type algorithm for computing nearest neighbors in hitting and commute times. In this chapter we have shown how to generalize this idea to compute top-k nodes in harmonic functions. This chapter along with the last one provides, 1) tractable base rankings from a query node using truncated hitting/commute times, and 2) fast reranking algorithms to improve upon the base ranking by incorporating user feedback.

Our main goal is efficient and effective ranking of nodes w.r.t a query in a large graph. We have achieved this by local neighborhood expansion algorithms. However there are two fundamental problems associated with these approaches. First, they can become significantly slower whenever high-degree nodes are encountered (a common phenomenon in real-world graphs). The second problem is that of computing similarities at query time when the graph is too large to be memory-resident. In such a case, we would not be allowed random access to memory, which is an important assumption in all of our algorithms.

This brings us to the next chapter. In a nutshell, we prove that turning high degree nodes into sinks results in only a small approximation error, while greatly improving running times. We deal with the second problem with a simple and obvious solution: to split the graph into clusters of nodes and store each cluster on a disk page; ideally random walks will rarely cross cluster boundaries and cause page-faults. We also show how to generalize our deterministic algorithm to this clustered representation of graphs on disk.

Chapter 5

Fast Nearest-neighbor Search in Disk Based Graphs

"You see," he explained, "I consider that a man's brain originally is like a little empty attic, and you have to stock it with such furniture as you choose. . . . Depend upon it there comes a time when for every addition of knowledge you forget something that you knew before. It is of the highest importance, therefore, not to have useless facts elbowing out the useful ones."

— Sir Arthur Conan Doyle's Sherlock Holmes.

5.1 Introduction

In this thesis, our main goal is to design fast algorithms for proximity search in large graphs. In chapter 3, we presented a local algorithm to compute approximate nearest neighbors in truncated hitting and commute times. Although these measures produce meaningful ranking, they can suffer from noise in data and ambiguity in the query. In chapter 4, we deal with this problem by incorporating relevance feedback provided by the user. We also present a fast local reranking algorithm which computes harmonic functions over this labeled set, and thus ranks the relevant results higher. One important assumption in these two chapters was that the graph can be loaded into main memory. A natural question at this point is, how do we obtain ranking when the graph is disk-resident. Also, these algorithms, like other local algorithms suffer from the presence of high degree nodes; we want to deal with this issue as well. These are the main components of this chapter.

There has been a lot of work on computing random walk based measures in large graphs. However, there are still limitations to what we can do when graphs become enormous. Some algorithms, such as streaming algorithms (Sarma et al. [2008]), must make

passes over the entire dataset to answer any query; this can be prohibitively expensive in online settings. Others perform clever preprocessing so that queries can be answered efficiently (Fogaras et al. [2004], Sarlós et al. [2006]). However these algorithms store information which can be used for computing a *specific* similarity measure, e.g. personalized pagerank for (Fogaras et al. [2004]). This chapter introduces analysis and algorithms which try to address the scalability problem in a generalizable way: not specific to one kind of graph partitioning nor one specific proximity measure.

Another broad class of algorithms estimate the similarity between the query node and other nodes by examining local neighborhoods around the query node (Berkhin [2006], Andersen et al. [2006a], Chakrabarti [2007], Sarlós et al. [2006], Chen et al. [2004]). The intuition is that in order to compute nearest neighbors, hopefully one would not need to look too far away from the query node. However, one fundamental computational issue with these techniques is the presence of very high degree nodes in the network. These techniques rely on updating the value of one node by combining that of its neighbors; whenever a high degree node is encountered, these algorithms have to examine a much larger neighborhood leading to severely degraded performance. Unfortunately, real-world graphs contain such high-degree nodes which, though few in number, are easily reachable from other nodes and hence are often encountered in random walks. Our first contribution is a simple transform of the graph that can mitigate the damage while having a provably bounded impact on accuracy. Indeed, we show that they *improve* accuracy in certain tasks like link prediction.

Another problem linked to large graphs is that algorithms can no longer assume that the entire graph can be stored in memory. In some cases, clever graph compression techniques can be applied to fit the graphs into main memory, but there are at least three settings where this might not work. First, social networks are far less compressible than Web graphs (Chierichetti et al. [2009]). Second, decompression might lead to an unacceptable increase in query response time. Third, even if a graph could be compressed down to a gigabyte (comfortable main memory size in 2009), it is undesirable to keep it in memory on a machine which is running other applications, and in which there are occasional user queries to the graph. A good example of this third case is the problem of searching personal information networks (Chakrabarti et al. [2005]), which integrates the user's personal information with information from the Web and hence needs to be performed on the user's own machine for privacy preservation (Dalvi et al. [2008]).

Is there an intuitive representation of a disk-resident graph such that any random walk based measure can be easily computed from this representation? The obvious solution is to split the graph into clusters of nodes and store each cluster on a disk page; ideally random walks will rarely cross cluster boundaries and cause page-faults. This *clustered representation* can be used to quickly simulate random walks from any graph node, and by extension, any similarity measure based on random walks. Still, while simulations

are computationally cheap, they have a lot of variation, and for some real-world graphs lacking well-defined clusters, they often lead to many page-faults. We propose a *deterministic* local algorithm guaranteed to return nearest neighbors in personalized pagerank from the disk-resident clustered graph. The same idea can also be used for computing nearest neighbors in hitting times. This is our second contribution.

Finally, we develop a fully external-memory clustering algorithm, named RWDISK, that uses only sequential sweeps over data files. This serves as a preprocessing step that yields the disk-resident clustered representation mentioned above, on top of which any nearest-neighbor algorithm can be run.

We present extensive experimental results on real-world graphs with up-to 86 million edges. We show how tackling the high degree nodes boost both computational efficiency and link prediction accuracy; the improved performance of the deterministic algorithm over vanilla Monte Carlo simulations; and finally the finer quality of clusters returned by our clustering algorithm compared to a popular in-memory clustering algorithm, METIS (Karypis and Kumar [1998]).

The chapter is organized as follows: in section 5.3.1, we theoretically show the effect of high degree nodes on personalized pagerank (PPV). We also show the same for discounted hitting times by presenting a new result which expresses discounted hitting times in terms of PPV. In section 5.3.2, we present a deterministic local algorithm to compute top k nodes in personalized pagerank using these clusters. In section 5.3.3, we describe our RWDISK algorithm for clustering a disk resident graph by only using sequential sweeps of files. We conclude with experimental results on large disk-resident Live Journal, DBLP and Citeseer graphs in section 5.4.

5.2 Background

In this section we will briefly describe interesting random walk based proximity measures, namely personalized pagerank and hitting times. We will also discuss the relevance of personalized pagerank for graph clustering.

PERSONALIZED PAGERANK. We will quickly recapitulate the intuition behind personalized pagerank, which has been discussed in detail in chapter 2. Consider a random walk starting at node a, such that at any step, the walk can be reset to the start node with probability α. The stationary distribution corresponding to this stochastic process is defined as the personalized pagerank vector (PPV) of node a. The entry corresponding to node j in the PPV vector for node a is denoted by $PPV(a, j)$. Large values of $PPV(a, j)$ is indicative of higher similarity/relevance of node j w.r.t a.

While this intuition leads directly to the definition of personalized pagerank in eq. (2.5),

we can also view personalized pagerank as a weighted sum of occupancy probabilities from timesteps $t = 0$ to $t = \infty$. This definition can also be found in the work by Jeh and Widom [2002a]. Recall that P is the row normalized probability transition matrix and $P^T v$ is the distribution after one step of random walk from v. Let's define x_t as the probability distribution over all nodes at timestep t. x_0 is defined as the probability distribution with 1.0 at the start node and zero elsewhere. By definition we have $x_t = P^T x_{t-1} = (P^T)^t x_0$. Let v_t be the partial sum of occupancy probabilities up-to timestep t. Now we can write PPV as:

$$v(j) = \sum_{t=1}^{\infty} \alpha(1-\alpha)^{t-1} x_{t-1}(j) = \lim_{t\to\infty} v_t(j) \tag{5.1}$$

Personalized pagerank has been shown to have empirical benefits in keyword search (Chakrabarti [2007]), link prediction (Liben-Nowell and Kleinberg [2003]), fighting spam (Joshi et al. [2007a]); there has been an extensive literature on algorithms for computing them locally (Berkhin [2006], Chakrabarti [2007]), off-line (Jeh and Widom [2002a], Fogaras et al. [2004], Sarlós et al. [2006]), and from streaming data (Sarma et al. [2008]) etc.

DISCOUNTED HITTING TIME. We have defined hitting times in chapter 2. This is a well-studied measure in probability theory (Aldous and Fill [200X]). Hitting times and other local variations of it has been used as a proximity measure for link prediction (Liben-Nowell and Kleinberg [2003]), recommender systems (Brand [2005]), query suggestion (Mei et al. [2008]), manipulation resistant reputation systems (Hopcroft and Sheldon [2007]) etc. In chapter 3, we have defined truncated hitting times in eq. (5.2), where only paths of length up-to a given horizon are considered. Since our ultimate goal is to show the effect of high degree nodes on hitting times, we will use the discounted variant, since it is easier to analyze.

Note that this is closer to the original hitting time definition, and is different from the generalized hitting time defined in (Graham and Zhao). Consider a random walk which, once started from i stops if node j is encountered, or with probability α. The expected time to hit node j in this process is defined as the α discounted hitting time from node i to node j, ($h_\alpha(i,j)$). Similar to the undiscounted hitting time, this can be written as the average of the hitting times of its neighbors to j.

$$h_\alpha(i,j) = \begin{cases} 1 + (1-\alpha)\sum_k P(i,k) h_\alpha(k,j) & \text{when } i \neq j \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

The maximum value of this quantity is $1/\alpha$, which happens when node j is never hit.

PERSONALIZED PAGERANK AND CLUSTERING. The relationship between random walks and good clusters has been exploited in many machine learning algorithms (Meila

and Shi [2001]). The main intuition is that, if a cluster is tightly knit, then a random walk started from a node in it tends to stay inside the cluster. Spielman and Teng [2004] formalize this relationship to produce good quality local clusters from a given node. The quality of a cluster can be measured by a number of well-known metrics. We will consider the conductance of a cluster (eq. 2.17), which is also used by Spielman and Teng [2004] and Andersen et al. [2006a].

In an undirected graph, the conductance of a subset of nodes S (i.e. a cluster), is defined as the ratio of the cross-edges to the total number of edges in S (or $V \setminus S$, if that has smaller number of edges than S). The smaller the conductance, the better is the cluster quality. If S is disconnected from the rest of the graph, its conductance is the minimum, i.e. 0. On the other hand, the worst conductance score can be 1 if S has no intra-cluster edges. The conductance of a graph is defined as the minimum conductance of all subsets S.

Let v_S is a probability distribution vector such that

$$v_S(i) = \begin{cases} d(i) / \sum_{i \in S} d(i) & \text{If } i \in S \\ 0 & \text{Otherwise} \end{cases} \tag{5.3}$$

Spielman and Teng [2004] show that, in a t step random walk starting from a random node (picked using the distribution 5.3) from inside a cluster S, the probability that it will end up outside S can be upper bounded by $t\Phi_V(S)$. This is very useful since we want to find clusters so that a random walk started from one stays mostly inside the cluster.

Spielman and Teng [2004] propose algorithm "Nibble", where the main idea is to start random walks from a number of seed nodes, and truncate the small probabilities at any step to obtain sparse representations of the distributions. The algorithm is guaranteed to return a local cluster with small conductance with high probability. The runtime is nearly linear in the size of the cluster it outputs. Andersen et al. [2006a] show that a local cut computed from personalized pagerank from a seed node leads to improved theoretical guarantees on the runtime and approximation quality of "Nibble".

5.3 Proposed Work

There are two main problems with nearest neighbor computation in large real world networks. First, most local algorithms for computing nearest neighbors suffer from the presence of high degree nodes. In section 5.3.1, we propose a solution that converts high degree nodes to sinks. This effectively stops a random walk once it hits a high degree node, thus preventing a possible blow-up in the neighborhood-size in local algorithms. Our results imply that in power law graphs, this does not affect the proximity measures significantly.

The second issue is that of computing proximity measures on large disk-resident graphs. While there are existing external-memory algorithms for computing random walks in large disk-resident graphs (Fogaras et al. [2004], Sarlós et al. [2006]), most of these store sketches aimed to compute one particular measure. Streaming algorithms (Sarma et al. [2008]) on the other hand require multiple passes over the entire data. While all these algorithms use interesting theoretical properties of random walks, they do not provide a generic framework for computing *arbitrary* random walk based proximity measures on the fly. One solution would be to cluster the graph and store each partition on a disk-page. Given such a clustered representation, one may easily simulate random walks, and thus compute nearest neighbors in hitting times, pagerank, simrank etc. Instead in section 5.3.2, we propose a deterministic local algorithm to compute nearest neighbors, which is later shown in section 5.4 to reduce number of page-faults compared to random simulations.

Finding a good clustering is a well-studied problem (Andersen et al. [2006a], Spielman and Teng [2004]). Good clusters will have few cross edges, leading to self-contained random walks and less page-faults. Sometimes a good clustering can be achieved by using extra features, e.g. URLs in the web-graph. However we are unaware of a fully external memory clustering algorithm in the general setting. Building on some ideas in prior literature (Andersen et al. [2006a], Sarlós et al. [2006]), we present an external memory clustering algorithm in section 5.3.3.

5.3.1 Effect of High Degree Nodes

Local algorithms estimate the similarity between the query node and others by examining local neighborhoods around the query node. These mostly rely on dynamic programming or power iteration like techniques which involve updating a node's value by combining that of its neighbors. As a result whenever a high degree node is encountered these algorithms have to examine a much larger neighborhood leading to performance bottlenecks. Sarlós et al. [2006] use rounding in order to obtain sparse representations of personalized pagerank and simrank. However before rounding, the simrank/pagrank vectors can become very dense owing to the high degree nodes. Berkhin [2006] and Chakrabarti [2007] maintain a priority queue to store the active neighborhood of the query node. Every time a high degree node is visited, all its neighbors need to be enqueued, thus slowing both algorithms down. Because of the power-law degree distribution such high degree nodes often exist in real world networks. Although there are only a few of them, due to the small-world property these nodes are easily reachable from other nodes, and they are often encountered in random walks. We will discuss the effect of high degree nodes on two proximity measures, personalized pagerank and discounted hitting times. Our analysis of the effect of degree on hitting time, and personalized pagerank is for the case of undirected graphs, although the main theorems, i.e. 5.3.1 and 5.3.5 hold for any graph.

EFFECT ON PERSONALIZED PAGERANK. The main intuition behind this analysis is that a very high degree node passes on a small fraction of its probability mass to the out-neighbors, which might not be significant enough to invest our computing resources. We argue that stopping a random walk at a high degree node does not change the personalized pagerank value at other nodes which have relatively smaller degree. First we show that the error incurred in personalized pagerank is inversely proportional to the degree of the sink node. Next we analyze the error for introducing a set of sink nodes. We turn a high degree node into a sink by removing all the outgoing neighbors and adding one self-loop with probability one, to have a well-defined probability transition matrix P. We do not change any incoming edges.

We denote by $PPV(\boldsymbol{r}, j)$ the personalized pagerank value at node j w.r.t start distribution $\boldsymbol{r}$, and $PPV(i, j)$ denotes ppv value at node j w.r.t a random walk started at node i. Let $\widehat{PPV}$ be the personalized pagerank w.r.t. start distribution r on the changed transition matrix.

Theorem 5.3.1. *In a graph G, if a node s is changed into a sink, then for any node $i \neq s$, the personalized pagerank in the new graph w.r.t start distribution $\boldsymbol{r}$ can be written as:*

$$\boxed{\widehat{PPV}(\boldsymbol{r}, i) = PPV(\boldsymbol{r}, i) - PPV(s, i)\frac{PPV(\boldsymbol{r}, s)}{PPV(s, s)}}$$

Given theorem 5.3.1 we will prove that if degree of s is much higher that than of i, then the error will be small. In order to do this we would need to examine the quantity $PPV(i, j)/PPV(j, j)$. Define the first occurrence probability $fa_\alpha(i, j)$. Consider a random walk which stops if it hits node j; if j is not hit, it stops with probability α. $fa_\alpha(i, j)$ is simply the probability of hitting a node j for the first time from node i, in this α-discounted walk. This is defined as:

$$fa_\alpha(i, j) = \begin{cases} (1 - \alpha) \sum_k P(i, k) fa_\alpha(j, k) & \text{when } i \neq j \\ 1 & \text{otherwise} \end{cases} \tag{5.4}$$

Lemma 5.3.2. *The personalized pagerank from node i to node j can be expressed as:*

$$PPV(i, j) = fa_\alpha(i, j) \times PPV(j, j)$$

Proof Sketch. Jeh and Widom [2002a] and Fogaras et al. [2004] prove that the personalized pagerank from node i to node j is simply the probability that a length ℓ path from i will end in j, where ℓ is chosen from a geometric distribution with probability $P(\ell = t) = \alpha(1 - \alpha)^{t-1}$. Note that these paths can have multiple occurrences of j. The main intuition is that, if we condition on the first occurrence of j, then $PPV(i, j)$ would

simply be probability of hitting j for the first time in an α discounted random walk, times the personalized pagerank from j to itself.

For the proof, look at appendix A.C. □

We use $d(s)$ and d_s interchangeably to denote weighted degree of s.

Lemma 5.3.3. *The error introduced at node $i \neq s$ by converting node s into a sink can be upper bounded by $\frac{d_i}{d_s}$.*

Proof Sketch. Note that by linearity of personalized pagerank vectors, we have:

$$PPV(\boldsymbol{r}, i) = \sum_k \boldsymbol{r}_k PPV(k, i)$$

Also as shown in appendix A.C,

$$PPV(s, i) \leq \frac{d_i PPV(i, s)}{d_s} \leq \frac{d_i}{d_s}$$

Now, the above statement can be proved by combining linearity with

$$PPV(k, s)/PPV(s, s) = fa_\alpha(k, s) \leq 1$$

(from lemma 5.3.2). □

Hence if d_s is much larger than d_i, then this error is small. Now we will present the error for converting a set of nodes S to a sink. The first step is to show that the error incurred by turning a number of high degree nodes into sinks is upper bounded by the sum of their individual errors. This expression can again be simplified to the result in the following lemma.

Lemma 5.3.4. *If we convert all nodes in set $S = \{s_1, s_2, ..., s_k\}$ into sinks, then the error introduced at node $i \notin S$ can be upper bounded by $\dfrac{d_i}{\min_{s \in S} d_s}$.*

The proofs for theorem 5.3.1 and lemma 5.3.4 can be found in appendix A.C.

In real world networks, the degree distribution often follows a power law, i.e. there are relatively fewer nodes with very large degree. And also most nodes have very low degree *relative* to these nodes. Hence we can make a few nodes into sinks and gain a lot of computational efficiency without losing much accuracy. We would like to mention that although the degree in the analysis is the weighted degree of a node, for experimental purposes we remove nodes with a large number of neighbors and not the weighted degree. This is because, the nodes with large un-weighted degrees are responsible for the computational inefficiency. Also, in our graphs the weights on the edges of a node are more or

less homogeneous, and hence we expect the distribution of the weighted and un-weighted degrees to be roughly the same.

EFFECT ON HITTING TIME. In order to see the effect of turning high degree nodes into sinks on discounted hitting times, we introduce the following result in this chapter.

Theorem 5.3.5. *The α-discounted hitting time $h_\alpha(i,j)$ is related to personalized pagerank by:*

$$\boldsymbol{h}_\alpha(i,j) = \frac{1}{\alpha}\left[1 - \frac{PPV(i,j)}{PPV(j,j)}\right]$$

Proof Sketch. First we show that $h_\alpha(i,j) = \frac{1}{\alpha}(1 - fa_\alpha(i,j))$. This can be easily verified by substituting this in eq. (5.2) for hitting time. This combined with lemma 5.3.2 gives the result. □

In this section we will only show the effect of deleting one high degree node on hitting time. The effect of removing a set of high degree nodes follows from the analysis of the last section. We denote by $\hat{h}_\alpha(i,j)$ the hitting time after we turn node s into a sink.

By combining theorems 5.3.1 and 5.3.5,we can upper and lower bound the change in hitting time, i.e. $\triangle h_\alpha(i,j) = \hat{h}_\alpha(i,j) - h_\alpha(i,j)$. A little algebra gives,

$$\begin{aligned}\alpha\triangle h_\alpha(i,j) &= \frac{\triangle PPV(i,j)}{PPV(j,j) - \triangle PPV(j,j)} - \frac{PPV(i,j)\triangle PPV(j,j)}{PPV(j,j)(PPV(j,j) - \triangle PPV(j,j))} \\ &= \frac{fa_\alpha(i,s)PPV(s,j)}{PPV(j,j) - \triangle PPV(j,j)} - \frac{fa_\alpha(i,j)fa_\alpha(j,s)PPV(s,j)}{PPV(j,j) - \triangle PPV(j,j)}\end{aligned} \tag{5.5}$$

When $\triangle PPV(j,j) < PPV(j,j)$, the first and second terms give the following lower and upper bounds on $\triangle h(i,j)$. Now note from lemma 5.3.2, that the upper bound on $\triangle PPV(j,j)$ is given by d_j/d_s, and we also have, $PPV(j,j) \geq \alpha$. So, if we have $d_j/d_s < \alpha$, then $\triangle PPV(j,j) < PPV(j,j)$ is satisfied.

$$\alpha\triangle h_\alpha(i,j) \leq \frac{PPV(s,j)}{PPV(j,j) - d_j/d_s} fa_\alpha(i,s) \leq \frac{d_j}{\alpha d_s - d_j}$$

Similarly,

$$\alpha\triangle h_\alpha(i,j) \geq \frac{-PPV(s,j)}{PPV(j,j) - d_j/d_s} fa_\alpha(i,j)fa_\alpha(j,s) \geq \frac{-d_j}{\alpha d_s - d_j}$$

Lemma 5.3.6. *If $d_j/d_s < \alpha$, then*

$$h_\alpha(i,j) - \frac{d_j}{\alpha(\alpha d_s - d_j)} \leq \hat{h}_\alpha(i,j) \leq h_\alpha(i,j) + \frac{d_j}{\alpha(\alpha d_s - d_j)}$$

Note that, if d_j/d_s is much smaller than α, then $\triangle h_\alpha(i,j) \approx \frac{d_j}{\alpha^2 d_s}$. This is generally the case for very large d_s, since α is usually set to be 0.1 or 0.2. This implies that turning a very high degree node into sink has a small effect on hitting time.

In this section we have given theoretical justification for changing the very-high degree nodes into sinks. We have shown its effects on two well known random walk based proximity measures. In the next two sections we would demonstrate algorithms to compute nearest neighbors on a clustered graph representation, and an external memory algorithm to compute clustering.

5.3.2 Nearest-neighbors in Clustered Graphs

Given a clustered representation, one can easily simulate random walks from a node, to obtain nearest neighbors in different proximity measures. While simulations are computationally cheap, they have a lot of variation, and for some real-world graphs they often lead to many page-faults, owing to the absence of well-defined clusters. In this section, we discuss how to use the clusters for deterministic computation of nodes "close" to an arbitrary query. As the measure of "closeness" from i, we pick the degree-normalized personalized pagerank, i.e. $PPV(i,j)/d_j$. d_j is the weighted degree of node j. This is a truly personalized measure, in the sense that a popular node gets a high score only if it has a very high personalized pagerank value. We will use the degree-normalized pagerank as a proximity measure for link prediction in our experiments as well.

We want to compute nearest neighbors in $PPV(i,j)/d_j$ from a node i. For an undirected graph, we have $PPV(i,j)/d_j = PPV(j,i)/d_i$. Hence it is equivalent to computing nearest neighbors in personalized pagerank to a node i. For an un-directed graph we can easily change these bounds to compute nearest neighbors in personalized pagerank *from* a node. For computing personalized pagerank *to a node*, we will make use of the dynamic programming technique introduced by Jeh and Widom [2002a] and further developed for computing sparse personalized pagerank vectors by Sarlós et al. [2006]. For a given node i, the PPV from j to it, i.e. $PPV(j,i)$ can be written as

$$PPV^t(j,i) = \alpha\delta(i) + (1-\alpha)\sum_{k \in nbs(j)} PPV^{t-1}(k,i)$$

Now let us assume that j and i are in the same cluster S. Hence the same equation becomes

$$PPV^t(j,i) = \alpha\delta(i) + (1-\alpha)\left[\sum_{k\in nbs(j)\cap S} P(j,k)PPV^{t-1}(k,i) + \sum_{k\in nbs(j)\cap \bar{S}} P(j,k)PPV^{t-1}(k,i)\right]$$

Since we do not have access to $PPV^{t-1}(k), k \notin S$, we will replace it with upper and lower bounds. The lower bound is simply zero, i.e. we pretend that S is completely disconnected to the rest of the graph. A random walk from outside S has to cross the boundary of S, $\delta(S)$ to hit node i. Hence $PPV(k,i) = \sum_{m\in\delta(S)} Pr_\alpha(X_m|X_{\delta(S)})PPV(m,i)$, where X_m denotes the event that *Random walk hits node m before any other boundary node for the first time*, and the event $X_{\delta(S)}$ denotes the event that the *random walk hits the boundary* $\delta(S)$. Since this is a convex sum over personalized pagerank values from the boundary nodes, this is upper bounded by $\max_{m\in\delta(S)} PPV(m,i)$. Hence we have the upper and lower bounds as follows:

$$lb^t(j,i) = \alpha\delta(i) + (1-\alpha)\sum_{k\in nbs(j)\cap S} lb^{t-1}(k,i)$$

$$ub^t(j,i) = \alpha\delta(i) + (1-\alpha)\left[\sum_{k\in nbs(j)\cap S} P(j,k)ub^{t-1}(k,i) + \{1 - \sum_{k\in nbs(j)\cap S} P(j,k)\}\max_{m\in\delta(S)} ub^{t-1}(m,i)\right]$$

Since S is small in size, the power method suffices for computing these bounds; one could also use rounding methods introduced by Sarlós et al. [2006] for computing the lower bounds. At each iteration, we maintain the upper and lower bounds for nodes within S, and the global upper bound $\max_{m\in\delta(S)} ub^{t-1}(m,i)$. In order to expand S we bring in the clusters for x of the *external* neighbors of $\text{argmax}_{m\in\delta(S)} ub^{t-1}(m,i)$. Once this *global upper bound* falls below a pre-specified small threshold γ, we use these bounds to compute approximate k closest neighbors in degree-normalized personalized pagerank.

The ranking step to obtain top k nodes using upper and lower bounds is simple: we return all nodes which have lower bound greater than the $(k+1)^{th}$ largest upper bound (when $k=1$, k^{th} largest is the largest probability). We denote this as ub_{k+1}. Since all nodes outside the cluster are guaranteed to have personalized pagerank smaller than the global upper bound, which in turn is smaller than γ, we know that the true $(k+1)^{th}$ largest probability will be smaller than ub_{k+1}. Hence any node with lower bound greater than ub_{k+1} is guaranteed to be greater than the $(k+1)^{th}$ largest probability. We use an additive

slack, e.g. $(ub_{k+1} - \epsilon)$ in order to return the top k *approximately* large ppv nodes. The reason for using an additive slack is that, for larger values of ub_{k+1}, this behaves like a small relative error, whereas for small ub_{k+1} values it allows a large relative slack, which is useful since we do not want to spend energy on the tail of the rank list anyways. In our algorithm, we initialize γ with 0.1 and keep decreasing it until the bounds are tight enough to return k largest nodes. Note that one could rank the probabilities using the lower bounds, and return top k of those after expanding the cluster a fixed number of times. This translates to a larger approximation slack.

Consider the case where we want to use this algorithm on a graph with high degree nodes converted into sinks. Since the altered graph is not undirected anymore, the ordering obtained from the degree normalized PPV from node i can be different from the ordering using PPV *to node* i. But using our error bounds from section 5.3.1 we can easily show that if the difference between two personalized pagerank values $\widehat{PPV}(a,i)$ and $\widehat{PPV}(b,i)$ is larger than $\frac{2d_i}{\min_{s_i \in S} d(s_i)}$, then a will have larger degree-normalized PPV value than b, i.e. $\widehat{PPV}(i,a)/d_a \geq \widehat{PPV}(i,b)/d_b$.

We have,

$$
\begin{aligned}
&[\widehat{PPV}(i,a)/d_a - \widehat{PPV}(i,b)/d_b] \\
&= \frac{PPV(i,a)}{d_a} - \frac{PPV(i,b)}{d_b} - \frac{\triangle PPV(i,a)}{d_a} + \frac{\triangle PPV(i,b)}{d_b} \\
&= \frac{\widehat{PPV}(a,i) + \triangle PPV(a,i)}{d_i} - \frac{\widehat{PPV}(b,i) + \triangle PPV(b,i)}{d_i} - \frac{\triangle PPV(i,a)}{d_a} + \frac{\triangle PPV(i,b)}{d_b} \\
&\geq \frac{\widehat{PPV}(a,i) - \widehat{PPV}(b,i)}{d_i} - \frac{\sum_{j=1}^{k} PPV(s_j,i) fa_\alpha(b,s_j)}{d_i} - \frac{\sum_{j=1}^{k} PPV(s_j,a) fa_\alpha(i,s_j)}{d_a} \\
&\geq \frac{\widehat{PPV}(a,i) - \widehat{PPV}(b,i)}{d_i} - \sum_{j=1}^{k} \frac{PPV(i,s_j)}{d_{s_j}} - \sum_{j=1}^{k} \frac{PPV(a,s_j)}{d_{s_j}} \\
&\geq \frac{\widehat{PPV}(a,i) - \widehat{PPV}(b,i)}{d_i} - \frac{2}{\min_{s_j \in S} d(s_j)}
\end{aligned}
$$

Without loss of generality, let us assume that $\widehat{PPV}(a,i) \geq \widehat{PPV}(b,i)$. Now the above expression says that, the ordering using degree-normalized PPV will be preserved if this difference is larger than the quantity $\frac{2d_i}{\min_{s_i \in S} d(s_i)}$.

Given that the networks follow a power law degree distribution, the minimum degree of the nodes made into sinks is considerably larger than d_i for most i, we see that the pairs which had a considerable gap in their original values should still have the same ordering. Note that for high degree nodes the ordering will have more error. However because of the

expander like growth of the neighborhood of a high degree node, most nodes are far away from it leading to an uninteresting set of nearest neighbors anyways.

5.3.3 Clustered Representation on Disk

So far we have discussed how to use a given clustered representation for computing nearest neighbors efficiently. Now we will present an algorithm to generate such a representation on disk. The intuition behind this representation is to use a set of anchor nodes and assign each remaining node to its "closest" anchor. Since personalized page-rank has been shown to yield good quality clusters (Andersen et al. [2006a]), we use it as the measure of "closeness". Our algorithm starts with a random set of anchors, and compute personalized pagerank from them to the remaining nodes. Since all nodes might not be reached from this set of anchors, we iteratively add new anchors from the set of unreachable nodes, and the recompute the cluster assignments. Thus our clustering satisfies two properties: new anchors are far away from the existing anchors, and when the algorithm terminates, each node i is guaranteed to be assigned to its closest anchor. Even though the anchors are chosen randomly, this should not affect the clustering significantly, because any node within a tight cluster can serve as the anchor for that cluster.

While clustering can be done based on personalized pagerank *from* or *to* a set of anchor nodes, one is not known to be better or worse than the other a priori. We use personalized pagerank *from the anchor nodes* as the measure of closeness. Sarlós et al. [2006] have presented a semi-external memory algorithm to compute personalized pagerank to a set of nodes. While the authors mention that their algorithm can be implemented purely via external memory manipulations, we close the loop via external memory computation of personalized pagerank from a set of anchor nodes (algorithm RWDISK). While we also use the idea of rounding introduced by the authors, the analysis of approximation error is different from their analysis. We provide proof sketches for the main results in this section, details of the proof can be found in appendix A.C.

We first present the basic algorithm, which we name *RWDISK-simple*. We demonstrate *RWDISK-simple* on a toy line graph. Then we sketch the scenarios where this simple algorithm will not work, and present the improved version, namely *RWDISK*.

ALGORITHM RWDISK-SIMPLE.

Algorithm *RWDISK-simple* computes PPV values from a set of anchor nodes based solely on passes on input files. In the subsequent sections, we will describe the full algorithm. First we will show how to compute x_t and v_t by doing simple join operations on files. We will demonstrate by computing PPV from node b in a line graph of 4 nodes (Figure 5.1). The RWDISK algorithm will sequentially read and write from four kinds of file. We will first introduce some notation. X_t denotes a random variable, which represents the

node the random walk is visiting at time t. $P(X_0 = a)$ is the probability that the random walk started at node a.

- The *Edges* file remains constant and contains all the edges in the graph and the transition probabilities. Treating *Edges* as an ASCII file with one line per edge, each line is a triplet $\{\texttt{sourcenode}, \texttt{destnode}, p\}$, where `sourcenode` and `destnode` are strings representing nodes in the graph, and $p = P(X_t = \texttt{destnode} | X_{t-1} = \texttt{sourcenode})$. Number of lines in *Edges* equals number of edges in the graph. *Edges* is sorted lexicographically by sourcenode.

- At iteration t, the *Last* file contains the values for x_{t-1}. Thus each line in *Last* is $\{\texttt{source}, \texttt{anchor}, \texttt{value}\}$, where `value` equals $P(X_{t-1} = \texttt{source} | X_0 = \texttt{anchor})$. *Last* is sorted by source.

- At iteration t of the algorithm, the file *Newt* will contain the values for x_t, i.e. each line is $\{\texttt{source}, \texttt{anchor}, \texttt{value}\}$, where `value` equals $P(X_t = \texttt{source} | X_0 = \texttt{anchor})$. *Newt*, once construction is finished, will also be sorted by source. Needless to say, the *Newt* of iteration t becomes the *Last* of iteration $t + 1$.

- The final file is *Ans*. At iteration t of the algorithm, the file *Ans* represents the values for v_t. Thus each line in *Ans* is $\{\texttt{source}, \texttt{anchor}, \texttt{value}\}$, where $\texttt{value} = \sum_{n=1}^{t} \alpha(1-\alpha)^{n-1} x_{n-1}(j)$. *Ans*, once construction is finished, will also be sorted by source.

We will now confirm that these files can be derived from each other by a series of merges and sorts. In step 1 of Algorithm RWDISK, using the simple 4-node graph in figure 5.1 with b as the sole anchor, we initialize *Last* as the single line $b, b, 1.0$, and *Ans* with the single line b, b, α.

In the next step, we compute the *Newt* file from *Last* and *Edges* by a disk-based matrix multiplication, which is a simple join operation of two files. Both *Last* and *Newt* sum to 1, since these are the occupancy probabilities at two consecutive timesteps. Once we have *Newt*, we multiply the probabilities by $\alpha(1-\alpha)^{t-1}$ (the probability that a random walk will stop at timestep t, if at any step the probability of stopping is α) and accumulate the values into the previous *Ans* file. Now the *Newt* file is renamed to *Last* file. Figure 5.2 is the same process on the new *Last*, i.e. old *Newt* file.

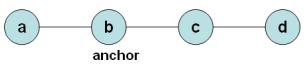
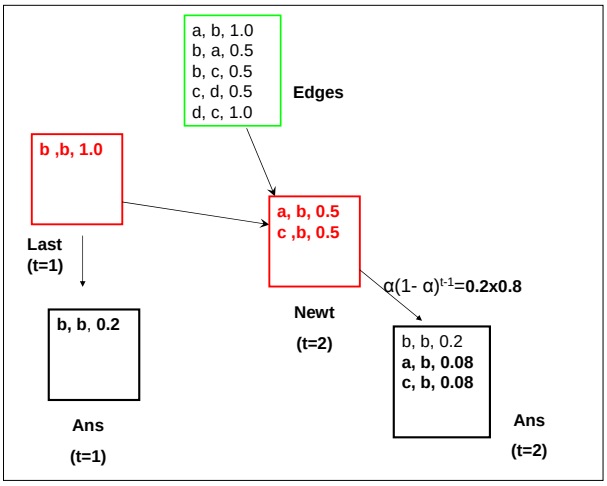
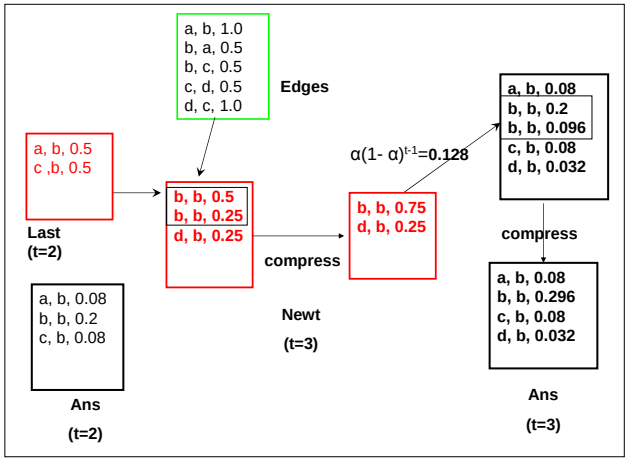


Figure 5.1: Line Graph



(A)



(B)

Figure 5.2: A. shows the first step of our algorithm on a line graph, and B. shows the second step. The input file (*Edges*) is in green, the intermediate files (*Last*, *Newt*) are in red and the output file (*Ans*) is in black.

Now let us look at some details. For any node y it can appear many times in *Newt*, since the values in *Last* from different in-neighbors of y accumulate in *Newt*. For example in the *Newt* file at timestep 2, probability mass accumulates in b from both its incoming neighbors $\{c,d\}$. Hence we need to sort and compress *Newt*, in order to add all the different values. Compressing is the simple process where sets of consecutive lines in the sorted file that have the same source and anchor (and each has its own value) are replaced by one line containing the given source and anchor, and the sum of the values. Each of sorting

and compression can happen with $O(n)$ sequential operations through the file (the former by means of bucket sort).

ALGORITHM RWDISK.

The problem with the previous algorithm is that some of the intermediate files can become very large: much larger than the number of edges. Let N and E be the total number of nodes and edges in the graph. Let d be the average outdegree of a node. In most real-world networks within 4-5 steps, it is possible to reach a huge fraction of the whole graph. Hence for any anchor the *Last*file might have N lines, if all nodes are reachable from that anchor. Therefore the *Last*file can have at most $O(A\mathrm{N})$ lines. Our ultimate goal is to create a pagesize cluster (partition) for each anchor. If roughly n_a nodes can fit per page, then we would need N/n_a anchors. Hence the naive file join approach will lead to intermediate files of size $O(\mathrm{N}^2/n_a)$. Since these files are also sorted in every iteration, this will greatly affect both the runtime and disk-storage.

Rounding for reducing file sizes: We address this serious problem by means of rounding tricks. In the step from *Newt* to *Last* at timestep t, we round all values below ϵ_t to zero. Elements of a sparse rounded probability density vector sums to at most one. By rounding we only store the entries bigger than ϵ_{t-1}. Hence the total number of nonzero entries can be at most $1/\epsilon_{t-1}$. In fact, since the probability of stopping decreases by a factor of $(1-\alpha)^t$ with t, we gradually increase ϵ_t, leading to sparser and sparser solutions.

As *Last* stores rounded x_{t-1} values for A anchors, its length can never exceed A/ϵ_{t-1}. Since *Newt* is obtained by spreading the probability mass along the out-neighbors of each node in *Last*, its total length can be roughly $A \times d/\epsilon_{t-1}$, where d is the average out-degree. For $A = \mathrm{N}/n_a$ anchors, this value is $(\mathrm{N}/n_a) \times d/\epsilon_{t-1} = \mathrm{E}/(n_a \times \epsilon_{t-1})$. Without rounding this length would have been N^2/n_a. The effect of introducing rounding on the intermediate file sizes is shown in Table 5.2.

In Table 5.1, we present this rounding algorithm using only sequential scans of datafiles on disk. At one iteration of *CreateNewtFile*, for each outneighbor `dst` of node `src` appearing in *Last* file, we append a line {`dst,anchor,prob × val`} to the *Newt* file. Since the length of *Last* is no more than A/ϵ_{t-1}, the total number of lines before sorting and compressing *Newt* can be $A \times d/\epsilon_{t-1}$. We avoid sorting a huge internal *Edges* every iteration by updating the *Newt* in a *lazy* fashion. We maintain a sorted *Newt* and append the triplets to a temporary buffer Tmp. Every time the number of lines exceed $B = 2A/\epsilon_{t-1}$, we sort the buffer, and merge it with the already sorted *Newt* file. Since the total number of lines in *Newt* can be as big as $A \times d/\epsilon_{t-1}$, this update happens roughly $d/2$ times, where d is the average out-degree.

The *update* function reads a triplet {`source,anchor,val`} from *Newt* and appends it after adjusting `val` by a factor of $\alpha(1-\alpha)^{iter-1}$ to *Ans*. Since *Ans* is not needed to

Table 5.1: Pseudocode for RWDISK

RWDISK(*Edges*, ϵ, α, Anchors)

1. Initialize: For each anchor $a \in$ *Anchors*,
   (a) Append to each empty file *Last* the line $a, 1.0$.
   (b) Append to each empty file *Ans* the line a, α.
2. $i = 1$.
3. Create Newt files from a pass over the *Last* and *Edges* files.
   Newt← **CreateNewtFile(*Last*,*Edges*)**.
4. If the *Newt* file is empty or $i >$ `maxiter`, return *Ans* after sorting and compressing it.
5. Update the *Ans* file.
   Ans← **Update(*Ans*,*Newt*,α,$i+1$)**.
6. Round values in *Newt* files and copy to *Last* files.
   Last← **Round(*Newt*,ϵ)**.
7. $\epsilon = \epsilon/\sqrt{1-\alpha}$
8. Increment counter i. Go to step 3.

CreateNewtFile(*Last*,*Edges*)

- Read triplets $\{src,anchor,val\}$ from *Last*with common prefix "src"
- Read triplets $\{src,dst,prob\}$ from *Edges*with common prefix "src"
- Append to a temporary buffer Tmp the triplets $\{dst,anchor,prob \times val\}$
- If number of lines in $Tmp \geq B$ sort and compress Tmp, and merge with already sorted file *Newt*

generate the intermediate probability distributions, we do not sort and compress it in every iteration. We only do so once at the end. How large can the *Ans* file become? Since each *Newt* can be as large as $A \times d/\epsilon_{t-1}$, and we iterate for $maxiter$ times, the size of *Ans* can be roughly as big as `maxiter` $\times A \times d/\epsilon_{t-1}$. Since we increase ϵ_t every iteration, $\epsilon_t \geq \epsilon, \forall t$. Here are the file-sizes obtained from RWDISK in a nutshell in Table 5.2.

APPROXIMATION ERROR OF RWDISK.

Table 5.2: The effect of introducing rounding

Algorithm	# anchors	*Last* Size	*Newt* Size	*Ans* Size
No-rounding	$\frac{\mathrm{N}}{n_a}$	$O(\frac{\mathrm{N}^2}{n_a})$	$O(\frac{\mathrm{N}^2}{n_a})$	$O(\frac{\mathrm{N}^2}{n_a})$
Rounding	$\frac{\mathrm{N}}{n_a}$	$O(\frac{\mathrm{N}}{\epsilon \times n_a})$	$O(\frac{\mathrm{E}}{\epsilon \times n_a})$	$O(\frac{\texttt{maxiter} \times \mathrm{E}}{\epsilon \times n_a})$

We present a proof sketch. The proofs of each lemma/theorem can be found in appendix A.C.

Error from rounding: The rounding step saves vast amounts of time and intermediate disk space, but how much error does it introduce? In this section, we will describe the effect of rounding on computing the degree-normalized personalized pagerank values. First we will bound the error in personalized pagerank. $\Psi_\epsilon(x)$ is an operation from a vector to a vector which rounds all entries of x below ϵ to 0. Thus we have,

$$\Psi_\epsilon(x[j]) = \begin{cases} 0 & \text{If } x[j] \leq \epsilon \\ x[j] & \text{Otherwise} \end{cases}$$

We will denote $\widehat{\boldsymbol{x_t}}$ as the approximated $\boldsymbol{x_t}$ distribution from the rounding algorithm at iteration t. We want to have a bound on $\boldsymbol{v} - \hat{\boldsymbol{v}}$, where $\boldsymbol{v}$ is the exact personalized pagerank (eq. 5.1). In order to do that, we will first bound $\boldsymbol{x_t} - \widehat{\boldsymbol{x_t}}$, and then combine these errors to get the bound on the final error vector. We will denote $\bar{\boldsymbol{\epsilon}}_t$ as a vector of errors accumulated at different nodes by rounding at time t. Note that $\bar{\boldsymbol{\epsilon}}_t$ is strictly less than the probability vector $\boldsymbol{x_t}$. Also note that $\widehat{\boldsymbol{x_t}}$ is strictly smaller than $\boldsymbol{x_t}$. . Hence we can prove the following recursive equation.

Lemma 5.3.7. *We have $\widehat{\boldsymbol{x_t}} = \Psi_{\epsilon_t}(P^T\widehat{\boldsymbol{x_{t-1}}})$. Let $\bar{\boldsymbol{\epsilon}}_t$ equal $P^T\widehat{\boldsymbol{x_{t-1}}} - \widehat{\boldsymbol{x_t}}$. At any node i we have $0 \leq \bar{\boldsymbol{\epsilon}}_t(i) \leq xI(x \leq \epsilon_t) \leq \epsilon_t$, where $I(.)$ is the indicator function, and $\boldsymbol{x} = P^T\widehat{\boldsymbol{x_{t-1}}}(i)$. Then we can prove $\boldsymbol{x_t} - \widehat{\boldsymbol{x_t}} \leq \bar{\boldsymbol{\epsilon}}_t + P^T(\boldsymbol{x_{t-1}} - \widehat{\boldsymbol{x_{t-1}}})$.*

We will use this lemma to bound the total error accumulated up to time t in the probability vector $\boldsymbol{x_t}$.

Lemma 5.3.8. *Let $\boldsymbol{E_t}$ denote $\boldsymbol{x_t} - \widehat{\boldsymbol{x_t}}$, the vector of errors accumulated up to timestep t. $\boldsymbol{E_t}$ has only non-negative entries. We have $\boldsymbol{E_t} \leq \sum_{r=1}^{t}(P^T)^{t-r}\bar{\boldsymbol{\epsilon}}_r$.*

The above lemma shows how the epsilon-rounding accumulates over time. From eq. (5.1) the total error incurred in the PPV value $\boldsymbol{E}$ can be bounded as follows.

Theorem 5.3.9. *Let $\boldsymbol{E}$ denote $\boldsymbol{v} - \hat{\boldsymbol{v}}$, and $PPV_\alpha(r)$ denote the personalized pagerank vector for start distribution $\boldsymbol{r}$, and restart probability α. We have $\boldsymbol{E} \leq \frac{1}{\alpha} PPV_\alpha(\bar{\boldsymbol{\epsilon}})$ where $\bar{\boldsymbol{\epsilon}}$ is a vector, with maximum entry smaller than $\frac{\alpha\epsilon}{1-\sqrt{1-\alpha}}$, and sum of entries less than 1.*

Theorem 5.3.9 indicates that our rounding scheme incurs an error in the personalized pagerank vector which is upper bounded by a constant times the stationary distribution of a α-restart walk whose start distribution is $\bar{\boldsymbol{\epsilon}}$. For an undirected graph, it can be shown (appendix A.C) that the total error at any node i can be, at most, $\frac{d(i)}{\delta} \frac{\epsilon}{1-\sqrt{1-\alpha}}$, where $d(i)$ is the weighted degree of node i, and δ is the minimum weighted degree. Since we are using *degree-normalized* personalized pagerank, the $d(i)$ at any node gets normalized leading to an error of $\frac{\epsilon}{\delta(1-\sqrt{1-\alpha})}$.

In fact, this result turns out to be very similar to that of Andersen et al. [2006a], which uses a idea similar to Berkhin [2006] for obtaining sparse representations of personalized pagerank vectors. The main difference is that RWDISK streams through the data without needing random access.

Error from early termination: Here is the error bound for terminating RWDISK early.

Theorem 5.3.10. *Let $\boldsymbol{v}_t$ be the partial sum of distributions up to time t. If we stop at $t = \texttt{maxiter}$, then the error is given by $\boldsymbol{v} - \boldsymbol{v}_{\texttt{maxiter}} = (1-\alpha)^{\texttt{maxiter}} PPV_\alpha(\boldsymbol{x}_{\texttt{maxiter}})$. where $PPV_\alpha(\boldsymbol{x}_{\texttt{maxiter}})$ is the personalized pagerank with start distribution $\boldsymbol{x}_{\texttt{maxiter}}$.*

This theorem indicates that the total error incurred by early stopping is $(1-\alpha)^{\texttt{maxiter}}$. Now we will combine theorems 5.3.9 and 5.3.10 to obtain the final approximation error guarantee.

Lemma 5.3.11. *If $\hat{\boldsymbol{v}}_{\texttt{maxiter}}$ is obtained from RWDISK with parameters ϵ, α, $\texttt{maxiter}$ and start distribution $\boldsymbol{r}$, then*

$$\boldsymbol{v} - \hat{\boldsymbol{v}}_{\texttt{maxiter}} \leq \frac{1}{\alpha} PPV_\alpha(\bar{\boldsymbol{\epsilon}}) + (1-\alpha)^{\texttt{maxiter}} PPV_\alpha(\boldsymbol{x}_{\texttt{maxiter}})$$

where $\boldsymbol{x}_{\texttt{maxiter}}$ is the probability distribution after $\texttt{maxiter}$ steps, when the start distribution is $\boldsymbol{r}$.

HIGH DEGREE NODES. In spite of rounding one problem with RWDISK is that if a node has high degree then, it has large personalized pagerank value from many anchors. As a result this node can appear in a large number of $\{\texttt{node}, \texttt{anchor}\}$ pairs in the *Last* file. After the matrix multiplication, each of these pairs now will lead to $\{\texttt{nb}, \texttt{anchor}\}$ pairs for each outgoing neighbor of the high degree node. Since we can only prune once the

entire *Newt* file is computed the size can easily blow up. This is why RWDISK benefits from turning high degree nodes into sinks as described before in section 5.3.1.

THE CLUSTERING STEP. We randomly pick about 1 percent of the nodes as anchor points, and compute personalized pagerank from them by the RWDISK algorithm. Each node is assigned to the anchor node which has the largest pagerank value to it, and this assignment defines our graph clustering. However there might be orphan nodes after one pass of the algorithm: nodes which no anchor can reach. We need every node to be placed in exactly one cluster, and so if there are orphans, we go ahead and pick another set of random anchor nodes from the orphans from step 1, and compute personalized pagerank from them by rerunning RWDISK. For any batch of anchors, we only store the information $\{$`src,closest-anchor,value`$\}$, where `src` is a node which is not an orphan. `closest-anchor` is the anchor with the maximum PPV value among *all* anchors seen so far, and `value` is the PPV value from that anchor to `src`. Now we reassign the nodes to the new anchors, in case a newly added anchor turns out to be closer to some node than its former anchor. We continue this process until there are no orphans left. The clustering satisfies two properties: 1) a new batch of anchors are far away from the existing pool of anchors, since we have picked them from nodes which have PPV value 0 from the pre-existing pool of anchors; 2) after R rounds if the set of anchors is S_R, then each node i is guaranteed to be assigned to its closest anchor, i.e. $\text{argmax}_{a \in S_R} PPV(a, i)$.

5.4 Results

We present our results in three steps: first we show the effect of high degree nodes on i) computational complexity of RWDISK, ii) page-faults in random walk simulations for an actual link prediction experiment on the clustered representation, and iii) link prediction accuracy. Second, we show the effect of deterministic algorithms for nearest-neighbor computation on reducing the total number of page-faults by fetching the right clusters. Last, we compare the usefulness of the clusters obtained from RWDISK w.r.t a popular in-memory algorithm METIS.

DATA AND SYSTEM DETAILS. We present our results on three of the largest publicly available social and citation networks (Table 5.3): a connected subgraph of the Citeseer co-authorship network, the entire DBLP corpus, and the online community Live Journal[1]. We use an undirected graph representation, although RWDISK can be used for directed graphs as well. The experiments were done on an off-the-shelf PC. We use a buffer of size 100 and the *least recently used* replacement scheme. Each time a random walk moves to a cluster not already present in the buffer, the system incurs page-faults. We use a

[1]The LiveJournal data has been obtained from "http://snap.stanford.edu/data/soc-LiveJournal1.html".

pagesize of $4KB$, which is standard in most computing environments. This size can be much larger, $1MB$ in some advanced architectures, and for those cases these results will have to be re-tuned.

Table 5.3: # nodes,directed edges in graphs

Dataset	Size of *Edges*	Nodes	Edges	Median Degree
Citeseer	$24MB$	$100K$	$700K$	4
DBLP	$283MB$	$1.4M$	$12M$	5
LiveJournal	$1.4GB$	$4.8M$	$86M$	5

5.4.1 Effect of High Degree Nodes

Turning high degree nodes into sinks have three-fold advantage: first, it drastically speeds up our external memory clustering; secondly, it reduces number of page-faults in random walk simulations done in order to rank nodes for link-prediction experiments; thirdly, it actually *improves* link-prediction accuracy.

EFFECT ON RWDISK. Table 5.4 contains the running times of RWDISK on the three

Table 5.4: For each dataset, the minimum degree, above which nodes were turned into sinks, and the total number of sink nodes, time for RWDISK.

Dataset	Sink Nodes		Time
	Min degree	Number	
Citeseer	None	0	1.3 hours
DBLP	None	0	$\geq$ 2.5 days
	1000	900	**11** hours
LiveJournal	1000	950	60 hours
	100	$134,000$	**17** hours

graph datasets. For Citeseer, RWDISK algorithm completed roughly in an hour without

introducing any sink nodes. For DBLP, without degree-deletion, the experiments ran for above 2.5 days, after which they were stopped. After turning nodes with degree higher than 1000, the time was reduced to 11 hours, a larger than 5.5 fold speedup. The Live-Journal graph is the largest and most dense of all three. After we made nodes of degree higher than 1000 into sinks, the algorithm took 60 hours, which was reduced to 17 (≥ 3 fold speedup) after removing nodes above degree 100. In Table 5.3 note that, for both DBLP and LiveJournal, the median degree is much smaller than the minimum degree of nodes converted into sink nodes. This combined with our analysis in section 5.3.1 confirms that we did achieve a huge computational gain without sacrificing the quality of approximation.

LINK PREDICTION. For link-prediction we use degree-normalized personalized pagerank as the proximity measure for predicting missing links. We pick the same set of 1000 nodes and the same set of links from each graph before and after turning the high degree nodes into sinks. For each node i we hold out $1/3^{rd}$ of its edges and report the percentage of held-out neighbors in top 10 ranked nodes in degree-normalized personalized pagerank from i. Only nodes below degree 100 and above degree 3 are candidates for link deletion, so that no sink node can ever be a candidate. From each node 50 random walks of length 20 are executed. Note that this is not AUC score; so a random prediction does much worse that 0.5 in these tasks.

From Table 5.5 we see that turning high-degree nodes into sinks not only decrease page-faults by a factor of ~ 7, it also boosts the link prediction accuracy by a factor of 4 on average. Here is the reason behind the surprising trend in link prediction scores. The

Table 5.5: Mean link-prediction accuracy and pagefaults

Dataset	Sink nodes	Accuracy	Page-faults
LiveJournal	none	0.2	1502
	degree above 100	**0.43**	**255**
DBLP	none	0.1	1881
	degree above 1000	**0.58**	**231**
Citeseer	none	0.74	69
	degree above 100	0.74	67

fact that the number of page-faults decrease after introducing sink nodes is obvious. This is because in the original graph, every time a random walk hits a high degree node, there is higher chance of incurring page-faults. We believe that the link prediction accuracy is related to quality of clusters, and transitivity of relationships in a graph. More specifi-

cally, in a *well-knit* cluster, two connected nodes do not just share one edge, they are also connected by many short paths, which makes link-prediction easy. On the other hand if a graph has a more expander-like structure, then in random-walk based proximity measures, everyone ends up being far away from everyone else. This leads to poor link prediction scores. In Table 5.5 one can catch the trend of link prediction scores from worse to better from LiveJournal to Citeseer. Our intuition about the relationship between cluster quality and predictability is reflected in Figure 5.3, where we see that LiveJournal has worse page-fault/conductance scores than DBLP, which, in turn has worse scores than Citeseer. Within each dataset, we see that turning high degree nodes into a sink generally helps link prediction, which is probably because it also improves the cluster-quality. Are all high-degree nodes harmful? In DBLP the high degree nodes without exception are common words which can confuse random walks. However the Citeseer graph only contains author-author connections, which are much less ambiguous than paper-word connections. There are not as many high degree nodes as compared to the other datasets, and the high degree nodes do not hamper link prediction accuracy very much. This might be the reason why introducing sink nodes does not change the link prediction accuracy.

5.4.2 Deterministic vs. Simulations

We present the mean and median number of pagefaults incurred by the deterministic algorithm in section 5.3.2. We have executed the algorithm for computing top 10 neighbors with approximation slack 0.005 for 500 randomly picked nodes. For Citeseer, we compute

Dataset	Mean Page-faults	Median Page-faults
LiveJournal	64	29
DBLP	54	16.5
Citeseer	6	2

Table 5.6: Page-faults for computing 10 nearest neighbors using lower and upper bounds

the nearest neighbors in the original graph, whereas for DBLP we turn nodes with degree above 1000 into sinks and for LiveJournal we turn nodes with degree above 100 into sinks. Both mean and median pagefaults decrease from LiveJournal to Citeseer, showing the increasing cluster-quality, as is evident from the previous results. The difference between mean and median reveals that for some nodes the neighborhood is explored much more in order to compute the top 10 nodes. Upon closer investigation, we find that for high degree nodes, the clusters have a lot of boundary nodes and hence the bounds are hard to tighten. Also from high degree nodes, all other nodes are more or less farther away. In

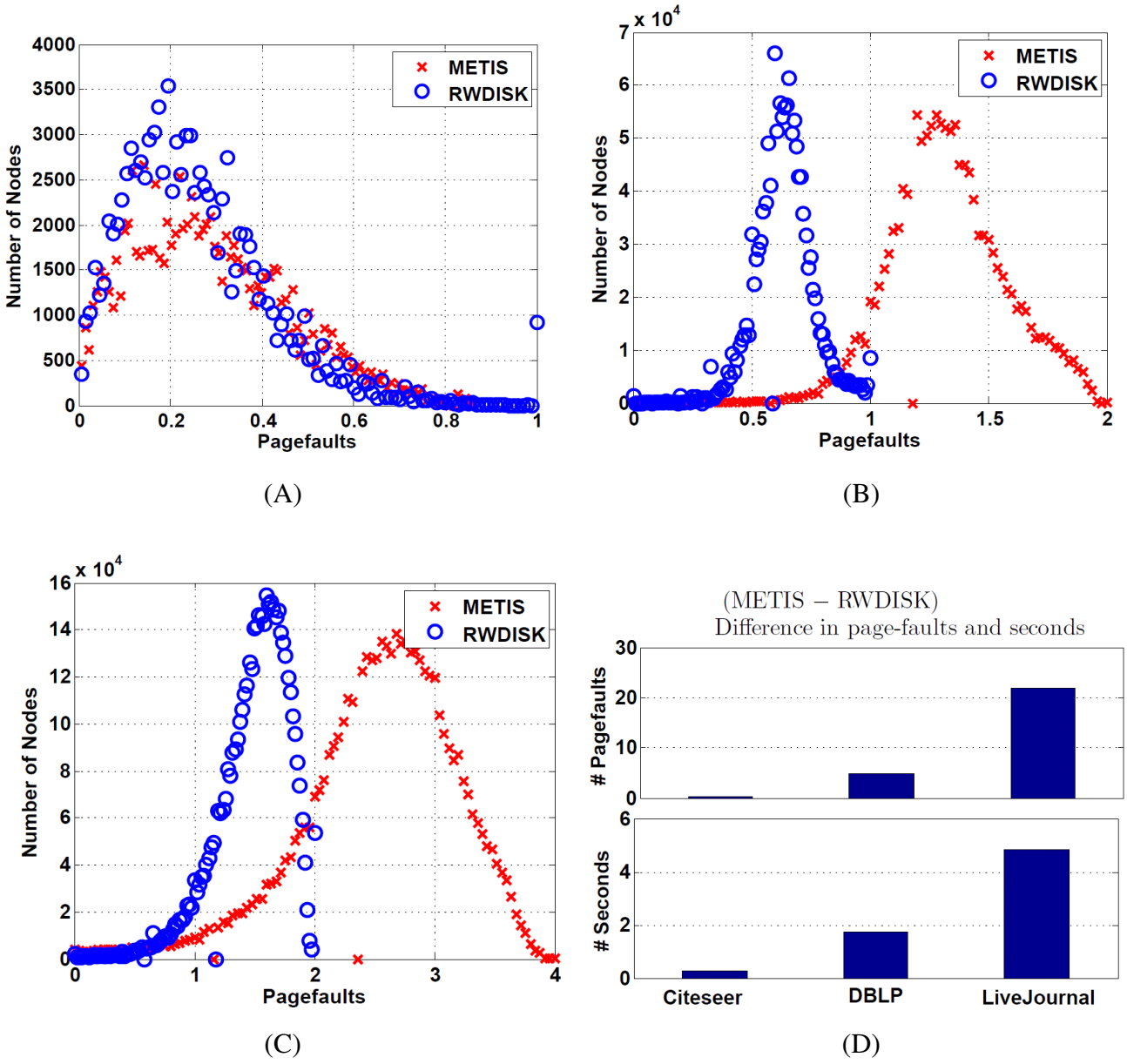


Figure 5.3: The histograms for the expected number of pagefaults if a random walk stepped outside a cluster for a randomly picked node. Left to right the panels are for Citeseer (Figure A), DBLP (Figure B) and LiveJournal (Figure C). Figure (D) has #Page-faults(METIS) − #Page-faults(RWDISK) per 20 step random walk in upper panel. Bottom Panel contains total time for simulating 50 such random walks. Both are averaged over 100 randomly picked source nodes.

contrast to random simulations (table 5.5), these results show the superiority of the deterministic algorithm over random simulations in terms of number of page-faults (roughly 5

fold improvement).

5.4.3 RWDISK vs. METIS

We use $\texttt{maxiter} = 30$, $\alpha = 0.1$ and $\epsilon = 0.001$ for PPV computation. We use PPV and RWDISK interchangeably in this section. Note that $\alpha = 0.1$ in our random-walk setting is equivalent to a restart probability of $\alpha/(2-\alpha) = 0.05$ in the lazy random walk setting of Andersen et al. [2006a].

We use METIS as a baseline algorithm (Karypis and Kumar [1998])[2], which is a state of the art *in memory* graph partitioning algorithm. We use METIS to break the DBLP graph into about $50,000$ parts; METIS requires about **20 GB** of RAM for this purpose. We also used METIS to partition LiveJournal into about $75,000$ parts; for this METIS requires about **50 GB** of RAM. Since METIS has been creating comparably larger clusters, we have tried to divide the LiveJournal graph into $100,000$ parts; however in this case the memory requirement increases to **80 GB** which is prohibitively large for us. In comparison, RWDISK can be executed on a $2-4$ GB standard computing unit. Table 5.3 contains the details of the three different graphs and Table 5.4 contains running times of RWDISK on these. Although the clusters are computed after turning high degree nodes into sinks, the comparison with METIS is done on the original graphs.

MEASURE OF CLUSTER QUALITY. A good disk-based clustering must combine two characteristics: (a) the clusters should have low conductance, and (b) they should fit in disk-sized pages. Now, the graph conductance ϕ measures the average number of times a random walk can escape outside a cluster (Spielman and Teng [2004]), and each such escape requires the loading of one new cluster, causing an average of m page-faults ($m = 1$ if each cluster fits inside one page). Thus, $\phi \cdot m$ is the average number of page-faults incurred by one step of a random walk; we use this as our overall measure of cluster quality. Note that m here is the expected size (in pages) of the cluster that a randomly picked node belongs to, and this is *not* necessarily the average number of pages per cluster.

Briefly, Figure 5.3 tells us that in a single step random walk, METIS will lead to similar number of pagefaults on Citeseer, about $1/2$ pagefaults more than RWDISK on DBLP and 1 more in $LiveJournal$. Hence in a 20 step random walk METIS will lead to about 5 more pagefaults than RWDISK on DBLP and 20 more pagefaults on LiveJournal. Note that since a new cluster can be much larger than a disk-page size, it is possible to make more than 20 pagefaults on a 20 step random walk in our paradigm. In order to demonstrate the accuracy of this measure, we simulate 50 random walks of length 20 from 100 randomly picked nodes from the three different graphs. We note the average page-faults and average time in wall-clock seconds. Figure 5.3(D) shows how many *more*

[2]The software for partitioning power law graphs has not yet been released.

pagefaults METIS incurs than RWDISK in every simulation. The wallclock seconds is the **total time** taken for all 50 simulations averaged over the 100 random nodes. These numbers exactly match our expectation from Figures 5.3(A), (B) and (C). We see that on Citeseer, METIS and RWDISK give comparable cluster qualities, but on DBLP and LiveJournal, RWDISK performs much better.

5.5 Summary and Discussion

In this chapter, we conclude our work on fast nearest neighbor search using random walk based proximity measures in large graphs. Link prediction, personalized graph search, fraud detection, and many such graph mining problems revolve around the computation of nearest neighbors of a given query node. As we show in chapter 2, a widely used class of similarity measures is based on random walks on graphs, e.g. personalized pagerank, hitting and commute times, and simrank. So far, we have looked at different components of the computational aspect of this problem. First, we present a branch and bound technique to compute nearest neighbors in truncated hitting and commute times (chapter 3); then we show how to incorporate user feedback in this framework for reranking search results (chapter 4). While these chapters address some interesting computational issues in relevance search, they also reveal two fundamental problems associated with these random walk based measures.

First, existing fast algorithms typically examine the local neighborhood of the query node which can become significantly slower whenever high-degree nodes are encountered (a common phenomenon in real-world graphs). Second, computing these similarities at query time is hard when the graph is too large to be memory-resident. This chapter introduces analysis and algorithms which try to address these issues. We take two steps. First, we identify the serious role played by high degree nodes in damaging computational complexity, and we prove that a simple transform of the graph can mitigate the damage with bounded impact on accuracy. Second, we apply the result to produce algorithms for the two components of general-purpose proximity queries on enormous graphs: algorithms to rank top-k neighbors by a broad class of random-walk based proximity measures including PPV, and a graph partitioning step to distribute graphs over a file system.

Chapters 3, 4, and 5 are focussed on algorithmic solutions to the problem of ranking with random walk based proximity measures. However, one interesting question is, why do these proximity measures perform well? Note that, random walk based measures basically examine the ensemble of paths between two nodes in a graph. There are popular heuristics which also look at paths between two nodes. For example number of common neighbors and the Adamic/Adar score look at paths of length two between two nodes. Length of the shortest path is also a common heuristic. How do we measure the goodness

of a heuristic? We will take link prediction for determining the performance of a measure. Link prediction has been extensively studied via empirical surveys. It also has been an integral part of the experimental evaluation of the previous chapters in this thesis.

There are some surprising common trends in the performances of these measures. For example, shortest path does poorly compared to other heuristics. Common neighbors does quite well, in fact sometimes, it performs almost as well as more complex measures. Adamic/Adar (see eq. 2.14), which is an weighted variant of common neighbors, generally performs better. Also, in sparse graphs, where there are not many common neighbors, measures like truncated hitting and commute times perform better than common neighbors.

Although these observations can be intuitively explained, one wonders if there is a theoretical framework to justify it. This naturally brings us to the next chapter. We provide what we believe to be the first formal connection between these intuitions and these path-based popular heuristics. We look at a familiar class of graph generation models in which nodes are associated with locations in a latent metric space and connections are more likely between a pair of nodes close in this latent space. We look at heuristics such as the number of common neighbors and its weighted variants (e.g. Adamic/Adar) which have proved successful in predicting missing links, but are not direct derivatives of latent space graph models.

Chapter 6

Theoretical Justification of Link-prediction Heuristics

You have your way. I have my way. As for the right way, the correct way, and the only way, it does not exist.

–Friedrich Nietzsche.

6.1 Introduction

In the previous chapters, we have developed an efficient framework for computing random-walk based similarities. As shown in the literature survey in chapter 2, these measures have been empirically shown to be effective for ranking nodes in a graph. Intuitively, these measures are useful, because they take into account the paths between two nodes. Essentially, the main idea is that two nodes are similar, if they are connected via many short paths in the graph. Take for example, popular similarity measures like number of common neighbors, Adamic/Adar etc. These look at paths of length two. It is also common to use the length of the shortest path as a measure of dissimilarity. There are some interesting common trends in the effectiveness of these measures in terms of ranking. Although we can make intuitive arguments to explain these, there has not been any work on formalizing these observations. This is the focus of this chapter. We consider path-based similarity measures, e.g. number of length k paths. These also include popular heuristics like the number of common neighbors ($k = 2$) and the Adamic/Adar score (weighted variant of common neighbors), etc. In order to evaluate how effective a heuristic or similarity measure is, we use link prediction. We have also used this extensively to evaluate the algorithms in this thesis.

Link prediction is a key problem in graph mining. It underlies recommendation sys-

tems (e.g., movie recommendations in Netflix, music recommendation engines like `last.fm`), friend-suggestions in social networks, market analysis, and so on. As such, it has attracted a lot of attention in recent years, and several heuristics for link prediction have been proposed Adamic and Adar [2003]. In-depth empirical studies comparing these heuristics have also been conducted Liben-Nowell and Kleinberg [2003] and Brand [2005], and some observations are made consistently: (1) a simple heuristic, viz., predicting links between pairs of nodes with the most common neighbors, often outperforms more complicated heuristics, (2) a variant of this heuristic that weights common neighbors using a carefully chosen function of their degrees Adamic and Adar [2003] performs even better on many graphs and (3) heuristics which use an ensemble of short paths between two nodes Katz [1953] often perform better than those which use longer paths. However, there has been little theoretical work on why this should be so. We present, to our knowledge, the first theoretical analysis of link prediction on graphs. We show how various heuristics compare against each other, and under what conditions would one heuristic be expected to outperform another. We are able to provide theoretical justifications for all of the empirical observations mentioned above.

We define the link prediction problem as follows. There is a latent space in which the nodes reside, and links are formed based on the (unknown) distances between nodes in this latent space. Individual differences between nodes can also be modeled with extra parameters. The quality of link prediction now depends on the quality of estimation of distance between points. We show how different estimators provide *bounds* on distance. Clearly, the tighter the bounds, the better we can distinguish between pairs of nodes, and thus the better the quality of link prediction.

While any latent space model can be used, we extend a model by Raftery et al. [2002] due to two characteristics: (1) it is simple to state and analyze, (2) yet, it is powerful enough to show all of the effects that affect estimation, such as node degree, lengths of paths, etc. Our results do not assume any degree distribution on the graph; in fact, they depend on very simple properties that should be generalizable to other models as well.

Our primary contributions are as follows:

- We formulate the link prediction problem as a problem of estimating distances between pairs of nodes, where the nodes lie at unknown positions in some latent space and the observed presence or absence of links between nodes provides clues about their distances.

- We show that the number of common neighbors between a pair of nodes gives bounds on the distance between them, with the upper bound on distance decreasing quickly as the count of common neighbors increases. This justifies the popular heuristic of predicting links simply by picking node pairs with the maximum number of common neighbors.

- Empirical studies Liben-Nowell and Kleinberg [2003] have shown that another popular heuristic Adamic and Adar [2003] that uses a carefully *weighted* count of common neighbors often outperforms the unweighted count. We present theoretical justification for this, and generalize it to other possible weighting schemes that should be similarly useful.

- Finally, another set of heuristics consider longer paths between pairs of nodes, e.g., hitting-time and other measures based on random walks. Our results here are twofold. (1) We show that while the number of long paths can, indeed, provide bounds on distance, these are looser than the bounds obtained if enough short paths (or ideally, common neighbors) exist. Thus, longer paths are more useful if shorter paths are rare or non-existent. (2) We also show that the bounds obtained from long paths can get much tighter given just the knowledge of *existence* of a short path. Thus, even the existence of a single short path can improve bounds obtained from long paths.

- Our results can be applied to any social network model where: nodes are distributed independently in some latent metric space; probability of a link satisfies *homophily*; given the positions links are independent of each other.

This chapter is organized as follows. In section 6.2 we introduce related work and background on link prediction and latent space models. Sections 6.3 and 6.4 consist of a formal relationship between popular heuristics like common neighbors and our graph model with same and distinct radii. In section 6.5, we analyze the implication of paths of length $\ell > 2$. Section 6.6 shows how to extend the analysis to handle non-determinism in the link generation process. In section 6.7, we summarize this chapter and discuss several implications of our work.

6.2 Review of Previous Empirical Studies

We describe the link prediction problem in section 2.5 of chapter 2. We quickly remind the readers of the link prediction problem, and of the results from previous empirical studies. Then we describe the latent space model we use, and conclude with the relation of this model to link prediction.

6.2.1 Link Prediction

Many real world graph-mining problems can be framed as link prediction. Popular applications include suggesting friends on Facebook, recommending movies (Netflix, MovieLens) or music (last.fm, Pandora) to users. Link prediction on informal office-network has

been shown to be useful for suggesting potential collaborators Raghavan [2002]. Also in intelligence analysis Schroeder et al. [2003], link-prediction can suggest potential involvement between a group of individuals, who do not have prior records of interaction.

In general popular graph-based proximity measures like personalized pagerank Jeh and Widom [2002a], hitting and commute times Aldous and Fill [200X], Adamic/Adar Adamic and Adar [2003] are used for link prediction on graphs. The experimental setup varies from predicting all absent links at once Liben-Nowell and Kleinberg [2003] to predicting the best link for a given node Brand [2005],Sarkar and Moore [2007].

These papers mostly use co-authorship graphs and movie-recommendation networks. We will sketch a few of the observations from earlier empirical evaluation. We would divide heuristics into roughly two parts, simple variants of the number of common neighbors (Adamic/Adar, Jaccard etc.), and measures based on ensemble of paths. Here is a summary of the most interesting observations from Liben-Nowell and Kleinberg [2003], Brand [2005], and Sarkar and Moore [2007].

1. The number of common neighbors performs surprisingly well on most data-sets, and in many cases beats more complex measures Liben-Nowell and Kleinberg [2003].

2. Adamic/Adar, which is analogous to common neighbors with a skewed weighting scheme mostly outperforms number of common neighbors Liben-Nowell and Kleinberg [2003].

3. Shortest path performs consistently poorly Liben-Nowell and Kleinberg [2003] and Brand [2005]. We have also noted this behavior.

4. Ensemble of paths which looks at long paths (hitting and commute times) does not perform very well Liben-Nowell and Kleinberg [2003] and Brand [2005].

5. Ensemble of paths which down-weights long paths exponentially (e.g. Katz, personalized pagerank) perform better than those which are sensitive to long paths Liben-Nowell and Kleinberg [2003] and Brand [2005].

While these are interesting results, there has not been any work which theoretically justifies this behavior of different heuristics for link prediction on social networks. In our work, we analyze a simple social network model to justify these empirical results.

6.2.2 Latent Space Models for Social Network Analysis

Social network analysis has been an active area of research in sociology and statistics for a long time. One important assumption in social networks is the notion of homophily. McPherson et al. [2001] write in their well-known paper, *"Similarity breeds connection. This principle—the homophily principle-structures network ties of every type, including marriage, friendship, work, advice, support, information transfer, exchange, comember-*

ship, and other types of relationship." More formally, there is a higher probability of forming a link, if two nodes have similar characteristics. These characteristics can be thought of as different features of a node, i.e. geographic location, college/university, work place, hobbies/interests etc. This notion of *social space* has been examined by McFarland and Brown [1973] and Faust [1988].

In 2002, Raftery et al. [2002] introduced a statistical model which explicitly associates every node with locations in a D-dimensional space; links are more likely if the entities are close in latent space. In the original model, the probability of a link between two nodes is defined as a logistic function of their distance. All the pairwise events are independent, conditioned on their latent positions, i.e. distances in the latent space. We alter this model to incorporate radius r in the exponent (for the `RHH` model $r = 1$). r can be interpreted as the sociability of a node. We name this model- the non-deterministic model (section 6.6).

$$\texttt{RHH} \text{ model:} \quad P(i \sim j | d_{ij}) = \frac{1}{1 + e^{\alpha(d_{ij} - 1)}}$$
$$\text{Our model:} \quad P(i \sim j | d_{ij}) = \frac{1}{1 + e^{\alpha(d_{ij} - r)}}$$

The model has two parameters α and r. Parameter $\alpha \geq 0$ controls the sharpness of the function whereas r determines the threshold. Setting $\alpha = \infty$ in our model, yields a simple deterministic model, on which we build our analysis. It may be noted that given the distances the links are deterministic; but given the links, inferring the distances is an interesting problem. In section 6.6, we show how this analysis can be carried over to the non-deterministic case with large but finite α. This assumption is reasonable, because low values of α leads to a random graph; $\alpha = 0$ is exactly the $\mathcal{G}_{N,1/2}$ random graph model. Clearly, it is impossible to perform better than a random predictor in a random graph. With distinct radii for each node, the deterministic model can be used for generating both undirected and directed graphs; however for simplicity we use a realistic directed graph model (section 6.4).

We assume that the nodes are uniformly distributed in a D dimensional Euclidian space. Hence $P(d_{ij} \leq x) = V(1)x^D$, where $V(1)$ is the volume of a *unit radius hypersphere*. This uniformity assumption has been made in earlier social network models, e.g. by Kleinberg [2000], where the points are assumed to lie on a two dimensional grid. In order to normalize the probabilities, we assume that all points lie inside a *unit volume hypersphere* in D dimensions. The maximum r satisfies $V(r) = V(1)r^D = 1$.

CONNECTION TO THE LINK PREDICTION PROBLEM. A latent space model is well-fitted for link prediction because, for a given node i, the most likely node it would connect to is the non-neighbor at the smallest distance. The measure of distance comes from the definition of the model, which could be:

1. Undirected graph with identical r. Here distance from node i to j is simply d_{ij}.

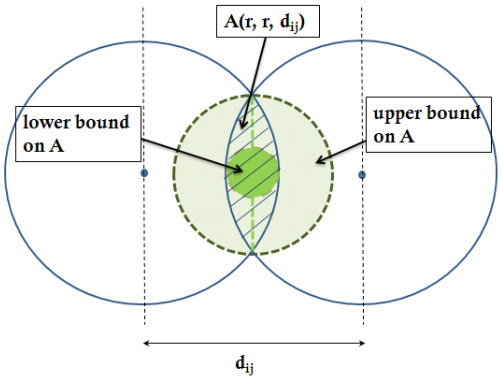


Figure 6.1: Common neighbors of two nodes must lie in the intersection $A(r, r, d_{ij})$.

2. Directed graph where i connects with j (or node j connects to node i) if d_{ij} is smaller than radius of j (or smaller that radius of i) leading to a distance of $d_{ij} - r_j$ (or $d_{ij} - r_i$).

In all these cases, the predicting distances between a pair of nodes is the key. While this can be obtained by maximizing the likelihood of the underlying statistical model, we show that one can obtain high probability bounds on distances from graph based heuristics. In fact we show that the distance to the node picked using a popular heuristic is within a small factor of the *true distance*. This factor quickly goes to zero as N becomes large. Although our analysis uses an extension of the `RHH` model to actually obtain the bounds on distances, the only property (in addition to the uniform and independently distributed latest positions) we use is of homophily in a latent metric space, i.e. if two nodes are *close* in some social space, then they are likely to form a link. Hence this idea should carry over to other social network models as well.

6.3 Deterministic Model with Identical Radii

Consider a simple version of the `RHH` model where all radii are equal to r, and $\alpha \to \infty$. This implies that two nodes i and j share a link (henceforth, $i \sim j$) iff the distance d_{ij} between them satisfies $d_{ij} < r$. Thus, given node positions, links are deterministic; however the node positions are still non-deterministic. While this might appear to be a strong constraint, we will show later in Section 6.6 that similar results are applicable even for finite but large α. We now analyze the simplest of heuristics: counting the common neighbors of i and j. Let there be N nodes in total.

Let $\mathcal{N}(i)$ be the set of neighbors of node i. Let Y_k be a random variable which is 1 if $k \in \mathcal{N}(i) \cap \mathcal{N}(j)$, and 0 otherwise. Given d_{ij}, for all $k \notin \{i, j\}$, the Y_k's are independent

since they only depend on the position of point k. Hence, we have

$$E[Y_k|d_{ij}] = P(i \sim k \sim j|d_{ij}) = \int\limits_{d_{ik},d_{jk}} P(i \sim k|d_{ik})P(j \sim k|d_{jk})P(d_{ik}, d_{jk}|d_{ij})d(d_{ij}) \tag{6.1}$$

In the deterministic model, this quantity is exactly equal to the volume of intersection of two balls of radius r centered at i and j (see Figure 6.1). Denote this volume by $A(r, r, d_{ij})$. Also, the observed value of $\sum_k Y_k$ is simply the number of common neighbors η. From now on we will drop the d_{ij} part when we write expectation for notational convenience. However any expectation in terms of area of intersection is obviously computed *given the pairwise distance* d_{ij}. Thus by using empirical Bernstein bounds Maurer and Pontil [2009], we have:

$$P\left[\left|\sum_k Y_k/N - E[Y_k]\right| \geq \sqrt{\frac{2\mathtt{var}_N(Y)\log 2/\delta}{N}} + \frac{7\log 2/\delta}{3(N-1)}\right] \leq 2\delta \tag{6.2}$$

$\mathtt{var}_N(Y)$ is the sample variance of Y, i.e. $\dfrac{\eta(1-\eta/N)}{N-1}$.

Setting $\epsilon = \sqrt{\dfrac{2\mathtt{var}_N(Y)\log 2/\delta}{N}} + \dfrac{7\log 2/\delta}{3(N-1)}$

$$\boxed{P\left[\frac{\eta}{N} - \epsilon \leq A(r, r, d_{ij}) \leq \frac{\eta}{N} + \epsilon\right] \geq 1 - 2\delta} \tag{6.3}$$

$A(r, r, d_{ij})$ is twice the volume of a spherical cap whose base is $d_{ij}/2$ distance away from the center:

$$A(r, r, d_{ij}) = 2\frac{\pi^{\frac{D-1}{2}} r^D}{\Gamma\left(\frac{D+1}{2}\right)} \int\limits_0^{\cos^{-1}\left(\frac{d_{ij}}{2r}\right)} sin^D(t)dt \tag{6.4}$$

Given the above bounds on $A(r, r, d_{ij})$, we can obtain bounds on d_{ij} by solving eq. (6.4) numerically. However, weaker analytic formulas can be obtained by using hyperspheres bounding this intersection, as shown in Figure 6.1. $V(r)$ is the volume of a D dimensional hypersphere of radius r.

$$\left(1 - \frac{d_{ij}}{2r}\right)^D \leq \frac{A(r, r, d_{ij})}{V(r)} \leq \left(1 - \left(\frac{d_{ij}}{2r}\right)^2\right)^{D/2} \tag{6.5}$$

Using this in eq. (6.3) gives us bounds on d_{ij}:

$$2r\left(1 - \left(\frac{\eta/N + \epsilon}{V(r)}\right)^{1/D}\right) \leq d_{ij} \leq 2r\sqrt{1 - \left(\frac{\eta/N - \epsilon}{V(r)}\right)^{2/D}}$$

USING COMMON NEIGHBORS IN LINK PREDICTION. Let us recall that in link prediction, we want to pick the node which is most likely to be a neighbor of i, and is not currently a neighbor (call this OPT). If we knew the positions, we would pick the non-neighbor with the minimum distance (d_{OPT}). However, since positions in latent space are unknown, we instead predict a link to the node that shares the most common neighbors with i (call this MAX). Here we will show that (Lemma 6.3.2) the distance to the node with largest common neighbors (d_{MAX}) is within an additive factor of d_{OPT}. This factor goes to zero as N increases. This again shows that, as N increases, link prediction using the number of common neighbors converges to the optimal prediction.

Let the number of common neighbors between i and OPT be η_{OPT}, and between i and MAX be η_{MAX}. Now we will try to relate d_{OPT} with d_{MAX}. Note that both these distances are larger than r. Denote the area of intersections of these two nodes with i as A_{OPT} and A_{MAX} respectively. Then, we have:

Lemma 6.3.1. *Define* $\epsilon_o = \sqrt{\frac{2\mathtt{var}_N(Y_{OPT})\log 2/\delta}{N}} + \frac{7\log 2/\delta}{3(N-1)}$ *and* $\epsilon_m = \sqrt{\frac{2\mathtt{var}_N(Y_{MAX})\log 2/\delta}{N}} + \frac{7\log 2/\delta}{3(N-1)}$*, where* Y_{OPT} *and* Y_{MAX} *denote the random variable for common neighbors between* i *and OPT, and* i *and MAX respectively.*

$$A_{OPT} \geq A_{MAX} \qquad\qquad P\left[A_{MAX} \geq A_{OPT} - \epsilon_o - \epsilon_m\right] \geq 1 - 2\delta$$

Proof. Using the high probability bounds from eq. (6.3) we have (w.h.p),

$$A_{MAX} - A_{OPT} \geq \frac{\eta_{MAX}}{N} - \epsilon_m - (\frac{\eta_{OPT}}{N} + \epsilon_o)$$

By definition, $\eta_{MAX} \geq \eta_{OPT}$. This and the high probability empirical Bernstein bound on A_{OPT} yield the result. □

This means that as N becomes large, the node with the highest number of common neighbors will be the optimal node for link prediction. Now we will give a bound on how far d_{OPT} is from d_{MAX}.

Theorem 6.3.2. *As before, define* $\epsilon_o = \sqrt{\frac{2\mathtt{var}_N(Y_{OPT})\log 2/\delta}{N}} + \frac{7\log 2/\delta}{3(N-1)}$ *. Also let* $\epsilon_m = \sqrt{\frac{2\mathtt{var}_N(Y_{MAX})\log 2/\delta}{N}} + \frac{7\log 2/\delta}{3(N-1)}$ *, and* $\epsilon_f = \epsilon_o + \epsilon_m$*.*

$$d_{OPT} \leq d_{MAX} \overset{w.h.p}{\leq} d_{OPT} + 2r\left(\frac{\epsilon_f}{V(r)}\right)^{1/D} \leq d_{OPT} + 2\left(\frac{\epsilon_f}{V(1)}\right)^{1/D}$$

Proof. The proof is deferred to appendix A.D. □

6.4 Deterministic Model with Distinct Radii

Until now our model has used the same r for all nodes. The degree of a node is distributed as $Bin(N, V(r))$, where $V(r)$ is the volume of a radius r. Thus r determines the degree of a node in the graph, and identical r will lead to a roughly regular graph. In practice, social networks are far from regular. In order to accommodate complex networks we will now allow a different radius (r_i) for node i. For this section, we will assume that these radii are given to us. The new connectivity model is: $i \to j$ iff $d_{ij} \leq r_j$, where $i \to j$ now represents a *directed* edge from i to j. While variants of this are possible, this is similar in spirit to a citation network, where a paper i tends to cite a well-cited paper j (with larger number of in-neighbors) than another infrequently cited paper on the same topic; here, r_j can be thought of as the measure of popularity of node j. Under this model, we will show why some link prediction heuristics work better than others.

As in the previous section, we can use common neighbors to estimate distance between nodes. We can count common neighbors in 4 different ways as follows:

(Type-1) All k, s.t. $k \to i$ and $k \to j$: all nodes which point to both i and j. The probability of this given d_{ij} is $P(d_{ik} \leq r_i \cap d_{jk} \leq r_j | d_{ij})$, which can be easily shown to be $A(r_i, r_j, d_{ij})$.

(Type-2) All k, s.t. $i \to k$ and $j \to k$: all nodes to which both i and j point. The probability of this given d_{ij} is $A(r_k, r_k, d_{ij})$.

(Type-3) All k, s.t. $i \to k$ and $k \to j$: all directed paths of length 2 from i to j. The probability of this given d_{ij} is given by $A(r_k, r_j, d_{ij})$.

(Type-4) All k, s.t. $j \to k$ and $k \to i$: all directed paths of length 2 from j to i. The probability of this given d_{ij} is given by $A(r_i, r_k, d_{ij})$.

If we count type-1 nearest neighbors, the argument from section 6.3 carries over, and if there are enough common neighbors of this type, we can estimate d_{ij} by computing $A(r_i, r_j, d_{ij})$. However, if both r_i and r_j are small, there might not be many common neighbors; indeed, if $d_{ij} > r_i + r_j$, then there will be no type-1 common neighbors. In such cases, we consider type-2 neighbors, i.e. the ones which both i and j point to. The analysis for type-3 and type-4 neighbors is very similar to that for type-2, and hence we do not discuss these any further. In the type-2 case, the radii r_k of the common neighbors play an important role. Intuitively, if both i and j point to a very popular node (high radius r_k), then that should not give us a lot of information about d_{ij}, since it is not very surprising. In particular, any type-2 common neighbor k leads to the following constraint: $d_{ij} \leq d_{ik} + d_{jk} \leq 2r_k$. Obviously, the bound is stronger for small values of r_k. This argues for weighting common neighbors differently, depending on their radii. We formalize this

intuition using a toy example now. The analysis in the following section can be generalized to graphs, where the radii of the nodes form a finite set.

Recall that common neighbors can be expressed as a sum of random variables Y_k, which is 1, if k is a common neighbor of i and j.

MOTIVATING EXAMPLE. Take a toy network where the nodes can have two different radii R and R', with $R < R'$. The total number of low radii nodes is $n(R)$, whereas that of large radii nodes is $n(R')$.

The expectation of the number of type-2 common neighbors will now have a mixture of $A(R, R, d_{ij})$ and $A(R', R', d_{ij})$. One solution is to estimate high probability bounds on distances from the two different classes of common neighbors separately, and then examine the intersection of these bounds. We will discuss a new estimator based on this intuition at the end of this section. The other solution is to look at weighted combinations of common neighbors from different radii. The weights will reflect how important one common neighbor is relative to another. For example, consider a pair of papers which both cite a book on introduction to algorithms (cited by 5000 other papers, i.e. higher radius), and a specific article on randomized algorithms (cited by 30 other papers, i.e. lower radius). The second article gives more evidence on the "closeness" or similarity of the pair. We will consider this approach next.

Suppose we observe $\eta(R)$ common neighbors of $n(R)$ nodes of small radius, and $\eta(R')$ common neighbors of $n(R')$ nodes of large radius, between pair of nodes i, j. The likelihood of these observations, given the pairwise distance d_{ij} is:

$$P(\eta(R), n(R), \eta(R'), n(R')|d_{ij}) = \prod_{r \in \{R,R'\}} \binom{N_r}{\eta_r} A(r, r, d_{ij})^{\eta_r} (1 - A(r, r, d_{ij}))^{N_r - \eta_r} \quad (6.6)$$

We want to rank pairs of nodes using the distance estimate d^*, which maximizes the likelihood of this partial set of observations. However, if $\eta(R) > 0$, the logarithm of the above is defined only when $d_{ij} \leq 2R$. To make the likelihood well-behaved, we introduce a small noise parameter β: node i connects to node j with probability $1-\beta$ (if $d_{ij} \leq r_j$), or with probability β (otherwise). Now, the probability of having a type-2 common neighbor of radius r will be $\beta + A(r, r, d_{ij})(1 - \beta)$. For ease of exposition we will denote this by $A_\beta(r, r, d_{ij})$. The new likelihood will be exactly as in eq. (6.6), except we will use A_β instead of A. Setting the derivative of the logarithm yields:

$$w(R, d^*)n(R)A_\beta(R, R, d^*) + w(R', d^*)n(R')A_\beta(R', R', d^*) = w(R, d^*)\eta(R) + w(R', d^*)\eta(R') \quad (6.7)$$

where, $w(R, d^*) = \dfrac{-\left.\frac{dA_\beta(R,R,d_{ij})}{d_{ij}}\right|_{d^*}}{A_\beta(R, R, d^*)(1 - A_\beta(R, R, d^*))}$. Note that the negative sign is only to make both sides positive, since A_β decreases with distance.

Using Leibnitz's rule on eq. 6.4, the derivative of A w.r.t d_{ij} can be written as:

$$A'(R,R,d_{ij}) = \frac{dA(R,R,d_{ij})}{d_{ij}} = \begin{cases} -C_D r^{D-1}\left(1-\frac{d_{ij}^2}{4r^2}\right)^{\frac{D-1}{2}} & \text{If } d_{ij} \leq 2r \\ 0 & \text{Otherwise} \end{cases} \quad (6.8)$$

$$A'_\beta(R,R,d_{ij}) = \frac{dA_\beta(R,R,d_{ij})}{d_{ij}} = (1-\beta)A'(R,R,d_{ij})$$

Suppose we could approximate $w(R,d_{ij})$ with some w_R that depends only on R and not on d_{ij}. Then, the RHS of eq. (6.7) can be obtained from the data alone. Also, it can be shown that the L.H.S. of eq. (6.8) is a monotonically decreasing function of d^*. Hence, a pair of nodes with higher RHS will have lower distance estimate, implying that ranking based on the RHS will be equivalent to ranking based on distance. All that remains is finding a good approximation w_R.

We start by bounding $A'(R,R,d_{ij})$ in terms of $A(R,R,d_{ij})$. Consider $D > 1$[1]. Combining eq. (6.8) with eq. (6.5) yields a lower bound: $A'(R,R,d) \geq c'_D \frac{A}{\sqrt{r^2-d^2/4}}$. For the upper bound, we note that the volume of the spherical cap can be lower bounded by the volume of a sphere in $D-1$ dimensions of radius $\sqrt{r^2-d^2/4}$, times the height $r-d/2$, times a constant depending only on D: $A(r,r,d) \geq k_D\left(r^2-\frac{d^2}{4}\right)^{\frac{D-1}{2}}\left(r-\frac{d}{2}\right)$ Combining with eq. (6.8) gives: $-A'(r,r,d) \leq k'_D \frac{A}{r-d/2}$. Given that β is extremely small, we have

$$\frac{1}{r}c''_D \frac{1}{\sqrt{1-\frac{d^2}{4r^2}}} \lesssim \frac{|A'_\beta(r,r,d)|}{A_\beta(r,r,d)(1-A_\beta(r,r,d))} \leq \frac{1}{r}k''_D \frac{1}{(1-V(r))(1-\frac{d}{2r})}$$

For a given distance d and increasing radius r, the weight $w(r,d)$ first decreases sharply but increases again once r becomes close to the maximum radius, i.e., $V(r) \approx 1$ (see Figure 6.2). Thus, it is high for both nodes of very low and very high radius. Clearly, the presence of a low-radius common neighbor gives strong evidence that d is small. On the other hand, the *absence* of a *very* high degree node gives strong evidence that d is very large. Note that, the presence of low radius common neighbors in the absence of very high radius common neighbors is extremely unlikely. This is because, if a pair of nodes are close enough to connect to a low radius node, they are also very likely to both be within the radius of some very high radius node.

Since $NV(r)$ is the expectation of the indegree of a node of radius r, such high-radius nodes are expected to have extremely high degrees. However, high-degree nodes in real-world settings typically connect to no more than $10-20\%$ of the set of nodes, which is why a practical weighting only needs to focus on situations where $1-V(r) \approx 1$. For relatively small d (which are the interesting candidates for link prediction), the weights $w(r,d)$ are

[1]When $D=1$, A' is constant, and $A(r,r,d) = 2r-d$.

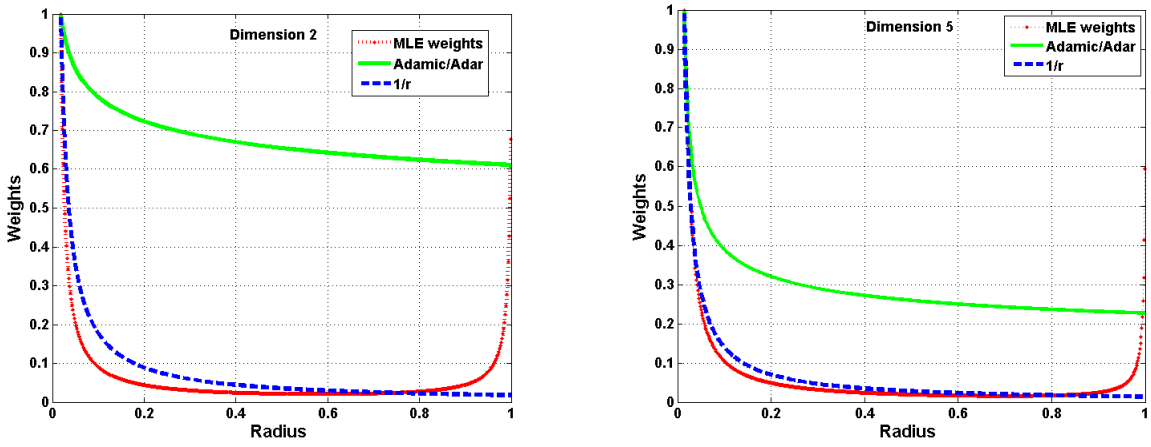


Figure 6.2: For a given distance d, we plot $w(r, d)$, $Adamic/Adar$ and $1/r$ with increasing r. Note that both axes are scaled to a maximum of 1. Note that $1/r$ closely matches $w(r, d)$ for a wide range of radii.

then well approximated by $w(r) = 1/r$ up to a constant. Note that this is identical to weighting a node by $1/(NV(r))^{1/D}$, i.e., essentially weighting a common neighbor i by $1/\deg(i)^{1/D}$.

Now we will discuss a popular weighting scheme, namely Adamic/Adar, which weights the common neighbors by $1/\log(\deg(i))$.

ADAMIC/ADAR: A POPULAR LINK PREDICTION HEURISTIC. In practice, the radius of a node is analogous to its degree, and hence it is natural to weight a node more if it has lower degree. The Adamic/Adar measure Adamic and Adar [2003] was introduced to measure how related two home-pages are. The authors computed this by looking at common features of the webpages, and instead of computing just the number of such features, they weighted the rarer features more heavily. In our social networks context, this is equivalent to computing similarity between two nodes by computing the number of common neighbors, where each is weighted inversely by the logarithm of its degree.

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

Liben-Nowell and Kleinberg [2003] have shown that this out-performs the number of common neighbors in a variety of social and citation networks, confirming the positive effect of a skewed weighting scheme that we observed in the motivating example.

We can analyze the Adamic/Adar measure as follows. In our model, the expected degree of a node k of radius r_k is simply $NV(r_k)$, so we set the weights as $w_k =$

$1/\log(NV(r_k))$. Let $\mathcal{S} = \sum_k w_k Y_k$, where random variable $Y_k = 1$ if k is a type-2 common neighbor of i and j, and zero otherwise. Clearly, $E[\mathcal{S}] = \sum_k w_k A(r_k, r_k, d_{ij}) = \sum_k A(r_k, r_k, d_{ij})/\log(NV(r_k))$. Let the minimum and maximum radii be $r_{\min}$ and $r_{\max}$ respectively. The following can be easily obtained from the Chernoff bound.

Lemma 6.4.1. $\frac{\mathcal{S}}{N}\left(1 - \sqrt{\frac{3\log(NV(r_{\max}))\ln(1/\delta)}{N \cdot A(r_{\max}, r_{\max}, d_{ij})}}\right) \leq \frac{E[\mathcal{S}]}{N} \leq \frac{\mathcal{S}}{N}\left(1 + \sqrt{\frac{3\log(NV(r_{\min}))\ln(1/\delta)}{N \cdot A(r_{\min}, r_{\min}, d_{ij})}}\right)$

Clearly, the error terms decay with increasing N, and for large N, we can tightly bound $E[\mathcal{S}]$. Since $E[\mathcal{S}]$ is monotonically decreasing function of d_{ij}, this translates into bounds on d_{ij} as well.

NEW ESTIMATORS. Based on the analysis of the motivating example discussed before, we can get a lot of intuition about a general graph. We have seen that low radius common neighbors imply that distance is small, whereas fewer high degree common neighbors in the absence of any low degree common neighbors imply that distance is large. Based on these observations, we will define the following two estimators.

Consider the estimate Q_R, which is simply the fraction of nodes with radius smaller than R that are type-2 neighbors of i and j. Let $n(R)$ be the number of nodes with radius less than R. We define Q_R as follows:

$$Q_R = \frac{\sum_{r_k \leq R} Y_k}{n(R)} \qquad \rightarrow \qquad E[Q_R] = \frac{\sum_{r_k \leq R} A(r_k, r_k, d_{ij})}{n(R)} \leq A(R, R, d_{ij}) \tag{6.9}$$

Q_R is large when many low-radius nodes are type-2 common neighbors of i and j. Application of Hoeffding bounds give:

$$P\left[Q_R \leq E[Q_R] + \sqrt{\frac{1}{2n(R)}\ln\left(\frac{1}{\delta}\right)}\right] \geq 1 - \delta$$

We pick R so that $E[Q_R]$ is *large enough*. While iterative algorithms can give an exact value of the upper bound on d_{ij}, we will also provide a simple intuitive bound:

$$Q_R \leq A(R, R, d_{ij}) + \sqrt{\frac{1}{2n(R)}\ln\left(\frac{1}{\delta}\right)} \leq V(R)\left(1 - \frac{d_{ij}^2}{4R^2}\right)^{D/2} + \sqrt{\frac{1}{2n(R)}\ln\left(\frac{1}{\delta}\right)}$$

$$\Rightarrow d_{ij} \leq 2R\sqrt{1 - \left(\frac{Q_R - \sqrt{\ln(1/\delta)/2n(R)}}{V(R)}\right)^{2/D}}$$

If we observe a large Q_R for a small R, then this upper bound gets smaller. This shows that a large number of neighbors of small degree gives a tighter upper-bound on d_{ij}. In the same spirit, we can define another estimator $T_{R'}$, such that

$$T_{R'} = \frac{\sum_{r_k \geq R'} Y_k}{n(R')} \qquad \rightarrow \qquad E[T_{R'}] = \frac{\sum_{r_k \geq R'} A(r_k, r_k, d_{ij})}{n(R')} \geq A(R', R', d_{ij})$$

A similar analysis yields a high probability lower-bound on d_{ij}:

$$T_{R'} \geq A(R', R', d_{ij}) - \sqrt{\frac{1}{2n(R')} \ln\left(\frac{1}{\delta}\right)} \geq V(R')\left(1 - \frac{d_{ij}}{2R'}\right)^D - \sqrt{\frac{1}{2n(R')} \ln\left(\frac{1}{\delta}\right)}$$

$$\Rightarrow d_{ij} \geq 2R'\left(1 - \left(\frac{T_{R'} + \sqrt{\ln(1/\delta)/2n(R')}}{V(R')}\right)^{1/D}\right)$$

Smaller values of $T_{R'}$ yield tighter lower bounds. This tells us that if many high degree nodes are *not* common neighbors of i and j then, we are more confident than i and j are far away.

While Q_R and $T_{R'}$ could be used with any R and R', we could also perform a sweep over the range of possible radii, computing bounds on d_{ij} for each radius using both estimators, and then retaining the best.

6.5 Estimators using Longer Paths in the Deterministic Model

The bounds on the distance d_{ij} described in the previous sections apply only when i and j have common neighbors. However, there will be no common neighbors if (for the undirected case) (a) $d_{ij} > 2r$, or (b) no points fall in the intersection area $A(r, r, d_{ij})$ due to small sample size N. In such cases, looking at paths of length $\ell > 2$ between i and j can yield bounds on d_{ij}. Such bounds can be useful even when common neighbors exist; in fact, we show that the mere existence of *one* common neighbor leads to stronger bounds for long paths.

We first discuss how d_{ij} can be upper-bounded using the observed number of simple ℓ-hop paths for any given $\ell > 2$. A stronger upper bound and a lower bound can be derived when i and j are known to have at least one common neighbor. We demonstrate this for $\ell = 3$, with a similar technique being applicable for longer paths as well. For the sake of simplicity, we restrict ourselves to the case of identical and known r.

AN UPPER BOUND FOR $\ell > 2$. Let $Y(i, k_1, \ldots, k_{\ell-2}, j) = 1$ if there is a simple path of length ℓ such that $i \sim k_1 \sim k_2 \sim \ldots k_{\ell-2} \sim j$, with no node being repeated in the path. Let $\mathcal{S}^{(\ell)}$ be the set of ordered sets of ℓ distinct elements from $\{1, \ldots, N\}$; thus, $\mathcal{S}^{(\ell)}$ represents all possible simple paths of length ℓ. Let $\eta_\ell(i, j)$ be the number of paths of length ℓ between i and j, given the distance between i, j. $\eta_\ell(i, j)$ is simply,

$$\eta_\ell(i, j) = \sum_{k_1, \ldots k_{\ell-2} \in \mathcal{S}^{(\ell-2)}} Y(i, k_1, \ldots, k_{\ell-2}, j | d_{ij}).$$

Note that, $\eta_\ell(i,j)$ is a function of the $N-2$ independent random variables $X_k, k \notin \{i,j\}$, for a given $X_i = x_i, X_j = x_j$, i.e. $f_{x_i,x_j}(X_k, k \notin \{i,j\})$.

We need to infer bounds on d_{ij} given the observed number of simple paths $\eta_\ell(i,j)$. Our first step is to bound the maximum degree Δ of any graph generated by the RHH model. Next, we use Δ to bound both the maximum possible value of $\eta_\ell(i,j)$ and the change that can be induced in it by moving any one point. Finally, we compute the expected value $E(f_{x_i,x_j})$, which is simply $E[\eta_\ell(i,j)]$. Combining these will give us a bound linking d_{ij} to the number of simple ℓ-hop paths.

Under the assumption that the positions X_k are uniformly distribution in the unit hypersphere, we can bound Δ, as follows:

Lemma 6.5.1. $\Delta < NV\left(1+\sqrt{\frac{\ln(N/\delta)}{2NV}}\right)$ *with probability at least* $1-\delta$.

Proof. The degree $\deg(k)$ of any node k is a binomial random variable with expectation $E[d(k)] = NV$, where V is the volume of a hypersphere of radius r. Thus, using the Chernoff bound, $d(k) < NV\left(1+\sqrt{\frac{\ln(N/\delta)}{2NV}}\right)$ holds with probability at least $(1-\delta/N)$. Applying the union bound on all nodes yields the desired proposition. □

Lemma 6.5.2. *For any graph with maximum degree* Δ*, we have:* $\eta_\ell(i,j) \leq \Delta^{\ell-1}$.

Proof. This can be proved using a simple inductive argument. If the graph is represented by adjacency matrix $\mathbf{M}$, then the number of length ℓ paths between i and j is given by $\mathbf{M}^\ell(i,j)$. Trivially $\mathbf{M}^2_{ij}$ can be at most Δ. This happens when both i and j have degree Δ, and their neighbors form a perfect matching. Assuming this is true for all $m < \ell$, we have: $\mathbf{M}^\ell(i,j) = \sum_p \mathbf{M}(i,p)\mathbf{M}^{\ell-1}(p,j) \leq \Delta^{\ell-2}\sum_p \mathbf{M}(i,p) \leq \Delta^{\ell-1}$ □

Lemma 6.5.3. *For* $\ell < \Delta$, $|\eta_\ell(i,j|X_1,\ldots,X_p,\ldots,X_N) - \eta_\ell(i,j|X_1,\ldots,\tilde{X}_p,\ldots X_N)| \leq (\ell-1)\cdot\Delta^{\ell-2}$

Proof. The largest change in $\eta_\ell(.)$ occurs when node p was originally unconnected to any other node, and is moved to a position where it can maximally add to the number of ℓ-hop paths between i and j (or vice versa). Consider all paths where p is m hops from i (and hence $\ell-m$ hops from j. From Lemma 6.5.2, the number of such paths can be at most $\Delta^{m-1}\cdot\Delta^{\ell-m-1} = \Delta^{\ell-2}$. Since $m \in \{1,\ldots,\ell-1\}$, the maximum change is $(\ell-1)\cdot\Delta^{\ell-2}$. □

The bounds in both Lemma 6.5.2 and 6.5.3 are tight, as can be seen by considering a clique of i, j, and $\Delta-1$ other nodes. Next, we compute the expected number of ℓ-hop paths. Define $A(r_1,r_2,d)$ as the volume of intersection of two balls of radii r_1 and r_2, whose centers are distance d apart (as always, the dimension D is implicit). Define $Pr_\ell(i,j)$ as the probability of observing an ℓ-hop path between points i and j.

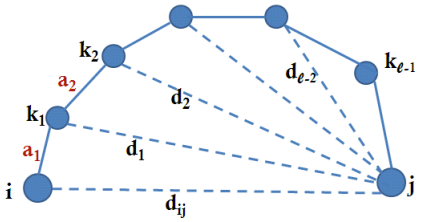


Figure 6.3: Triangulation for bounding d_{ij} using ℓ-hop paths.

Theorem 6.5.4. $E[\eta_\ell(i,j)] \leq \Delta^{\ell-1} \prod_{p=1}^{\ell-1} A(r, p \times r, (d_{ij} - (\ell - p - 1)r)_+)$, *where* x_+ *is defined as* $\max(x, 0)$.

Proof. Consider an ℓ-hop path between nodes i and j as in figure 6.3. The solid lines are the edges in the path, whereas the dotted lines are the distance variables we will introduce in order to compute the probability of this path. For clarity of notation, let us denote the distances d_{ik_1}, d_{ik_2} etc. by a_1, a_2, up to $a_{\ell-1}$. These are all less than r, since the corresponding edges are present. We also denote the distances d_{jk_1}, d_{jk_2}, etc. by d_1, d_2, up to $d_{\ell-1}$. From the triangle inequality, $d_{\ell-2} \leq a_{\ell-1} + a_\ell \leq 2r$, and by induction, $d_k \leq (\ell - k)r$. Similarly, $d_1 \geq (d_{ij} - a_1)_+ \geq (d_{ij} - r)_+$, and by induction, $d_k \geq (d_{ij} - kr)_+$. We can now compute the probability of observing an ℓ-hop path:

$$
\begin{aligned}
&Pr_\ell(i,j) = P(i \sim k_1 \sim \ldots \sim k_{\ell-1} \sim j | d_{ij}) = P(a_1 \leq r \cap \ldots \cap a_\ell \leq r | d_{ij}) \\
&= \int_{d_1,\ldots,d_{\ell-2}} P(a_1 \leq r, \ldots, a_{\ell-1} \leq r, d_1, \ldots, d_{\ell-2} | d_{ij}) \\
&= \int_{d_1=(d_{ij}-r)_+}^{(\ell-1)r} \cdots \int_{d_{\ell-2}=(d_{ij}-(\ell-2)r)_+}^{2r} P(a_{\ell-1} \leq r, a_\ell \leq r | d_{k-2}) P(a_{\ell-2} \leq r, d_{\ell-2} | d_{\ell-3}) \ldots P(a_1 \leq r, d_1 | d_{ij}) \\
&\leq A(r, r, (d_{ij} - (\ell-2)r)_+) \times A(r, 2r, (d_{ij} - (\ell-3)r)_+) \times \ldots \times A(r, (\ell-1)r, d_{ij}) \\
&\leq \prod_{p=1}^{\ell-1} A(r, p \times r, (d_{ij} - (\ell - p - 1)r)_+) \qquad (6.10)
\end{aligned}
$$

Since there can be at most $\Delta^{\ell-1}$ possible paths (from Lemma 6.5.2), the theorem statement follows. □

Corollary 6.5.5. *For* 3*-hop paths, we have:*

$$
\begin{aligned}
Pr_3(i,j) &\leq A(r, r, (d_{ij} - r)_+)) \cdot A(r, 2r, d_{ij}) \\
E[\eta_3(i,j)] &\leq \Delta^2 \cdot A(r, r, (d_{ij} - r)_+)) \cdot A(r, 2r, d_{ij})
\end{aligned}
$$

Theorem 6.5.6.

$$\eta_\ell(i,j)$$

$$\leq (NV)^{\ell-1} \left[\prod_{p=1}^{\ell-1} A(r, p\times r, (d_{ij} - (\ell-p-1)r)_+) + \frac{(\ell-1)\sqrt{\frac{1/\delta}{2}}}{\sqrt{N}V\left(1+\sqrt{\frac{\ln(N/\delta)}{2NV}}\right)} \right] \left(1+\sqrt{\frac{\ln(N/\delta)}{2NV}}\right)^{\ell-1}$$

with probability at least $(1-2\delta)$.

Proof. From McDiarmid's inequality McDiarmid [1989], we have:

$$\eta_\ell(i,j)$$

$$\leq E[\eta_\ell(i,j)] + (\ell-1)\Delta^{\ell-2}\sqrt{\frac{N\ln(1/\delta)}{2}}$$

$$\leq \Delta^{\ell-2}\left[\Delta\textstyle\prod_{p=1}^{\ell-1} A(r, p\times r, (d_{ij} - (\ell-p-1)r)_+) + (\ell-1)\sqrt{\frac{N\ln(1/\delta)}{2}}\right]$$

$$\leq (NV)^{\ell-1}\left[\textstyle\prod_{p=1}^{\ell-1} A(r, p\times r, (d_{ij} - (\ell-p-1)r)_+) + \frac{(\ell-1)\sqrt{\frac{\ln(1/\delta)}{2}}}{\sqrt{N}V\left(1+\sqrt{\frac{\ln(N/\delta)}{2NV}}\right)}\right]\left(1+\sqrt{\tfrac{\ln(N/\delta)}{2NV}}\right)^{\ell-1}$$

where Theorem 6.5.4 is applied in the step 2, and Lemma 6.5.1 in step 3. Note that as N increases, the second term in the summation decays, yielding tighter bounds. □

BOUNDING d_{ij}. Theorem 6.5.6 yields an upper bound d_{ij} as follows. Only the first term in the summation depends on d_{ij}, and this term decreases monotonically with increasing d_{ij}. Thus, a simple binary search can give us the value of d_{ij} that achieves the equality in Theorem 6.5.6, and this is an upper bound on d_{ij}.

A looser but analytic bound can be obtained by upper-bounding all but one of the $A(.)$ terms by 1. For example, using $A(r, 2r, d_{ij}) \leq 1$ in Corollary 6.5.5 yields $E[\eta_3(i,j)] \leq \Delta^2 A(r, r, (d_{ij}-r)_+)$. Using this in McDiarmid's inequality yields a bound of the form

$$A(r, r, (d_{ij}-r)_+) \geq \frac{\eta_3(i,j)}{c(N,\delta)} - c'(N,\delta) \Rightarrow d_{ij} \leq r + 2r\sqrt{1 - \left(\frac{\eta_3(i,j)/c(N,\delta) - c'(N,\delta)}{V(r)}\right)^{2/D}}$$

In general, bounds for ℓ-hop paths are of the form $d_{ij} \leq \ell r(1 - g(\eta_\ell(i,j), \epsilon))$. Thus, for some $\ell' > \ell$, $\eta_{\ell'}(i,j)$ needs to be much larger than $\eta_\ell(i,j)$ for the bound using ℓ' to be stronger than that for ℓ. In particular, this shows that when enough common neighbors are present (i.e., 2-hop paths), looking at longer paths is unlikely to improve bounds and help link prediction, thus theoretically confirming the empirical observations of Liben-Nowell and Kleinberg [2003].

BETTER BOUNDS WHEN SHORTER PATHS EXIST. While the above applies in the general case, it suffers from two weaknesses: (1) the upper bound on $E[\eta_\ell(i,j)]$ (and hence on d_{ij}) derived in Theorem 6.5.4 gets progressively weaker as ℓ increases, and (2) we could not derive a useful lower bound on $E[\eta_\ell(i,j)]$ (the trivial lower bound is zero, which is achieved when $d_{ij} = \ell r$). Both of these can be remedied if we know that *at least* one path of length less than ℓ exists. We demonstrate the idea for $\ell = 3$, but it can be used for larger ℓ in a similar fashion. First, we prove two bounds on the probability $Pr_3(i,j)$ of observing a 3-hop path between two points whose distance is d_{ij}.

Lemma 6.5.7. *If there exists any 3-hop path between i and j, then, for any $d' \in [(d_{ij} - 2r)_+, 2r]$,*

$$A(r, d', d_{ij}) \cdot A(r, r, d')$$

$$\leq \mathbf{Pr_3(i,j)}$$

$$\leq [A(r, r, (d_{ij} - r)_+) - A(r, r, d')]\, A(r, d', d_{ij}) + A(r, r, d') \cdot A(r, 2r, d_{ij})$$

Proof. Consider all points that are within distance r of point i, and within a distance range $(x, x + \Delta x)$ of point j; here, Δx refers to an infinitesimal change in x. Since these points are within r of i, they can be the first hop in a 3-hop path from i to j. Also, since they are all equidistant from j, and the probability of a common neighbor between two points depends only on their distance, all of these points have the same probability $A(r, x, d_{ij})$ of forming a common neighbor with j. Let $p(r, x, d_{ij})$ denote the probability density function for such points. Triangle inequalities imply that $2r \geq x \geq (d_{ij} - r)_+$ for any 3-hop path to exist. Then,

$$\begin{aligned} Pr_3(i,j) &= \int_{(d_{ij}-r)_+}^{2r} p(r, x, d_{ij}) \cdot A(r, r, x) dx \geq \int_{(d_{ij}-r)_+}^{d'} p(r, x, d_{ij}) \cdot A(r, r, x) dx \\ &\geq A(r, r, d') \int_{(d_{ij}-r)_+}^{d'} p(r, x, d_{ij}) dx \geq A(r, d', d_{ij}) \cdot A(r, r, d') \end{aligned}$$

This proves the lower-bound on $Pr_3(i,j)$. The proof for the upper bound is similar:

$$\begin{aligned} Pr_3(i,j) &= \int_{(d_{ij}-r)_+}^{d'} p(r,x,d_{ij}) \cdot A(r,r,x)dx + \int_{d'}^{2r} p(r,x,d_{ij}) \cdot A(r,r,x)dx \\ &\leq A(r,r,(d_{ij}-r)_+) \cdot A(r,d',d_{ij}) + A(r,r,d') \cdot [A(r,2r,d_{ij}) - A(r,d',d_{ij})] \\ &\leq [A(r,r,(d_{ij}-r)_+) - A(r,r,d')]\, A(r,d',d_{ij}) + A(r,r,d') \cdot A(r,2r,d_{ij}) \end{aligned}$$

□

The difficulty in using these bounds arises from the fact that $d' \in [(d_{ij}-r)_+, 2r]$, and d_{ij} is unknown. When we only know that *some* 3-hop path exists, then $d_{ij} \leq 3r$, and hence $d' = 2r$ is the only value of d' that is guaranteed to lie within the required interval. In fact, using $d' = 2r$ yields exactly the statement of Corollary 6.5.5. However, suppose we also knew that at least one 2-hop path exists. Then, $d_{ij} \leq 2r$, and so any d' in the range $r \leq d' \leq 2r$ is valid. In particular, we can use $d' = r$ to get the following bounds.

Theorem 6.5.8. *When $d_{ij} \leq 2r$, then*

$$\begin{aligned} & A(r,r,d_{ij}) \cdot A(r,r,r) \\ & \leq \mathbf{Pr_3(i,j)} \\ & \leq [A(r,r,(d_{ij}-r)_+) - A(r,r,r)]\, A(r,r,d_{ij}) + A(r,r,r) \cdot A(r,2r,d_{ij}) \end{aligned}$$

Lemma 6.5.9. *The upper bound in Theorem 6.5.8 is smaller (and hence, better) than the upper bound in Corollary 6.5.5.*

Proof. The proof follows from basic algebraic manipulations. □

Thus, the presence of even one common neighbor between i and j offers better bounds using 3-hop paths. In addition, we also have a lower bound that was unavailable in the general case. These translate to sharper bounds on d_{ij}, which can be obtained via binary search as described previously. Similar results can be obtained for paths of length $\ell > 3$.

OBSERVATIONS. Our analysis of ℓ-hop paths yields the following observations. (1) When short paths are non-existent or rare, the bounds on d_{ij} that we obtain through them can be loose. Longer paths can be used to yield better bounds in such cases. (2) As ℓ increases, more and more long paths need to be observed before the corresponding bound on d_{ij}

becomes comparable or better than bounds obtained via shorter paths. (3) Even the *existence* of a short path can improve upper bounds obtained by all longer paths. In addition, lower bounds on d_{ij} can also be obtained. (4) The *number* of paths is important to the bound. Link prediction using the length of the shortest path ignores this information, and hence should perform relatively poorly, as observed by Liben-Nowell and Kleinberg [2003], Brand [2005] and Sarkar and Moore [2007].

6.6 The Non-deterministic Case

All of the previous sections have assumed that, given the positions of points, the corresponding graph could be inferred exactly. In terms of the `RHH` model introduced in section 6.2, this corresponds to setting $\alpha \to \infty$. In this section, we investigate the effects of finite α. Our analysis shows that while bounds become looser, the results are still qualitatively similar.

The core idea underlying almost all of our previous results has been the computation of the probability of two nodes i and j having a common neighbor. For the deterministic case, this is simply the area of intersection of two hyperspheres, $A(r, r, d_{ij})$, for the case when all nodes have the same radius r. However, in the non-deterministic case, this probability is hard to compute exactly. Instead, we can give the following simple bounds on $Pr_2(i, j)$, which is the probability of observing a common neighbor between two nodes i and j that are distance d_{ij} apart and have identical radius r.

Theorem 6.6.1.

$$Pr_2(i,j) \;>\; \frac{1}{4}\left(A(r,r,d_{ij}) + 2e^{-\alpha d_{ij}} \cdot \left(V(r) - A(r,r,d_{ij})\right)\right)$$

$$Pr_2(i,j) \;<\; \begin{cases} A(r,r,d_{ij}) + 2V(r) \cdot \dfrac{\left[1 - \left(\frac{D}{\alpha r}\right)^D\right]}{\frac{\alpha r}{D} - 1} & \textit{(for } \alpha r > D\textit{)} \\ A(r,r,d_{ij}) + 2D \cdot V(r) & \textit{(for } \alpha r = D\textit{)} \\ A(r,r,d_{ij}) + 2V(D/\alpha) \cdot \dfrac{\left[1 - \left(\frac{\alpha r}{D}\right)^D\right]}{1 - \frac{\alpha r}{D}} & \textit{(for } \alpha r < D\textit{)} \end{cases}$$

Proof. Details of the proof are given in appendix A.D. □

OBSERVATIONS AND EXTENSIONS. The importance of theorem 6.6.1 is that the probability of observing a common neighbor is still mostly dependent on the area of intersection of two hyperspheres, i.e. $A(r, r, d_{ij})$. However, there is a gap of a factor of 4 between the lower and upper bounds. This can still be used to obtain reasonable bounds on d_{ij} when

enough common neighbors are observed. However, when we consider longer paths, the gap increases and we might no longer be able to get strong bounds.

The reason for this is that theorem 6.6.1 only uses the fact that probability of linking i and j is at least $1/2$ when d_{ij} is less than r. This statement is applicable to all α. However, we typically want to perform link prediction only when α is large, as small values of α yield graphs that are close to random and where no link prediction methods would work. For the case of large α, we can get much stronger lower bounds and close the factor-of-4 gap, as follows.

In order to compute the probability $Pr_2(i,j)$, we need to integrate the product of the link probabilities over the intersection of the two hyperspheres of radius r around nodes i and j. Let this region be denoted by $S(i,j)$.Suppose that, instead of integrating over $S(i,j)$, we integrate over a smaller subset $S'(i,j)$. While the volume of $S'(i,j)$ would be smaller, the minimum probabilities inside that subset could be much higher, leading to a better overall lower-bound. We consider $S'(i,j) = \{x_k | d_{ik} < r', d_{jk} < r'\}$ to be the intersection of two hyperspheres of radius $r' < r$, centered on i and j. Then, using eq. (6.4) we have,

$$\begin{aligned}
&Pr_2(i,j,x_k \in S'(i,j)) \\
&\geq \left(\tfrac{1}{1+e^{\alpha(r'-r)}}\right)^2 \cdot \text{vol}(S'(i,j)) \\
&\geq \left(\tfrac{1}{1+e^{\alpha(r'-r)}}\right)^2 V(r') \int_0^{\cos^{-1}\left(\frac{d_{ij}}{2r}\right)} sin^D(t)dt
\end{aligned}$$

Ideally, we would like to pick r' to maximize this, but vol$(S'(i,j))$ depends on d_{ij} as well. Noting that and the effect of d_{ij} is restricted to the last term only, we propose the following formulation:

$$\text{Pick } r' \text{ to maximize } \left(\frac{1}{1+e^{\alpha(r'-r)}}\right)^2 \cdot V(r') \tag{6.11}$$

Lemma 6.6.2. *If $\alpha > D/r$, then $r' < r$.*

Thus, for large enough α, we can find a good r' which can improve the gap between upper and lower bounds of $Pr_2(i,j)$. The optimal r' gets closer to r as α increases, but its exact value has to be obtained numerically.

6.7 Summary and Discussion

In the previous chapters we have designed fast algorithms for computing random-walk based proximity measures for link prediction in large graphs. Some of our observations on behaviors of different graph-based proximity measures are aligned with previous empirical

studies. These observations, put together, bring us to this question. Is there a theoretical framework for explaining them? Ideally we should be able to connect characteristics of graphs with properties of the measures to reveal which measure will work well on which class of graphs. In order for that, we need a generative model for link-formation.

In this chapter, we use a simple generative model and present, to our knowledge the first theoretical study of link prediction and the heuristics commonly used for that purpose. We formalize the link prediction problem as one of estimating distances between nodes in a latent space, where the observed graph structure provides evidence regarding the unobserved positions of nodes in this space. We present theoretical justifications of two common empirical observations: (1) the simple heuristic of counting common neighbors often outperforms more complicated heuristics, (2) a variant that weighs common neighbors by the inverse of the logarithm of their degrees (Adamic and Adar [2003]) often performs better. We show that considering longer paths is useful only if shorter paths (especially, common neighbors) are not numerous enough for the bounds obtained from them to be tight enough. However, the bounds obtained from longer paths can be made significantly tighter if even a single short path is known to exist.

Chapter 7

Concluding Remarks

I have seen too much not to know that the impression of a woman may be more valuable than the conclusion of an analytical reasoner.
– Sir Arthur Conan Doyle's Sherlock Holmes.

In this thesis our main focus is on investigating some useful random walk based proximity measures. We have started with chapter 2, which builds necessary background for understanding random walk based measures of proximity; describes popular proximity measures, fast algorithms for computing them, and real world applications. The motivation behind our work can be derived from the array of interesting questions this chapter brings up. For example: using these measures can we design local ranking algorithms with formal guarantees? Can we enumerate the properties of a graph, which make local search inefficient? How can one handle graphs that are too large to be memory-resident? These questions, and the interesting directions that result from our solutions constitutes most of this thesis.

While we have addressed some computational and theoretical questions that arise in relevance-search in large graphs, this hardly ends the quest for the holy grail of personalized search. Rather, it opens up some natural directions for future research. First we will summarize the main results, and then conclude with our thoughts on future work.

7.1 Summary of Our Work

In chapter 3, we provide a fast local algorithm to compute nearest neighbors of a query node in truncated hitting and commute times with high probability. We investigate the compactness properties of these measures. In particular, we try to obtain an answer to this question: given a query node, is there a small neighborhood which contains other nodes in the graph that are significantly similar to the query? Our results indicate that

this is indeed true. We then show how to "discover" this neighborhood by pruning off most of the graph. Our algorithms bring together deterministic neighborhood expansion techniques and simple Monte Carlo sampling.

Although random walk based measures are often effective for relevance search in graphs, often their usefulness is curbed by ambiguous queries, and noisy data. One simple solution to this is to incorporate user feedback. In chapter 4, we use a short-term variant of harmonic functions for reranking nodes in a graph based on user feedback. Our algorithmic framework from chapter 3 is completely generalizable to this problem setting. Using quantifiable entity disambiguation tasks, we show the superiority of harmonic functions over ranking mechanisms which do not take negative information into account.

One large component of this thesis has been local search algorithms for ranking. One problem with local algorithms is posed by very high degree nodes. In real-world networks, these nodes are few in number (power-law), but easily reachable (small-world property). We show how a simple transformation of the high degree nodes in the graph can lead to much faster search, while incurring bounded error in some random walk based measures. Although the analysis is done for undirected graphs, we expect the intuition to be true even for directed graphs. We also show that discounted hitting times are closely related to personalized pagerank. In fact they preserve the interesting properties of personalized pagerank, while automatically normalize out the effect of individual popularity. The detailed analysis can be found in chapter 5.

The other two components of chapter 5 are focussed around ranking in disk-resident graphs. We propose a simple clustered representation of the graph on disk, and show that our framework of branch and bound algorithms carry over to this representation as well. We also present a clustering algorithm which only makes sequential sweeps of the external memory graph, and does not rely on random access to the memory.

So far, we have focussed on the computational aspect of random walk based proximity measures. These measures have been empirically shown to be effective for ranking entities in a graph. Essentially, they examine the paths from one node to another. Other popular path-based similarity measures are the number of common neighbors, Adamic/Adar, the length of the shortest path, etc. There has been much work on graph-based similarity search, both in terms of clever algorithms and new intuitively meaningful measures. In-depth experimental surveys have compared the effectiveness of existing measures in terms of link prediction, which has also been extensively used by us in this thesis. There are some interesting common trends in these surveys, in terms of which measure outperforms which other measure. The question is, what are there reasons behind these trends, and how do we formalize them? In chapter 6 we provide, to our knowledge the first formal connection between generative models for link-formation and useful graph-based heuristics for link prediction. Our analysis explains many empirical observations made by us, and other authors in previous studies.

7.2 Future Directions

For computing nearest neighbors using hitting and commute times, we proposed a branch and bound algorithm (chapter 3) which adaptively finds the right neighborhood. We saw that the main idea of this algorithm can be applied for computing ranking using many other random walk based measures, like the truncated variant of harmonic functions, discounted hitting times and personalized pagerank. It seems to be the case that, any measure, which considers short paths, and at any node, can be expressed as the average of the values of the neighbors of the node (shifted and scaled by constants), can be computed using this technique. For example, in case of the truncated harmonic function, the shift is zero, and the scaling is one. For truncated hitting times, the shift is one, and scaling constant is one. For discounted hitting times the shift is one and the scaling constant is the discount factor. For personalized pagerank the shift is zero, and the scaling factor depends on the restart probability. All these measures involve short term walks, either via truncation, or discounting, or the restart probability. It would be interesting to actually formalize the sufficient and necessary conditions to apply the GRANCH framework. This would let us add this framework to our grand arsenal of computational tools, so that we can use it as often as possible.

In chapter 4, we present an algorithm for re-ranking which refines the ranking by incorporating user-feedback. 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 learning problems. The question we want to ask is: find the smallest set of nodes, which would lead to most informative ranking, if labeled by an user. It would be interesting to pose this as a combinatorial problem, and obtain interesting approximation algorithms to quickly identify the set of nodes that we want to present to the user.

While in chapter 5 we have presented an approach to deal with high degree nodes, we believe that this can be improved further. For example, instead of turning all out-going edges of a high out-degree node into self-edges, one could alter only the low probability transitions. In fact, this leads us to another question. We did not utilize the term in the error which represents the personalized pagerank from the query node to the node to be deleted. Essentially, we ignored the fact that the definition of noisy/un-necessary links can depend on the query. For example, for a query on "soccer" might regard the links about "tennis" as un-necessary. This can also be seen from the basic intuition of HITS and SALSA. High out-degree nodes (hubs) are useful, when they are pointing to high indegree (authority) nodes, where the notion of hubs and authorities is query-specific.

Our clustering algorithm RWDISK (chapter 5) is currently not constrained to produce page-size clusters. In future work, we intend to experiment with a highly optimized implementation designed to respect true disk page size and hope to give results on graphs with billions of edges. One important future direction will be to characterize the tradeoffs

and limitations of algorithms in this external-memory setting. For example, in our case the number of page-faults depends on the number of cross-edges, the number of nodes in each page, the size of the graph, etc. Hence we intend to identify the formal relationship between these elements and apply that for obtaining better theoretical guarantees on runtime performance of our algorithms.

Finally, using a generative latent space model, we have shown how to explain some interesting behaviors of graph-based heuristics in relation to link prediction. We believe that our work is generalizable to other class of small-world graphs and it would be interesting to enumerate the exact relationship between the predictive power of different link prediction heuristics and the model parameters of other generative models.

Liben-Nowell and Kleinberg [2003] and Brand [2005] have empirically shown that hitting and commute times have some properties which curb their effectiveness for ranking. These properties are believed to stem from long range paths. It would be interesting to theoretically justify this observation. Zhu [2006] shows that there is a relationship between the popular regularization framework of semi-supervised learning on graphs with random walk based measures. Agarwal and Chakrabarti [2007] explore this connection for obtaining generalization guarantees for popular random walk based measures. It would be interesting to use this framework for explaining the above phenomenon, in addition to the different behavior of different measures for link prediction.

As mentioned in section 2.5, it is common practice to empirically evaluate different proximity measures using link prediction tasks. One interesting observation resulting from our experiments was: different proximity measures perform differently on *different tasks* on different graphs. We have tried to enumerate the relationship between properties of a graph with characteristics of different measures in the last chapter. Liben-Nowell and Kleinberg [2003] discuss this in detail, where the properties of different graphs are studied to better explain the behavior of different measures on link-prediction tasks. It would be also interesting to take into consideration the characteristics of the tasks. If we can theoretically connect the dots among properties of graphs, the nature of a prediction task, and the characteristics of different measures, then it 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 Facebook 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.

Appendix A

Appendix

A.A Proofs for Chapter 3

First we will present proofs for the theorems involving the GRANCH algorithm. We will prove some results first, which will be used to show that the optimistic bounds on hitting time can never decrease by expanding the neighborhood. We then also show that the pessimistic bounds never increase, which is much more complicated. We then prove the theorems for the sampling algorithm for computing hitting times.

A.A.1 Proofs for the GRANCH Algorithm

In order to prove the main theorem we will prove some results first. The main theorem will be proven using these results.

Lemma A.A.1. *For optimistic and pessimistic hitting times, we have the following result for all $i \in S$, where S is the local neighborhood for j in the APset:*

$$ho^T(i,j) \leq 1 + ho^{T-1}(i,j) \tag{A.1}$$
$$hp^T(i,j) \leq 1 + hp^{T-1}(i,j) \tag{A.2}$$

Proof. We will prove this for the optimistic bounds via induction. The proof for the pessimistic bounds is similar. Recall that $ho^1(i,j) = 1$, if $i \neq j$, and 0 otherwise. Also, $lb^1(S)$, i.e. the minimum time to get to j from outside S in one step, is 1. Now, $ho^2(i,j) = 1 + \sum_{k \in S} P(i,k) ho^1(k,j) + (1 - \sum_{k \in S}) lb^1(S)$. If there is an edge from $i \to j$, then $ho^2(i,j) = 2 - P(i,j)$, whereas if there is no such edge, then $ho^2(i,j) = 2$. Hence for $T = 1$ we have $ho^{T+1}(i,j) \leq 1 + ho^T(i,j)$.

For the inductive step, we have:

$$
\begin{aligned}
&ho^T(i,j) - ho^{T-1}(i,j) \\
&= \sum_{k\in S} P(i,k)\left[ho^{T-1}(k,j) - ho^{T-2}(k,j)\right] + (1-\sum_{k\in S} P(i,k))\left[lb^{T-1}(S) - lb^{T-2}(S)\right] \\
&\leq \sum_{k\in S} P(i,k) + (1-\sum_{k\in S} P(i,k))\left[lb^{T-1}(S) - lb^{T-2}(S)\right] \quad \text{By induction} \\
&\leq \sum_{k\in S} P(i,k) + (1-\sum_{k\in S} P(i,k))\left[\min_{m\in\delta(S)} ho^{T-2}(m,j) - \min_{m\in\delta(S)} ho^{T-3}(m,j)\right] \\
&\leq \sum_{k\in S} P(i,k) + (1-\sum_{k\in S} P(i,k))\left[\min_{m\in\delta(S)} (1+ho^{T-3}(m,j)) - \min_{m\in\delta(S)} ho^{T-3}(m,j)\right] \quad \text{By induction} \\
&\leq \sum_{k\in S} P(i,k) + (1-\sum_{k\in S} P(i,k)) \\
&= 1
\end{aligned}
$$

This completes the proof. □

Lemma A.A.2. *Let S is the neighborhood around j. Now node a is added to it to yield a new neighborhood S_1. Now, $ho^T(a,j) \geq 1 + \min_{m\in\delta(S)} ho^{T-1}(m,j)$.*

Proof. Using the definition of optimistic hitting times, we have

$$ho^T(a,j) = 1 + \sum_{k\in S} P(i,k) ho^{T-1}(k,j) + (1-\sum_{k\in S} P(i,k))\left[1 + \min_{m\in\delta(S_1)} ho^{T-2}(m,j)\right]$$

The above equation uses the fact that, the neighbors of k which are in S_1, are essentially in $\delta(S)$ (boundary nodes of S by definition). The other neighbors are outside S. Clearly, in the first part we have, $ho^{T-1}(k,j) \geq \min_{m\in\delta(S)} ho^{T-1}(m,j)$. For the second part, we have:

$$
\begin{aligned}
1 + \min_{m\in\delta(S_1)} ho^{T-2}(m,j) &= \min_{m\in\delta(S_1)} (1 + ho^{T-2}(m,j)) \\
&\geq \min_{m\in\delta(S_1)} ho^{T-1}(m,j) \quad \text{Using lemma A.A.1} \\
&= \min(ho^{T-1}(a,j), \min_{m\in\delta(S)} ho^{T-1}(m,j))
\end{aligned}
$$

Let $p = \sum_{k\in S} P(i,k)$, and $X = \min_{m\in\delta(S)} ho^{T-1}(m,j)$. Now we have

$$ho^T(a,j) \geq 1 + pX + (1-p)\min(ho^{T-1}(a,j), X) \tag{A.3}$$

We will show that $ho^{T-1}(a,j) \geq X$, which will complete the proof.

Assume $ho^{T-1}(a,j) \leq X$. We will now prove that this leads to $ho^{T-1}(a,j) = X$.

From eq. (A.3) we have,

$$ho^T(a,j) \geq 1 + pX + (1-p)ho^{T-1}(a,j)$$

Using lemma A.A.1 we have $1 + ho^{T-1}(a,j) \geq ho^T(a,j)$. Thus

$$\begin{aligned} &1 + ho^{T-1}(a,j) \geq 1 + pX + (1-p)ho^{T-1}(a,j) \\ &\Rightarrow ho^{T-1}(a,j) \geq X \end{aligned}$$

But our assumption was that $ho^{T-1}(a,j) \leq X$. Hence, it must be the case that $ho^{T-1}(a,j) = X$. Hence, we have $ho^{T-1}(a,j) \geq X$. □

PROOF OF PART 1 OF THEOREM 3.3.2. For a given neighborhood S, with boundary $\delta(S)$,

$$ho^T(i,j) = 1 + \sum_{k \in S} P(i,k) ho^{T-1}(k,j) + \left[1 - \sum_{k \in S} P(i,k)\right] lb^{T-1}(S) \tag{A.4}$$

Where

$$lb^T(S) = 1 + \min_{m \in \delta(S)} ho^{T-1}(m,j)$$

Now let $S_1 = S \cup \{A\}$, where A is the new set of node that was added to S. We will denote $ho^T_S(i,j)$ to denote hitting time from i to j computed on subset S. We want to show,

$$ho^T_{S_1}(i,j) \geq ho^T_S(i,j), \quad \forall i \in S$$

We will prove that by adding one node a to S, the optimistic bounds only increase. Adding a set of A nodes is equivalent to adding one node at a time. Since the optimistic bounds of the nodes already in S can never decrease, the result is proven for a set of nodes A.

We will prove this by induction.

Inductive assumption:

Assume that $\forall t < T, \forall i \in S, ho^t_{S_1}(i,j) \geq ho^t_S(i,j)$. We want to show that $ho^T_{S_1}(i,j) \geq ho^T_S(i,j)$.

Base case:

For $t = 1$ we have, $\forall i \neq j, ho^1_S(i,j) = 1$, and $ho^1_S(j,j) = 0$.These values do not change after adding nodes to S, because, in less than 1 step no other node than j itself can reach j. Now assume that S contains all incoming neighbors of j (this is how GRANCH

initializes S). It can be easily verified that $ho_S^2(i,j) = 2 - P(i,j)$. Now, by adding nodes to S, these values will not change, since all the new nodes will be 2 hops away from j.

Proof of Inductive step:

WLOG let a be a neighbor of i. We will break up the difference between $ho_{S_1} - ho_S$ in three parts, and prove that all three parts are non-negative. If a is not a direct neighbor of i, then part 3 will become zero.

We can rewrite eq. (A.4) as:

$$\begin{aligned} & ho_S^T(i,j) \\ &= 1 + \sum_{k \in S} P(i,k) ho_S^{T-1}(i,j) + \left[1 - \sum_{k \in S} P(i,k)\right] lb^{T-1}(S) \\ &= 1 + \sum_{k \in S} P(i,k) ho_S^{T-1}(i,j) + \left[1 - \sum_{k \in S_1} P(i,k)\right] lb^{T-1}(S) + P(i,a) lb^{T-1}(S) \end{aligned} \tag{A.5}$$

Now combining equations (A.5) and (A.4) on ho_{S1} we have,

$$\begin{aligned} ho_{S_1}^T(i,j) - ho_S^T(i,j) = & \sum_{k \in S} P(i,k) \left[ho_{S_1}^{T-1}(i,j) - ho_S^{T-1}(i,j)\right] && \text{Part 1} \\ & + (1 - \sum_{k \in S_1} P(i,k)) \left[lb^{T-1}(S_1) - lb^{T-1}(S)\right] && \text{Part 2} \\ & + P(i,a) \left[ho_{S_1}^{T-1}(a,j) - lb^{T-1}(S)\right] && \text{Part 3} \end{aligned}$$

Proof that part 1 is non-negative: Part 1 is larger than 0 via the induction. We will prove the same for part 2 and part 3.

Proof that part 2 is non-negative: Let $m^* = \text{argmin}_{m \in \delta(S_1)} ho_S^{T-2}(m,j)$. Now, if $m^* \in S$, then we have,

$$\begin{aligned} lb^{T-1}(S_1) - lb^{T-1}(S) &= ho_{S_1}^{T-2}(m^*,j) - \min_{m \in \delta(S)} ho_S^{T-2}(m,j) \\ &\geq ho_S^{T-2}(m^*,j) - \min_{m \in \delta(S)} ho_S^{T-2}(m,j) \quad \text{Via induction, since } m^* \in \delta(S) \\ &\geq 0 \quad \text{Since } m^* \in \delta(S). \end{aligned}$$

Now, if $m^* \notin S$, i.e. $m^* = a$, then we have the following:

$$
\begin{aligned}
& lb^{T-1}(S_1) - lb^{T-1}(S) \\
& = ho_{S_1}^{T-2}(a,j) - \min_{m \in \delta(S)} ho_S^{T-2}(m,j) \\
& \geq (ho_{S_1}^{T-1}(a,j) - 1) - \min_{m \in \delta(S)} ho_S^{T-2}(m,j) \quad \text{Lemma A.A.1} \\
& \geq \min_{m \in \delta(S)} ho_{S_1}^{T-2}(m,j) - \min_{m \in \delta(S)} ho_S^{T-2}(m,j) \quad \text{Using lemma A.A.2} \\
& \geq \min_{m \in \delta(S)} ho_S^{T-2}(m,j) - \min_{m \in \delta(S)} ho_S^{T-2}(m,j) \quad \text{Since } m \in S \text{, we can use the inductive assumption} \\
& \geq 0
\end{aligned}
$$

This proves that part 2 is non-negative.

Proof that part 3 is non-negative:

$$
\begin{aligned}
ho_{S_1}^{T-1}(a,j) - lb^{T-1}(S) &\geq 1 + \min_{m \in \delta(S)} ho_{S_1}^{T-2}(m,j) - lb^{T-1}(S) \quad \text{Using lemma A.A.2} \\
&\geq 1 + \min_{m \in \delta(S)} ho_S^{T-2}(m,j) - lb^{T-1}(S) \quad \text{Via induction since } m \in S \\
&= lb^{T-1}(S) - lb^{T-1}(S) \\
&\geq 0
\end{aligned}
$$

This completes the inductive step.

PROOF OF PART 2 OF THEOREM 3.3.2. For a given neighborhood S, with boundary $\delta(S)$,

$$
hp^T(i,j) = 1 + \sum_{k \in S} P(i,k) hp^{T-1}(k,j) + \left[1 - \sum_{k \in S} P(i,k) \right] (T-1) \tag{A.6}
$$

Now let $S_1 = S \cup \{A\}$, where A is the new set of node that was added to S. We will denote $hp_S^T(i,j)$ to denote hitting time from i to j computed on subset S. We want to show,

$$
hp_{S_1}^T(i,j) \leq hp_S^T(i,j), \quad \forall i \in S
$$

We will prove that by adding one node a to S, the pessimistic bounds only decrease. Adding a set of A nodes is equivalent to adding one node at a time. Since the pessimistic bounds of the nodes already in S can never increase, the result is proven for a set of nodes A.

We will prove this exactly as we proved the property of the optimistic bounds, i.e. via induction.

Inductive assumption

Assume that $\forall t < T, \forall i \in S, hp^t_{S_1}(i,j) \leq hp^t_S(i,j)$. We want to show that $hp^T_{S_1}(i,j) \leq hp^T_S(i,j)$.

Base case:

For $t = 1$ we have, $\forall i \neq j, hp^1_S(i,j) = 1$, and $hp^1_S(j,j) = 0$. These values do not change after adding nodes to S, because, in less than 1 step no walk from any other node than j itself can reach j. Now assume that S contains all incoming neighbors of j (this is how GRANCH initializes S). It can be easily verified that $hp^2_S(i,j) = 2 - P(i,j)$. Now, by adding nodes to S, these values will not change, since all the new nodes will be 2 hops away from j.

Proof of the inductive step: We will obtain this proof using similar ideas as before. WLOG let a be a neighbor of i. We will break up the difference between $hp_{S_1} - hp_S$ in three parts, and prove that all three parts are less than or equal to zero. Proving this for the pessimistic bounds is much easier, because by definition, part 2 will be zero. If a is not a direct neighbor of i, then part 3 will become zero.

We can rewrite eq. (A.6) as:

$$\begin{aligned} & hp^T_S(i,j) \\ & = 1 + \sum_{k \in S} P(i,k) hp^{T-1}_S(i,j) + \left[1 - \sum_{k \in S} P(i,k)\right](T-1) \\ & = 1 + \sum_{k \in S} P(i,k) hp^{T-1}_S(i,j) + \left[1 - \sum_{k \in S_1} P(i,k)\right](T-1) + P(i,a)(T-1) \end{aligned} \tag{A.7}$$

Now combining equations (A.7) and (A.6) on hp_{S_1} we have,

$$\begin{aligned} hp^T_{S_1}(i,j) - hp^T_S(i,j) = & \sum_{k \in S} P(i,k) \left[hp^{T-1}_{S_1}(i,j) - hp^{T-1}_S(i,j)\right] && \text{Part 1} \\ & + (1 - \sum_{k \in S_1} P(i,k)) \left[(T-1) - (T-1)\right] && \text{Part 2} \\ & + P(i,a) \left[hp^{T-1}_{S_1}(a,j) - (T-1)\right] && \text{Part 3} \end{aligned}$$

Part 1 is less than zero via induction. Clearly part 2 is zero. Also part 3 is less than zero, since $hp^t(a,j) \leq t$. Thus the inductive step is also proven.

A.A.2 Proofs for the Sampling Algorithm

PROOF OF THEOREM 3.4.1:. We provide a bound on the number of samples M required in order to give an ϵ-correct answer with probability $1 - \delta$. We denote the estimate of a

random variable x by $\hat{x}$ from now on. Let us denote the first arrival time at node u from node i on the r^{th} trial by $X^r(i,u)$. Define $X^r(i,u) = T$, if the path does not hit u on trial r.

Note that

$$\hat{h}^T(i,u) = \sum_r X^r(i,u)/M$$
$$E[\hat{h}^T(i,u)] = h^T(i,u)$$

$\{X^r(i,u) \in [1,T], r = 1:M\}$ are i.i.d. random variables. The Hoeffding bound gives the following:

$$P(|\hat{h}^T(i,u) - h^T(i,u)| \geq \epsilon T) \leq 2\exp(-\frac{2M(\epsilon T)^2}{T^2}) = 2\exp(-2M\epsilon^2)$$

Since we want the probability of error to be low, setting the above to be less than a small value δ gives us the desired lower bound of $\frac{1}{2\epsilon^2}\log(\frac{2}{\delta})$.

PROOF OF COROLLARY 3.4.2:. Now we want the probability of a bad estimate for any u to be low. We upper bound this error probability using union and Hoeffding bounds and set the upper bound to be less than a small value δ. Hence we have

$$P(\exists u \in \{1,\ldots,n\}, |\hat{h}^T(i,u) - h^T(i,u)| \geq \epsilon T) \leq 2n\exp(-2M\epsilon^2) \leq \delta$$

This gives the lower bound of $\frac{1}{2\epsilon^2}\log(\frac{2n}{\delta})$.

PROOF OF THEOREM 3.4.3:. Consider a sampled path of length T starting from i. We define $X^r(i,u)$ as before. For two arbitrary nodes u and v, WLOG let $h^T(i,u) > h^T(i,v)$.

The idea is to define a random variable whose expected value will be equal to $h^T(i,u) - h^T(i,v)$.

We define a random variable $Z^r = X^r(i,u) - X^r(i,v)$. $\{Z^r \in [-(T-1), T-1], r = 1:M\}$ are i.i.d. random variables. Note that $E(Z^r) = h^T(i,u) - h^T(i,v)$.

The probability that the ranking of u and v will be exchanged in the estimated h^T values from M samples can be given by:

$$P(\hat{h}^T(i,u) < \hat{h}^T(i,v)) = P(\sum_{r=1}^{M} Z_r/M < 0)$$

Using the Hoeffding bound, we can show that this probability is smaller than

$$P(\sum_{r=1}^{M} Z_r/M < 0)$$

$$\leq \exp\left(-2M(h^T(i,u) - h^T(i,v))^2/(2T)^2\right)$$

$$= \exp\left(-M(h^T(i,u) - h^T(i,v))^2/2T^2\right)$$

Let $v_1, v_2, \ldots, v_k$ be the top k neighbors of i in exact truncated hitting time.

$$Pr(\exists j \leq k, q > k, \hat{h}^T(i,v_j) > \hat{h}^T(i,v_q))$$

$$\leq \textstyle\sum_{j\leq k} \sum_{q>k} Pr(\hat{h}^T(i,v_j) > \hat{h}^T(i,v_q))$$

$$\leq \textstyle\sum_{j\leq k} \sum_{q>k} \exp(-\frac{M(h^T(i,v_q) - h^T(i,v_j))^2}{2T^2})$$

$$\leq nk \exp(-\frac{M(h^T(i,v_{k+1}) - h^T(i,v_k))^2}{2T^2})$$

Let $\alpha = h^T(i,v_{k+1}) - h^T(i,v_k)$. Setting the above probability to be less than δ gives us the desired lower bound of $\frac{2T^2}{\alpha^2}\log(nk/\delta)$ on M.

PROOF OF LEMMA 3.4.5:. Let S be a set of nodes. Let $q = \text{argmin}_{k\in S}\, h(j,k)$. Let $m = \text{argmin}_{k\in S}\, \hat{h}(j,k)$. We know that $h(j,q) \leq h(j,m)$, since q is the true minimum, and $\hat{h}(j,m) \leq \hat{h}(j,q)$, since m is the node which has the minimum estimated h-value. Using the sample complexity bounds from theorem 3.4.1, we have,

$$\hat{h}(j,m) \leq \hat{h}(j,q) \overset{w.h.p}{\leq} h(j,q) + \epsilon T \tag{A.8}$$

For the other part of the inequality we have,

$$\hat{h}(j,m) \overset{w.h.p}{\geq} h(j,m) - \epsilon T \geq h(j,q) - \epsilon T \tag{A.9}$$

Using equations (A.8) and (A.9) we get:

$$h(j,q) - \epsilon T \overset{w.h.p}{\leq} \hat{h}(j,m) \overset{w.h.p}{\leq} h(j,q) + \epsilon T$$

Using S to be the set of nodes outside the neighborhood of j yields lemma 3.4.5.

A.B Proofs for Chapter 4

PROOF OF THEOREM 4.5.1:. Denote the probability of an event by $Pr\{.\}$, and the transition matrix by P. $d(i)$ denotes the weighted degree of a node in an undirected graph. Define $S = \{i | f^T(i) \geq \alpha\}$, and L_p as the set of positive labels. We have,

$$
\begin{aligned}
f^T(i) &= Pr_i(\text{Hitting a '+' before a '}-\text{' in } T \text{ steps}) \\
&\leq Pr_i(\text{Hitting a '+' in } T \text{ steps}) \\
&= \sum_{t=1}^{T} Pr_i(\text{Hitting a '+' for the first time in exactly } t \text{ steps}) \\
&\leq \sum_{t=1}^{T} \sum_{p \in L_p} Pr_i(\text{Hitting } p \text{ in exactly } t \text{ steps}) \\
&\leq \sum_{t=1}^{T} \sum_{p \in L_p} P^t(i,p) \\
&\leq \sum_{p \in L_p} \frac{d(p)}{d(i)} P^t(p,i) \qquad \text{(A.10)}
\end{aligned}
$$

The final step of equation (A.10) is due to a straightforward use of the reversibility of a random walk in an undirected graph (Aldous and Fill [200X]). Now we will upper bound the total probability of hitting a positive label before a negative label from the set of nodes S.

$$
\begin{aligned}
\sum_{i \in S} f^T(i) &\leq \sum_{i \in S} \sum_{t=1}^{T} \sum_{p \in L_p} \frac{d(p)}{d(i)} P^t(p,i) \\
&\leq \frac{1}{min_{i \in S} d(i)} \sum_{t=1}^{T} \sum_{p \in L_p} d(p) \sum_{i \in S} P^t(p,i) \\
&\leq \frac{\sum_{p \in L_p} d(p)}{min_{i \in S} d(i)} T
\end{aligned}
$$

Combining the definition of S and the above equation, we get:

$$\alpha|S| \leq \frac{\sum_{p \in L_p} d(p)}{min_{i \in S} d(i)} T$$

$$\rightarrow |S| \leq \frac{\sum_{p \in L_p} d(p)}{min_{i \in S} d(i)} \frac{T}{\alpha}$$

A.C Proofs for Chapter 5

SYMMETRY OF DEGREE NORMALIZED PPV IN UNDIRECTED GRAPHS. This follows directly from the reversibility of random walks.

$$\begin{aligned} v_i(j) &= \alpha \sum_{t=0}^{\infty} (1-\alpha)^t P^t(i,j) \\ &= \frac{d_j}{d_i} \alpha \sum_{t=0}^{\infty} (1-\alpha)^t P^t(j,i) \\ \Rightarrow \frac{v_i(j)}{d_j} &= \frac{v_j(i)}{d_i} \end{aligned} \tag{A.11}$$

PROOF OF THEOREM 5.3.1. Personalized pagerank of a start distribution $\boldsymbol{r}$ can be written as

$$PPV(\boldsymbol{r}) = \alpha \boldsymbol{r} + \alpha \sum_{t=1}^{\infty} (1-\alpha)^t (P^T)^t \boldsymbol{r} = \alpha \boldsymbol{r} + (1-\alpha) PPV(P^T \boldsymbol{r}) \tag{A.12}$$

By turning node s into a sink, we are *only* changing the s^{th} row of P. We denote by $\boldsymbol{r_s}$ the indicator vector for node s. For $\boldsymbol{r} = \boldsymbol{r_s}$ we have personalized pagerank from node s. Essentially we are subtracting the entire s^{th} row of matrix P, i.e. $P^T \boldsymbol{r_s}$ and adding back $\boldsymbol{r_s}$.

This is equivalent to subtracting the matrix $\boldsymbol{v}\boldsymbol{u}^T$ from P, where $\boldsymbol{v}$ is $\boldsymbol{r_s}$ and $\boldsymbol{u}$ is defined as $P^T \boldsymbol{r_s} - \boldsymbol{r_s}$. This gives:

$$PPV(\boldsymbol{r}) = \alpha (I - (1-\alpha) P^T + (1-\alpha) \boldsymbol{u}\boldsymbol{v}^T)^{-1} \boldsymbol{r}$$

Let, $\mathcal{M} = I - (1-\alpha) P^T$. Hence $PPV(\boldsymbol{r}) = \alpha \mathcal{M}^{-1} \boldsymbol{r}$. A straightforward application of the Sherman Morrison lemma gives

$$\widehat{PPV}(\boldsymbol{r}) = PPV(\boldsymbol{r}) - \alpha(1-\alpha) \frac{\mathcal{M}^{-1} \boldsymbol{u}\boldsymbol{v}^T \mathcal{M}^{-1}}{1 + (1-\alpha) \boldsymbol{v}^T \mathcal{M}^{-1} \boldsymbol{u}} \boldsymbol{r}$$

Note that, $\mathcal{M}^{-1}\boldsymbol{u}$ is simply $1/\alpha[PPV(P^T\boldsymbol{r_s}) - PPV(\boldsymbol{r_s})]$ and $\mathcal{M}^{-1}\boldsymbol{r}$ is simply $1/\alpha PPV(\boldsymbol{r})$.

Also $\boldsymbol{v}^T PPV(\boldsymbol{r})$ equals $PPV(\boldsymbol{r}, s)$. Combining these facts with eq. (A.12) yields the following:

$$\widehat{PPV}(\boldsymbol{r}) = PPV(\boldsymbol{r}) - [PPV(\boldsymbol{r_s}) - \boldsymbol{r_s}]\frac{PPV(\boldsymbol{r}, s)}{PPV(\boldsymbol{r_s}, s)}$$

This leads to the element-wise error bound in theorem 5.3.1.

PROOF OF LEMMA 5.3.2. Recall that personalized pagerank from node i to node j can be written as

$$PPV(i,j) = \alpha I(i == j) + (1-\alpha)\sum_k P(i,k)PPV(k,j) \tag{A.13}$$

Also, from eq. 5.4, we have

$$fa_\alpha(i,j) = \begin{cases} (1-\alpha)\sum_k P(i,k)fa_\alpha(k,j) & \text{When } i \neq j \\ 1 & \text{Otherwise} \end{cases}$$

Now, when $i = j$, we have $PPV(j,j) = fa_\alpha(j,j)PPV(j,j)$. When $i \neq j$, substituting $PPV(k,j) = fa_\alpha(k,j)PPV(j,j)$ in the RHS of eq. A.13 we have

$$\begin{aligned} PPV(i,j) &= (1-\alpha)\sum_k P(i,k)fa_\alpha(k,j)PPV(j,j) \\ &= fa_\alpha(i,j)PPV(j,j) \end{aligned}$$

Since both PPV and fa_α are linear systems, this proves the lemma.

PROOF OF LEMMA 5.3.4. For proving this lemma, we use a series of sink node operations on a graph and upper bound each term in the sum. For $j \leq k$, $S[j]$ denotes the subset $\{s_1, ..., s_j\}$ of S. Also let $G \setminus S[j]$ denote a graph where we have made each of the nodes in $S[j]$ into a sink. $S[0]$ is the empty set and $G \setminus S[0] = G$. Since we do *not* change the outgoing neighbors of any node when we turn a node into a sink, we have $G \setminus S[j] = (G \setminus S[j-1]) \setminus s_j$. This leads to:

$$\begin{aligned} PPV^{G\setminus S[k-1]}(\boldsymbol{r}, i) - PPV^{G\setminus S[k]}(\boldsymbol{r}, i) &\leq \frac{PPV^{G\setminus S[k-1]}(s_k, i)PPV^{G\setminus S[k-1]}(\boldsymbol{r}, s_k)}{PPV(s_k, s_k)} \\ &\leq PPV^{G\setminus S[k-1]}(s_k, i) \end{aligned}$$

The last step can be obtained by combining linearity of personalized pagerank with lemma 5.3.2. Now, using a telescoping sum:

$$PPV^G(\boldsymbol{r}, i) - PPV^{G\setminus S[k]}(\boldsymbol{r}, i) \leq \sum_{j=1}^{k} PPV^{G\setminus S[j-1]}(s_j, i)$$

The above equation also shows that by making a number of nodes into sinks, the personalized pagerank value w.r.t any start distribution at a node can only decrease, which intuitively makes sense. Thus each term $PPV^{G\setminus S[k-1]}(s_k, i)$ can be upper bounded by $PPV^{G}(s_k, i)$, and $PPV^{G\setminus S[k-1]}(\boldsymbol{r}, s_k)$ by $PPV^{G}(\boldsymbol{r}, s_k)$. This gives us the following sum, which can be simplified using (A.11) as,

$$\sum_{j=1}^{k} PPV(s_j, i) = \sum_{j=1}^{k} \frac{d_i PPV(i, s_j)}{d(s_j)} \leq \frac{d_i}{\min_{s\in S} d_s}$$

The last step follows from the fact that $PPV(i, s_j)$, summed over j has to be smaller than one. This leads to the final result in lemma 5.3.4.

PROOF OF LEMMA 5.3.7. We have:

$$\widehat{\boldsymbol{x_t}} = \Psi_{\epsilon_t}(P^T \widehat{\boldsymbol{x_{t-1}}})$$

Also $\bar{\boldsymbol{\epsilon}}_t$ denotes the following difference:

$$\bar{\boldsymbol{\epsilon}}_t = P^T \widehat{\boldsymbol{x_{t-1}}} - \widehat{\boldsymbol{x_t}}$$

Let $y = P^T \hat{x}_{t-1}(i)$. Note that $\bar{\boldsymbol{\epsilon}}_t(i)$ is 0 if $y \geq \epsilon_t$, and is y if $y < \epsilon_t$. Since $\bar{\boldsymbol{\epsilon}}_t(i) \leq P^T \hat{x}_{t-1}(i), \forall t$.

ϵ_t has the property that its sum is at most 1, and the maximum element is at most ϵ_t.

This gives:

$$\begin{aligned}
\boldsymbol{x_t} - \hat{\boldsymbol{x}}_t &= \boldsymbol{x_t} - \Psi_{\epsilon_t}(P^T \widehat{\boldsymbol{x_{t-1}}}) \\
&= \boldsymbol{x_t} - (P^T \widehat{\boldsymbol{x_{t-1}}} - \bar{\boldsymbol{\epsilon}}_t) \\
&\leq \bar{\boldsymbol{\epsilon}}_t + P^T(\boldsymbol{x_{t-1}} - \widehat{\boldsymbol{x_{t-1}}})
\end{aligned}$$

PROOF OF LEMMA 5.3.8. Let us denote $\boldsymbol{E_t} = \boldsymbol{x_t} - \hat{\boldsymbol{x}}_t$ by the vector of errors accumulated *up to timestep* t. Note that this is always going to have non-negative entries.

$$\begin{aligned}
\boldsymbol{E_t} &\leq \bar{\boldsymbol{\epsilon}}_t + P^T \boldsymbol{E_{t-1}} \\
&\leq \bar{\boldsymbol{\epsilon}}_t + P^T \bar{\boldsymbol{\epsilon}}_{t-1} + (P^T)^2 \bar{\boldsymbol{\epsilon}}_{t-2} + ... \\
&\leq \sum_{r=1}^{t} (P^T)^{t-r} \bar{\boldsymbol{\epsilon}}_r
\end{aligned}$$

PROOF OF THEOREM 5.3.9. By plugging the result from lemma 5.3.8 into eq. (5.1), the

total error incurred in the PPV value is

$$
\begin{aligned}
\boldsymbol{E} &= |\boldsymbol{v} - \hat{\boldsymbol{v}}| \\
&\leq \sum_{t=1}^{\infty} \alpha(1-\alpha)^{t-1}(\boldsymbol{x_t} - \widehat{\boldsymbol{x}_t}) \\
&\leq \sum_{t=1}^{\infty} \alpha(1-\alpha)^{t-1}\boldsymbol{E_t} \\
&\leq \sum_{t=1}^{\infty} \alpha(1-\alpha)^{t-1}\sum_{r=1}^{t}(P^T)^{t-r}\bar{\boldsymbol{\epsilon}}_r \\
&\leq \sum_{r=1}^{\infty}\sum_{t=r}^{\infty} \alpha(1-\alpha)^{t-1}(P^T)^{t-r}\bar{\boldsymbol{\epsilon}}_r \\
&\leq \sum_{r=1}^{\infty}(1-\alpha)^{r-1}\sum_{t=r}^{\infty} \alpha(1-\alpha)^{t-r}(P^T)^{t-r}\bar{\boldsymbol{\epsilon}}_r \\
&\leq \sum_{r=1}^{\infty}(1-\alpha)^{r-1}\left[\sum_{t=0}^{\infty} \alpha(1-\alpha)^{t}(P^T)^{t}\right]\bar{\boldsymbol{\epsilon}}_r \\
&\leq \frac{1}{\alpha}\left[\sum_{t=0}^{\infty} \alpha(1-\alpha)^{t}(P^T)^{t}\right]\sum_{r=1}^{\infty} \alpha(1-\alpha)^{r-1}\bar{\boldsymbol{\epsilon}}_r
\end{aligned}
$$

where we use:

$$\epsilon_r = \epsilon_{r-1}/\sqrt{1-\alpha} = \epsilon/(\sqrt{\alpha-1})^{r-1}$$

Note that $\bar{\boldsymbol{\epsilon}}_r$ is a vector such that $|\bar{\boldsymbol{\epsilon}}_r|_\infty \leq \epsilon_r \leq \epsilon/(\sqrt{1-\alpha})^{r-1}$, and $|\bar{\boldsymbol{\epsilon}}_r|_1 \leq 1$.

Let $\bar{\boldsymbol{\epsilon}} = \sum_{r=1}^{\infty} \alpha(1-\alpha)^{r-1}\bar{\boldsymbol{\epsilon}}_r$. Clearly entries of $\bar{\boldsymbol{\epsilon}}$ sum to at most 1, and the largest entry can be at most $\alpha\sum_{r=1}^{\infty}(1-\alpha)^{r-1}\frac{\epsilon}{\sqrt{1-\alpha}^{r-1}} = \frac{\epsilon\alpha}{1-\sqrt{1-\alpha}}$.

Now we have $\boldsymbol{E} \leq \frac{1}{\alpha}PPV_\alpha(\bar{\boldsymbol{\epsilon}})$. The above is true because, by definition $\left[\sum_{t=0}^{\infty} \alpha(1-\alpha)^{t}(P^T)^{t}\right]\bar{\boldsymbol{\epsilon}}$ equals the personalized pagerank with start distribution $\bar{\boldsymbol{\epsilon}}$, and restart proba-

bility α. We will analyze this for an undirected graph:

$$\begin{aligned}
PPV_\alpha(\bar{\boldsymbol{\epsilon}}, i) &= \sum_j \sum_{t=0}^{\infty} \alpha(1-\alpha)^t (P^T)^t(i,j)\bar{\boldsymbol{\epsilon}}(j) \\
&\leq \max_j \bar{\boldsymbol{\epsilon}}(j) \sum_{t=0}^{\infty} \alpha(1-\alpha)^t \sum_j P^t(j,i) \\
&= \max_j \bar{\boldsymbol{\epsilon}}(j) \sum_{t=0}^{\infty} \alpha(1-\alpha)^t \sum_j \frac{d_i P^t(i,j)}{d_j} \\
&\leq \frac{d_i}{\delta} \max_j \bar{\boldsymbol{\epsilon}}(j) \sum_{t=0}^{\infty} \alpha(1-\alpha)^t \sum_j P^t(i,j) \\
&\leq \frac{d_i}{\delta} \max_j \bar{\boldsymbol{\epsilon}}(j) \sum_{t=0}^{\infty} \alpha(1-\alpha)^t \\
&\leq \frac{d_i}{\delta} \max_j \bar{\boldsymbol{\epsilon}}(j)
\end{aligned}$$

The fourth step uses the reversibility of random walks for undirected graphs, i.e. $d_i P^t(i,j) = d_j P^t(j,i)$, where d_i is the weighted degree of node i. δ is the minimum weighted degree. Thus we have $E(i) \leq \frac{d_i}{\delta} \frac{\epsilon}{1-\sqrt{1-\alpha}}$.

PROOF OF THEOREM 5.3.10. Let r be the start distribution, and v_t be the partial sum upto time t. We know that $\forall t \geq \texttt{maxiter}$, $x_t = (P^T)^{t-\texttt{maxiter}} x_{\texttt{maxiter}}$

$$\begin{aligned}
\boldsymbol{v} - \boldsymbol{v}_{\texttt{maxiter}} &= \sum_{t=\texttt{maxiter}+1}^{\infty} \alpha(1-\alpha)^{t-1} \boldsymbol{x}_{t-1} \\
&= (1-\alpha)^{\texttt{maxiter}} PPV_\alpha(\boldsymbol{x}_{\texttt{maxiter}})
\end{aligned}$$

PROOF OF LEMMA 5.3.11. Let $\hat{\boldsymbol{v}}$ be rounded PPV values when `maxiter` equals ∞, and $\hat{\boldsymbol{v}}_{\texttt{maxiter}}$ the rounded values from RWDISK. We have

$$\boldsymbol{v} - \hat{\boldsymbol{v}}_{\texttt{maxiter}} = (\boldsymbol{v} - \hat{\boldsymbol{v}}) + (\hat{\boldsymbol{v}} - \hat{\boldsymbol{v}}_{\texttt{maxiter}})$$

Using the same idea as that of theorem 5.3.10 with the fact that $\widehat{x_t(i)} \leq x_t(i), \quad \forall i$, we can upper bound the difference $\hat{\boldsymbol{v}} - \hat{\boldsymbol{v}}_{\texttt{maxiter}}$ by $(1-\alpha)^{\texttt{maxiter}} PPV_\alpha(\boldsymbol{x}_{\texttt{maxiter}})$. This, combined with theorem 5.3.9 gives the desired result.

A.D Proofs for Chapter 6

PROOF OF THEOREM 6.3.2. Define $\epsilon_o = \sqrt{\frac{2\texttt{var}_N(Y_{OPT})\log 2/\delta}{N}} + \frac{7\log 2/\delta}{3(N-1)}$, $\epsilon_m = \sqrt{\frac{2\texttt{var}_N(Y_{MAX})\log 2/\delta}{N}} + \frac{7\log 2/\delta}{3(N-1)}$, and $\epsilon_f = \epsilon_o + \epsilon_m$. We want to prove:

$$d_{OPT} \leq d_{MAX} \overset{w.h.p}{\leq} d_{OPT} + \left(\frac{2\epsilon_f}{V(1)}\right)^{1/D}$$

The lower bound follows from the definition of d_{OPT}. For the upper bound, define the function $A^{-1}(y) = d_{ij}$ s.t. $A(r, r, d_{ij}) = y$. Note that this function is well-defined since $A(.)$ is a monotonic function of d_{ij}. Using Leibniz's rule on equation 6.4 and some algebraic manipulation, we get

$$\frac{dA^{-1}(y)}{dy} = -\left[\frac{C_D r^D}{2r}\left(1 - \frac{\left(A^{-1}(y)\right)^2}{4r^2}\right)^{\frac{D-1}{2}}\right]^{-1}$$

Thus, the slope of $A^{-1}(y)$ is always negative, (and hence $A^{-1}(.)$ is monotonically decreasing), and has the highest magnitude when $A^{-1}(y)$ is largest, i.e., when $A^{-1}(y) = 2r$. Now,

$$\begin{aligned} d_{MAX} - d_{OPT} &= A^{-1}(A_{MAX}) - A^{-1}(A_{OPT}) \\ &\leq 2r - A^{-1}\left(A_{OPT} - A_{MAX}\right) \\ &\leq 2r - A^{-1}(\epsilon_f) \quad \text{(from Theorem 6.3.1)} \end{aligned} \tag{A.14}$$

Let $d = A^{-1}(\epsilon_f)$. Then,

$$\begin{aligned} \epsilon_f &= A(r, r, d) \geq V(r)\left(1 - \frac{d}{2r}\right)^D \\ \Rightarrow d &\geq 2r\left(1 - \left(\frac{\epsilon_f}{V(r)}\right)^{1/D}\right) \end{aligned} \tag{A.15}$$

Substituting eq. (A.15) in eq. (A.14) we get the desired result.

PROOF OF THEOREM 6.6.1. We want to prove:

$$
\begin{aligned}
Pr_2(i,j) &> \frac{1}{4}\left(A(r,r,d_{ij}) + 2e^{-\alpha d_{ij}} \cdot (V(r) - A(r,r,d_{ij}))\right) \\
Pr_2(i,j) &< \begin{cases} A(r,r,d_{ij}) + 2V(r) \cdot \dfrac{\left[1 - \left(\dfrac{D}{\alpha r}\right)^D\right]}{\dfrac{\alpha r}{D} - 1} & (\text{for } \alpha r > D) \\ A(r,r,d_{ij}) + 2D \cdot V(r) & (\text{for } \alpha r = D) \\ A(r,r,d_{ij}) + 2V(D/\alpha) \cdot \dfrac{\left[1 - \left(\dfrac{\alpha r}{D}\right)^D\right]}{1 - \dfrac{\alpha r}{D}} & (\text{for } \alpha r < D) \end{cases}
\end{aligned}
$$

We use the following facts. When $x \leq 0$, $1/2 \leq 1/(1+e^x) < 1$, and when $x \geq 0$, $e^{-x}/2 \leq 1/(1+e^x) < e^{-x}$. Now, $Pr_2(i,j)$ is the probability that a randomly generated point k with position x_k links to both i and j. Let us divide the range of x_k into four parts:

(a) part $S(i,j) = \{x_k | d_{ik} \leq r, d_{jk} \leq r\}$,

(b) part $S(i,\neg j) = \{x_k | d_{ik} \leq r\} \setminus S_{i,j}$,

(c) part $S(\neg i,j) = \{x_k | d_{jk} \leq r\} \setminus S_{i,j}$,

(d) part $S(\neg i,\neg j)$ elsewhere.

We will first prove the lower bound and then the upper bound.

LOWER BOUND ON $Pr_2(i,j)$. Let $p(x_k)$ be the p.d.f (probability density function) of point k being at position x_k, and $Pr_2(i,j,x_k)$ be the probability that $i \sim k \sim j$ and that k is at position x_k. We have:

$$
\begin{aligned}
Pr_2(i,j,x_k \in S(i,j)) &= \int_{x_k \in S(i,j)} \frac{1}{1+e^{\alpha(d_{ik}-r)}} \frac{1}{1+e^{\alpha(d_{jk}-r)}} p(x_k) dx_k \quad \text{(A.16)} \\
&\geq \frac{1}{4} \int_{x_k \in S(i,j)} p(x_k) dx_k \geq \frac{1}{4} A(r,r,d_{ij}) \quad \text{(A.17)}
\end{aligned}
$$

where $A(r,r,d_{ij}) = \int_{x_k \in S(i,j)} p(x_k) dx_k$ is the volume of intersections of two hyperspheres of radius r centered at i and j. The second-last step follows from both $d_{ik} - r$ and $d_{jk} - r$

being less than zero. Similarly,

$$
\begin{aligned}
& Pr_2(i, j, x_k \in S(i, \neg j)) \\
& = \int_{x_k \in S(i,\neg j)} \frac{1}{1 + e^{\alpha(d_{ik}-r)}} \frac{1}{1 + e^{\alpha(d_{jk}-r)}} p(x_k) dx_k \\
& > \frac{1}{4} \int_{\{x_k | d_{ik} \leq r, d_{jk} > r\}} e^{-\alpha(d_{jk}-r)} p(x_k) dx_k \\
& > \frac{1}{4} \int_{\{x_k | d_{ik} \leq r, d_{jk} > r\}} e^{-\alpha(d_{ij}+r-r)} p(x_k) dx_k \quad \text{(because } d_{jk} < d_{ik} + d_{ij} < d_{ij} + r\text{)} \\
& > \frac{1}{4} \cdot e^{-\alpha d_{ij}} \cdot (V(r) - A(r, r, d_{ij}))
\end{aligned}
$$

since $\int_{x_k \in S(i,\neg j)} p(x_k) dx_k = V(r) - A(r, r, d_{ij})$. The same formula holds for part $S(\neg i, j)$. So, even without considering part $S(\neg i, \neg j)$, we have:

$$
Pr_2(i, j) > \frac{1}{4} \left(A(r, r, d_{ij}) + 2e^{-\alpha d_{ij}} \cdot (V(r) - A(r, r, d_{ij})) \right)
$$

UPPER BOUND ON $Pr_2(i, j)$. We will use the following facts. The volume of a hypersphere of radius x is given by $V(x) = c_D x^D$ where c_D is some constant that depends on the dimension D. Hence, the probability density function for a point k being at distance x from a given point j is $\frac{d}{dx} V(x) = c_D \cdot D \cdot x^{D-1}$.

We have:

$$
\begin{aligned}
Pr_2(i, j, x_k \in S(i, j)) &= \int_{x_k \in S(i,j)} \frac{1}{1 + e^{\alpha(d_{ik}-r)}} \frac{1}{1 + e^{\alpha(d_{jk}-r)}} p(x_k) dx_k \\
&< \int_{x_k \in S(i,j)} 1 \cdot p(x_k) dx_k < A(r, r, d_{ij})
\end{aligned}
\tag{A.18}
$$

$$Pr_2(i,j,x_k \in S(i,\neg j) \cup S(\neg i,\neg j))$$

$$< \int\limits_{\{x_k | d_{jk} > r\}} \frac{1}{1+e^{\alpha(d_{jk}-r)}} p(x_k)dx_k < \int\limits_{\{x_k | d_{jk} > r\}} e^{-\alpha(d_{jk}-r)} p(x_k)dx_k$$

$$< \int\limits_{d_{jk} > r} e^{-\alpha(d_{jk}-r)} c_D \cdot D \cdot (d_{jk})^{D-1} d(d_{jk})$$

$$< \frac{c_D D e^{\alpha r}}{\alpha^D} \int\limits_{y=\alpha r}^{\infty} e^{-\alpha y} y^{D-1} dy \quad \text{(using } y = \alpha d_{jk}\text{)}$$

$$< \frac{c_D D e^{\alpha r}}{\alpha^D} \Gamma(D, \alpha r) \quad (\Gamma \text{ is the upper incomplete-gamma function})$$

$$< \frac{c_D D!}{\alpha^D} \sum_{k=0}^{D-1} \frac{(\alpha r)^k}{k!} \quad \text{(expansion of } \Gamma(.) \text{ function)}$$

$$< \frac{c_D D^D}{\alpha^D} \sum_{k=0}^{D-1} \left(\frac{\alpha r}{D}\right)^k \quad \text{(using } D!/k! < D^{D-k}\text{)}$$

$$< \begin{cases} V(r) \cdot \dfrac{\left[1 - \left(\dfrac{D}{\alpha r}\right)^D\right]}{\dfrac{\alpha r}{D} - 1} & \text{(for } \alpha r > D\text{)} \\ D \cdot V(r) & \text{(for } \alpha r = D\text{)} \\ V(D/\alpha) \cdot \dfrac{\left[1 - \left(\dfrac{\alpha r}{D}\right)^D\right]}{1 - \dfrac{\alpha r}{D}} & \text{(for } \alpha r < D\text{)} \end{cases}$$

A similar formula holds for $Pr_2(i,j,x_k \in S(\neg i,j) \cup S(\neg i,\neg j)$. Summing over all of these gives an upper bound on $Pr_2(i,j)$.

Bibliography

Jacob Abernethy, Olivier Chapelle, and Carlos Castillo. Web spam identification through content and hyperlinks. In *AIRWEB '08*, 2008. ISBN 978-1-60558-085-2. 4.2

Lada A. Adamic and Eytan Adar. Friends and neighbors on the web. *Social Networks*, 25, 2003. 2.2.2, 6.1, 6.2.1, 6.4, 6.7

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. 2.2.3, 7.2

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. 2.4.2

David Aldous and James Allen Fill. *Reversible Markov Chains and Random Walks on Graphs*. Book in preparation, `http://www.stat.berkeley.edu/~aldous/book.html`, 200X. 2.1, 2.2, 2.2.1, 3.2.3, 5.2, 6.2.1, A.B

Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, 2006a. 2.2.4, 2.2.4, 2.3.2, 5.1, 5.2, 5.2, 5.3, 5.3.3, 5.3.3, 5.4.3

Reid Andersen, Fan Chung, and Kevin Lang. Local Partitioning for Directed Graphs Using PageRank. *Algorithms and Models for the Web-Graph*, 2006b. 2.2.4

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. 2.3.3

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. 2.2.4

A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-based keyword search in databases. In *VLDB*, 2004. URL `citeseer.ist.psu.edu/balmin04objectrank.html`. (document), 1.2, 2.1, 2.3.2, 2.4, 2.4.2, 4, 3.5.2, 1

P. Berkhin. Bookmark-Coloring Algorithm for Personalized PageRank Computing. *Internet Mathematics*, 2006. 2.1, 2.3.2, 5.1, 5.2, 5.3.1, 5.3.3

Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7), 1970. 2.2.1

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. 5

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*, 2001. 2.2.1

M. Brand. A Random Walks Perspective on Maximizing Satisfaction and Profit. In *SIAM '05*, 2005. (document), 1.1, 1, 1, 2.1, 2.2.1, 2.3.1, 2.4.3, 2.5.1, 1, 3.1, 3.1, 5.2, 6.1, 6.2.1, 6.5, 7.2

S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proc. WWW*, 1998. 2.2.1

Andrei Z. Broder. On the resemblance and containment of documents. In *In Compression and Complexity of Sequences (SEQUENCES97*, pages 21–29. IEEE Computer Society, 1997. 2.2.2

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. 2.2.1

Wray L. Buntine. Operations for learning with graphical models. *Journal of Artificial Intelligence Research*, 2:159–225, 1994. 2.4.2

Soumen Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *Proc. WWW*, New York, NY, USA, 2007. 2.1, 2.3.2, 2.4, 2.4.2, 3, 3.5.2, 3.5.2, 3.6, 1, 4.6, 5.1, 5.2, 5.3.1

Soumen Chakrabarti and Alekh Agarwal. Learning parameters in entity relationship graphs from ranking preferences. In *In ECML/PKDD, volume 4213 of LNCS*, pages 91–102, 2006. 3.6

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. 5.1

A. K. Chandra, P. Raghavan, W. L. Ruzzo, and R. Smolensky. The electrical resistance of a graph captures its commute and cover times. In *ACM Symposium on Theory of Computing (STOC)*, pages 574–586, 1989. 2.2.1

Yen-Yu Chen, Qingqing Gan, and Torsten Suel. Local methods for estimating pagerank values. In *Proceedings of the Conference on information and knowledge management*, 2004. 5.1

Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessandro Panconesi, and Prabhakar Raghavan. On compressing social networks. In *KDD '09*, 2009. 5.1

Fan Chung. Laplacians and the cheeger inequality for directed graphs. *Annals of Combinatorics*, 2005. 1

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. 2.4.2

Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005. 2.3.2

Bhavana Bharat Dalvi, Meghana Kshirsagar, and S. Sudarshan. Keyword search on external memory data graphs. *Proc. VLDB Endow.*, 2008. 5.1

P. G. Doyle and J. L. Snell. *Random Walks and Electric Networks.* The Mathematical Assoc. of America., 1984. 2.2.1, 2.2.1, 2.2.3

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. 2.2.1

Katherine Faust. Comparison of methods for positional analysis: Structural and general equivalences. *Social Networks*, 1988. 6.2.2

D. Fogaras and B. Rácz. Scaling link-based similarity search. In *Proceedings of the 14th Int'l World Wide Web Conference*, 2005. 2.3.2

D. Fogaras, B. Rcz, K. Csalogny, and Tams Sarls. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics*, 2004. 2.1, 2.3.2, 2.4, 7, 3.4.1, 5.1, 5.2, 5.3, 5.3.1

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. 2.2.1, 2.4.3, 1

Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 2003. 4.2

Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999. 2.4.2

David Gibson, Jon Kleinberg, and Prabhakar Raghavan. Clustering categorical data: an approach based on dynamical systems. *The VLDB Journal*, 8(3-4), 2000. 2.2.1

L. Gorelick, M. Galun, E. Sharon, R. Basri, and A. Brandt. Shape representation and classification using the poisson equation. In *CVPR*, 2004. 2.4, 2.4.1

J.C. Gower. A general coefficient of similarity and some of its properties. *Biometrics*, 27: 857–871, 1971. 2.2.2

Leo Grady. Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11), 2006. 2.1, 2.4.1, 4.1

Leo Grady and Eric L. Schwartz. Isoperimetric graph partitioning for data clustering and image segmentation. In *IEEE PAMI*, 2006. 2.4.1

Fan Chung Graham and Wenbo Zhao. Pagerank and random walks on graphs. In *"Fete of Combinatorics"*. 5.2

Zoltán Gyöngyi, 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. 2.4.4, 4.1, 4.2

Zoltn Gyongyi, Hector Garcia-Molina, and Jan Pedersen. Combating web spam with trustrank. In *VLDB*, pages 576–587, 2004. 2.4

David Harel and Yehuda Koren. On clustering using random walks. In *Foundations of Software Technology and Theoretical Computer Science 2245*. Springer-Verlag, 2001. 2.2.4

T. Haveliwala. Topic-sensitive pagerank. In *Proceedings of the Eleventh International World Wide Web Conference*, 2002. 2.1, 2.3.2, 2.4, 7

David Heckerman, Chris Meek, and Daphne Koller. *Probabilistic Entity-Relationship Models, PRMs, and Plate Models*, chapter 7. The MIT Press, 2004. 2.4.2

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. 7

John Hopcroft and Daniel Sheldon. Manipulation-resistant reputations using hitting time. Technical report, Cornell University, 2007. 2.1, 5.2

G. Jeh and J. Widom. Scaling personalized web search. In *Stanford University Technical Report*, 2002a. (document), 2.1, 2.3.2, 2.3.2, 2.1, 2.4, 7, 5.2, 5.2, 5.3.1, 5.3.2, 6.2.1

Glen Jeh and Jennifer Widom. Simrank: A measure if structural-context similarity. In *ACM SIGKDD*, 2002b. 2.2.1

Rong Jin, Hamed Valizadegan, and Hang Li. Ranking refinement and its application to information retrieval. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, 2008. ISBN 978-1-60558-085-2. 4.1, 4.2

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. 2.4.2

W. Johnson and J. Lindenstrauss. Extensions of lipschitz maps into a hilbert space. *Contemporary Mathematics*, 26:189–206, 1984. 2.3.1, 3.1

Amruta Joshi, Ravi Kumar, Benjamin Reed, and Andrew Tomkins. Anchor-based proximity measures. In *International World Wide Web Conference*, 2007a. 5.2

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*, 2007b. 2.4.4, 4.1, 4.2

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. 2.2.1

George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal of Scientific Computing*, 20(1):359–392, 1998. 5.1, 5.4.3

L. Katz. A new status index derived from sociometric analysis. In *Psychometrika*, 1953. 2.1, 2.2.2, 2.4, 2.5.1, 6.1

J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *ACM Symposium on Theory of Computing (STOC)*, 2000. 1, 6.2.2

Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5), 1999. 2.2.1, 2.2.1

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. 2.2.1

Ioannis Koutis and Gary L. Miller. A linear work, $o(n^{1/6})$ time, parallel algorithm for solving planar laplacians. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007. 4.2

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. 2.4.2

Amy N. Langville and Carl D. Meyer. Deeper inside pagerank. *Internet Mathematics*, 2003. 2.2.1

R. Lempel and S. Moran. Salsa: the stochastic approach for link-structure analysis. *ACM Trans. Inf. Syst.*, 19(2), 2001. 2.2.1, 3

Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. *ACM Transactions on Graphics*, 23:689–694, 2004. (document), 2.1, 2.4.1, 4.1, 4.2

David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *CIKM*, 2003. 1, 2.2.1, 2.2.2, 2.2.2, 2.3.1, 2.5.1, 5, 3.2.1, 3.5.1, 5.2, 6.1, 6.2.1, 6.4, 6.5, 6.5, 7.2

László Lovász. Random walks on graphs: a survey. *Combinatorics, Paul Erdos is Eighty, J. Bolyai Math. Soc.,Vol. II*, 1996. 2.2.1, 3.2.1

Andreas Maurer and Massimiliano Pontil. Empirical bernstein bounds and sample-variance penalization. In *Conference on Learning Theory*, 2009. 6.3

Colin McDiarmid. On Method of Bounded Differences. *Surveys in Combinatorics*, 1989. 6.5

D. D. McFarland and D. J. Brown. Social distance as a metric: a systematic introduction to smallest space analysis. *Bonds of Pluralism: The Form and Substance of Urban Social Networks*, 1973. 6.2.2

Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 2001. 6.2.2

Qiaozhu Mei, Dengyong Zhou, and Kenneth Church. Query suggestion using hitting time. In *CIKM '08*, 2008. 2.1, 2.3.1, 2.4.2, 3.1, 5.2

Marina Meila and Jianbo Shi. A random walks view of spectral segmentation. In *AISTATS*, 2001. 2.2.4, 5.2

S. Milgram. The small world problem. In *Psychology Today*, 1967. 3.2.3

Einat Minkov, William W. Cohen, and Andrew Y. Ng. Contextual search and name disambiguation in email using graphs. In *SIGIR '06*, 2006. 2.4.2

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. 2.2.1

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. 2.2.1

A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, 2001. URL `citeseer.ist.psu.edu/ng01spectral.html`. 2.2.4

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. 2.4.4, 4.2

Huaijun Qiu and Edwin R. Hancock. Image segmentation using commute times. In *Proc. BMVC*, 2005. 2.4, 2.4.1

Huaijun Qiu and Edwin R. Hancock. Robust multi-body motion tracking using commute time clustering. In *ECCV'06*, 2006. 2.4.1

A. E. Raftery, M. S. Handcock, and P. D. Hoff. Latent space approaches to social network analysis. *J. Amer. Stat. Assoc.*, 15:460, 2002. 6.1, 6.2.2

Prabhakar Raghavan. Social networks: From the web to the enterprise. *IEEE Internet Computing*, 2002. 6.2.1

C. Rao and S.Mitra. *Generalized inverse of matrices and its applications*. John Wiley and Sons, 1971. 2.2.1

M. Saerens, F. Fouss, L. Yen, and P. Dupont. The principal component analysis of a graph and its relationships to spectral clustering, 2004. URL `citeseer.ist.psu.edu/saerens04principal.html`. 2.1, 2.2.4, 2.3.1, 3.1

Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986. ISBN 0070544840. 2.2.2

Purnamrita Sarkar and Andrew Moore. A tractable approach to finding closest truncated-commute-time neighbors in large graphs. In *Proc. UAI*, 2007. 1, 6.2.1, 6.5

Purnamrita Sarkar and Andrew Moore. Fast nearest-neighbor search in disk-resident graphs. Technical report:10-005, Machine Learning Department, Carnegie Mellon University, 2010. 1

Purnamrita Sarkar and Andrew W. Moore. Fast dynamic reranking in large graphs. In *International World Wide Web Conference*, 2009. 1

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. 1

Purnamrita Sarkar, Deepayan Chakrabarti, and Andrew Moore. Theoretical justification of popular link prediction heuristics. In *Conference on Learning Theory*, 2010. 1

Tamás Sarlós, Adrás A. Benczúr, Károly Csalogány, Dániel Fogaras, and Balázs Rácz. To randomize or not to randomize: space optimal summaries for hyperlink analysis. In *WWW*, 2006. 2.1, 2.3.2, 7, 5.1, 5.2, 5.3, 5.3.1, 5.3.2, 5.3.3

Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. Estimating pagerank on graph streams. In *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2008. 5.1, 5.2, 5.3

J. Schroeder, J. J. Xu, and H. Chen. Crimelink explorer: Using domain knowledge to facilitate automated crime association analysis. In *ISI*, pages 168–180, 2003. 6.2.1

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. 2.2.4

A. Smola and R. Kondor. Kernels and regularization on graphs. In Springer Verlag, editor, *Learning Theory and Kernel Machines*, 2003. URL `citeseer.ist.psu.edu/smola03kernels.html`. 2.2

D. Spielman and N. Srivastava. Graph sparsification by effective resistances. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2008. 2.3.1, 3.1

D. Spielman and S. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2004. 2.2.4, 2.2.4, 2.3.1, 3.1, 4.1, 4.2, 5.2, 5.2, 5.3, 5.4.3

Martin Szummer and Tommi Jaakkola. Partially labeled classification with markov random walks. In *Advances in Neural Information Processing Systems*, volume 14, 2001. 2.2.3

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. 2.4.2

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. 2.2.1

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. 2.4.2

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. 2.2.1, 2

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. 2.2.1

Jeffrey D. Ullman and Jennifer Widom. *A first course in database systems*. Prentice-Hall, Inc., 1997. ISBN 0-13-861337-0. 2.4.2

Baoning Wu and Kumar Chellapilla. Extracting link spam using biased random walks from spam seed sets. In *WWW*, 2007. 2.4

Zuobing Xu, Ram Akella, and Yi Zhang. Incorporating diversity and density in active learning for relevance feedback. In *ECIR*, 2007. 4.6.6

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. URL `citeseer.ist.psu.edu/yen05clustering.html`. 2.2.4

X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML, volume 20*, 2003. (document), 1, 2.1, 2.2.3, 2.2.3, 2.4.2, 4.1, 4.1

Xiaojin Zhu. *Semi-supervised learning with graphs*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2005. Chair-Lafferty, John and Chair-Rosenfeld, Ronald. 2.3.3

Xiaojin Zhu. Semi-supervised learning literature survey, 2006. 2.2, 2.2.3, 2.2.4, 2.3.3, 4.2, 7.2

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. 2.3.3

Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Carnegie Mellon