Efficient Computation of First-Order Markov Measures

on Large Evolving Graphs

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Prasanna K. Desikan

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Dr. Jaideep Srivastava, Adviser

January 2009

DEDICATION

To my parents,
Kalyani and Desikan

To my wife,
Smitha

ACKNOWLEDGEMENTS

This dissertation has required a tremendous amount of motivation, patience, support and appropriate guidance. I am very grateful and thankful to my adviser, Professor Jaideep Srivastava, who was instrumental in providing the required motivation, support and expert advice at every stage of this dissertation. He has been a great source of inspiration, encouragement, and support through the best and the toughest times. It has been an extreme pleasure to work with him. His expert views on academic and industry research always provided insights and stressed a broader scope of research. Such a perspective had a positive influence on my capacity to identify problems and propose solutions. His skill in providing the right guidance at appropriate times has ensured continuous research progress. I would strongly recommend every student interested in pursuing a graduate study in his areas of interest to consider him as an adviser.

I would also like to thank Professor Vipin Kumar for participating in and providing great ideas and feedback during the initial stages of my research. His support and encouragement have been tremendous. His availability during late hours at Army Center for research discussions was very helpful.

I would like to thank my committee members, Professor John Carlis and Professor Richard McGehee, who provided critical feedback during my preliminary oral exam and final dissertation, enabling me to make it more polished and complete.

I would like to make special mention of Professor Ravi Janardan. I thank him greatly for motivating me to pursue graduate study in the Department of Computer Science, University of Minnesota. I have always appreciated his selfless advice to expose myself to a wider variety of topics within computer science and to explore my interests before deciding to work in his area of interest. Having a mechanical engineering background

similar to his, I was initially tempted to pursue research in rapid prototyping. Without his selfless advice, I would probably be writing a different dissertation.

I would like to also thank my colleagues at the various stages of my doctoral degree program: Pang Ning Tan, Hui Xiong, Sandeep Mane, Colin Delong, and Nishith Pathak. They have supported me and played a valuable role by participating in research discussions and providing feedback

Special thanks to Georganne Tolaas and Liz Freppert, who have always been willing to answer questions and provide appropriate administrative support in a timely fashion to help complete the various administrative requirements for a doctoral degree.

I would like to thank my family for encouraging me to pursue this doctoral degree and providing me with the support and motivation needed to complete it. I would also like to thank all my friends from Osmania University (Rowdies), University of New Hampshire and University of Minnesota for their support and encouragement.

Finally, the completion of my dissertation would not have been possible without the love and support of my wife, Smitha. I am extremely grateful to her for her patience, motivation, and faith, which have helped me immensely, especially during the last two years of work on this dissertation.

ABSTRACT

A useful information infrastructure can be created from various types of data by modeling the set of data objects as a set of nodes, and the relationship (e.g. transition or interaction) between them as a set of links. Such a representation is commonly referred to as a link graph, and the analytical techniques associated with such representations are called link analysis techniques. These techniques have played a crucial role in improving applications such as Web searches, Web-based social networks, and cyber security. Link analysis techniques have been successfully applied to derive popular measurements such as PageRank, TrustRank, and SpamRank, which belong to a class of first-order Markov measures. A first-order Markov measure for a node is a measure that depends only on the set of nodes with direct links to it. A key characteristic of the representation of data as link graphs is that these graphs evolve as the corresponding real world phenomena change over time. Understanding such phenomena requires a study of the evolution of their representative graphs and the measures associated with them. However, the large size of these graphs and their evolution over time pose challenges in the computation of measures such as PageRank.

This dissertation presents techniques for efficient computation for First-Order Markov Measures (FOMM) on large graphs that evolve over time. PageRank is used as a representative of the class of first-order Markov measures. The objective is to develop work reduction techniques to reduce the computation time for graphs representing relationships between nodes at a single point in time (static graphs) and graphs that evolve over time (evolving graphs). A “divide and conquer” strategy is proposed for the efficient computation of PageRank. This can be achieved by using a novel graph-partitioning technique to divide the graph into a set of subgraphs, the measure for each of which can be computed independently. This strategy differs from existing approaches in that it can be combined with any existing enhancements to PageRank. This approach also leads naturally into the development of an incremental approach for the computation of such ranking metrics, given that these large graphs evolve over a period of time. Finally, I/O efficient methods are proposed to compute first-order Markov measures on large static and evolving graphs. The experimental results for a static single graph (graph at a single time instance) as well as for the incremental computations in evolving graphs, illustrate the usefulness of the presented approach. This dissertation opens new directions for research such as (a) exploration of efficient computational techniques for other classes of measures (b) incremental maintenance of evolving graphs, and (c) analysis of graph growth models and dynamically changing change graphs.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Motivation

The science of networks and its applicability across various domains—such as the study of the human brain's neurological network, computer networks, social networks, and the World Wide Web (hereafter, Web)—is well known [B2002]. The behavior and nature of networks and the various phenomena they represent can be understood by applying the underlying theory of networks and network evolution. A network is essentially comprised of a set of entities and the interactions or relations among them, which represent an underlying phenomenon. Typical examples include social networks (relations or interactions among people), computer networks (connections among computers), chemical networks (interactions among chemical compounds), and the World Wide Web (hyperlinks connections among Webpages). Networks are mathematically modeled as graphs, with the sets of vertices representing the entities and the set of edges the interactions/relations among them. The study of the nature and properties of such networks is commonly referred to as *network analysis* or *link analysis.* Typical examples of link analysis techniques include hyperlink analysis for the Web [SDK04], social network analysis [Kre02], and link analysis for computer security [SHJ+02].

The large scale of these representative graphs, and their evolution in conjunction with the evolution of the underlying phenomena that they represent, pose computational challenges. The focus of this dissertation is to address the computational challenges involved in computing a certain class of relevant measures derived from the application of link analysis techniques. This dissertation will focus primarily on the Web as an application domain for link analysis, since the Web-graph has similar properties to most other real networks and also is large enough to encounter scalability issues. The contributions of this dissertation, however, are not domain-specific and are applicable to networks from other domains.

The rest of this chapter is organized as follows. The following section first describes basic terminology used in link analysis techniques to help familiarize the reader with these standard terms, and then provides a brief overview of Web mining and how link analysis techniques play a significant role in this process. Section 1.3 describes the knowledge models used in link analysis. Section 1.4 introduces the first-order Markov measure, which is the main focus of this dissertation. Section 1.5 concludes with a definition of the problem and an overview of the approach taken by this dissertation to solve the problem.

1.2 Terminology: Properties and Models of Graph Evolution

The study of evolving graphs up to this point has focused on two dimensions—properties of the graphs across time, and growth models that simulate an evolution. Below is a brief overview of the most popularly studied properties and growth models, with a definition of common terminology.

The most basic set of properties for any graph are its *order* (the number of vertices in the graph) and *size* (number of edges in the graph). Some of the other key properties of interest are discussed below.

Average connected distance: This is defined as the average of the length of the shortest paths from vertex p to vertex q, for all pairs of vertices p and q. This property assumes significance to describe the *small-world effect*, which suggests that most pairs of vertices in a graph representing real-world interactions are connected by a short path through the graph. The most well-known experiment using this model was conducted by Stanley Milgram [M1967]; his results suggested that any two persons in this world are at most a path length of six away from each other.

Clustering Coefficient: This is defined as the mean probability that two vertices that share a neighbor will themselves be neighbors. This property has been referred to in the domain of sociology as *network density*. It is related to small-world effect and has been studied by Watts and Strogatz [WS1998].

Degree Distribution: The degree of a vertex in a graph is the number of incident edges on the vertex. If a graph is directed, then a vertex will have an in-degree (number of edges pointing to the vertex) and an out-degree (number of vertices to which the vertex points). Let p_k of a graph be defined as the fraction of vertices in the whole graph with degree *k*. A histogram of all vertices of degree *k* will reveal a distribution of degrees of vertices in the whole graph. This distribution is commonly referred to as *degree distribution*.

Researchers who studied complex networks were motivated by necessity to model them as random graphs. Early work in modeling random graphs was carried out by Erdős and Rényi, who proposed the Erdős-Rényi model for random graphs [ER1960, ER1961]. According to this model, given a graph G of order *n*, every pair of vertices is connected by an edge with a probability *p*. The average degree of a vertex in such a graph is *np* and the average number of edges is $n(n-1)\cdot p/2$. Such graphs have a binomial distribution of degrees. In a random graph, since the edges are distributed randomly, the clustering coefficient equals the probability that a pair of nodes connect to each other. It is suggested by Watts and Strogatz [WS1998] that in real networks, the clustering coefficient is typically much larger than it is in a random network of equal numbers of nodes and edges.

Albert and Barabasi [AB2002] demonstrate how scale-free networks differ from the random graph models proposed earlier, based on two criteria. In a random graph, the starting point was a graph of *n* vertices, and an edge was added between a pair of vertices chosen uniformly at random. Barabasi suggested a model of *growth* and *preferential attachment*. The scale-free graphs always start with a single vertex, and vertices and edges were added to the graph to facilitate its growth. Moreover, when a new vertex was added, the edge was added not to a vertex chosen uniformly at random, but had a "preference" to be connected to a node with a higher degree. In other words, the probability that a vertex attaches to another vertex depends on the degree of the vertex. This implies that the probability that a vertex has a degree *i* proportional to $1/i^{\alpha}$, for some $\alpha > 1$. Such a degree distribution follows the *power-law distribution*. This is different from the *Poisson distribution* that results from the Erdős-Rényi model.

It was empirically proved by Leskovec et al. [LKF05] that in most real networks, the average out-degrees grow over time (referred to as *densification power law*) while the diameter shrinks over time. This was contrary to the assumptions of earlier models, such as the small-world model, in which the average out-degree remains constant and the diameter increases slowly as the network grows over time. They proposed two models to capture such behavior. The community guided attachment model covered the densification power law. The basic idea of this model is that given a particular community structure, the probability that a new edge forms within that community is higher than the probability that an edge forms across communities. The forest-fire model captured the behavior of the densification power law and the shrinking diameter observed in real networks. This model suggests that a newly arrived vertex randomly chooses a vertex from the existing nodes. It then follows the edges of that vertex and randomly chooses a subset of these edges to identify a new set of vertices, to which the new vertex will add an edge. This process essentially continues, simulating the behavior of a forest fire.

1.3 Web Mining: Analysis of Evolving Graphs

Most network evolution models focus primarily on modeling the growth of graphs. However, the necessity to study temporally evolving graphs, and to mine information from this evolution, has gained popularity—as discussed by Desikan et al. [DS2004A]. Some of the key issues that arose in this study are discussed below.

Properties of Interest: One of the key challenges in graph evolution has been to identify the key properties whose change is of interest. The basic properties that are most frequently studied are *order* (number of vertices) and *size* (number of edges), as well as *diameter* and *degree distributions*. For example, it has been revealed that a large number of real networks, such as the Web, follow a *power law distribution*, and have a similar distribution even as the graph evolves, even though other properties such as diameter may shrink. Properties of interest also depend on the scope of analysis. Once the change has been defined, efficient ways to detect the change need to be developed, especially for large graphs. The analysis of the temporal behavior can be classified at three levels that are discussed below in detail.

Single Node Analysis: The behavior of the graph data can be modeled at the level of a single node, by monitoring the behavior of each node in the graph. For each single node that is labeled, properties based on its content, structure, and usage can be computed. For example, in the World Wide Web, over a period of time the content of the page represented as a node can change, indicating a change in the topic addressed by the Webpage. In addition, the structural significance of the node can be derived by computing properties that are purely structure based, such as in-degree, out-degree, or relevance ranking scores like authority score, hub score, or PageRank score. Usage data pertaining to a single node during a given time period is a reflection of its popularity. Such popularity tends to portray a trend that can help predict popular topics.

Sub-graph Analysis: The changing sub-graph patterns also evoke interest. These sub-graphs may represent different communities and abstract concepts that evolve over time. The idea of mining frequent sub-graphs has been applied with either a large graph or a set of small graphs as input [16]. However, the study of evolving graphs adds a temporal dimension to the study of sets of sub-graphs at different time instances.

It can be seen that the mining of sequential patterns of sub-graphs might provide useful information in profiling the change behavior and could help to predict an upcoming trend or an abnormal behavior. These changes may occur due to a shift in the strategy of an organization or to the launch of a new product by an e-commerce site. Mining such patterns poses interesting challenges to the research community, because of the need to develop algorithms that work efficiently despite the graphs' large size and the number of such graphs that are needed to model the temporal behavior.

Whole Graph Analysis: While single node analysis and sub-graph analysis tend to reveal specific information, analysis at the level of the whole graph reveals higher level concepts. The primary properties that have evoked interest in the research community are based on the average degree of the node, the diameter of the graph, and the degree distribution.

The basic assumption is that the network is "growing," and hence the nodes and edges are incremental in nature. The properties that are of interest may thus depend on the phenomenon being modeled and the analysis that needs to be performed. For some networks, such as email connections, the concept of connections within a given time frame will be the most appropriate model for understanding the behavior of traffic within the network. In such cases, the graphs may not be completely connected and can contain multiple components in a given time period. Also, certain properties, such as the number of components, the distribution of frequent sub-graphs, and so forth, may provide valuable information apart from properties such as order, size, and average degree.

Desikan et al. [DS2004A] studied the behavior of Web-graphs over time. They observed that by clustering vertices of Web usage graphed over time it is possible to identify a "concept" that is "popular" during a time period. Such trends are not evident either from pure Web structure–based or Web usage–based techniques. The interesting patterns were revealed when the graph was analyzed across time. Secondly, they proposed a time-aware relevance-ranking metric *Page-Usage-Popularity*, and showed that it was effective in detecting topics that were "recently popular" versus those that were "historically significant."

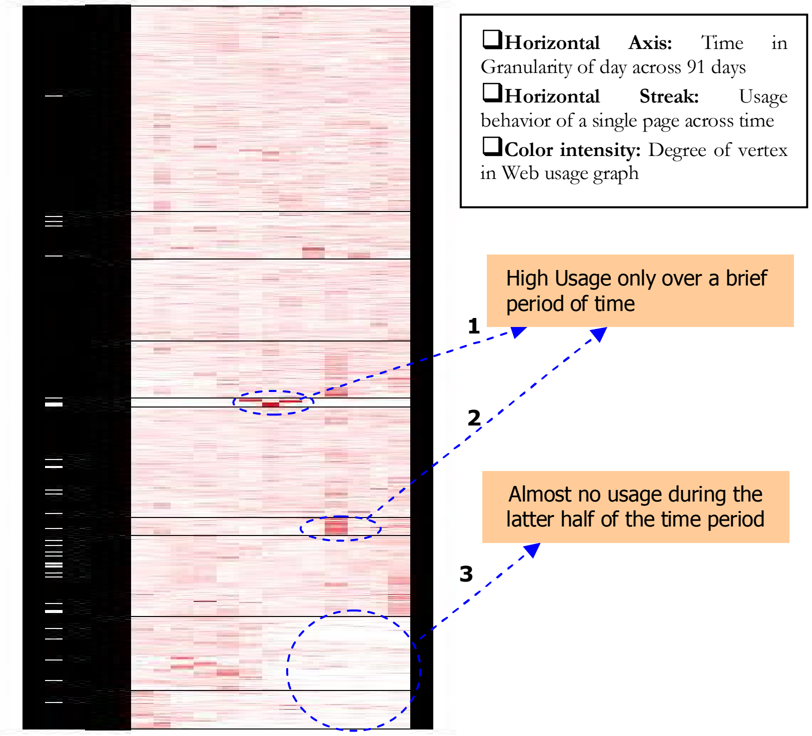


Figure 1. Clustering of Vertices of Web Usage Graphed Over Time.

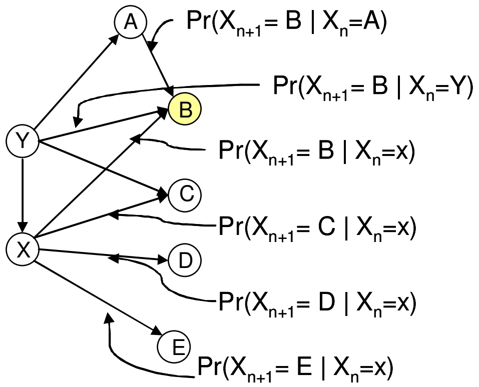
Computational Issues: Given that this study deals with large graphs, computation of certain measures and properties can be done efficiently by partitioning the graph into smaller sub-graphs. Such partitioning may not only help to speed up computations due to the smaller size of the graph, but also to handle memory more efficiently. The key issues lie in finding an optimal way to partition a given graph. It may depend on the class of properties that are being measured. A related issue for evolving graphs is that of incremental computation in order to avoid re-computation on the whole graph when only a part of it has changed. There are two kinds of incremental computation: those that result in an approximate measure, and those that compute the exact measure.

Storage Issues: Given that this study deals with large graphs, storage of such graphs also becomes an issue. It is imperative to devise optimal ways of partitioning the graph and storing it in different files so that the reading and computation of various properties are more efficient. For incremental computation, as discussed above, other challenges include the devising of optimal partitioning for incremental computation, and the development of optimal maintenance strategies for such partitions.

Visualization: Visualization of large graphs that evolve over time is a challenge. The key issues are how to depict the change, and how changes at different levels (such as node, sub-graph, and whole graph) can be viewed.

1.4 First-Order Markov Measure

Markov measures can be described as measures that are derived out of Markov models. The previous sub-section discusses in detail the various measures that have resulted from Markov models. Markov measures depend on the order of the Markov chain, since the order determines the number of past states that the system in its current state depends on to transition to a future state. Among the different kinds of measures that depend on Markov models, the measures that depend on the first-order Markov chain are popular due to their simplicity, ease of computation, stability, and better predictability.


A
$Pr(X_{n+1}= B \mid X_n=A)$
$Pr(X_{n+1}= B \mid X_n=Y)$
B
Y
$Pr(X_{n+1}= B \mid X_n=x)$
C
$Pr(X_{n+1}= C \mid X_n=x)$
X
D
$Pr(X_{n+1}= D \mid X_n=x)$
E
$Pr(X_{n+1}= E \mid X_n=x)$

$$\boxed{\begin{aligned} Pr(X_{n+1}=B) = {} & Pr(X_{n+1}=B|X_n=A)*Pr(X_n=A) + \\ & Pr(X_{n+1}=B|X_n=Y)*Pr(X_n=Y) + \\ & Pr(X_{n+1}=B|X_n=X)*\ Pr(X_n=X); \end{aligned}}$$

Figure 2. First-Order Markov Measure.

Let X_i represent a state of a node x at time instance i. A *first-order Markov measure* can thus be defined as:

$$\Pr(X_{n+1} = x \mid X_0 = x_0, X_1 = x_1, \ldots, X_n = x_n) = \Pr(X_{n+1} = x \mid X_n = x_n)$$

Figure 2 illustrates the same for node B, where the probability that a node is in state B depends on the probability that the node was in state A, state X, state Y, and on the respective transition probabilities.

1.5 Thesis Overview

This section provides an overview of the thesis in three parts: the hypothesis, the scope, and the contributions. Each of these parts is discussed below in more detail.

Hypothesis: "Computational efficiency of First-Order Markov Measures (FOMM) on large evolving graphs can be improved using work reduction techniques such as graph partitioning and incremental computation."

Scope: The scope of the thesis can be defined in terms of the class of computational techniques for efficiency and the measure of interest. *Computational efficiency* refers to a reduction in the overall time required to compute first-order Markov measures. This time reduction can be achieved in two ways: through *parallel computing* techniques and through *work reduction* techniques. This thesis focuses on work reduction techniques.

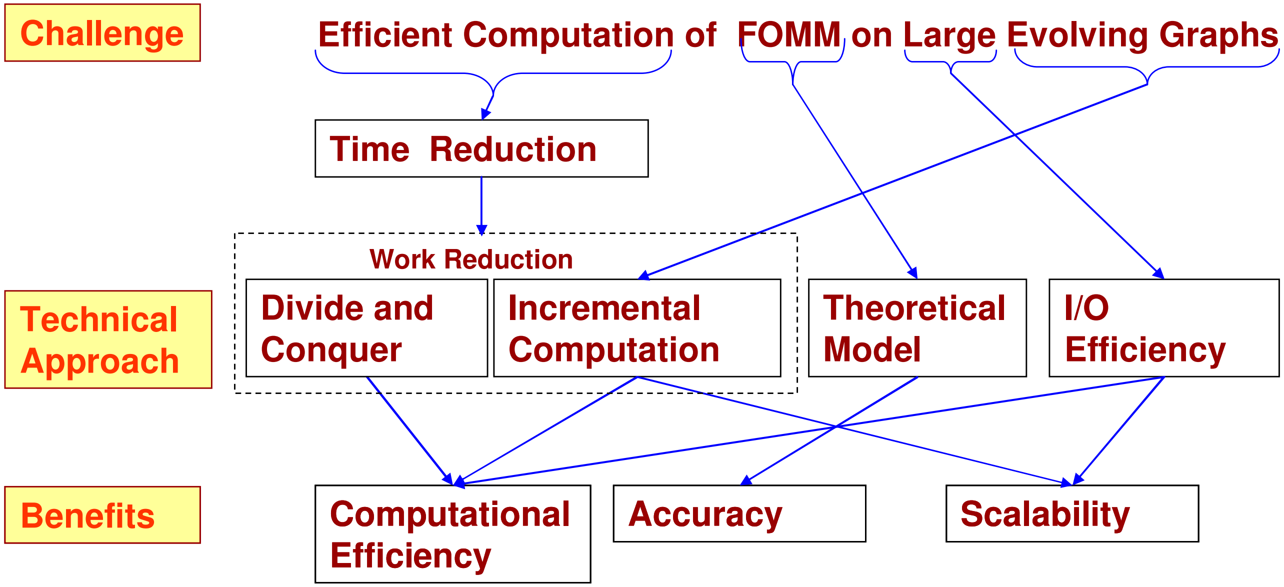


Figure 3. Roadmap of Thesis

While this dissertation addresses the class of first-order Markov measures, it focuses on PageRank as an example of such a measure. This is because the benefits and applicability of PageRank are well understood, since it has been successful as a relevance measure for the popularity or importance of Webpages. The value of PageRank as a measure is that it reveals the authority of a vertex in a graph. In addition, PageRank is a relatively stable measure, because control of it is not in hands of a creator of a state in the Markov chain. While collaborative spamming of this measure, for example a "Google Bomb," can be achieved, it is far more stable than degree-based measures such as out-degree, or other measures such as hubs and authorities, where the creator of a Webpage has control over the number of outgoing links and hence is able to influence any measure based on that number. Finally, the key reason for choosing an existing measure such as PageRank whose utility is well accepted is that it is computationally expensive to compute such measures in large graphs such as the Web.

Contributions: The contributions of this dissertation are the development of models and techniques for the efficient computation of first-order Markov measures. The contributions can be summarized as follows:

- *Methodology for link analysis techniques*: In this survey, a taxonomy for classifying the research on hyperlink analysis is introduced. Key dimensions, namely *knowledge models, measures and algorithms,* and *analysis scope* are identified. Each of these dimensions is described in detail, and it is shown how they form the core components of any application of hyperlink analysis. Existing literature is classified in terms of this taxonomy, thereby illustrating where they are similar and where they complement each other. A rather pleasing consequence of the taxonomy is that it leads naturally to a methodology for applying hyperlink analysis to an application that has been described.

- *Theoretical model for partitioning graphs*: The second contribution of this dissertation is to identify patterns of graph partitions for which any first-order Markov measure can be computed independently without having to compute the measure for the whole graph at once. A key challenge in identifying this model is that the measure of a particular state (or vertex) is dependent on the set of possible previous states (vertices pointing to the given vertex). Traditional graph partition algorithms aim to find an optimal partition that gives a minimum cut partition. However, a min-cut does not help when computing measures such as first-order Markov measures.

- *Divide and conquer computation*: This dissertation proposes a *divide and conquer* strategy for the efficient computation of PageRank. This strategy is different from contemporary improvements to traditional graph partition algorithms in that it can be combined with any existing enhancements to PageRank, giving way to an entire class of more efficient algorithms. The proposed approach outlines a method for identifying partitions in a given graph that adheres to the conceptual model mentioned above.

- *Incremental computation*: This dissertation also proposes a method to incrementally compute PageRank for a large graph that is evolving. It should be noted that the rate of change for the Web-graph is rather slow compared to its size. This approach once again exploits the underlying principle of the partitioning mentioned earlier to incrementally compute PageRank for the evolving Web-graph. The approach has shown a significant increase in the speed of computational cost, as the computation involves only the (small) portion of the Web-graph that has undergone change. The approach is quite general, and can be used to incrementally compute (on evolving graphs) any metric that satisfies the first-order Markov property.

- *I/O efficient computation*: Finally, this dissertation proposes I/O efficient techniques to compute first-order Markov measures on large graphs. The techniques proposed address computation for a large graph at a given time instance, and computation for an evolved graph. The techniques use the approaches of divide and conquer and incremental computation, in conjunction with a binning strategy, to handle I/O efficiency. The proposed approaches are generic and can be applied to any generic graph without the requirement of knowledge of the domain it represents.

Chapter No.	**Chapter Title**	**Key Associated Research Paper**	
1	Introduction	*Proceedings:*	WebKDD 2004
		Paper Title:	Mining Temporally Evolving Graphs
2	Background and Related Work	*Proceedings:*	Semantic Computing Chapter 2008
		Paper Title:	Link Analysis in Web Mining: Techniques and Applications
3	Work Reduction Technique for Large Static Graph	*Proceedings:*	ICWE 2006
		Paper Title:	Divide and Conquer Approach for Efficient PageRank Computation
4	Incremental Computation on Large Evolving Graphs	*Proceedings:*	WWW 2005
		Paper Title:	Incremental PageRank Computation
5	Improving I/O Efficiency	*Proceedings:*	WebKDD 2008
		Paper Title:	I/O Efficient Computation for First-Order Markov Measures
6	Case Study: Analyzing Network for Email Spam Detection	*Proceedings:*	ICDM 2004
		Paper Title:	Analyzing Network Traffic to Detect Email Spamming Machines

Figure 4. Thesis Organization

1.6 Thesis Organization

The dissertation is organized as follows. This chapter primarily discusses the motivation behind this research. It provides an introduction from a graph-theoretic perspective to the study of evolving graphs, various properties of interest, and to growth models. It continues the discussion on evolving graphs by presenting various scopes of analysis and domain-specific scenarios, highlighting the importance of studying evolving graphs from an application perspective. The first-order Markov model is described, and the related first-order Markov measure is defined. The defined measure is the focus of the study of this dissertation. The hypothesis, scope, and contributions of the dissertation are summarized in Section 1.6.

Chapter 2 provides a background on state-of-the-art link analysis techniques, a classification of such techniques, and a methodology to develop future techniques. In addition, the discussion points to the future directions that remain, leading to the focus on the challenges addressed in this dissertation. Chapter 3 presents the theoretical model for work reduction techniques and an approach to leverage the model and divide the graph into partitions for which computation can be carried out independently. Chapter 4 addresses incremental computation on evolving graphs based on the principle discussed in Chapter 3. Chapter 5 presents an algorithm to efficiently identify a partition from a given changed portion of a graph, and provides a detailed description of I/O efficient techniques that combine binning strategies for vertices with the divide and conquer approach and the incremental computational technique. A case study that highlights why the evolving graphs and the measurement of first-order Markov measures on evolving graphs are useful is presented in Chapter 6. The final chapter presents the conclusions of this study and future directions for computational techniques, evolving graphs, and link analysis.

Chapter 2

LINK ANALYSIS FOR THE WEB: TECHNIQUES AND APPLICATIONS

2.1 Introduction

Link analysis is part of a broader research area in data mining known as *Web mining*, which is the process of applying data mining techniques to extract useful information from Web data. The types of data collected and utilized in Web mining include *content data, structure data,* and *usage data* [SCD+00]. The field of Web mining can therefore be divided according to these data types into three interrelated categories [SCD+00, KB2000]:

- **Web content mining** is the process of extracting useful information from the contents of Web documents. The information may consist of text, images, audio, video, or structured records such as lists and tables. Closely related research areas that make use of Web content include Information Retrieval (IR) and Natural Language Processing (NLP).

- **Web structure mining** is concerned with mining two types of structural data found in Web documents—*anchor tag data*, which define the links (or edges) in a Web-graph between one or more Webpages (or nodes), and the tree-based *HTML/XML tag data* defining the layout of Webpages.

- **Web usage mining** is the application of data mining techniques to discover interesting usage patterns from server logs (e.g., Apache, IIS). Usage data typically captures the identity or origin of Web users along with their browsing behavior at a Web site. Examples of usage data include IP addresses, URL references, and Webpage access times of the Web site visitors.

The primary interest in link analysis is the sub-area of Web structure mining that utilizes anchor tag data, and hence the Web-graph created by a set of interlinked Web documents. However, link analysis can be further enriched by leveraging data from all the categories of Web mining. Some examples of how link analysis can be used—both with and without additional Web mining data—follow:

- Assigning authority to a collection of Webpages. When combined with Web content data, authority can be subdivided according to content topics.

- Understanding Web-graph structure through the examination of various graph patterns, such as co-citations, co-references, bipartite graphs, and so forth.

- Improving the efficiency of *crawling*—the process of indexing a collection of Webpages—by scheduling pages that need to be crawled before others, according to various link analysis–generated metrics.

- Predicting user-browsing behavior and creating improved recommendation systems by combining link analysis with Web usage data.

2.1.1 Web Structure Terminologies

In general, the Web can be modeled as a directed graph containing a set of nodes connected by directed edges. A basic overview of terminology used in modeling a Web-graph, as described by Broder et al. [B2000], follows:

- *Web-graph:* A directed graph that represents the Web.

- *Node:* Each Webpage is a node of the Web-graph.

- *Link:* Each hyperlink on the Web is a directed edge of the Web-graph.

- *In-degree*: The in-degree of a node, *p,* is the number of distinct links that point to *p.*

- *Out-degree*: The out-degree of a node, *p,* is the number of distinct links originating at *p* that point to other nodes.

- *Directed Path*: A sequence of links, starting from a page *p* that can be followed to reach a page *q.*[1]

- *Shortest Path:* Of all the paths between nodes *p* and *q,* one containing fewest links.

- *Diameter*: The maximum of all the shortest paths between a pair of nodes *p* and *q,* for all pairs of nodes *p* and *q* in the Web-graph.

- *Average Connected Distance*: Average of the lengths of the shortest paths from node *p* to node *q*, for all pairs of nodes *p* and q [AJB99]. Broder et al. [B2000] observed that this definition could result in an infinite average connected distance if there is at least one pair of nodes *p* and *q* that have no existing path between them. They thus proposed a revised definition: "the average connected distance is the expected length of the shortest path, where expectation is uniform choices from a set of all ordered pairs, (*p*,*q*) such that there exists a path from *p* to *q*."

2.1.2 Related Work

The past decade has seen a growing interest in the research on Web mining, and on link analysis in particular. Etzioni [E1996] first classified the area of Web mining using a process-centric view into three phases, namely resource discovery, information

[1] A link can be traversed in only one direction, i.e., from its source to its destination

extraction, and generalization. Cooley et al. [CMS97] took a data-centric approach to define and categorize Web mining based on the type of data, namely content, structure, and usage. This approach has gained in popularity and is widely accepted [SCD+00, KB2000]. Various overviews addressing different angles, such as the comparison of different data mining and statistical techniques [C2000], the importance of links and the interesting graph patterns that they form [ERC+00], and key link-based metrics [H2000], have been published.

Each Webpage is associated with keywords that are found in in-links to that page. In existing methodologies, a Webpage is assumed to be equally knowledgeable of all such keywords. Thus, a major limitation of these (and similar) ranking algorithms is that they assume that a Webpage with high authoritative weight is very knowledgeable of all terms related to it. (This is known as topic drift [RD2002].) Philosophically speaking, a Webpage may not be equally informative about all related topics. Initial approaches to address this limitation include heuristic methods for differentially weighting links (Chakrabarati et al. [CDG+98]; Bharat and Henzinger [BH1998]). Haveliwala [H2002] proposed a modified PageRank algorithm, called *topic-sensitive PageRank.* In this approach, ranks are computed separately for each topic, thus each page has a vector of authoritative weights for all topics found in it. The main disadvantage of this approach is that it assumes independence of topics; i.e., a particular topic is uncorrelated to every other topic, therefore separate ranking is assumed to be sufficient. In addition, the issue of topic identification from the Web-graph was not addressed in their approach and instead topics manually selected from Open Directory Project (http://www.dmoz.org/) were used. Richardson and Domingos [RD2002] proposed a query-dependent PageRank approach that removed the assumption of the random surfer from the PageRank algorithm. However, for scalability, their query-dependent PageRank is computed separately for each term, thus assuming the terms are independent As a side note, Richardson and Domingos' approach was the PageRank analog of Cohn and Hoffman's [CH2001] probabilistic variation for the HITS algorithm.

Rafiei and Mendelzon [RM2000] proposed a process to determine the topics on which a Webpage is considered to be authoritative. Their approach was based on modifying the HITS algorithm to simultaneously rank Webpages as well as identify topics based on link information. Xue et al. [XCM+03] showed how implicit relationships between Webpages can be captured by using user access patterns and then using ranking algorithms. Other approaches to PageRank computing are related to personalizing search results using user behavior (profile), such as a scaled personalization that utilizes users' bookmarks (Jeh and Widom [JW2002]); a user-feedback based authorities list (Chang et al. [CCM00]); and query chains based on the sequence of users' search queries (Radlinski and Joachims [RJ2005]). Similar issues related to link analysis can also be found in [DDS2009].

2.2 Knowledge Models

Most research in link analysis starts with a basic model upon which different measures are applied, and the targeted application objective is achieved by a more specific computation technique or algorithm. These models either relate to the basic information unit or the Web property that is the focus of the application.

2.2.1 Graph Models

In this section, the discussion revolves around the fundamental graph patterns that represent different fundamental concepts and serve as information units while mining the Web. These patterns can be classified based on whether a single node is involved or multiple nodes participate in the pattern. Link analysis literature points to the following fundamental patterns that form the basis for most further analysis:

Single Node Models: Single Node Models are graph structures consisting of a single node and the links pointing to or away from it.

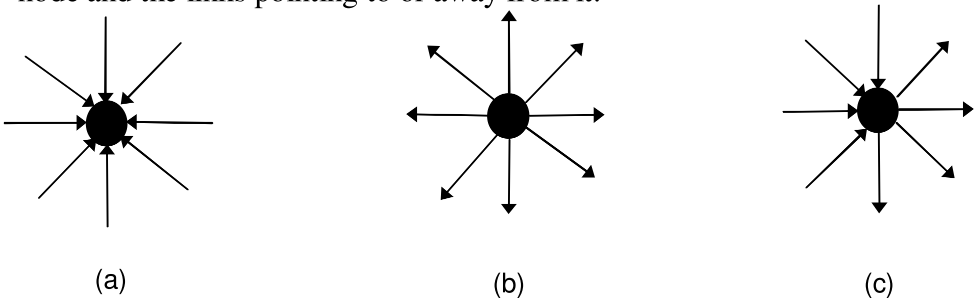


Figure 5. Single Node Models. (a) represents a pure *authority* page. (b) represents a pure *hub* page. (c) represents more a typical Webpage that will have both a *hub* score and an *authority* score associated with it.

Authority: An *authority* page is a Webpage that is pointed to by a set of other related Webpages.

Hub: A *hub* page is a Webpage that points to a set of other related Webpages. A *good hub* is a one that points to many *good authorities*, while a *good authority* is one that is pointed to by many *good hubs*. The notion of *hubs* and *authorities* was first introduced by Kleinberg [K1998]. Knowledge models for a single page are often used to determine the quality of a Webpage [PBM+98, LM2000, RM2000].

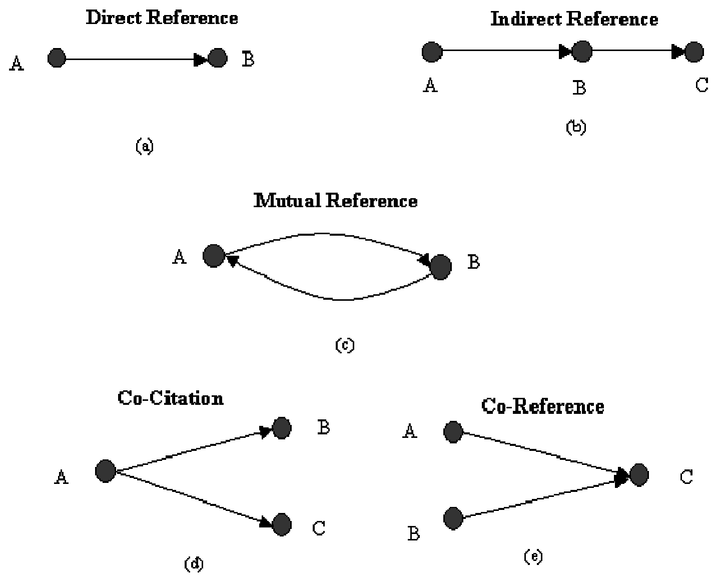


Figure 6. Multiple Node Models with simple structures. They have also been discussed in [ERC+00] as graph patterns.

Multiple Node Models: Multiple Node Models deal with graph structures that contain a set of nodes and the links that connect them. Some of these graph structures or patterns have also been discussed by Efe et al. [ERC+00]. Examples of these models are given in Figure 6.

Direct Reference: A direct reference refers to a situation where a node A is pointed to directly by an adjacent node B. In Figure 2(a), “B” is *directly referred* by “A,” indicating that “A” and “B” may address a common topic and may be related.

Indirect Reference: An indirect reference refers to a concept whereby node A is pointed to or referred directly by an adjacent node B and node B is pointed to or referred directly by another adjacent node C. In this scenario, node A is said to be indirectly referred by node C. In Figure 2(b), “A” directly refers “B” and “B” directly refers “C.” Thus, “A” *indirectly refers* “C,” indicating that “A” and “C” could be related.

Mutual Reference: When two nodes A and B point to each other directly, then they are said to mutually reference each other. This also indicates a strong relevance between the two pages. In Figure 2(c), “A” and “B” *mutually reference* each other.

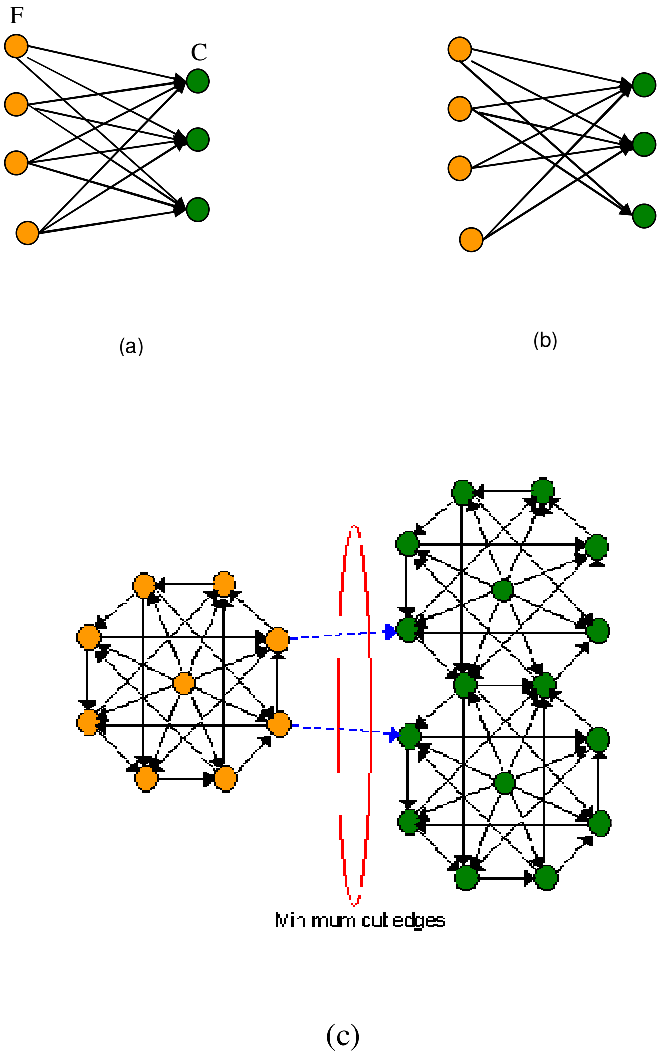
Co-citation: When a node A points to two other nodes B and C, then node A is said to be co-citing node B and node C. On the Web, such co-citation intuitively could indicate a similarity between page B and page C. In Figure 2(d), “A” is *co-citing* “B” and “C.” Thus, it is possible that “B” and “C” have some similarity.

Co-reference: When two nodes B and C point to a node A, then node A is said to be co-referenced by node B and node C. On the Web, such co-citation intuitively indicates a possible similarity between page B and page C. In Figure 2(e), “C” is *co-referenced* by “A” and “B,” suggesting possible relatedness between “A” and “B.”

Directed Bipartite Graph: A graph whose node set can be partitioned into two disjointed sets *F* and *C,* where every directed edge in the graph is from a node *u* in *F* to a node *v* in *C*.

Complete Bipartite Graph: A bipartite graph that contains all possible edges between a vertex of *F* and a vertex of *C*.

Bipartite Core: A core (*i, j*) is a complete directed bipartite sub-graph with at least i nodes from *F* and at least *j* nodes from *C*. With reference to the Web-graph, the *i* pages that contain the links are referred to as *fans* and the *j* pages that are referenced are the *centers. Fans* and *centers* in a *bipartite core* are viewed as *hubs* and *authorities* in the Web-graph. For a set of pages related to a topic, a bipartite core can be found that represents the hubs and authorities for the topic. Hubs and authorities are important since they serve as good sources of information for the topic in question.



(a)

(b)

(c)

Figure 7. Multiple Node Models with more complex structures. (a) fans (F) or hubs (left nodes) and the centers (C) or authorities (right nodes). (b) Bipartite graph. In (c), Web Communities as defined by Flake et al. [FLG00]

Community: The *community* is a core of central authoritative pages linked together by hub pages [GKR98]. It has also been defined as a collection of Webpages such that each member node has more hyperlinks (in either direction) within the community than outside the community [FLG00].

Markov Models: The Web-graph is viewed with nodes as a set of states, and edges as state transitions. The set of states and transitions can be modeled using a Markov Model. The underlying principle of an "m" order Markov chain is that given the current state of a system, the evolution of the system in the future depends only on the present state and the past "m-1" states of the system. First-order Markov models have been used to model the browsing behavior of a typical user on the Web. PageRank [PBM+98] and Randomized HITS [NZJ01] use the random walk process based on the Markov model. The user randomly chooses to either jump to a new page or to follow a link—*out-link* in the case of PageRank, and *in-link* or *out-link* depending on the time-step in the case of the Randomized HITS approach. Other approaches, e.g., SALSA [LM2000], have also incorporated the Markovian random walk. Zhu et al. [ZHH02] use Markov chains to predict links for adaptive Web sites. The modeling of a Web surfer's activity based on Markov models, which essentially involves traversing links, has been used significantly in link analysis.

2.3 Measures and Algorithms

This section discusses some of the more popular and interesting link analysis techniques in the Web domain, where hyperlink information has been useful in describing everything from the properties of a single Webpage to the entire Web-graph of the Internet.

2.3.1 HITS: Hubs and Authorities in Web Search

Hubs and *authorities*, as mentioned earlier, together constitute a bipartite graph that has directed edges linking hubs to authorities. The computed hub and authority scores for each Webpage indicate the extent to which that page serves as an *authority* on a topic, or as a *hub* which can reference good *authority* pages. These scores are computed using the HITS algorithm [K1998], described in more detail below. This procedure is done *after* retrieving a set of candidate Webpages, however. Prior to that, a query is first issued to a search engine and a set of relevant documents is retrieved, called the *root set*. The *root set* is then grown to include the in-link and out-link Webpages of the Webpages in the root set. This expanded set is called the *base set*. An adjacency matrix, **A**, is formed such that if there exists at least one hyperlink from page i to page j, then $\mathbf{A}_{i,j} = 1$, else $\mathbf{A}_{i,j} = 0$. The HITS algorithm is then used to determine the hubs and authorities scores.

HITS Algorithm

```
Let a be the vector of authority scores and h be the vector of hub scores
$\mathbf{A} = [1,1,\ldots.1]$, $\mathbf{h} = [1,1,\ldots..1]$ ;
do
  $\mathbf{a}=\mathbf{A}^T\mathbf{h}$;
  $\mathbf{h}=\mathbf{A}\mathbf{a}$;
  Normalize $\mathbf{a}$ and $\mathbf{h}$;
while $\mathbf{a}$ and $\mathbf{h}$ do not converge (reach a convergence threshold)
$\mathbf{a}^* = \mathbf{a}$;
$\mathbf{h}^* = \mathbf{h}$;
return $\mathbf{a}^*$, $\mathbf{h}^*$

The vectors $\mathbf{a*}$ and $\mathbf{h*}$represent the authority and hub weights
```

Algorithm 1. HITS Algorithm

The HITS algorithm is shown above, and can be described as follows: let **A** be an adjacency matrix such that if there exists at least one hyperlink from page *i* to page *j*, then $\mathbf{A}_{i,j} = 1$, else $\mathbf{A}_{i,j} = 0$. The vectors $\boldsymbol{a}^*$ *and* $\boldsymbol{h}^*$ correspond to the principal eigenvectors of $\boldsymbol{A}^T \boldsymbol{A}$ and $\boldsymbol{A}\ \boldsymbol{A}^T$. The stability of the HITS algorithm in response to small perturbations [NZJ01, PBM+98] is determined by the *eigengap* of **S**, which is defined as the difference between the largest and second largest eigenvalues. The authors found the HITS algorithm to be less stable than that of Google's PageRank, and [NZJ01, PBM+98] propose two modifications to HITS to address this issue.

The first modification, an algorithm called *Randomized HITS,* introduces a bias factor based on time-steps (odd or even) to determine authority and hub scores. It can be viewed as a random surfer tossing a coin with a bias, $\in$. This bias is the probability that, at any point in time, the surfer will jump to a new page chosen uniformly at random. With a probability 1-$\in$, the surfer will follow an out-link if it is an odd time-step, and will traverse an in-link if it is an even time-step. The authority weight of the page is the chance that a surfer visits that page at an odd time-step *t*. The second algorithm is called *Subspace HITS*, and it asserts that hub and authority scores are determined by the *subspace* spanned by the eigenvectors, rather than the individual eigenvectors themselves. The bias factor (or subspace) generated by the eigenvectors is considered more stable in response to perturbation than the original HITS algorithm.

In general, HITS has been found to be successful for queries regarding topics that are well-represented in the Web in terms of linkage density. Often, when a query regarding a more focused topic is issued, HITS returns results for a more general topic. As a result, *topic drift* becomes a problem since the user issuing the query “drifts” away from the specific topic described in the query.

HITS has also been researched and extended by other researchers. Chakrabarti et al.. [CDG+98] modified Kleinberg’s hub and authority scores by using text-based weights in the adjacency matrix while calculating the scores. Bharat and Henzinger [BH1998] suggested that edge weights should be modified such that if *k* edges on a document on the first host point to a single document on the second host, each edge is given an *authority weight* of 1/*k*. Similarly, if a document on a host is pointing to *l* documents on another host, then each edge is given a weight of 1/*l*. This addresses the problem of "mutually reinforcing relationships" between hosts. The CLEVER project at IBM [CLEVER] has enhanced the original HITS-based measures and used this enhancement for link-driven applications, such as Web crawling, Webpage categorization, and the identification of Web communities.

2.3.2 PageRank

PageRank is a well-known algorithm for ranking hyperlinked documents, and was developed by Larry Page and Sergey Brin [PBM+98] as the backbone of their then-fledgling search engine, Google [GOOG, BP1998]. The key idea behind PageRank is that a page's rank is proportional to the sum of the ranks of its in-linking pages. Put another way, the most highly-ranked Webpages will have a large number of highly-ranked Webpages linking to them. More formally, the rank of a page *p* can be written as follows:

$$PR(p) = \frac{(1-d)}{N} + d \times \left(\sum_{\forall q \in Inlinks(p)} \frac{PR(q)}{OutDegree(q)} \right) \quad (1)$$

Here, *N* is the number of nodes in the Web-graph, *q* is a Webpage corresponding to an in-link of *p*, and *Out-degree(q)* is the number of outgoing links from page *q*.

Intuitively, this approach can be understood as a stochastic analysis of a random walk on the Web-graph. In this model, a surfer browses Webpages by clicking on links until he or she becomes bored and chooses a different Webpage at random from bookmarked pages, or by typing in a URL. The first term in the right hand side of the equation, *(*1 – *d)/n*, corresponds to this event—that the Web surfers stops clicking links and chooses another page at random. Naturally, the second term pertains to the former behavior—that the surfer continues to browse by clicking links. Additionally, the second term formalizes the prior statement that highly-ranked Webpages will be linked to by other highly-ranked Webpages, and for such Webpages, this term essentially corresponds to the probability that a random surfer will arrive at page *p* from any source (as the first term is constant). In both terms, *d*, the dampening factor, is present and is the probability that any random surfer at page *p* will continue to traverse links on that page instead of choosing another URL at random.

The popularity and utility of PageRank has inspired research, which has led to several interesting studies and/or modifications. Haveliwala [H1999] discussed efficient methods to scale the implementation of PageRank using large graphs on machines with limited memory. The stability of PageRank and other ranking metrics was discussed in a number of studies [NZJ01, BRR+01, PBM+98]. The authors found that as long as Webpages with high PageRank scores are not modified or perturbed (i.e., either more links are added or certain links removed), the PageRank scores resulting from such perturbation/modification will not be significantly different from the original scores. This stability under perturbation is due to the modeling of a surfer arriving at a page “out of the blue” (i.e., typing in a URL, or choosing a bookmark, of a page chosen at random from a uniform distribution).

As mentioned previously, the PageRank algorithm is based on the *random surfer* model, which is implemented as a Markov chain (or random walk). To do this, the PageRank measure is computed iteratively for a given Web document corpus. This can also be done using matrix computations similar to the HITS algorithm. However, the difference lies in the entries of the matrix $\boldsymbol{A}$, which, in PageRank, contains transition probabilities. An (i, j) element in the matrix represents the probability that the link from page *i* to page *j* will be chosen. As such, for the initial values, the element (i, j) = 0 if there is no link from page i to page *j*, else it is 1/*Out-degree(i)*, where *Out-degree(i)* is the out-degree of page *i* as defined in the PageRank equation above.

The PageRank Algorithm

```
Set $\mathbf{PR} \leftarrow [r_1, r_2, .....r_N]$, where $r_i$ is some initial rank of page I, and $N$ the number of webpages in the graph;
  $d \leftarrow 0.15$; $\mathbf{D} \leftarrow [1/N.......1/N]^T$;
  $\mathbf{A}$ is the adjacency matrix as described above;
  do
    $\mathbf{PR_{i+1}} \leftarrow \mathbf{A^T*PR_i}$ ;
    $\mathbf{PR_{i+1}} \leftarrow d* \mathbf{PR_{i+1}} + (1-d)*\mathbf{D}$;
    $\delta \leftarrow \| \mathbf{PR_{i+1}} - \mathbf{PR_i} \|_1$
  while $\delta < \varepsilon$, where $\varepsilon$ is a small number indicating the convergence threshold
  return PR.
```

The vector **PR** represents the global ranking of all the *N* webpages in the Web-graph.

Algorithm 2. Basic PageRank Algorithm

2.4 Applications of Link Analysis

Link analysis has been used in a wide variety of applications, including Webpage ranking, Web crawling, Web community identification, recommendation systems, and browsing personalization. Here several applications of link analysis will be described, as well as some potential applications made possible by Web 2.0.

2.4.1 Webpage Ranking

Perhaps the most well known of all link analysis applications is that of Webpage ranking. To do this, Markovian techniques such as PageRank or HITS are applied to a Web-graph until the probability distribution over the set of all Webpages in the corpus stabilizes.

Each Webpage’s probability corresponds to its PageRank. During a search, Webpages containing the search query terms are returned in descending order of these probabilities.

2.4.2 Fraud Analysis

Fraudulent attempts to unjustly obtain property on Web sites have been increasing. Although a great deal of effort has been expended in investigating and preventing Internet fraud, criminals have shown they are also capable of quickly adapting to existing defensive methods and they continue to create more sophisticated ways of perpetrating fraud. Much Internet-based fraud is carried out cooperatively with multiple associates. For example, in online auction shilling, fake customers (who are associates of a fraudulent seller, also known as “shills”) pretend not to have any connection with the seller and raise the bid price so that the seller's item is sold at a higher price than its real value. Alternatively, a seller can replicate this process alone by using multiple accounts, each associated with a different computer with its own IP address, and pretend to be different bidders.

In order to detect such fraudulent activity, link analysis can be used to uncover latent relationships between associates by finding sub-graphs of similar topology. Since a number of frauds may be perpetrated by the same group of associates, identifying similar fraudulent activity is possible using these techniques.

2.4.3 Knowledge Modeling

Compact structures for capturing information entities, such as topics and the relationships between them, can be efficiently represented as directed or undirected graphs. Additionally, in the Web domain, it is possible to use link analysis to combine this graph-based topic information with the traditional Web-graph by assigning a node to each unique Webpage/topic combination existing in a document corpus. This is accomplished by applying the rule retention/deletion principles from association rule analysis for graph pruning. Ranking methods such as PageRank or HITS can be run on the resulting graph, which can be used for Web search and recommendation systems. Thus, it is inherently well-suited to the problem of finding topical authorities given a particular topical context.

2.4.4 Semantic Web and Social Networks

Semantic Web is an enabling technology for capturing the knowledge models of social networks. They complement each other to provide a very useful tool to manage the social information and beliefs of people and their interactions. Hope et al. [HNT06] discuss a mechanism to integrate data from various sources on the Web, such as using Web mining to extract social network information, or using communication logs and actor profiles from social network sites such as Orkut or Friendster.

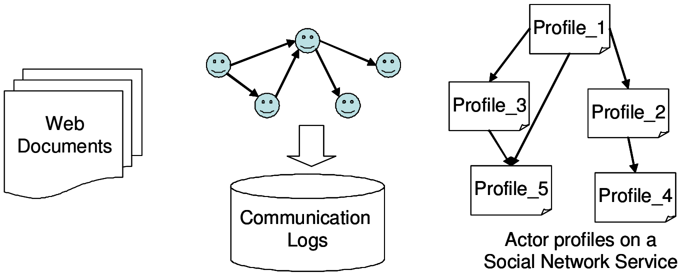


Figure 8. Integration of Data Sources for Social Networks.

Two of the key prominent projects in this field are the Friend-of-a-Friend (FOAF) project (http://www.foaf-project.org) and the Flink System. Ding et al. [DFJ05] describe some of the key challenges in analyzing social networks in a semantic web. A key challenge identified was the need for common ontologies that help create a more common and standard mode of knowledge representation and sharing. The missing links between RDF documents do not provide an efficient way to summarize and understand the social models. Semantic Web also has a lot of noise that needs to be cleaned before any rigorous analysis can provide a high utility value. Another key challenge is the integration of the trust networks and knowledge authority of topics. Mika [M2005] discusses the Flink system that gathers information from various sources such as Webpages, emails, and social sites such as FOAF and extracts and presents a Web-based representation of these social networks.

2.4.5 Criminal Network Analysis

Knowledge gained by applying Social Network Analysis (SNA) to criminal networks aids law enforcement agencies in fighting crime proactively. Xu and Chen [XC2005] provide a good overview of the challenges and the current state-of-the-art technology in criminal network analysis from a data mining and a social networks perspective. Although criminal networks are large and dynamic, they are characterized by uncertainty. There is often a need to integrate information from multiple sources (criminal incidents) to discover regular patterns of structure, operation, and information flow. The covert nature of criminal networks leads to insufficient, inconsistent, and sometimes conflicting data. This leads to challenges in data pre-processing. Data mining helps to identify the existence of communities and patterns of communication within these communities. Social networks provide complementary information to determine the importance of the interactions and the relevance of individual persons in the network, using measures such as betweeness and centrality. However, computing SNA measures like centrality is NP-hard, and various approximation techniques have been developed [CKS02]. A key challenge in this area is the visualization of networks. Xu and Chen [XC2005] discuss some of the existing tools. It is also believed that some of the visualization tools, such as Naviz [PPT+02], that have been developed for the Web can be used in this domain to analyze large scale data.

2.5 Methodology for Hyperlink Analysis

The methodology for using hyperlink analysis for a particular application can be described as the following sequence of steps:

- First, the needs of the application need to be analyzed to determine the type of information it needs from hyperlink analysis. For example, the Web search application requires that pages that are relevant to a user query be ranked in some order of importance. The information model here is a ranked list of URLs. In some cases, a process model may be required in addition to the information model.

- Next, the metric(s) that need to be calculated to quantify various aspects of the information model must be determined. For example, for Google, the metric is PageRank, while in the HITS approach, it is HubScore and AuthorityScore. As newer applications of hyperlink analysis are discovered, new metrics will have to be developed to suit their needs.

- Then algorithms to compute the selected metrics need to be selected/designed. The Google approach uses a (bounded scope) graph traversal algorithm to compute the PageRank metrics of pages relevant to a user query. The HITS approach has developed a new algorithm called Hypertext Indexing and Topic Selection (HITS), which primarily uses the algorithm to compute eigen values of large sparse matrices.

Hyperlink analysis metrics and algorithms for computing them are intimately tied together, and each time a metric is designed, there will usually be a need to design an algorithm for it.

- Next, the analysis scope relevant to the application must be determined. The choices are single page level, groups of pages and links, or an entire graph. Similar analysis can be done with varying scopes for different applications.

- Finally, it must be decided if hyperlink analysis is to be done by itself, or in conjunction with Web content and Web usage analyses. If the latter, then the results must be integrated.

Following such an approach can help to better leverage the growing body of techniques and experiences with hyperlink analysis. The survey "Hyperlink Analysis," by Desikan et al. [DSK+02], presents a detailed classification of literature using such a methodology.

Chapter 3

WORK REDUCTION TECHNIQUE FOR A LARGE STATIC GRAPH.

3.1 Introduction

As evident from the discussions in previous chapters, graphs are a common way to represent interactions and relationships between various objects. The growth of the technology sector, especially the Internet, has enabled users to collect huge amounts of data with relative ease. This is largely because of the World Wide Web. Webpage creation, crawling mechanisms, and cheap disk space have enabled the generation of large graphs. While the sizes of these graphs grow, the key properties of interest of such graphs, such as node relevance, remain the same for applications such as search engines.

Problem Definition: Given a large graph $G<V, E>$, to develop a work reduction technique for efficient computation of first-order Markov measures such as PageRank.

Principal idea: Partition the graph into subgraphs such that computation on each such partition can be done separately to save on the overall computation cost.

This chapter is organized as follows. Graph partitioning techniques are reviewed in section 3.2 and highlight the drawbacks of current techniques in order to address the problem under consideration. Section 3.3 talks in detail about how the graphs were partitioned based on the criteria. Section 3.4 presents the experimental results.

3.2 Traditional graph partitioning techniques

Graph partitioning has been used traditionally to leverage parallel-computing techniques. This is so that a given large graph can be partitioned in a way to minimize the dependencies between the vertices of different partitions with a goal of optimizing parallel computation. Various kinds of balancing constraints are applied while determining the partitions [GGK+03, SKK02]. Given a graph $G = <V, E>$, where V is the set of vertices and E the set of edges, this determines the connectivity between the nodes. Both vertices and edges can be weighted, where $|v|$ is the weight of a vertex v, and $|e|$ is the weight of edge e. Then, the graph partitioning problem consists on dividing G into k disjointed partitions. The goal is to minimize the number of cuts in the edges of the partition, and also to reduce the imbalance of the weight of the subdomains. The weight of a subdomain is the sum of the weights of the vertices allocated in it.

Optimal graph partitioning problem is NP-hard [GJS74]. Hence, research in this area has resulted in heuristic based approximate algorithms that can be applied to large graphs. Graph partitioning algorithms can be classified into *Sequential* and *Parallel Algorithms*

[Fjä98]. While some sequential algorithms tend to give high-quality partitions, they also tend to be too slow for large graphs. This has led to the development of a number of parallel algorithms for graph partitioning. Most early graph partitioning techniques focused on graphs that represent finite element meshes. Such graphs tend to have a uniform degree distribution and do not represent the scale free nature of networks such as social networks, Web-graph, citations, and so forth. Abourjeli and Karypis [AG2005] addressed the issue of partitioning power-law graphs using a multi-level partitioning technique.

However, most of the research in graph partitioning algorithms has either addressed graphs with uniform degree distribution or with undirected graphs. The constraints of the computation of first-order Markov measures for graph partitioning have not been particularly addressed by any of the existing literature. The goals of partitioning for computing first-order Markov measures are different than the goals of the minimum cut approach. The key goal is to identify a partition (which is referred to as a "red" patch) such that there are no incoming links to that partition from any other partition. The optimization criteria would be to obtain as many such partitions (red patches) and minimize the partitions (which are referred as "yellow patches") that have incoming edges but no outgoing edges to other partitions. This dissertation provides a partitioning approach that results in a set of red patches and yellow patches. While this approach does not necessarily guarantee an optimal partition, it provides a mechanism to improve the computational cost significantly.

3.3 Divide and Conquer Approach

The proposed approach makes use of the fact that the PageRank is based on a first-order Markov model. To summarize the model presented in Chapter 3: If a vertex belonging to one set cannot be reached from a vertex belonging to any other set, then the score on this vertex would depend only on the vertices of the set to which it belongs. This is because in a first-order Markov model, the present state depends upon one previous state, and to arrive at the present state, it is imperative to have an incoming link from a previous vertex. This leads to the idea that the PageRank of the vertices belonging to a set A does not depend on the PageRank of the vertices from another set B if there is no incoming links from vertices in set B to vertices in set A. In such a scenario, the PageRank of vertices in set A (referred to as a “red patch”) could be computed independently of the PageRank of vertices in set B (referred to as a “yellow patch”).

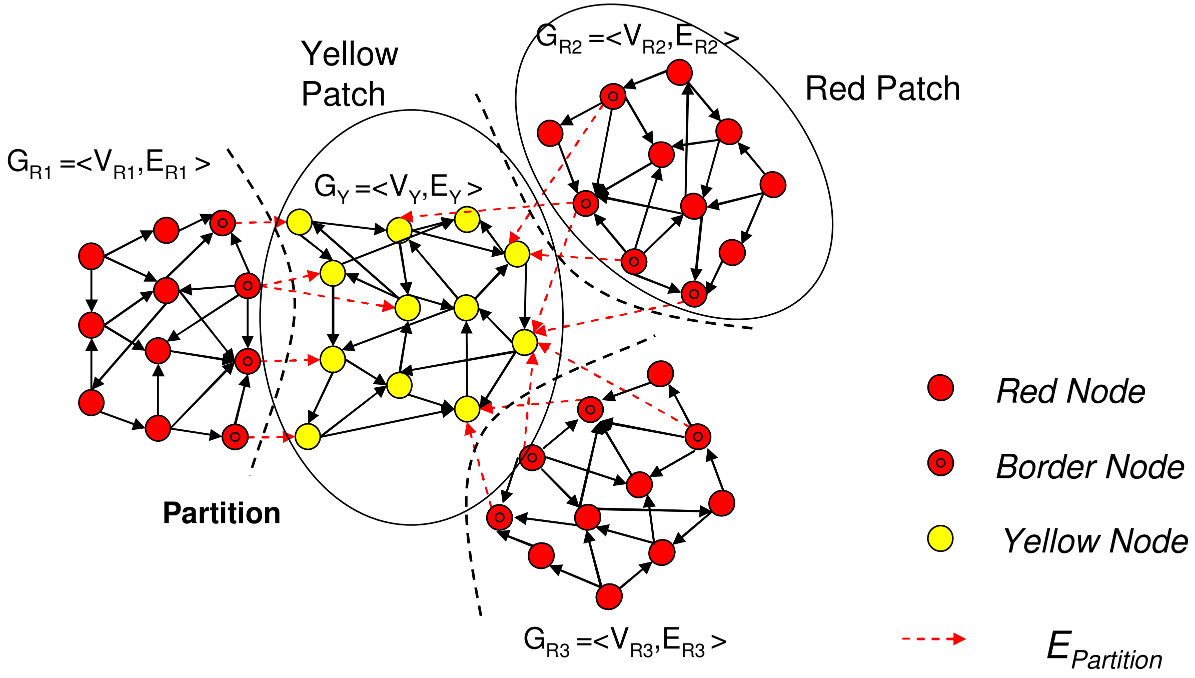


Figure 9. Overview of Work-Reduction for First-order Markov Model Measure.

The above criteria is used to divide the graphs into partitions of “red patches” and “yellow patches.” Once graph is partitioned into sets of “red patches” and “yellow patches,” one can then compute the PageRank of vertices in the “red patches” independently and follow it with the computation of the PageRank for vertices in the “yellow” patch. Such an approach has two advantages. First, it reduces the size of the problem by reducing the size of the graph into smaller subgraphs of “red patches” and “yellow patches.” Such a reduced problem size helps in fitting the graph in the main memory without requiring a machine of high RAM capacity. Second, since the computation of “red patches” can be carried out independently, this process can be parallelized, leading to further optimization and saving on computation time. However, parallelization issues are not dealt with in this dissertation.

3.4 Theoretical Concepts

Let us consider a graph, $G = \langle V, E \rangle$. The idea is to partition graph G into components, $G_1, G_2, \ldots\ldots, G_k$, such that:

(a) $\bigcup_{i=1}^{k} V_i = V; \left(V_i \underset{i \neq j}{\cap} V_j \right) = \phi$

(b) $\left(\left(\bigcup_{i=1}^{k} E_i \right) \cup E_{partition} = E \right); \left(E_i \underset{i \neq j}{\cap} E_j \right) = \phi$

For PageRank, which is based on a first-order Markov model, further constraints apply that prevent the cyclic flow of information, such as for a given partition, G_i:

$$\neg\exists\ e_{xy} \in E \ni v_x \in G_i \wedge v_y \in G - G_i$$

Such a partition, G_i corresponds to the definition of a “red patch” described earlier. In Figure 9, the graph G is partitioned into four partitions such that G_{R1}, G_{R2}, G_{R3} correspond to the “red patches” discussed earlier. G_Y corresponds to the “yellow” patch.

The discussion will now describe the scheme to compute PageRank on such a partitioned graph. Let a graph be partitioned into k “red patches,” G_{R1} to $G_{RK,}$ and a single yellow patch G_Y. The edges from the “red patches” to the “yellow patch” (represented as dotted edges in the figure) form the set $E_{partition}$. In a given “red patch,” G_{RI} , let the vertices (represented by annular circles in the figure) that are the source end of an edge belonging to $E_{partition}$ be denoted as $V_{border,RI}$.

Let us define a graph, G’ such that:

$G' = \langle V', E' \rangle$ where

$$V' = \left\{ \bigcup_{ri=r1}^{rk} V_{border,ri} \cup V_y \right\}; E' = E_{partition} \cup E_y$$

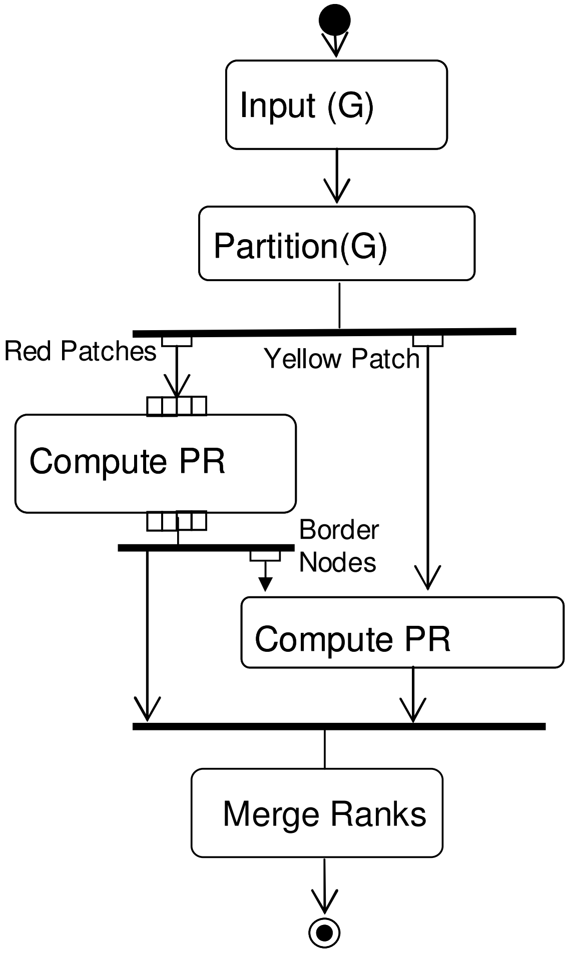


Figure 10. Activity Diagram for Divide and Conquer Approach.

This G’ corresponds to the yellow patch. The PageRank for the whole graph G can now be computed by computing PageRanks independently for all individual “red patches” G_{R1} to G_{RK} and then computing for the graph, G’ as defined above.

3.5 Methodology

Figure 2 presents the overall methodology for computing PageRank on single large static graph. The first key step is to partition the graph and extract patches that have the properties described in the previous sections. Ideally, the user is free to design and use any patch extraction algorithm, provided the patches extracted have the desired properties. A key point to be noted about patch extraction is that it can be pipelined along with the crawling or with a pre-processing step. However, there is an approach that would have a minimal cost to extract patches, though we do not guarantee that the extracted patches represent the optimum patch extraction for PageRank computation as a whole. The steps of the algorithm are illustrated in the figure.

Consider that there are three sets of vertices colored red, yellow, and black. Red colored vertices are those which belong to a red patch. Yellow colored vertices belong to a yellow patch. The black colored vertices are the unexplored vertices. I will now describe in steps how to extract red patches and yellow patches.

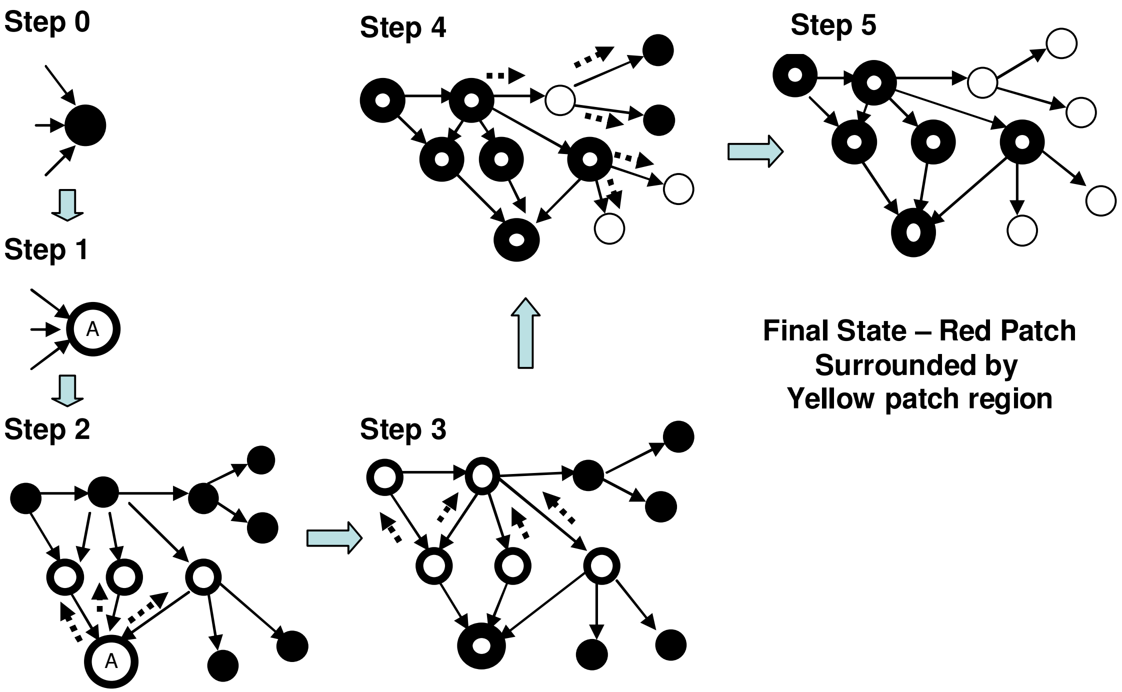


Figure 11. Red Patch Identification.

- **Step 0:** Initially, since all the vertices are unexplored, there are no red and yellow colored vertices, and all the vertices in the graph are colored black.

- **Step 1**: Randomly pick a black colored vertex.

- **Step 2**: Perform a reverse BFS on this vertex—i.e. explore all the ancestors of this vertex by traversing along the incoming links. Color all the black vertices that encounter red. The reverse BFS does not stop until no further traversal is possible or a red vertex is encountered.

- **Step 3**: Label this set of red vertices as a red patch. Select all “peripheral” vertices—i.e. vertices in this red patch that have edges crossing over to any vertex(s) outside this red patch.

- **Step 4**: Perform a normal BFS on each of the peripheral vertices, coloring all the black vertices encountered as yellow. Each of the BFSs continue until no further traversal is possible or a yellow vertex is encountered. It is not possible for any of the BFSs to encounter a red vertex. Thus, one colors all the descendents of each of the peripheral vertices yellow.

- **Step 5**: At this point, there is a red patch surrounded by a yellow patch region. Repeat the entire procedure from Step 1–4 (Steps 1–4 are illustrated in Figure 3), each time extracting a different red patch, until all the black vertices are exhausted. Note that in step 2 it is neither possible to encounter a red vertex from a previously extracted red patch nor a yellow colored vertex. Also in step 4, it is not possible to encounter a red colored vertex from a previously extracted red patch; however, one may encounter a yellow vertex. It is observed that the red patches extracted in steps 1–5 contain a small fraction of the vertices in the graph. An ideal situation would be one where the entire graphic partitioned into equal sized red patches—i.e. no yellow patch. This is because PageRank is computed for the red patches in parallel, and the more the vertices in the red patches, the more will be the gain in efficiency. Even though the ideal situation may not be realized, it is still desirable to include as much possible of the graph as red patches. Therefore, the second part of the algorithm expands the size—i.e. includes as many vertices as possible in the above obtained red patches.

- **Step 6**: Pick a red patch, call it R.

- **Step 7:** Perform a reverse BFS on each of the yellow colored children of the peripheral vertices of this red patch. If during the process of traversal, a red vertex belonging to red patch R is encountered, then all the vertices encountered so far are colored red and included in the red patch R. If a red colored vertex from some other red patch is encountered, then the red patch R cannot be expanded along this path and the reverse BFS is abandoned, leaving all vertices as they were. If one encounters a dead end—i.e. a yellow vertex with no incoming links—then again all vertices encountered so far are colored red and included in the red patch R.

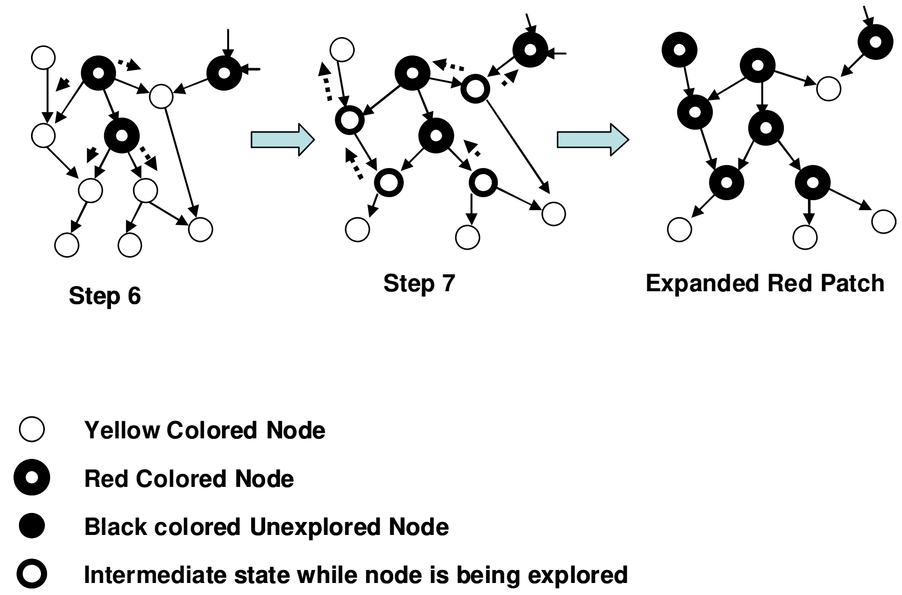

Step 6
Step 7
Expanded Red Patch
Yellow Colored Node
Red Colored Node
Black colored Unexplored Node
Intermediate state while node is being explored

Figure 12. Red Patch Expansion.

- **Step 8**: Perform Steps 6 and 7 for each red patch (Steps 6–7 are illustrated in figure 4).

- **Step 9**: Repeat Steps 6–8 until no change is observed in the size of any of the red patches—i.e. no red patch can be expanded any more.

- **Step 10**: Return the list of red patches and the yellow colored vertices. These are then used for parallel PageRank computation as explained earlier.

After patch extraction, the graph can be treated for dangling vertices by adding a self-loop to each dangling vertex, or one may also choose to delete these vertices. The first option will have no effect on the patches extracted. The second option may result in certain vertices being deleted; however, it will not result in a situation where the properties of the patches extracted are violated. One may also choose to perform these operations before the patch extraction process. Adding edges from a dangling vertex to every other vertex has an adverse effect on the partitioning scheme. In such a case, every vertex will have an incoming link from a dangling vertex. Therefore, every red patch will have to contain these dangling vertices. Since all red patches are disjointed, it is not possible to have more than one red patch. Therefore, in spite of its popularity, this technique for handling dangling vertices will not work with the partitioning scheme.

A good partitioning scheme would be one with patches of a good profile. A good profile is a set of patches that maximize the total number of vertices included in red patches, minimize the size of the largest red patch, and minimize the skew in the red patch sizes. Note that the patch extraction algorithm presented does not necessarily result in an optimal patch extraction for PageRank computation. For instance, no steps are taken to prevent skew in the red patches sizes. Also note that the profile of the patches (i.e.

properties such as number of patches, sizes of patches, and skew in patch size) that are extracted from a graph depends on the black colored vertices that are picked in step 1. In this case, it was chosen to pick a vertex randomly. This is because there may be a more optimal or "intelligent" picking scheme to come up with better patch profile when compared to a random one such as the one presented in this paper.

3.6 Experiments and Results

In this section, the results for the graph partitioning scheme for a static single graph are presented. The experiments were conducted in a machine with an i686 processor running Linux, with 1 GB RAM. These experiments were conducted on synthetic and real datasets. The synthetic data was generated using GTGraph generator [BM2006]. Two kinds of graphs were generated: Power law graphs and Random graphs. Input Parameters varied for the data including the Order (V) and Size (E) of the graph. The real dataset consisted of the link graph for the http://www.cs.umn.edu website. The graph contained a total of 52,183 edges and 12,209 vertices. Prior to partitioning, dangling vertices were taken care of by adding self-loops to each of them.

A convergence threshold of 10^{-8} and a dampening factor of 0.1 for all PageRank computations were used. To examine the experimental accuracy, the L1-norm was computed for the PageRank score vectors returned by the two methods, which was found to be an average less than 10^{-4}. The small error may have been the result of numerical computing issues, and may also depend on the number of iterations and the convergence rate when computed as a whole graph versus a computation for convergence as a small subgraph.

The experiments on the synthetic datasets reveal that the time required by the work reduction technique is more pronounced when the graph is small. In addition, as the size of the graph increases, the percentage reduction increases, even though there is a reduction in computational time. This is due to the formation of small red patches and a large red patch. It was noticed that as the edge size was increased in the generator and specified the number of nodes, the density of the power-law graph increased, and hence the size of the yellow-patch remained large and dominated the computational cost.

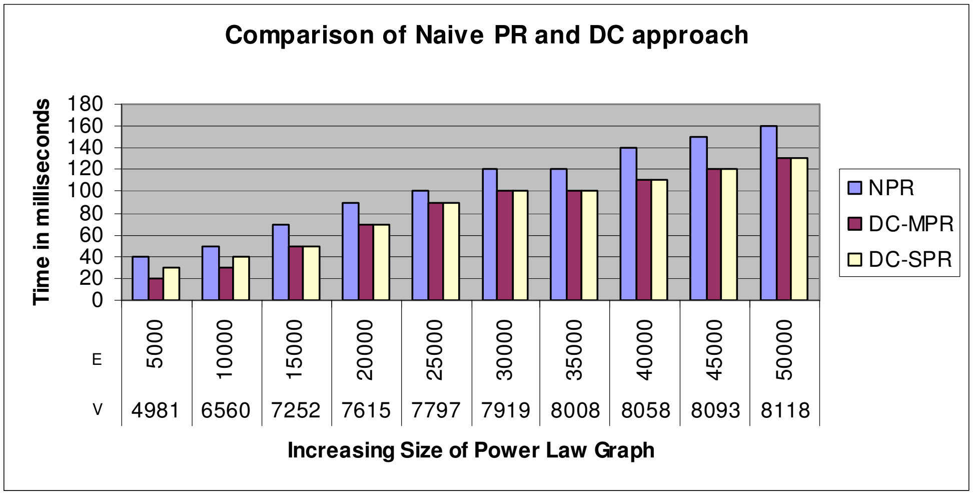


Figure 13 : Performance comparison with increasing size for Power Law Graphs

- 🟦 **NPR:** Naïve PageRank
- 🟥 **DC-MPR:** DC Approach with Red Patch PRs computed in parallel
- 🟨 **DC-SPR:** DC Approach with Red Patch PRs computed sequentially

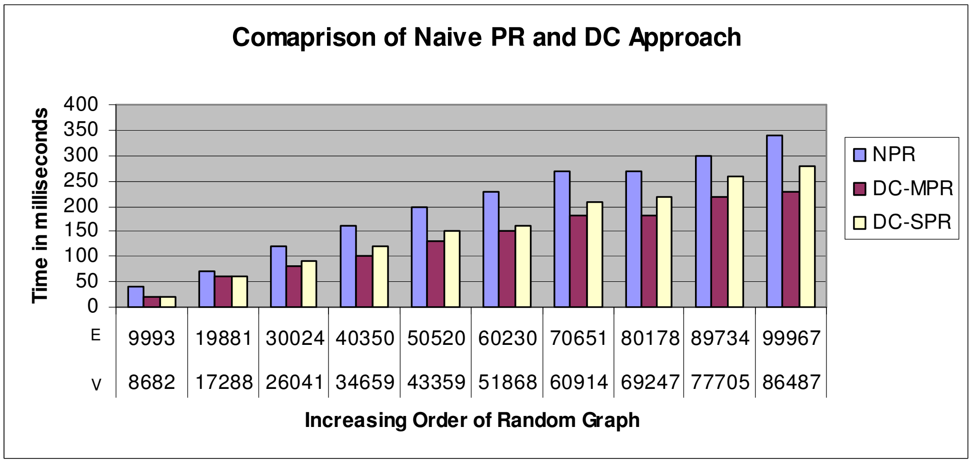
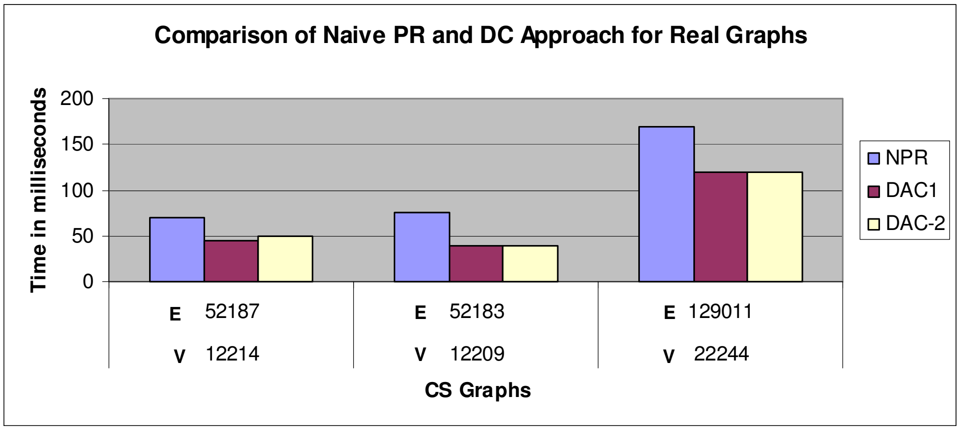


Figure 14. Performance Comparison with Increasing Order for Random Graphs.

The parallel PageRank computations for a Static Single Graph were again run on the link graph for http://cs.umn. The results were simulated for a parallel execution by using the maximum of the runtimes of the PageRank computations of all red patches in place of the runtime of computing PageRank of all red patches in parallel. This is a valid assumption, as there was no process intercommunication.



■ **NPR:** Naïve PageRank
■ **DC-MPR:** DC Approach with Red Patch PRs computed in parallel
□ **DC-SPR:** DC Approach with Red Patch PRs computed sequentially

V	E	NR	vMR	eMR	totvR	toteR	vY	eY	RPE	MRP	totRP	YRP	NPR	DAC-1	DAC-2
12214	52187	326	350	2075	1518	5084	10696	47103	10	10	10	30	70	45	50
12209	52183	326	350	2075	1518	5085	10691	47098	10	10	10	20	75	40	40
22244	129011	343	315	1852	1643	5501	20601	123510	30	10	10	80	170	120	120

Figure 15. Performance Comparison with Increasing Size for Power Law Graphs.

The results on the real dataset that represents the computer science website confirm the findings from the experiments on the synthetic dataset that the work reduction technique is faster, but the effect of time-reduction gradually reduces as the size of the graph increases and the size of the yellow patch dominates the computation cost.

3.7 Conclusions

In this chapter, a "divide and conquer" approach was provided for the purpose of partitioning a graph in order to enable efficient computation for measures and metrics based on the first-order Markov model. In addition, a theoretical framework to demonstrate how such a partitioning scheme can be used to divide the problem into smaller problems was presented. The experimental results show that such an approach improves the computation over using naive algorithms by a significant amount.

Chapter 4

INCREMENTAL COMPUTATION ON LARGE EVOLVING GRAPHS

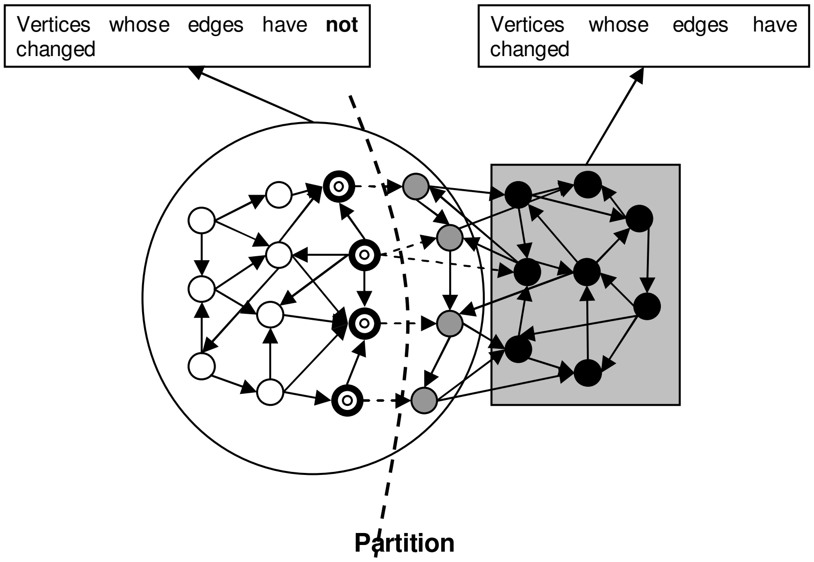
4.1 Introduction

An important dimension of Web mining is the evolution of the Web-graph. The Web is changing over time, and so are the interaction of users on (and with) the Web, suggesting the need to study and develop models for the evolving Web Content, Web Structure and Web Usage. The study of this evolution of the Web requires computing the various existing measures for the Web-graph at different time instances. A straightforward approach would be to compute these measures for the whole Web-graph at each time instance. However, given the size of the Web-graph, this is becoming increasingly unfeasible. Furthermore, if the percent of nodes that change during a typical time interval when the Web is crawled by search engines is not high, a large portion of the computation cost may be wasted on re-computing the scores for the unchanged portion. Hence, there is a need for computing metrics incrementally in order to save on the computation costs.

To study changes in graph structure over time, techniques for incremental computations depend on the underlying knowledge model that defines a metric [DSK+02]. For example, the computation of hub and authority scores is based on mutual reinforcement of nodes, and hence a change in the in-degree or out-degree of a node may affect its score. Mutual reinforcement makes hub and authority scores a second order model. However, for PageRank, whose random surfer model is based on the first-order Markov property, the change in out-degree of the node does not affect the score of the node. Hence, the level of penetration of change in scores due to a change in the degree of a node is not as high in PageRank as it is in hub and authority scores. The focus of this chapter is on incremental computation of first-order Markov measures such as PageRank.

Problem Definition: Given snapshots of evolving graphs at two consecutive time instances— G_1,G_2 —to compute a first-order Markov measure of the graph at the second time instance, $G_{2,}$ in a cost effective manner.

This chapter is organized as follows. The overview of the approach for incremental computation is presented in Section 4.2. The theory is discussed in Section 4.3, and Section 4.4 provides the methodology for the computing. The experiments and results are presented and analyzed in Section 4.5. Section 4.6 discusses the conclusions.


Vertices whose edges have **not** changed
Vertices whose edges have changed
Partition

○	v_l	Vertex on left partition, which remains unchanged
◎	v_b	Vertex on the border of left partition from which there are outgoing edges to the right partition
◍	v_{ra}	Vertex on right partition, which remains unchanged but whose PageRank gets affected by vertices in changed partition
●	v_{rc}	Vertex on right partition, which has changed or has been added.

$$G_2' = \left\langle (V_l \bigcup V_b), E_l \right\rangle$$

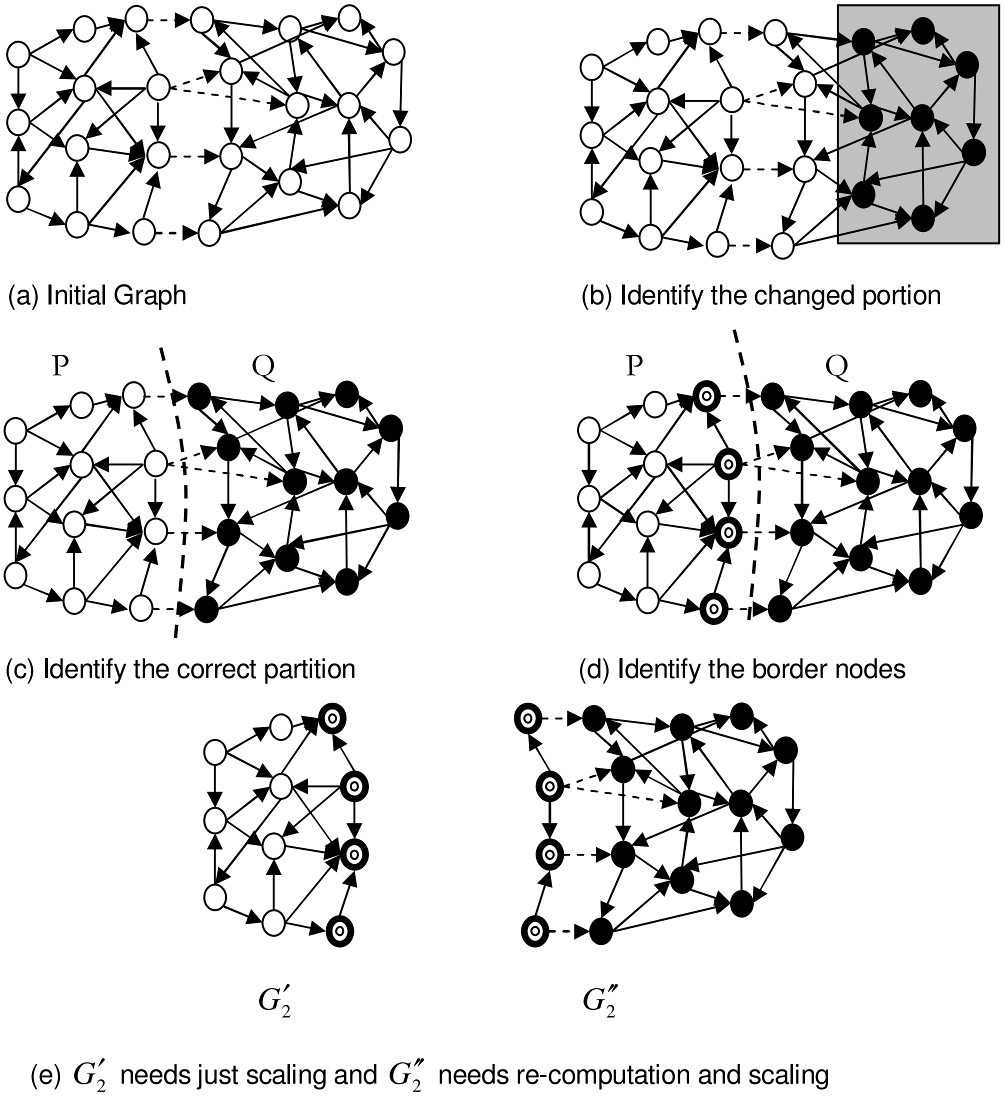
$$G_2'' = \left\langle (V_b \bigcup V_{ra} \bigcup V_{rc}), (E_{part} \cup E_{rc}) \right\rangle$$

Figure 16. Partitioning Technique.

4.2 Overview of Approach

The principle idea of this approach is to find a partition, *P*, of the graph $G\langle V, E\rangle$ such that there are no incoming links from a partition, *Q* (includes all changed nodes), to *P*. In such a case, the PageRank of the partition, *Q,* is computed separately and later scaled and merged with the rest of the graph to get the actual PageRanks of vertices in *Q*. The scaling is done with respect to the number of vertices in the partition, *P* ($n(P)$), to the total number of nodes in the whole graph ($n(P \cup Q) = V$). The PageRank of the partition *Q* is computed, taking the border vertices that belong to the partition *P* and have edges pointing to the vertices in partition *Q*. The PageRank values of partition *P* are obtained by simple scaling.

This basic idea of partitioning the Web-graph, and computing the PageRanks for individual partitions and merging, works extremely well when incrementally computing PageRank for a Web-graph that has evolved over time. Given the Web-graphs at two consecutive time instances, one must first determine the portion of the graph that has changed. Figure 16 illustrates the approach for partitioning the graph such that computation of PageRank for only one partition is necessary.



(a) Initial Graph

(b) Identify the changed portion

(c) Identify the correct partition

(d) Identify the border nodes

(e) G_2' needs just scaling and G_2'' needs re-computation and scaling

Figure 17. Steps to Identify Partitions for Computation.

4.3 Theoretical Concepts

The whole concept is illustrated in Figure 2. Let the graph at the new time be $G_2\langle V,E\rangle$ and

Let,

v_{rc} = Vertex on the right partition which has changed, or has been a new addition.

Therefore, the set of changed vertices can be represented as,

$$V_{rc} = \{v_{rc}, \forall v_{rc} \in V\}$$

Let V_{lu} represent the set of vertices on the left, which have not changed.

$$V_{lu} = V - V_{rc}$$

v_{ra} = Vertex on the right partition which remains unchanged, but whose PageRank is affected by vertices in the changed component.

These set of vertices can be denoted by the set,

$$V_{ra} = \{v_{ra}, \forall v_{ra} \in V\}$$

In order to identify a partition for which PageRank has to be re-computed, one must identify all vertices that are not already in V_{rc}. In order to do so, for every vertex in V_{rc}, it was essential to perform a BFS to find out all vertices, V_{ra} (defined below) reachable from V_{rc}. A union of these sets of vertices, V_r, is the set of all vertices for which PageRank has to be recomputed. Hence,

$$V_r = V_{rc} \cup V_{ra}$$

$$E_r = \{e_{x,y} | x \in V_r \ and \ y \in V_r\}$$

Let the graph containing the set of vertices, V_r and the edges among them, E_r be denoted as G_r . Hence,

$$G_R = \langle V_r, E_r \rangle$$

Let the set of vertices that remains unaffected, which form the left side of the partition, be denoted by V_U . By construction there is,

$$V_U = V - V_r$$

Let E_l be the set of edges between all vertices that belong to V_l . Hence,

$$E_U = \{e_{x,y} | x \in V_l \ \& \ y \in V_l\}$$

The graph for which re-computation is not necessary can thus be defined as,

$$G_2' = \langle V_U, E_U \rangle$$

Let V_b be the set of vertices in G_2' that have outgoing edges to vertices belonging to V_r .

$$V_b = \{v_b | \ \exists \ e_{x,y} \ni x \in V_l \, and \ \ y \in V_r\}$$

The set of partitioning edges can be defined as,

$$E_{Part} = \{e_{x,y} \mid x \in V_b, y \in V_r\},$$

Hence the set of vertices whose PageRank has to be computed in the incremental approach corresponds to the partition G_2'', and can be denoted as,

$$V_A = V_r \bigcup V_b$$

$$E_A = E_{Part} \bigcup E_r$$

The graph for which PageRank is recomputed can thus be defined as,

$$G_2'' = \langle V_A, E_A \rangle$$

Now, we know that the graph G_2' has remained unchanged from the previous time instance and that the PageRank of vertices in this partition is not affected by the partition, G_R. The distribution of PageRank values for the nodes in partition G_2' is going to be the same as it was for the corresponding nodes in the previous time instance G_1. Thus, the PageRank of the vertices in partition G_2' could be calculated by simply scaling the scores from the previous time instance. The scaling factor will be $|V(G_1)|/|V(G_2)|$. The PageRank for the partition G_2'' can be computed using the regular PageRank Algorithm and scaled for the size of the graph, G. Since the percentage change within the structure of the Web is not high, the computation of the changed portion will be a smaller graph compared to the whole Web. Moreover, the existence of such partitions is also suggested by the bow-tie model of the Web [KRR+00], where about 27 percent of Web contributes to the influx. It should also be noted that when computing PageRanks for the changed portion, in order to maintain the stochastic property of the incremental matrix, one must scale the PageRanks of nodes in V_b such that they correspond to the number of nodes for which the PageRank is actually computed. Additionally, the out-degree of these border nodes that have edges in partition G_2' are also taken into account, since the way they distribute their PageRanks to nodes in partition G_2'' will depend on their out-degree.

4.4 Methodology

This section desciribes the incremental algorithm to compute PageRank. The initial step is to read the graph at a new instance and determine the vertices that have changed. This does not require additional time, as it can be computed as one reads the new graph. Thus, after reading the graph, one can assume that there are two sets of vertices—one containing the vertices which have changed from a previous time instance, and the other containing vertices that have remain unchanged.

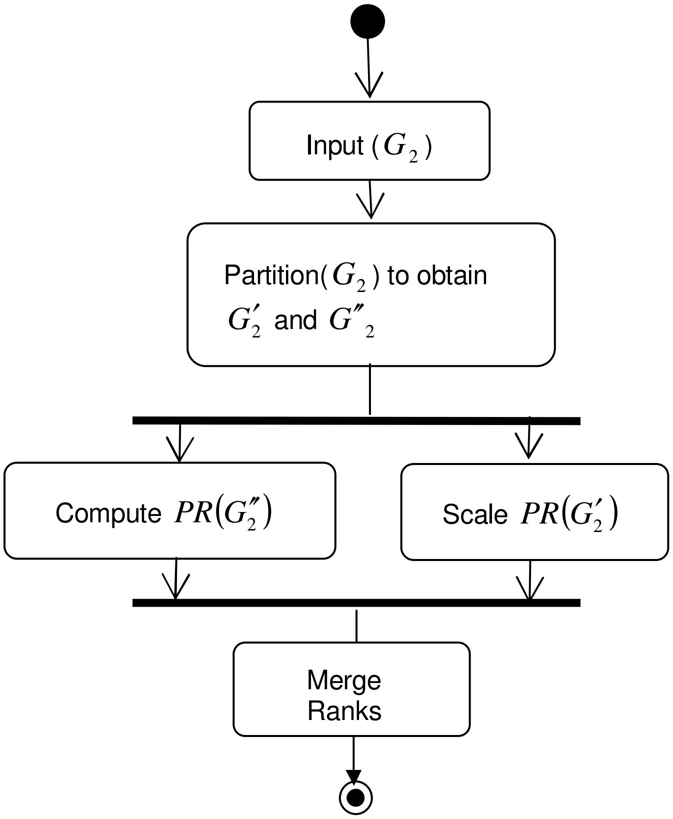


Figure 18. Activity Diagram for Incremental Computation

Step 1: Initialize a list L that represents a list of vertices for which PageRank has to be re-computed.

Step 2: A change in a vertex induces a change in the PageRank distribution of all its children. All such changed vertices, V_{rc}, are in the queue, Q_c. In this step, the set of “changed vertices” is extended to a partition to include the set of vertices, V_{ra}, whose PageRanks are affected by vertices in V_{rc}. The set of vertices, V_{ra} is pushed into the list L.

Step 3: For the remaining vertices, V_l, there is no change in their PageRank distribution. The PageRank is simply obtained by scaling the previous PageRank scores. The scaling factor is $|V(G_1)|/|V(G_2)|$. In addition, all the border vertices, V_b, from this set of unchanged vertices that point to a changed vertex, will influence the PageRank value of that changed vertex. Thus, these too must be included in the list L, as their PageRank scores will be required for computing the PageRank scores for the changed vertices.

Step 4: Now the original PageRank computation algorithm along on the vertices that are in list L, and colored violet (i.e. vertices which have changed) to get the new PageRank values for these changed vertices. Thus, one ends up localizing the changed partition to a certain sub-graph of the web, which consists of all changed vertices, and then the basic PageRank algorithm is performed only on this changed sub-graph. The PageRank value for the rest of the vertices is simply a matter of scaling the previous values. It is important to take steps to ensure that the stochastic property of transition matrix is performed.

It should be noted that this sub-graph thus obtained also contains border vertices whose PageRank would not have changed with the graph evolution except for the scaling factor. Including these border nodes along with their scaled PageRank for the new graph is important for correctness of the PageRank of vertices in the changed portion. In addition, since the PageRanks of these nodes are known and can be used for the starting vector, it helps in faster convergence.

```
IPR
Step 1 – Initialize a list of vertices to $L$
Step 2 – Pop a Vertex $v$ from $Q_c$
    2.1 For all the children of $v$
    if children of $v \in$ list $Q_u$
            remove them from $Q_u$
            push them in $Q_c$
    2.2. Push $v$ in $L$ and repeat step 2 till
    queue $Q_c$ is empty
Step 3 – For each element in list $Q_u$
    3.1 Take the element and scale the previous pagerank value to get
    new pagerank value.
    3.2 Look up whether any of the children, of the vertex contained
    in $Q_u$ is contained in $L$, if so remove this element of, $Q_u$ copy it
    in a list $L_b$.
Step 4 – Perform PageRank on graph, $G_2''$ whose vertices are $(L \cup L_b)$
```

Algorithm 3. Incremental PageRank Algorithm.

4.5 Experiments and Results

In this section the results for the incremental computation on large evolving graphs is presented. The experiments were conducted in a machine i686 processor running Linux with 1 GB RAM. The experiments were conducted on synthetic and real datasets. The synthetic data was generated using GTGraph Generator [BM2006]. Two kinds of graphs were generated: power law graphs and random graphs. It should be noted that the model for partition or the algorithm for partitioning does not rely on the nature of the graph. Hence, the approach is applicable to any class of graphs. The experiments are aimed at validating this claim by applying the approach to both random graphs as well as power-law graphs. The goal is to identify at what point it is beneficial to use the incremental approach, given the growth models of such graphs are different. Based on the chosen synthetic graph generator, the input parameters varied for the data include the Order (V), Size (E) of the graph.

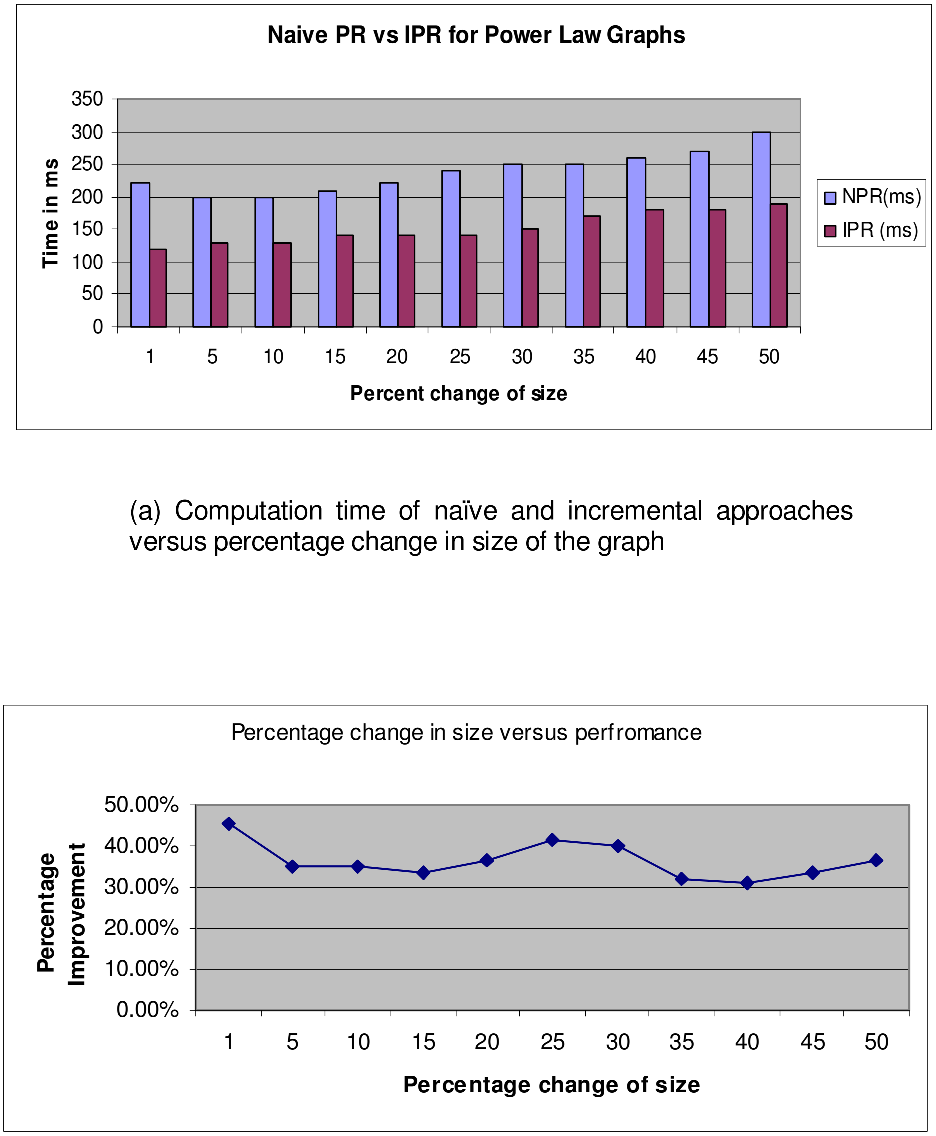
Another important issue in the computation of PageRank is the handling of dangling nodes. Dangling nodes are nodes with no outgoing edges. These nodes tend to act as rank sink, as there is no way for rank to be distributed among the other nodes. The suggestion made initially to address this problem was to iteratively remove all the nodes that have an out-degree of zero, and compute the PageRank on the remaining nodes [PBM+98]. The reasoning here was that dangling nodes do not affect the PageRank of other nodes. Another suggested approach was to remove the dangling nodes during the initial computation and add them back during the final iterations of the computation [KHM+03]. Other popular approaches to handling dangling nodes are to add self loops to dangling nodes[JW2003, EMT04] and to add links to all nodes in the graph, G from each of the dangling node to distribute the PageRank of the dangling node uniformly among all nodes[PBM+98].

For the power-law graphs, the initial graph G_{t_0} at time instance t_0 had an order of 27557 ($|V|$) and a size of 50000 ($|E|$). Subsequent graphs that modeled the evolution were constructed by varying percentage increases in size as the parameter. Since the power-law graphs follow a certain distribution, it is not possible to specify the exact desired order and size and retain the power-law properties. Hence, the variation considered was on the size, since it is the size which influences the computation of PageRank more than the order of the graph. IPR performs better with an increase in the size of power-law graphs. Percentage improvement varies little with percentage change in size, and an average 40 percent improvement can be seen in this case.

For random graphs, the starting graph had an order of 8,585 and a size of 9,993. The percentage change in the size varied from 100 percent to 1,000 percent, and the computation time for the naïve approach versus the incremental approach was measured. It was observed that IPR performs better with an increase in the size of random graphs until there is a change of around 200 percent increase in size. After that, the percentage improvement decreases with the percentage change in size. This is due to the expansion to large patch equivalent to graph at new instance.

The real dataset consisted of the link graph for the http://www.cs.umn.edu website. The graph contained a total of 52,183 edges and 12,209 vertices. Prior to partitioning, dangling vertices were taken care of by adding self-loops to each of them.

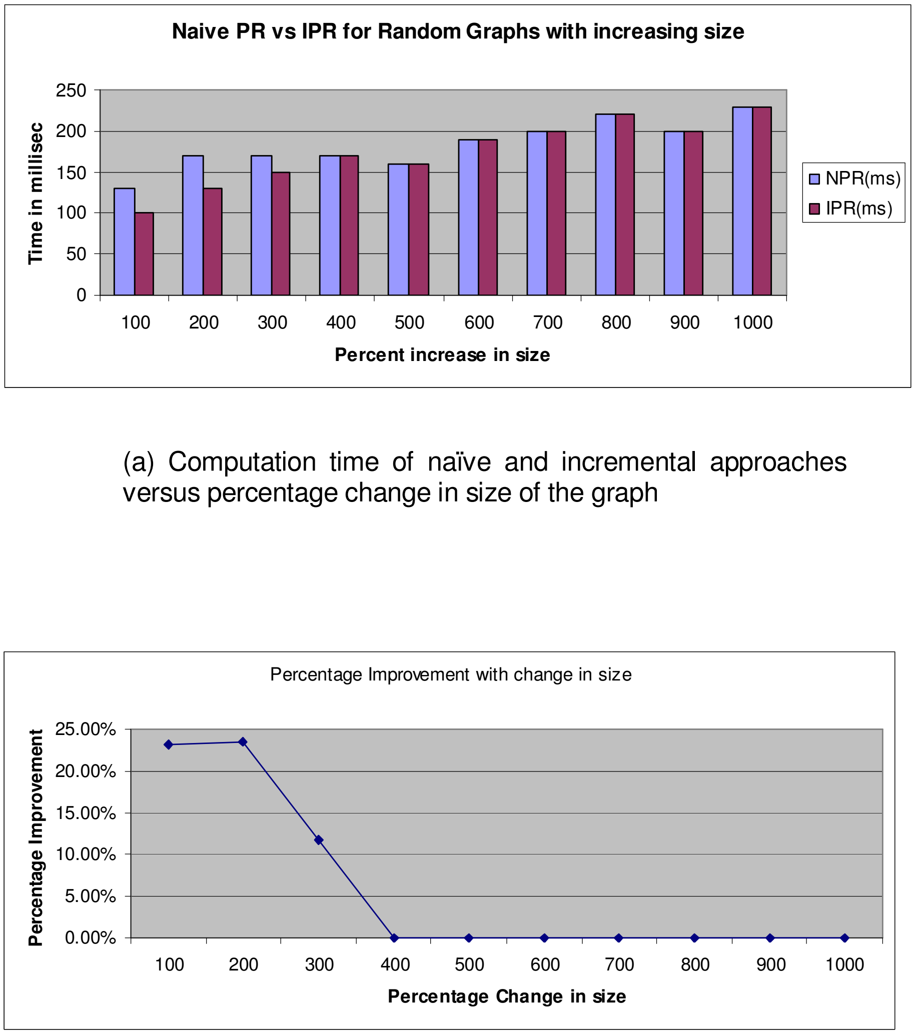
These results are from actual experiments conducted on the Computer Science and Institute of Technology websites. For the Computer Science website, in the first time interval of eight days, there seemed to be a significant change in the structure of the website—about 60 percent of the pages had changed their link structure. It was discovered that such a sea change occurred because a whole subgraph that contained the documentation for Matlab help was removed. The incremental approach still, however, performed 1.86 times faster than the naïve PageRank. Similarly, for a period of ten days, the incremental approach performed around 1.75 times faster. For a period of two days, the improvement was 8.65 times faster.



(a) Computation time of naïve and incremental approaches versus percentage change in size of the graph

(b) Percentage improvement in performance versus percentage change in size of the graph

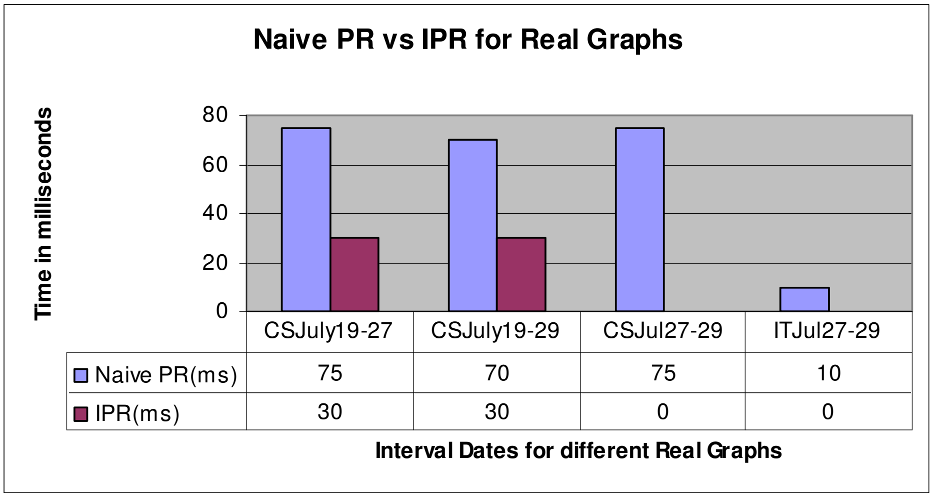
Figure 19. Performance Variation of Power-law Graphs.



(a) Computation time of naïve and incremental approaches versus percentage change in size of the graph

(b) Percentage improvement in performance versus percentage change in size of the graph

Figure 20. Performance Variation of Random Graphs.



	V(First)	E(First)	V(Second)	E(Second)	V(Red)	V(Green)	V(Yellow)
CSJuly19-27	22244	129011	12209	52183	4266	7943	8127
CSJuly19-29	22244	129011	12214	52187	4849	7365	7550
CSJul27-29	12214	52187	12209	52183	11529	680	731
ITJul27-29	4596	15052	4596	15052	4596	0	0

Figure 21. Performance of IPR on real datasets

The Institute of Technology website typically is a website that does not change very often. There was no change over a period of two days in the Web structure. Since there was no change detected, it was not necessary to compute the PageRank for the graph at the new time instance. By this measure, it was 11 times faster.

4.6 Conclusions

This chapter has provided an approach to compute PageRank incrementally for evolving graphs. The key observation is that the evolution of the Web-graph is slow, with large parts of it remaining unchanged. By carefully delineating the changed and unchanged portions, as well as the dependence across them, it is possible to develop efficient algorithms for computing the PageRank metric incrementally. This generic approach can be applied to any algorithm that has been developed for efficient computation of the PageRank metric. Experimental results show significant speedup in computation of PageRank using the approach as compared to naive approach. Moreover, in the incremental approach, if the partitioned sub-graph that has changed is small, the whole PageRank computation might perhaps be performed in the main memory.

Chapter 5

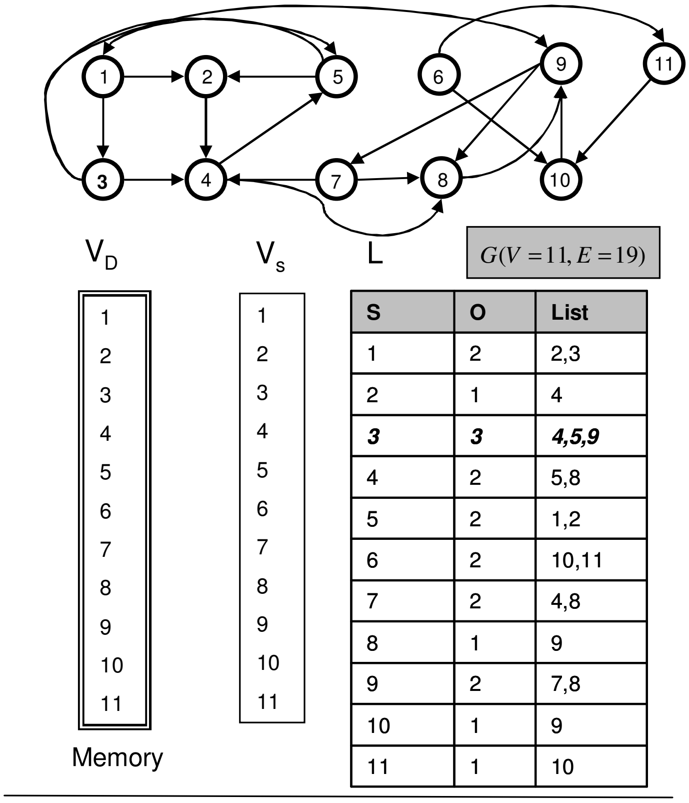
IMPROVING I/O EFFICENCY

In this chapter the focus is on algorithms to improve the I/O efficiency of computation of first-order Markov Models

5.1 Deeper inside PageRank computation

The key idea of PageRank is that a page has high rank if it is pointed to by many highly ranked pages. PageRank computation is carried out using the iterative power method where the graph is represented as an adjacency matrix. The existence of a stationary vector of PageRank in the iterative power method is guaranteed only if the Web-graph is strongly connected and is aperiodic. To ensure the condition of strong connectedness, the dampening factor is introduced, which assigns a uniform probability to jumping to any page. In a graph theoretic sense it is equivalent of adding an edge between every pair of nodes with a transition probability of d/n. This transition probability vector can also be used for personalization. The aperiodic property is also guaranteed for the Web-graph by proper handling of the dangling nodes and ensuring they do not act as rank sinks. Different approaches have been taken to handle dangling nodes, which act as a rank sink, such as to iteratively remove all the nodes that have an out-degree of zero, to remove the dangling nodes while computation initially and add them back during the final iterations of the computation [H1999], to add self loops to dangling nodes [JW2003, EMT04] and to add links to all nodes in the graph, G from each of the dangling node. Dangling nodes are handled by adding self loops to all nodes. Once these issues are taken care of the convergence of the PageRank vector using power method is guaranteed.

Figure 22 illustrates the cost of a PageRank computation using the basic algorithm. The destination vector resides in the memory. During a single iteration, each line read from the link file representing the adjacency matrix, the source node and the list of nodes in its adjacency list is read. The contribution of source node to these lists of nodes is then computed using the PageRank score of the source node (stored in the source vector file) from the previous iteration and its out-degree. The next line is then read and the process repeats till all the lines of the link files are read. At the end of this iteration PageRank scores of all the nodes are computed and stored in destination vector in the memory. This destination vector now has to be written to disk so that it can be used as the source vector for the next iteration. Hence the total I/O cost would be: $|V_S| + |L| + |V_D|$.

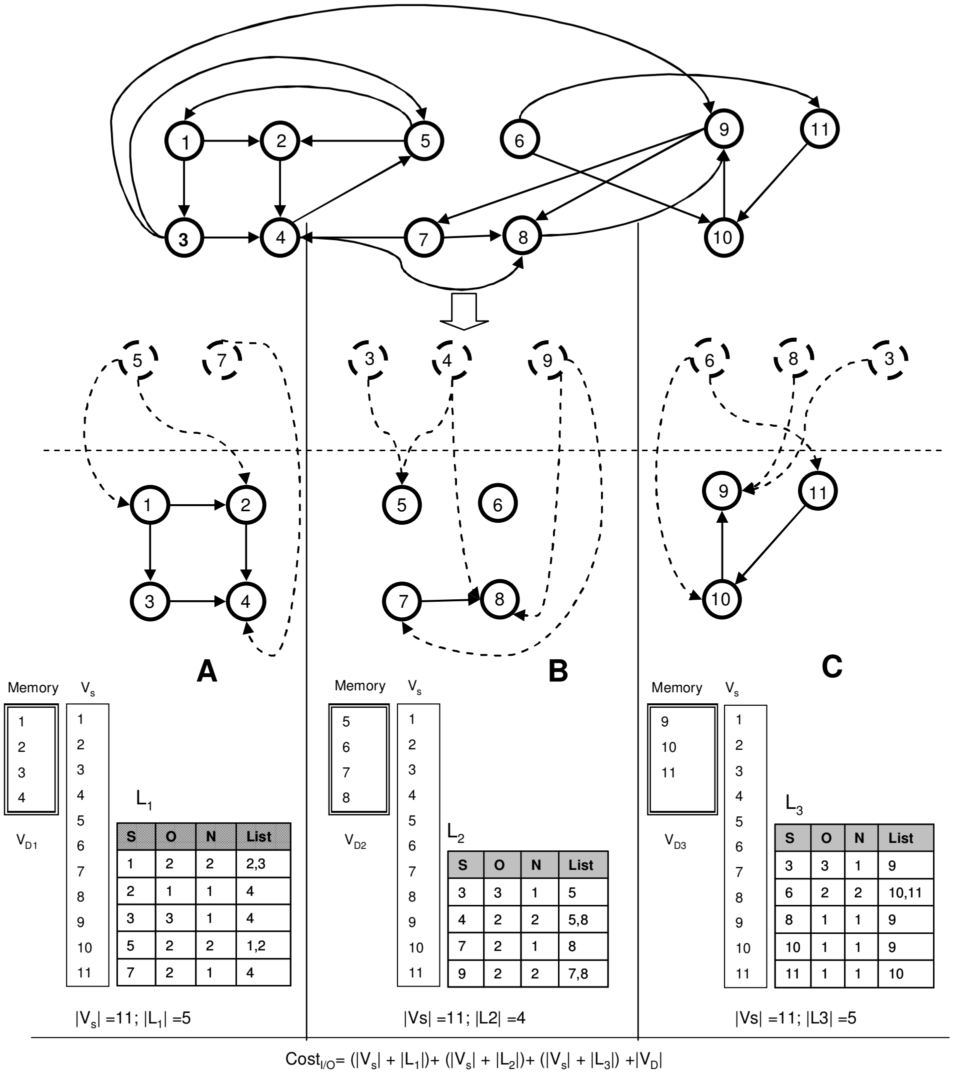


S	O	List
1	2	2,3
2	1	4
3	***3***	***4,5,9***
4	2	5,8
5	2	1,2
6	2	10,11
7	2	4,8
8	1	9
9	2	7,8
10	1	9
11	1	10

$Cost_{I/O} = |V_s| + |L| + |V_D|$

Figure 22. Basic PageRank Computation Approach

Haveliwala [H1999] proposed a memory efficient approach of computing PageRank when the destination vector entirely does not fit in the memory. The key idea behind this approach was to partition the graph in such a way that each partition contains nodes whose PageRank will be computed and the edges among these nodes and also the nodes from other partitions which have edges to nodes in this partition. This ensures that in a partition there are a set of nodes whose all incoming nodes are captured, which translates into the fact that for a given set of nodes the PageRank contribution from all its incoming neighbors are captured. And if the size of these nodes can fit in the memory, regular PageRank algorithm could be applied to compute the PageRank scores of these nodes. And this process has to be repeated for each partition thus ensuring the complete computation of PageRank scores of all partitions. In this approach, a given node's children are split so that each set in the split belongs to one of the partitions which are designated for a set of destination nodes to which these children belong to. For example from Figure 23 it is observed that 3→4, 5, 9 is split as 3→4, 5 of partition B and 3→ 9 of partition C This split results in increased storage in terms of the Adjacency matrix and hence the number of I/Os required for reading the graph increases. Also the source vector has to be read for computation in every partition. If there are 'n' number of partitions the total I/O cost is : $n.|V_S| + |V_D| + |L|.(1 + \delta)$ where δ account for the number of additional lines in the graph file due to splitting the adjacency list of a node according to the partitions.



Memory (V_{D1}): 1, 2, 3, 4 — V_s: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

L_1

S	O	N	List
1	2	2	2,3
2	1	1	4
3	3	1	4
5	2	2	1,2
7	2	1	4

$|V_s| = 11$; $|L_1| = 5$

Memory (V_{D2}): 5, 6, 7, 8 — V_s: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

L_2

S	O	N	List
3	3	1	5
4	2	2	5,8
7	2	1	8
9	2	2	7,8

|Vs| =11; |L2| =4

Memory (V_{D3}): 9, 10, 11 — V_s: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

L_3

S	O	N	List
3	3	1	9
6	2	2	10,11
8	1	1	9
10	1	1	9
11	1	1	10

|Vs| =11; |L3| =5

$Cost_{I/O} = (|V_s| + |L_1|) + (|V_s| + |L_2|) + (|V_s| + |L_3|) + |V_D|$

Figure 23. Block PageRank Computation Approach (also called as Block PageRank (BPR))

5.2 The Approach

In the approach to improve the efficiency of I/O based computation of PageRank one firstly categorizes the problem based on the two scenarios:

- Single Large Graph
- Evolving Graphs

PageRank computation can be made efficient by parallelizing computations or reducing the work to compute. The proposed approach to both classes of problems mentioned above is to reduce the amount of work to compute. In order to do so, one should consider reducing the key bottleneck of PageRank computation - number of iterations required to converge. For a single iteration all the edges of the graph have to be traversed, and hence the cost of a single iteration cannot be improved. However, the overall number of iterations can be reduced by leveraging the fact certain portions of the graph converge faster and that there are faster approaches to compute PageRank when the graph fits in the memory. In the following subsections, it will be discussed how each class of problem to make the computation more efficient is approached. It should be noted that while the proposed approach for a single graph is similar to the approach proposed by using the block-structure of the web [KHM+03], it differs in the fact that it is more generic not restricted to a graphs from the Web where blocks can be approximated us doing the directory structure of the Website. Secondly, this approach relies on earlier work of using a divide-and-conquer approach [DPS+06] and incremental pagerank computation

[DPS+05] to expedite PageRank computation of the local partitions. The approach is also described in a related paper by the author [DS2008]. Also the related work mentioned [KHM+03] does not address the issue of evolving graphs.

5.2.1 Single Large Graph

Once the memory configuration is known, the graph can be partitioned according to either the number of entries from the destination vector or according to the size of a subgraph that will fit in the memory. Different graph clustering and graph partitioning can be applied to find a partitioning that eventually minimizes the δ factor. However, one should also consider the cost of partitioning the graph with respect to the cost of PageRank computation. Most graph partitioning and clustering algorithms assume the graph to fit in the memory, which is not true for large graphs such as the Web. It is assumed that the nodes to be binned as the graph is being crawled depending on the predetermined number of nodes the bin can hold. While this may not be the optimal binning strategy this ensures virtually no additional binning or partitioning cost for PageRank computation. The next step is to compute the PageRank of nodes belonging to a bin (or the subgraph formed due to the binning) using the divide and conquer approach [DPS+06].

The divide and conquer approach relies on the fact that certain portions of the graph can be identified such that the PageRank of those portions can be computed independently. This overview is illustrated in Figure 24.

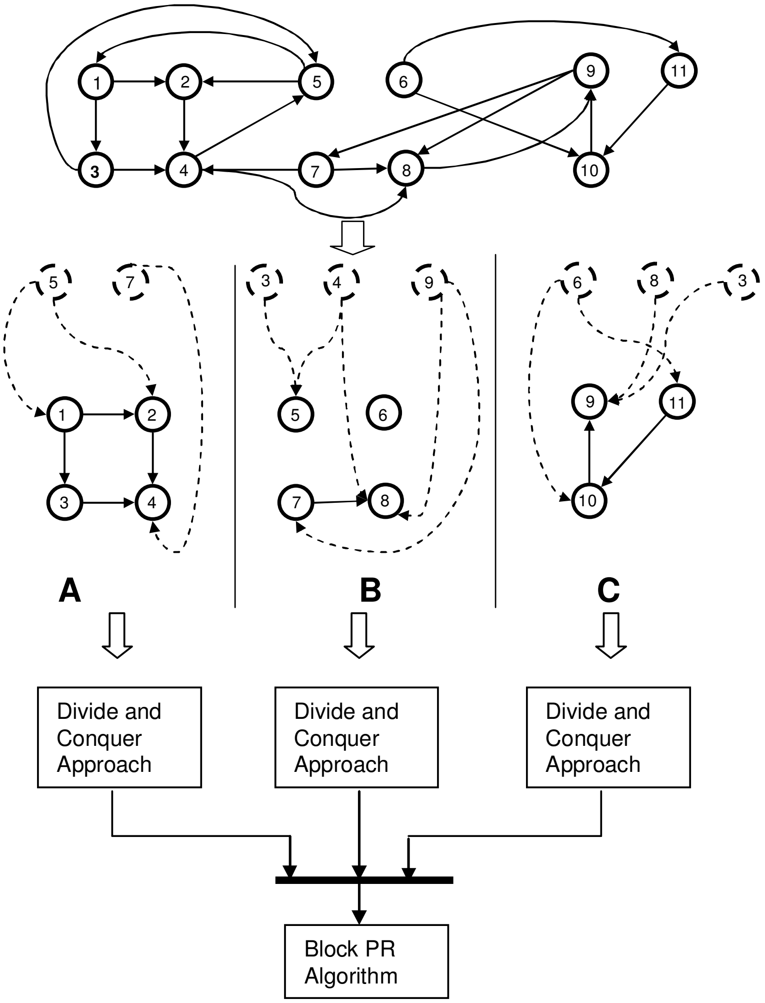

1
2
5
6
9
11
3
4
7
8
10
5
7
1
2
3
4
A
3
4
9
5
6
7
8
B
6
8
3
9
11
10
C
Divide and Conquer Approach
Divide and Conquer Approach
Divide and Conquer Approach
Block PR Algorithm

Figure 24. . Overview of approach for PR computing for a Single Large Graph

The first step is to identify portions of graphs whose PageRank computation is not affected by rest of the graph as there are no incoming links to these portions from the rest of the graph. This can be achieved by doing a Breadth-First-Search following incoming links from a randomly picked node. For further details, refer to the original paper [DPS+06] or Chapter 3 of this dissertation. The computation of PageRank for each partition has to be completed to obtain the approximate PageRank Vector which can be used as starting point for the global computation.

A key strategy applied here is to limit the number of iterations for the computation of local PageRank by specifying a lower threshold for the L_1 norm or setting a low threshold for the maximum number of iterations. Finally, the PageRank vector obtained by the above computations was retrieved and used as the starting vector and applied to the block PageRank to compute the PageRank. In doing so, the PageRank vector converges at a faster rate and reducing in the overall number of iterations that saves more time than the time spent on computing the PageRank for the individual partitions using the divide-and-conquer approach. It is noteworthy that the approach is generic and it does not rely on grouping nodes by domain. However, at the same time the approach is open to any such grouping that can improve the performance additionally.

5.2.2 Evolving graphs

The next focus is on computation of PageRank for evolving graphs. The scope of the problem is kept to changing of graph from one time instance to the next time instance. The scope is not expanded to computing simultaneously the PageRank of series of graphs over a time period.

It is proposed that an incremental computation method be used for computing PageRank of graph *G*, which has evolved from a time instance *t* to *t*+Δ*t*. Nodes and edges are referred to at time instance *t* as 'old' and nodes and edges that have surfaced in time instance *t*+Δ*t* as 'new' in the discussions below. The first step is to identify the portions of the graphs that have changed, which can be pipelined with crawling. The changes can be broadly considered into three categories:

- **Changes exclusive to new nodes**: For all the new nodes in the new time instance, this case involves only edges between the new nodes. This can be viewed as a new subgraph that is formed due to the evolving graph. This is depicted in Figure 25 by nodes '16' to '23' and the edges between them. These nodes are considered to belong to a new partition. The destination vector in this partition will consists only of these new nodes.

- **Changes exclusive to old nodes**: These changes include change in the portion of the graph that existed at time instance. These can include addition of new edges within a partition or between partitions or deletion of existing edges. The only changes these will result is in the change in the link file size of memory efficient PageRank computation which will increase with new edges and decrease with deleted edges. The destination vector of a particular partition will still remain the same if only the edges change. New nodes are added into a new partition and if any nodes are deleted, the size of the destination vector would decrease. However, none of this would impact the fact that the PageRank of these nodes can be computed independently to obtain a global approximation for a good starting vector. The key point to be noted is it is necessary to recompute local PageRank for all the partitions whose edge information have changed, such as partition where we have missing edge between node ‘13’ and ‘15’ as illustrated in Figure 25.

- **Changes involving both old and new node**: The third case involves new edges between the old nodes and new nodes. This can be split further into two cases:

  Outgoing edge from new node to old node: This case arises when a new directed edge exists between newly formed nodes to an already existing node. This case is illustrated by the edge between node ‘19’ and node ‘20’ in Figure 25. Such a scenario does results in the increase in the link file size of the both the new partition and the partition to which the old node belongs to. This would mean it is imperative to re-compute the PageRank scores of the nodes in the partition in the previous time instance in addition to computing the PageRank scores of the new partition.

  Incoming edge from old node to new node: This case arises when there is a new edge from an old node to a new node. This is illustrated by the edge from nodes ‘9’, ‘10’, ‘11’, ‘13’ and ‘15’ in Figure 25. This would mean that it is necessary to compute only the PageRank of the new partition to accommodate this change.

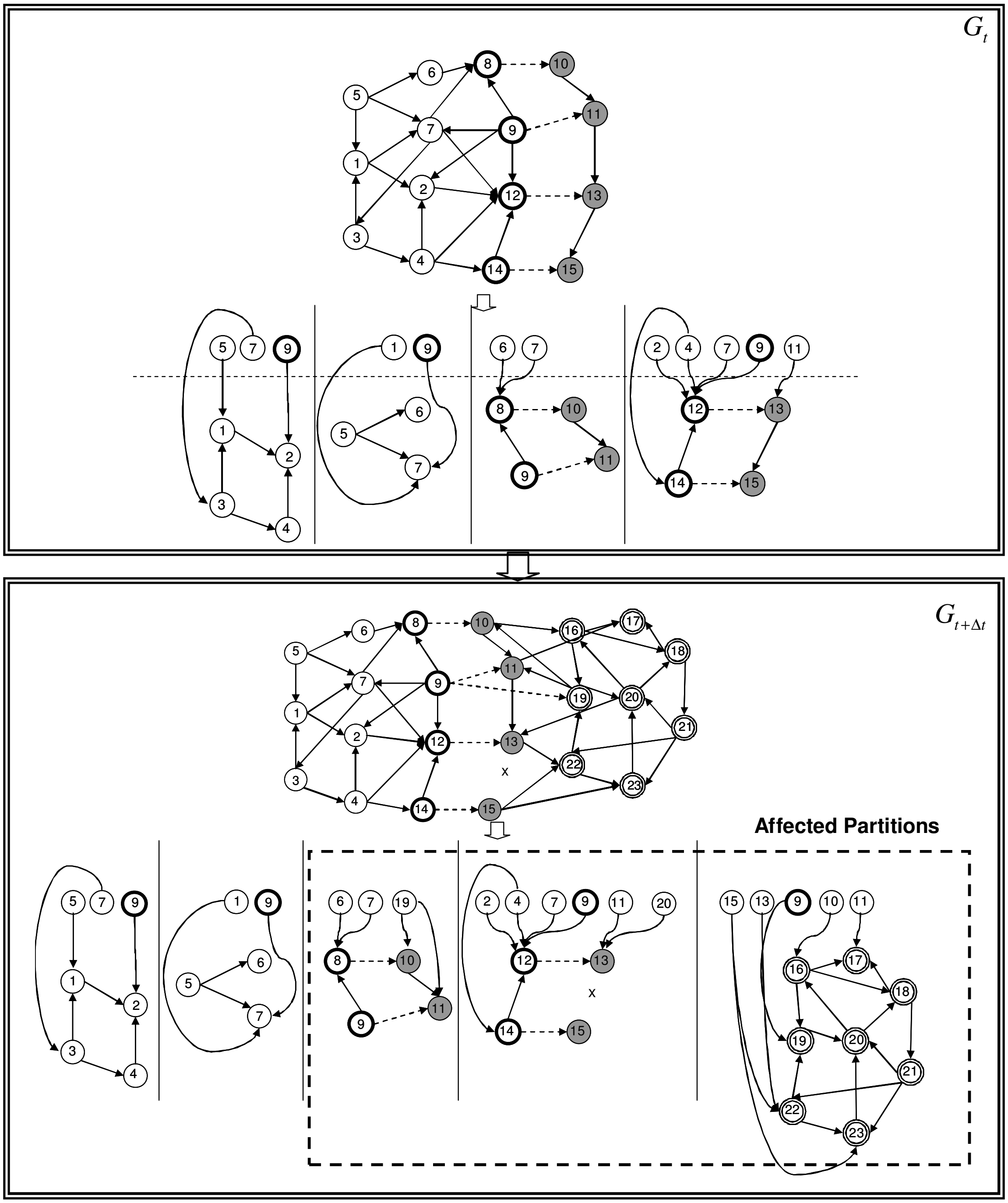


Figure 25. Capturing graph evolution as changes across old partitions and new partition

The above paragraphs discussed approaches to bin the new nodes into a new partition and how to accommodate all other changes with existing partitions. How these changes are going to affect the partitions and as a result identify the partitions that would need re-computing, was also taken into account. As a final step, the incremental PageRank algorithm proposed in earlier work [DPS+05] is used to re-compute the local PageRanks of the partitions that have undergone a change. And basic or memory-efficient approach was used to compute the local PageRank of the new nodes. These results are used to start as a good approximation to Global PageRank vector. Also, if there are partitions which remain totally unaffected because there has been no changes within the partition and also there is no incoming edge to the partition, one can skip those partitions use the ideas of incremental page rank to compute only for the remaining partitions and scale the values accordingly to obtain the PageRank.

5.3 Experiments and Results

The experimental set up is briefly discussed in this section, as well as results and the findings. A GT Graph generator was used to generate graphs of various sizes for the experiments. The number of edges in a graph is a key parameter, as the GTGraph generates a graph with as specified edges there are only certain numbers that can be the order of the graph to satisfy power-law requirements.

Table 1. Computation time of PR using different approaches for Single Static Graphs

V	**E**	**Num Bins**	**BPR-Iter**	**BPR-Time**	**DAC - Time**	**DAC-BPR-**	**DAC-BPR-Time**	**Total Time**	**% Time Saved**
5572643	2E+07	10	50	***5293.33***	1042.67	30	3128.00	***4170.67***	21.21
6336079	3E+07	10	51	***8098.80***	1407.60	30	4692.00	***6099.60***	24.69
6810171	4E+07	10	55	***11645.33***	2085.33	31	6464.53	***8549.87***	26.58
7132733	5E+07	10	59	***15615.33***	3128.00	32	8341.33	***11469.33***	26.55
7363573	6E+07	10	62	***19691.20***	4379.20	33	10322.40	***14701.60***	25.34

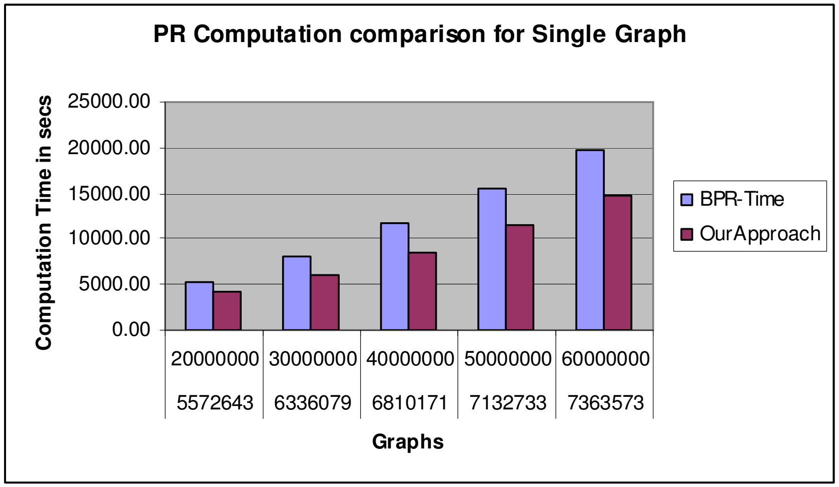


Figure 26. Plot of computation of PageRank using Memory Efficient Approach

In the first set of experiments, graphs of size 20 million to 60 million were generated and partitioned and the nodes were binned into 10 partitions. It should be noted that the size of each partition is not necessarily equal since the binning criteria is based on the number of nodes in the local destination vector.

Table 1 reflects the actual order and size of the graphs and the various run times. Figure 6 shows the perspective of how much one saves and what the trend of saving is as the size of the graph increases. The experiments reveal that the block Page Rank ('BPR') requires more time for PageRank computation than using the divide and conquer approach (DAC) to individual partitions. To arrive at an approximate global PageRank, the threshold of DAC approach was set to be 10^{-6} or maximum of 30 iterations. This enables saving time on computing local partitions but at the same time arrive at a good approximation. The threshold for convergence of actual PageRank is set to be 10^{-8}. Using the approach, one saves around 25% of the time required to compute the PageRank using the block PageRank method. This percentage of time saved using the approach (DAC-BPR Total Time) grows slightly with increase in the size of the graphs and seems to stabilize thereafter. This is due to the fact that the total time is dominated by the computation of the PageRank after determining the approximate global PageRank Vector. However, it is suspected that two factors might affect the performance variation is the size of the bin and the number of partitions.

For the experiments on evolving graphs, experiments were conducted along essentially three data points. Graph changing size from 20 million to 30 million, graph changing size from 30 million to 40 million and graph changing from 40 million to 50 million (see Table 2). The results shown in Table 3 indicate that the computation with respect to basic computation of PageRank is significantly lesser—saving from 30 percent to 40 percent of the time. However, the time saved from performing a PageRank computation on the new instance using the approach proposed for single static graph discussed in the previous section is not that high – with savings of approximately 10%. However, it was assumed a huge change in the percentage of the size of the graph, and the number of partition being small, the changes affected all partitions in some way. This resulted in computation of PageRank of all partitions. If the percentage of change is less and the graph is initially divided not many more partitions, one could reduce the number of partitions that need to be recomputed drastically and save much more time. Hopefully more experimentation might shed more light on this issue in the next couple of months.

Table 2. Experimental set up for incremental computation

Graph (t)	**V (First)**	**E (First)**	**Graph (t+1)**	**V (Second)**	**E (Second)**	**Bins Affected**
G1	5572643	20000000	G2	6336079	30000000	10
G2	6336079	30000000	G3	6810171	40000000	10
G3	6810171	40000000	G4	7132733	50000000	10

Table 3. Computation of PR using various approaches

Graph Evolution	BPR	BPR-DAC	IPR	%Save from BPR	%Save from BPR-DAC
G1->G2	8098.80	6099.60	5489.64	32.22	10.00
G2->G3	11645.33	8549.87	6912.88	40.64	19.15
G3->G4	15615.33	11469.33	9556.05	38.80	16.68

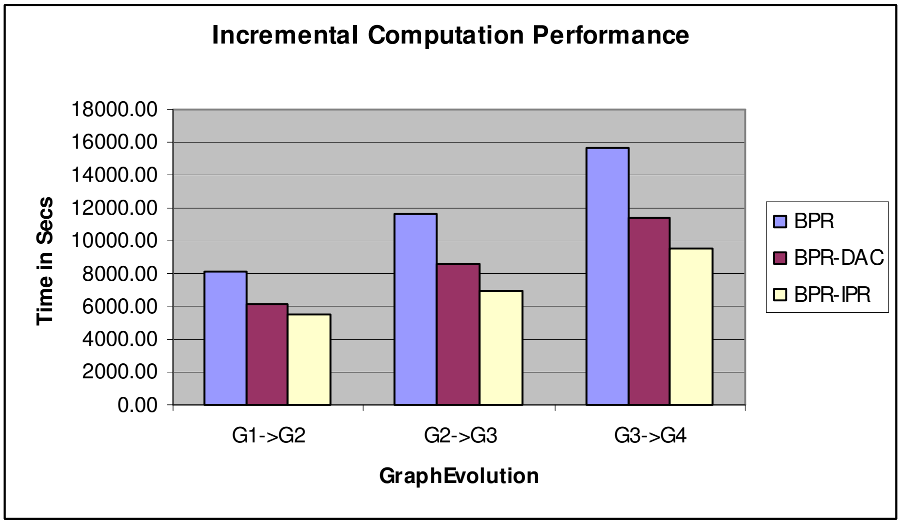


Figure 27. Incremental computation of evolving graphs

Table 4. Performance variation with graph percent change – 10 bins

Percent Change	Partitions Affected	BPR	BPR-DAC	IPR	%Save from BPR	%Save from BPR-DAC
1	1	5366.36	4209.88	1417.05	73.59	66.34
2	2	5446.13	4247.81	2247.65	58.73	47.09
4	3	5645.39	4324.97	2842.65	49.65	34.27
8	3	5959.81	4479.51	3401.56	42.93	24.06
16	5	6208.69	4787.92	3629.72	41.54	24.19

Table 5. Performance variation with graph percent change – 20 bins

Percent Change	Partitions Affected	BPR	BPR-DAC	IPR	%Save from BPR	%Save from BPR-DAC
1	2	5417.13	4252.38	1500.45	72.30	64.72
2	3	5496.59	4299.62	2331.38	57.59	45.78
4	5	5676.34	4367.97	2979.07	47.52	31.80
8	8	6003.62	4523.21	3508.59	41.56	22.43
16	11	6263.87	4847.13	3775.57	39.72	22.11

Table 6.Performance variation with graph percent change – 40 bins

Percent Change	Partitions Affected	BPR	BPR-DAC	IPR	%Save from BPR	%Save from BPR-DAC
1	3	5466.43	4317.53	1545.46	71.73	64.20
2	5	5543.75	4345.67	2375.61	57.15	45.33
4	11	5749.56	4428.59	3009.05	47.66	32.05
8	17	6105.38	4586.82	3653.52	40.16	20.35
16	22	6314.22	4937.94	3950.94	37.43	19.99

Table 7. Performance variation with graph percent change – 40 bins

Percent Change	Partitions Affected	BPR	BPR-DAC	IPR	%Save from BPR	%Save from BPR-DAC
1	5	5576.47	4417.53	1564.01	71.95	64.60
2	11	5645.13	4435.67	2412.33	57.27	45.62
4	22	5865.37	4538.59	3186.64	45.67	29.79
8	34	6159.48	4675.7	3711.20	39.75	20.63
16	45	6417.93	5007.9	4064.81	36.66	18.83

Tables 4 to 7 indicate the variation of performance as the percentage of graph changes and the number of partitions that the graph is split into changes. Figure 28 illustrates the fact that as the percentage change in the graph increases, the proposed measures still have a significant improvement in the computational cost. However, this improvement in the cost decreases non-linearly with increase in percentage change in the graph. This is due to the fact that the number of partitions needed to recomputed increases.

Figure 30 below illustrates variation of the computational cost as the number of bins or partitions increases. It can be seen for given percentage of change – which is 1% change for Figure 30 – as we increase the number of bins the computational costs also increase almost linearly. This is due the number of redundant edges as the number of bins increase.

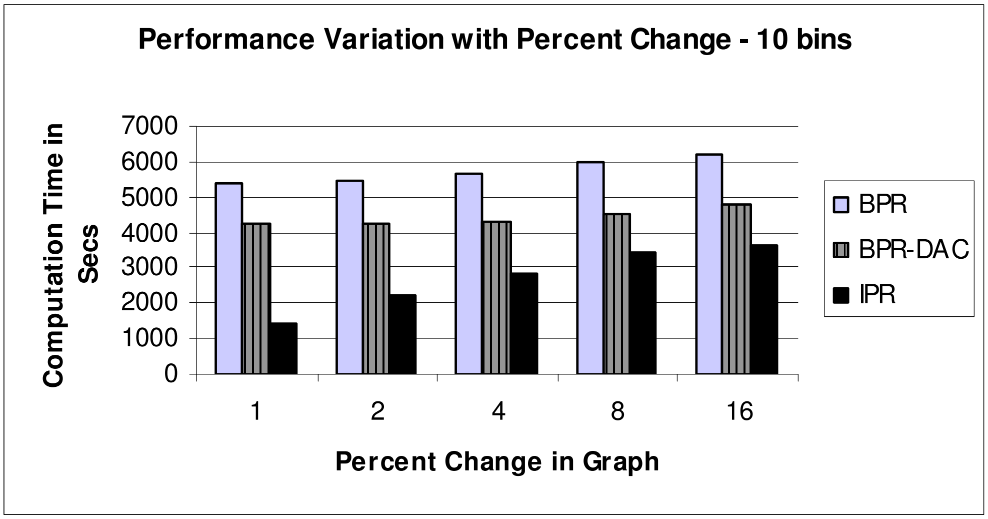


Figure 28. Performance variation with graph percent change

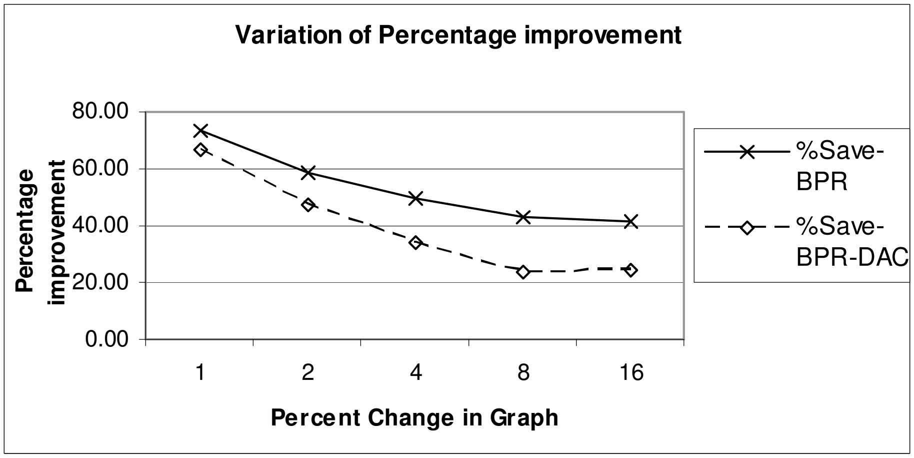


Figure 29. Percentage improvement with percent change in graph.

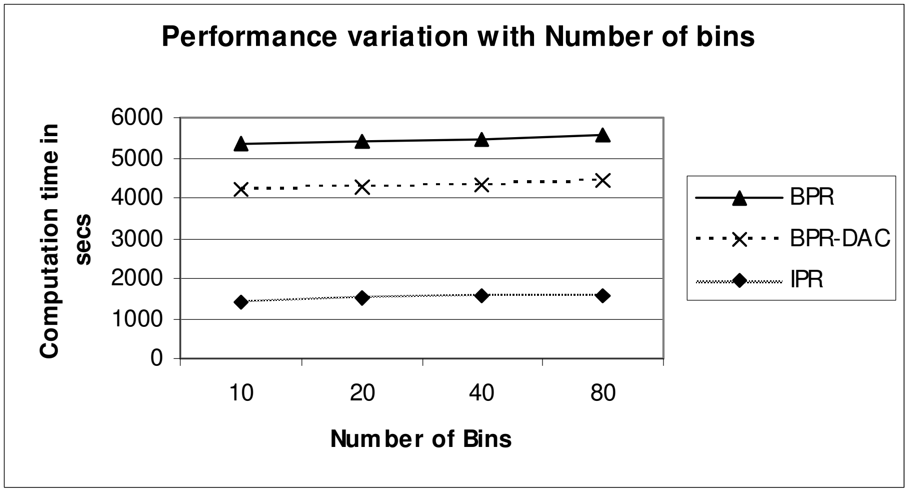


Figure 30. Performance variation with Number of Bins

5.4 Conclusions

This chapter has addressed the problem of efficient computation of PageRank for which the entire graph cannot fit in the memory. In this respect, two classes of problems have been addressed: computation for a large single graph and computation for large graphs that evolve. The approaches are based on the criteria that reducing the number of iterations by starting with a PageRank vector close to the final PageRank vector can save time. While this idea of saving on iterations is itself not new, the way one approaches the problem using the novel divide and conquer approach for individual partitions and proposing schemes that are independent of the nature of the graph (power-law or random) and independent of the domain the graph represents (such as the Web) are key contributions for efficient computing of single large static graph. The approach for efficient computation of evolving graphs is a key contribution that shows the effectiveness of the scheme to bin the changed portion and use the earlier incremental page rank version for computing local PageRank on individual partitions that have changed. The experiments show that one saves on computation even though the current experiments are not designed to show the best effects of the approach by presenting a more realistic scenario of change. However, further improvement can be achieved by parallelizing the computation of individual partitions instead of sequentially computing as done so in current experiments. It has been demonstrated that one can save time without having to explicitly use any binning strategy such as grouping by domains and computing block ranks. However, domain specific low-cost binning strategies, such as grouping nodes by website domain in the Web domain, can help improve the overall performance as revealed by earlier work [DPS+06].

An important area of future research is on an ideal partitioning or binning strategy that helps in improving the computation time. For example, it is possible to find clusters of graphs using graph partitioning approaches it would reduce the number of inter-partition edges and hence the cost of PageRank computation by reducing the δ factor mentioned in section 3. There exists many graph partitioning algorithms that assume the graph to fit in the memory. In this case, if one can fit the graph in the memory, one could use the basic divide and conquer approach to compute the PageRank. Hence, it would be interesting to explore I/O based graph partitioning techniques that can help in saving the overall computation of PageRank like measures. Another key area of future work is to see how some of these techniques can be extended for other link based relevance measures, such as hubs and authorities.

Chapter 6

CASE STUDY:ANALYZING NETWORK TRAFFIC TO DETECT E-MAIL SPAMMING MACHINES

6.1 Introduction

Cyber Security has emerged as one of the key areas of research interest with increase in information stored online and the vulnerability to attacks of such an information infrastructure. Over the years, the dependency on information infrastructure has increased, and so has their sophistication and potency. There have been intelligent and automated tools that exploit vulnerabilities in the infrastructure that arise due to flaws in protocol design and implementation, complex software code, mis-configured systems, and inattentiveness in system operations and management. The most common exploit seen is the buffer-overflow attack [CWP+00].

Technological advancements on the Internet have contributed very significantly in making information exchange very easy across the globe. E-Mail is the most popular medium for individuals to communicate with each other. However, such an effective communication medium is being increasingly abused. According to a recent survey, the number of spam mails has increased from 8% in 2001 and 50% in 2004 [GH2004]. This alarming increase in the rate of spam mails is of concern for operational as well as security reasons. The total estimated cost incurred due to spamming was around $10B/yr in US (2002) [GH2004]. To the cyber-security community, this is of concern, especially when machines inside a sensitive network are sending spam or huge amounts of information to the outside. Also, of interest are machines from outside the network that try to scan to use the exploits in the machines inside the network. It is very critical to differentiate such machines from those that are sending mail normally.

In this chapter, the issue of identifying the machines that are sending spam, or machines that have been compromised and are being used as a spam relay is addressed. Note that the focus is not on identifying individual users who send spam, or filtering an e-mail as spam based on its content. There has been work in such areas which is not directly related to the work discussed in this paper [K2003, SDH+98, SHW+03]. Recent work on detection of spam Trojans suggests the use of signature and behavior based techniques [San2004]. However, using signatures will fail to detect novel attacks at an early stage and require looking into message content. Dealing with such problems would require availability of data that would be sensitive with respect to security and privacy which limits the applicability of these techniques. The techniques have been implemented as a part of the MINDS project [EEL+04] and initial work was published in a research paper[DS2004b].

In section 6.2, various kinds of data that can be analyzed from e-mail traffic, and the levels of privacy involved will be discussed. Section 6.3 gives a brief overview of link analysis techniques that can be applied for network security. The approaches are explained in detail in Sections 6.4 and 6.5. Results of experimental evaluation of the approaches are presented in Section 6.6. Section 6.7 discusses other works that are related to this topic. Finally, Section 6.8 will point to future directions.

6.2 E-Mail Architecture and Privacy Issues

Electronic Mail is technically a file transfer from one machine to another and is initiated by the sender. The architecture of this service is illustrated in Figure 31. The *Mail Client* is responsible for creating the message files and sending and receiving them at the host level.

The *Mail Client* handles the part of transferring a file to or from a mail server. The *Mail Server* handles the message files received from various mail clients within its network, and transfers them to the Internet where other mail transfer agents transfer the files to the mail servers of respective destinations. A receiving *Mail Server* is responsible for putting the received message files in mailboxes of the respective users. The *Mail Client* at the recipient end can retrieve the message files from the *Mail Server*. The transfer of messages between a mail server and other mail transfer agents within the Internet takes place via a TCP connection using the SMTP protocol. The transfer between a client and the local mail server uses protocols such as POP or IMAP. It should be noted that all emails do not necessarily pass through the mail server and a client can open a connection on a different port and communicate directly to another machine. The border router collects all information about the network connections made in and out of the network.

It can be seen that with this architecture, data can be collected at different points. Data collected at such point reveals different kinds of information and with different granularity and privacy levels. The kinds of information that can be extracted will now be discussed, as well as the respective levels of privacy intrusion. The darkness of the shaded boxes indicates the level of privacy intrusion in Figure 31.

Mail Client Data: The data that can be collected at this level is primarily the files that have been transferred and received. These files contain information about all the people the user sent mail to or received mail from, the date and time of such transfer. Mail clients also contain meta data such as the folders in which these files are stored, the mails that been replied to, forwarding, and more recently introduced concept of ‘conversations’. Other interesting information that can be obtained at a meta level is the contact information from the address book. Such data has high level of privacy intrusiveness.

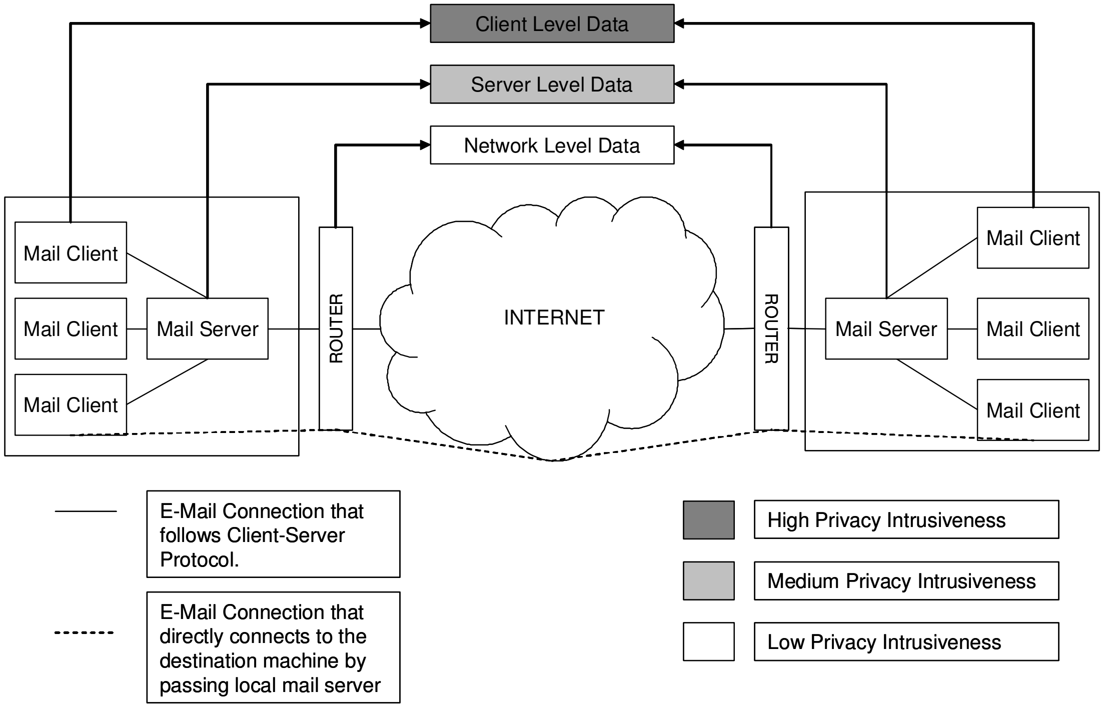

Client Level Data
Server Level Data
Network Level Data
Mail Client
Mail Client
Mail Server
Mail Client
ROUTER
INTERNET
ROUTER
Mail Client
Mail Server
Mail Client
Mail Client
E-Mail Connection that follows Client-Server Protocol.
E-Mail Connection that directly connects to the destination machine by passing local mail server
High Privacy Intrusiveness
Medium Privacy Intrusiveness
Low Privacy Intrusiveness

Figure 31. Architecture of electronic mail

Mail Server Data: The data that can be obtained at this level is the set of all files that have been transferred. These files can reveal who communicated with whom, when and about what topic. The level of granularity is fine, as everything that has been exchanged between the sender and receiver of email is known. The main difference between the data at the Mail Server level versus the Mail Client Level is the meta-data for each user discussed earlier. The level of privacy intrusion still remains high, as all information about the content of the file exchanged is available.

Network Level Data: These include data that can be collected at the network interface levels. The two main kinds of such data are the Tcpdump data and Netflow data. Tcpdump data contains a log of all the packets that passed the network sensor, including the packet content. Thus, the data provides a fine level of information granularity, which can lead to high level of privacy intrusiveness, though analyst may not be able to figure out the exact conversation if secure protocols such as SSL are used. Netflow data on the other hand is collected from routers (e.g. Cisco, Juniper). Each flow is a summary of traffic traveling in one direction in a session. When the router tears down a flow, a flow record is created. This flow record contains basic information about the connection, such as source/destination IP/ports, number of packets/bytes transferred, protocol used, and cumulative OR of TCP flags. However, flow records do not contain payload information. An email service connection that uses the SMTP protocol typically has the destination port as 25. The Netflow data has medium granularity of information and the privacy intrusiveness is at a much lower level as compared to the data obtained at the client level or the server level.

6.3 Link analysis techniques for network intrusion detection

Link analysis can be viewed as primarily used for two purposes namely, integration of different data sources, and profiling the system or user interactions. For example, Netflow data gives traffic flowing in one direction and hence a directed graph can be built at the level of an IP address or port. If one uses TCP dump data, additional information about the content will be available and one can weigh the nodes and links accordingly to get a better picture of actual traffic. The traffic data will help in building graphs that reflect system interactions. Link analysis can then be used to find 'communities' of systems that have similar interactive behavior patterns. At the host level, syslogs can be used to model the sequence of commands (or the applications executed one after other can be connected by a link) as a graph and profile the host based on the command-command graphs. A mapping between the user (or a machine) and the list of commands issued (executed) will enable the profiling of users (machines) that execute these commands (run the applications) frequently. For example, analysis of a bipartite structure, with users (machines) as one set and the commands (applications) as the other set, would identify a group of users (machines) with similar behavior patterns. Information from server logs such as the web server or the database server can also be integrated. Link analysis techniques can be applied BGP router information to identify communities of networks that have similar usage pattern, and also key router locations that need to be monitored. The trade-off in privacy for the various kind of data was discussed in the earlier section.

Most techniques in link analysis have so far concentrated on identifying prominent normal behavior. Other techniques such as attack graphs [SHJ+02] have modeled possible plans based on a formal logic approach and have an underlying assumption that all events are observable. This makes them incapable of detecting novel attacks. Hence, there is a need to define measures for anomalous behavior in the link graph terminology to help detect attacks. Furthermore, most techniques developed so far have been related to static graphs. However, the network topology keeps changing and so do user patterns, and hence there is a need to develop robust techniques for evolving graphs. For long-term analysis, historical data of attacks or anomalous behavior can be collected and used to identify nodes that have been prominent 'perpetrators' and nodes that have been most 'vulnerable'. In summary Link Analysis Techniques for Network Security can be used to:

- Identify nodes (machines) and edges (connections) that are anomalous in behavior.
- Identify nodes highly likely to be possible sources of attack or are vulnerable over a period of time.

- Identify ‘communities’ of machines involved in ‘normal’ as well as ‘anomalous’ connections.

- Study the changing behavior of connections by analyzing temporal behavior of graphs.

6.4 Proposed approach

E-mail servers traditionally send and receive mails from other e-mail servers. Thus, e-mail servers among themselves form a community due to interactions with each other. More precisely, they form among themselves a dense bipartite graph. One may utilize this behavior of e-mail servers to profile normal versus anomalous behavior. In the following sub-section, an existing approach to identify such bipartite graphs that have been used in other domains such as the web will be discussed. It will then be described how one may utilize this to detect anomalous behavior of e-mail servers.

Existing link analysis techniques fail to detect machines that send spam or are used to relay spam. Most techniques are used to mine for behavior that is normal and dense within a community, as opposed to anomalous or rare behavior. To detect e-mail spamming machines one must differentiate their behavior from those of the e-mail servers. Both of them will tend to have high outgoing traffic. However, an e-mail server tends to send e-mails to only other e-mail servers whereas a spamming machine sends mail to all machines. This behavioral aspect to detect the potential perpetrators will be made use of.

The following sequence of steps is followed:

1. Pre-process the netflow data and construct the graph for e-mail connections.
    - Graphs can be constructed for patterns that represent other kind of services like ftp.
    - Node can be an IP or AS or port or any combination depending on the problem. Analysis is done at an IP Level.
2. Perform the HITS Algorithm on the generated graph.

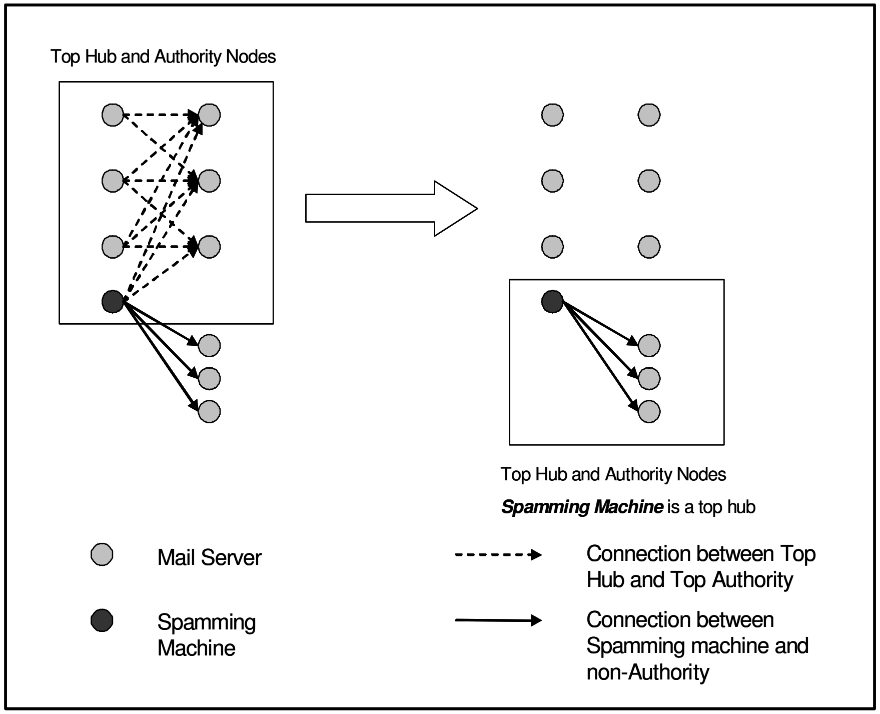

Top Hub and Authority Nodes
Top Hub and Authority Nodes
Spamming Machine is a top hub
Mail Server
Spamming Machine
Connection between Top Hub and Top Authority
Connection between Spamming machine and non-Authority

Figure 32. Identifying spamming machines

- The nodes with top hub and authority scores represent typical e-mail servers

3. Remove edges between top k% of hubs to top k% authorities.
   - These top k % connections correspond to normal e-mail traffic between regular mail servers that have high hub and authority score.
4. Perform the HITS algorithm on the resultant graph.
   - A simple out-degree also works fine on the resultant graph.
5. The new scores are the *Perpetrator Scores*.
   - Spamming machines obtain high rank compared to other e-mail servers.

It can be seen that the approach is two-fold. Firstly, it identifies connections between regular mail servers. Such connections form a dense bipartite graph between servers, assigning them high hub and authority scores. All such connections that contribute to normal e-mail traffic are then removed. Note, only the edges are deleted and not the nodes. This eliminates normal e-mail server behavior. The second step identifies machines that behave like servers and have high traffic that does not correspond to regular e-mail connections. These machines are most likely spamming, since they send mails to a lot of other machines that do not take part in regular e-mail connections. Since no node is deleted, such an approach also helps to identify e-mail servers that are affected and sending spam. Figure 32 illustrates this concept clearly.

6.5 Rank evolution

The evolution of the network graph is analyzed at a single node level. For each node, its rank is determined based on its *Perpetrator Score*(*PScore*) and call it *Perpetrator Rank.* Then another metric is defined based on its *Perpetrator Rank*(PR) called Perpetrator Height. The height is a measure of 'how far' a node is from an infinitely low ranked node. For a node i at a time t, its *Perpetrator Height* can be expressed as:

$$PHeight_{it}=\log_2(1+1/PR)$$

Note that for a top ranked node, *PR*=1 and its *PHeight*=1. For a node with almost infinite rank, PR=∞ , and its *PHeight* would be zero. Then study the rate of change in the rank of a node over time. The change for a time period Δ *t* can be defined as:

$$v = \Delta\ PHeight/\Delta\ t$$

Since the interest of this study is in the change and not in a negative or a positive change in the rank (for the present work), take the square of *v* for the analysis of how the node behaves. It is then essential to weigh the node according to the perpetrator score, *PScore*. This is done because a small change in a highly ranked node or a big change in a low ranked node is more interesting than a small or moderate change in a low ranked node. Now it is possible to define a quantity Rank Energy of a node as:

$Rank\ Energy = Weight^{*}\ v^2$

This measure would be a good indicator of any rapid changes in the network behavior of machines. Such a rapid change would be of particular interest to the security analyst as it may indicate machines suddenly spamming or a mail server going down. Also, though *PScore* is presently used to weigh the node, the node can be weighed on other factors such as inside the network versus outside the network. The weight factor can be a vector of properties inherent to the node. The strength of the approach lies in its ability to detect anomalous behavior at an early stage.

6.6 Experimental evaluation

Experiments were performed to evaluate two kinds of analyses. Firstly, the focus is on identifying potential perpetrators given netflow data for a 10 minute time window. Second, we observed at a 3 hour time period and analyzed the rank evolution of each node. Details are discussed in the following sections.

6.6.1 Analysis at a single time instance

The first dataset was netflow data for the University for a 10 minute window from 07:10 to 07:20 hrs on June 17th, 2004. The total number of flows during this time period was 856470, with 228276 distinct IPs. Of these the number of connections that used SMTP protocol for E-Mail was 10368, with 1633 distinct IPs.

Using the approach described in section 6.4, the nodes are ranked according to their perpetrator scores. It was found that all main email servers were ranked low. Among those that were ranked on the top were, small e-mail servers that did not have traffic to the scale of the main e-mail servers. Most importantly, it was possible to detect a machine, at address 134.84.S.44, that was known to be sending spam during that time period. This particular machine was ranked 2nd when ordered according to Perpetrator Score. It was also noticed that once the edges between the top hub and top authorities were removed, a simple out-degree of the resultant graph also gave a fair measure of anomalous behavior. The rank of this machine according to authority scores was 1563, indicating that it was sending mails and not receiving them. The results are shown in Figure 33.

Total Flows: 856470
Email Flows: 10368
Distinct IPs (Total): 228276
Distinct IPs (Email): 1633

At this time, **134.84.S.44** was known to be sending spam. All of the other hosts were known, good email servers that were sending email

Sorted by Hub Score

IP Address	Authority Score	Hub Score
128.101.X.109	0	0.728289
134.84.S.44	**0**	**0.033964**
160.94.X.36	0	0.02685
160.94.X.35	0	0.02016
160.94.X.35	0	0.016173
160.94.X.36	0	0.014935
160.94.X.36	0	0.014778
128.101.X.119	0	0.013571
160.94.X.67	0	0.011118
160.94.X.33	0	0.010552
160.94.X.35	0	0.007896
160.94.X.33	0	0.006688
134.84.X.117	0	0.006529
128.101.X.10	0	0.005942
134.84.X.172	0	0.005282
134.84.X.4	0	0.005127
128.101.X.21	0	0.005016
128.101.X.1	0	0.004601
160.94.X.33	0	0.004492
160.94.X.100	0	0.004374

Sorted by Outdegree

IP Address	Indegree	Outdegree
128.101.X.109	0	363
160.94.X.36	1	176
134.84.S.44	**0**	**147**
160.94.X.35	1	112
160.94.X.36	1	106
160.94.X.36	1	103
128.101.X.119	0	99
160.94.X.35	1	92
160.94.X.35	1	60
160.94.X.33	0	45
160.94.X.33	0	45
160.94.X.33	0	36
128.101.X.10	0	33
134.84.X.4	0	28
134.84.X.2	0	26
128.101.X.2	0	26
134.84.X.172	0	25
160.94.X.11	0	24
160.94.X.34	0	22
128.101.X.104	0	21

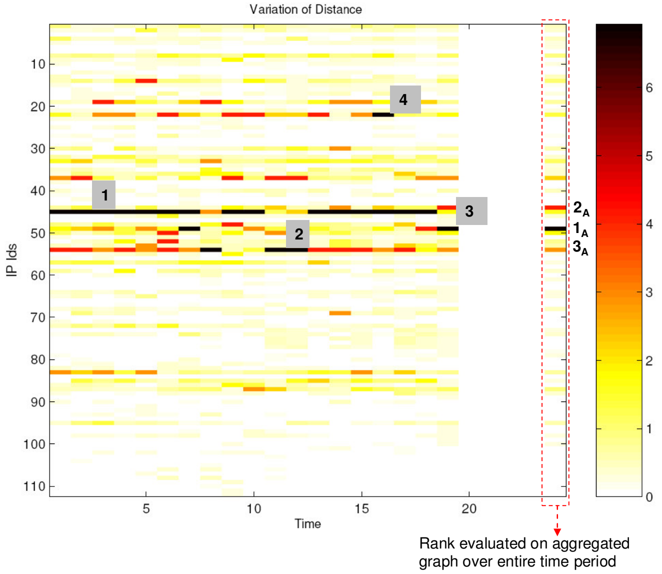
Figure 33. Identifying perpetrators

6.6.2 Analysis of rank evolution

The second dataset was netflow data for the University for a three hour time period from 7am to 10am on July 21st. Graphs were constructed for each ten minute period, to obtain a set of eighteen graphs for this time period. The results are depicted in Figure 34.

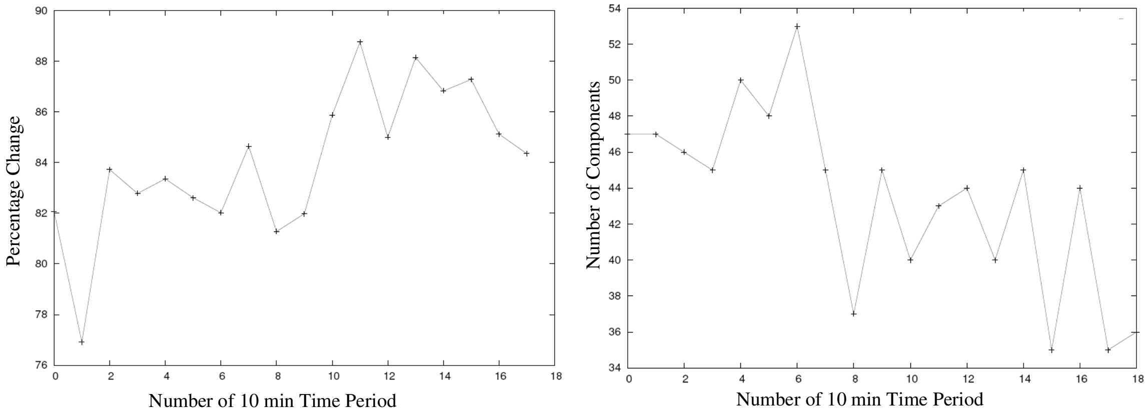
Perpetrator Scores using PageRank as measure were generated for each time instance and determined the rank of each node for that time period. The shading is a reflection of node rank. The top ranked node has a darker shade. Each column indicates one time period, and each row is an IP. For an IP not present in a time period, a default score of zero was assigned. Thus, the picture on the left indicates the variation of rank of the nodes. The last column is ranking of the node for the aggregated time period.

It can be seen that the sudden changes in the node ranks, for certain machines, can be eclipsed by high change in one node, when computed for an aggregated time period. For example, the ranking order computed studying the evolution reveals ranked node "4" in Figure 34, which did not show up when ranking was performed on an aggregated graph. The security analyst at computer science department confirmed that the ranked nodes using the evolution approach during that time period were of interest, asserting the value of such a ranking method.



Rank evaluated on aggregated graph over entire time period

Figure 34. Variation of PageRank Distance of IPs across time



(a)

(b)

Figure 35(a) Variation percent change in graph across time period. (b) Number of components in the graph instances across time.

Figure 35 illustrates how the graph properties over different time instances vary. It can be seen that there percentage change in the graph is really high across time period. Also of interest is the graph is not well connected. At any time instance there are a number of components for a given graph. This seems typical behavior of very dynamic connections such as usage graphs or network connections. PageRanks were computed using BPR-DAC approach for these set of graphs.

6.7 Related Work

E-Mail Spamming has been a prominent area of research and different approaches have been taken to solve this problem. The two main class of problems studied have been 'spam email filtering' and 'detection and prevention of virus/worm intrusion and spreading'. Spam analysis can be broadly classified into content based techniques and flow statistics based techniques. There are commercial products that use signatures developed by analyzing the content [Bri2008]. Collaborative filtering approaches have also been developed by analyzing the content [Clo2008]. Classification based approaches that use heuristics or rules such as SpamAssasin [Spa2008] are also popular. MSN8[SDH+98] uses Bayesian based approaches to classify e-mails as spam. However, all these techniques have high privacy intrusiveness as they analyze the e-mail content.

Behavior based techniques such as the E-Mail Mining Toolkit [SHW+03] use user profiles to construct user cliques and analyze the e-mail attachment statistics for detection of e-mail worms or viruses. However, such techniques also need to obtain data at least at the mail server level and have a medium level of privacy intrusiveness. Sandvine Incorporated [San2004] suggests the use of behavior based techniques coupled with signature based techniques for detection of spam Trojans. However, signature based methods fail to detect novel attacks at an early stage and such an approach would require looking into message content, raising privacy concerns. Also, the technical details of behavior based approach in the work are not clearly described.

The goal in this work is not to identify individual users sending spam or classifying an individual email as a spam. Instead, the focus is on detecting machines that are sending spam and capturing e-mail traffic that does not necessarily pass through an e-mail server or use a particular user id or a mail client. Compared to ‘receiver based’ approaches such as content filtering, and ‘sender based’ approaches such as IP blocking; the approach is in the complementary area of ‘transport based’ approaches where the e-mail is suppressed by stopping the misbehaving mail system machine. In addition to being less privacy intensive, it is believed that this is also a new and complementary approach to spam reduction.

6.8 Conclusions

This chapter has presented a case study for studying evolving graphs and utility of computing PageRank like measures for such graphs over time. It has been proposed that an approach to detect anomalous behavior in E-Mail traffic at the network level, with low privacy intrusiveness. Finally, a framework for studying evolving graphs has been presented, along with an explanation of how it can be applied to network traffic for early detection suspicious behavior. The work was restricted to that of a single node.

Chapter 7

CONCLUSIONS AND FUTURE DIRECTIONS

The final chapter of this dissertation provides a summary of what can be learned from this dissertation. The study of evolving graphs helps in understanding and profiling the behavior of the phenomenon it models. A key basic step in this study involves identifying properties of interest in a given graph. For a given graph, there are certain basic properties such as *order* and *size*, derived properties such as *degree distributions,* and certain conceptual properties which are based on certain knowledge models. The study of such properties is discussed extensively in Chapter 2, which provides a generic approach to developing link analysis techniques. This dissertation addresses challenges involved in the study of large evolving graphs with a particular focus on the computation of first-order Markov measures, which have gained huge popularity because of their utility in a variety of domains.

The key challenges necessary to handle the problem mentioned above were: developing efficient work-reduction techniques, handling large graphs, and handling large graphs as they evolve. The initial challenge was to identify a theoretical framework to partition graphs for the computation of first-order Markov measures. Based on this framework, work reduction techniques for handling large graphs at a given time instance and large evolving graphs were proposed. An extension of this work to handle i/o based computation for large graphs was developed. Experimental results of the proposed approaches show significant improvements for power-law graphs. For random graphs, the benefits of the proposed approach wanes as the percentage change in the evolved graph increases. This is due to the fact that a certain percentage of change affects the whole graph, and hence the measure needs to be computed for the whole graph irrespective of the fact that certain portions of the graph have not changed.

7.1 Contributions and benefits

- The definition of first-order Markov measures is introduced in chapter 1.

- A survey of link analysis techniques and classification of the literature on link analysis for Web is presented. *Key benefit*: Identification of key dimensions that span the design space for Hyperlink Analysis. Illustrates where existing approaches are similar and where they complement each other. Points to a methodology for applying hyperlink analysis techniques.

- A theoretical framework for graph partitioning for first-order Markov measures is defined. *Key benefit*: Model assures accuracy of measure computed using such a partition.

- A divide and conquer approach to use the above framework to improve computational efficiency is proposed. *Key benefit*: This approach reduces computational cost.

- An incremental computational approach using the defined theoretical framework is proposed. *Key benefit*: Accuracy of the resulting computed measure and computational efficiency.

- I/O based techniques for large single graphs or large evolving graphs. *Key benefit*: Scalability of computation.

7.2 Future Directions

The work and study involved in this dissertation provide certain areas that need to be explored further, with definite scope of interest and benefits from such explorations. These directions are broadly discussed below.

- **Medium term goals**

Computational methods for relevance measures, such as Hubs and Authorities remains to be explored. While this dissertation examines efficient computation techniques of first order Markov measures, other relevance measures applicable to Web domain and measures used on other networking domains, such as social networks, computer networks. Also, current link analysis techniques have focused on measures for a single graph. Most link analysis techniques for evolving graphs have focused on studying the behavior of these measures over a period of time. However, there is a clear lack of measures that model time as an integrated part of the model. The study of evolving graphs requires new models and measures that are time-aware. Once such models are identified, there is a scope for exploring algorithms for computing such *time aware measures*.

- **Long term goals**

Graph evolution models have primarily focused on how networks grow or evolve over a period of time. However, all real networks involve growth. The concepts of birth, decay and rate of change has not been entirely explored. While citation networks, the World Wide Web, and certain biological networks can be modeled as growing networks, not all modes of interactions between objects are about growth. Usage based graphs, contrary to creator based structural graphs, depend on the time interval in which they are captured, reflecting the usage behavior in the particular time interval. Typical examples include traffic on transportation network, Web usage graphs, or telephone calls. Patterns of behavior of such networks across time are not captured by graph-based models.

The issue of graph storage has captured the attention of researchers for a long time. However, with enabling technologies to collect and store large graphs over a period of time, there are multiple challenges. The challenges are not just about how to store a large graph, but how to store the graph as it evolves. Incremental maintenance of such graphs as they evolve is a challenge. More specifically, it becomes a challenge when an application requires computing a class of measures. When this occurs, the storage models need to not only provide appropriate access for computation of such measures, but also provide efficient maintenance and access for computing such measures over time.

BIBILIOGRAPHY

[AB2002] R. Albert, A. L. Barabási. Statistical mechanics of complex networks. Reviews of Modern Physics 74, 47-97 (2002).

[AJB99] R. Albert, H. Jeong, and A. L. Barabasi. Diameter of the World Wide Web, Nature 401: 130-131, Sep 1999.

[AK2005] A. B. Amine and G. Karypis. Multi-level Algorithms for Partitioning Power-law Graphs [R]. Technical Report, Department of Computer Science, University of Minnesota, 2005.

[Bri2008] BrightMail,
http://www.symantec.com/business/products/family.jsp?familyid=brightmail

[B2000] A. Broder et al, Graph Structure in the Web. In the Proc. 9th WWW Conference 2000.

[B2002a] A. L. Barabasi, Linked: The New Science of Networks. Cambridge, Massachusetts: Perseus Publishing, 2002.

[BH1998] K. Bharat and M. R. Henzinger, Improved algorithms for topic distillation in hyperlinked environments. In Proceedings of the 21st International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 104–111, 1998

[BM2006] D. A. Bader and K. Madduri. GTgraph: A suite of synthetic graph generators. http://www.cc.gatech.edu/~kamesh/GTgraph.

[BP1998] S. Brin and L. Page, The anatomy of a large-scale hypertextual Web Search engine. In 7th International World Wide Web Conference, Brisbane, Australia, 1998.

[BRR+01] A. Borodin, G. O. Roberts, J. S. Rosenthal, and P. Tsaparas (2000), Finding Authorities and Hubs From Link Structures on the World Wide Web, WWW10 Proceedings , Hongkong, May 2001

[Clo2008] CloudMark, http://www.cloudmark.com/

[C2000] S. Chakrabarti. Data Mining for hypertext: A tutorial survey. ACM SIGKDD Explorations, 1(2): 1-11, 2000.

[CCM00] H. Chang, D. Cohn, and A. McCullum. “Learning to Create Customized Authority Lists.” In Proceedings of the 17th International Conference on Machine Learning, 127–134, 2000.

[CDG+98] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan. “Automatic resource compilation by analyzing hyperlink structure and associated text”. Proceedings of the Seventh International World Wide Web Conference, 1998.

[CH2001] D. Cohn and T. Hofmann. “The missing link—a probabilistic model of document content and hypertext connectivity.” Advances in Neural Information Processing Systems, 13, 2001.

[CKS02] T. Carpenter, G. Karakostas, and D. Shallcross, Practical issues and algorithms for analyzing terrorist networks, Proc. WMC 2002, 2002

[CLEVER]The CLEVER project, IBM, http://www.almaden.ibm.com/cs/k53/clever.html

[CMS97] R. Cooley, B. Mobasher, and J. Srivastava, Web Mining: Information and Pattern Discovery on the World Wide, In Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'97), Nov 1997.

[CWP+00] C. Cowan, P. Wagle, C. Pu, S. Beattie, and J. Walpole, “Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade”, DARPA Information Survivability Conference and Expo (DISCEX), Hilton Head Island SC, January 2000.

[DDS09] P. Desikan ,C. DeLong, and J. Srivastava "Link Analysis in Web Mining: Techniques and Applications", submitted to Semantic Computing (Editors: Phillip Sheu, Heather Yu, C. V. Ramamoorthy, Arvind K. Joshi, and Lotfi A. Zadeh) IEEE Press/Wiley, 2009.

[DFJ05] L. Ding, T. Finin and A. Joshi. Analyzing Social Networks on the Semantic Web. IEEE Intelligent Systems,2005.

[DPS+05] P. Desikan ,N. Pathak, J. Srivastava and V. Kumar "Incremental PageRank computation on evolving graphs", In: 14th WWW Conference on May 10–14, 2005.

[DPS+06] P. Desikan, N. Pathak, J. Srivastava and V. Kumar, "Divide and Conquer Approach for Efficient PageRank Computation", ICWE, Palo Alto, July 2006.

[DSK+02] P. Desikan, J. Srivastava, V. Kumar, P. N. Tan, "Hyperlink Analysis—Techniques & Applications", Army High Performance Computing Center Technical Report, 2002.

[DS2004a] P. Desikan and J. Srivastava, "Mining Temporally Evolving Graphs", WebKDD, Seattle (2004).

[DS2004b] P. Desikan and J.Srivastava, "Analyzing Network Traffic to Detect E-Mail Spamming Machines", ICDM Workshop on Privacy and Security Aspects of Data Mining, Brighton, UK, November 2004..

[DS2008] P. Desikan and J. Srivastava, "I/O efficient computation of First Order Markov Measures for Large and Evolving Graphs", WebKDD, Las Vegas (2008).

[E1996] O. Etzioni The world wide web: Quagmire or goldmine. Communications of the ACM, 39(11):65-68, 1996

[EEL+04] L. Ertoz, E. Eilertson, A. Lazarevic, P. Tan, J. Srivastava, V. Kumar, P. Dokas, The MINDS—Minnesota Intrusion Detection System, "Next Generation Data Mining", MIT /AAAI Press 2004.

[ER1960] P. Erd'os, and A. R´enyi, On the evolution of random graphs, Publications of the Mathematical Institute of the Hungarian Academy of Sciences 5, 17–61 (1960).

[ER1961] P. Erd'os, P. and A. R´enyi, On the strength of connectedness of a random graph, Acta Mathematica Scientia Hungary 12, 261–267 (1961).

[EMT04] N. Eiron, K. McCurley, J. Tomlin: Ranking the Web frontier. In: Proc. 13th conference on World Wide Web, ACM Press (2004) 309–318

[ERC+00] K. Efe, V. Raghavan, C. H. Chu, A. L. Broadwater, L. Bolelli, S. Ertekin (2000), The Shape of the Web and Its Implications for Searching the Web, International Conference on Advances in Infrastructure for Electronic Business, Science, and

Education on the Internet-Proceedings.-Aug. 2000.

[Fjä98] P. Fjällström. Algorithms for Graph Partitioning: A Survey in Linköping Electronic Articles in Computer and Information Science, Vol. 3

[FLG00] G.W. Flake, S. Lawrence, C. L. Giles . Efficient Identification of Web Communities. Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. August 2000, pp. 150–160

[GGK+03] A. Grama, A. Gupta, G. Karypis, and V. Kumar. Introduction to Parallel Computing: Design and Analysis of Algorithms, 2nd Edition. Addison Wesley Publishing Company, Redwood City, CA, 2003.

[GH2004] J. Goodman, G. Hulten, “Junk E-mail Filtering”, Tutorial, KDD 2004

[GJS74] R. M. Garey, D. S. Johnson, and L. Stockmeyer, Some simplified NP-complete problems. In Proceedings of the Sixth Annual ACM Symposium on theory of Computing (Seattle, Washington, United States, April 30–May 02, 1974

[GKR98] D. Gibson, J. Klienberg, and P. Raghavan. Inferring web communities from link topology. In Proc. 9th ACM Conference on Hypertext and Hypermedia, 1998

[Goog] http://www.google.com

[H1999] T. Haveliwala, Efficient Computation of PageRank in Technical Report, Stanford University, CA, 1999.

[H2000] M. Henzinger, Link Analysis in Web Information Retrieval, ICDE Bulletin Sept 2000, Vol 23. No.3

[H2002] T. Haveliwala. “Topic-sensitive pagerank”. In Proceedings of the Eleventh international Conference on World Wide Web, 517–526, 2002.

[HNT06] T. Hope, T. Nishimura and H. Takeda. An integrated method for social network extraction. In Proceedings of 15th international conference on World Wide Web, USA, 2006.

[JW2003] G. Jeh and J. Widom. “Scaling Personalized Web Search”. In 12th

International World Wide Web Conference , 2003.

[K1998] J. M. Kleinberg, Authoritative Sources in Hyperlinked Environment, in Proc. of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 668-667, 1998

[K1999] J. Klienberg et al. The web as a graph: measurement models and methods. In Proc ICCC 1999.

[K2003] V. Krebs, "Data Mining Email to Discover Social Networks and Communities of Practice", http://www.orgnet.com/email.html, 2003

[KB2000] R. Kosala and H. Blockeel, "Web Mining Research: A Survey," SIG KDD Explorations, vol. 2, pp. 1–15, July 2000

[KHM+03] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub. Exploiting the block structure of the web for computing. Technical report, Stanford University, Stanford, CA, 2003.

[KRR+00] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. The Web as a graph. In ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pages 1–10, 2000.

[LKF05] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (Chicago, Illinois, USA, August 21– 24, 2005).

[LM2000] R. Lempel and S. Moran. The stochastic approach for link-structure analysis (SALSA) and the TKC effect. In the 9th International World Wide Web Conference, May 2000.

[M1967] S. Milgram, The small world problem, Psychology Today 2, 60–67 (1967).

[M2005] P. Mika. Flink: Using Semantic Web Technology for the Presentation and Analysis of Online Social Networks, Journal of Web Semantics 3(2), Elsevier, 2005.

[NSW01] Newman, M. E. J., Strogatz, S. H., and Watts, D. J., Random graphs with arbitrary degree distributions and their applications, Phys. Rev. E 64, 026118 (2001).

[NZJ01a] A. Y. Ng, A. X. Zheng, and M. I. Jordan (2001), Stable algorithms for link analysis. Proc. 24th International Conference on Research and Development in Information Retrieval (SIGIR), 2001.

[NZJ01b] A. Y. Ng, A. X. Zheng, and M. I. Jordan (2001), Link Analysis, Eigenvectors and Stability, IJCAI 01.

[PBM+98] L. Page, S. Brin, R. Motwani and T. Winograd (1998) The PageRank Citation Ranking: Bringing Order to the Web. Stanford Digital Library Technologies, Working Paper 1999 January 1998.

[PPT+02] B. Prasetyo, I. Pramudiono, K. Takahashi, and M. Kitsuregawa,. Naviz: Website Navigational Behavior Visualizer. In Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (May 06–08, 2002).

[RD2002] M. Richardson and P. Domingos. "The Intelligent Surfer: Probabilistic Combination of Link and Content Information in PageRank". In Proceedings of the 2001 Neural Information Processing Systems (NIPS) Conference, 1441–1448, 2002.

[RJ2005] F. Radlinski and T. Joachims." Query chains: learning to rank from implicit feedback". In Proceeding of the Eleventh ACM SIGKDD international Conference on Knowledge Discovery in Data Mining, 239–248, 2005.

[RM2000] D. Rafiei, A.O. Mendelzon (2000), What is this Page Known For? Computing Webpage Reputations, In Proceedings of Ninth International WWW Conference, Amsterdam, May 2000.

[San2004] Sandvine Incorporated, "Trend analysis: Spam trojans and their impact on broadband service providers",
http://www.sandvine.com/solutions/pdfs/spam_trojan_trend_analysis.pdf, June 2004

[Spa2008] SpamAssassin, http://spamassassin.apache.org/

[SCD+00] J. Srivastava, R. Cooley, M. Deshpande, and P. N. Tan. Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data (2000), SIGKDD

Explorations, Vol. 1, Issue 2, 2000.

[SDH+98] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz, “A Bayesian Approach to Filtering Junk E-mail” Learning for Text Categorization: Papers from the 1998 Workshop.

[SHJ+02] O. Sheyner, S. Jha, J. Haines, R. Lippmann, and J. M. Wing , “Automated Generation and Analysis of Attack Graphs”, Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA, May 2002.

[SHW+03] S.J. Stolfo, S. Hershkop, K. Wang, O. Nimeskern, and C.W. Hu, "A Behavior-based Approach to Securing Email Systems". "Mathematical Methods, Models and Architectures for Computer Networks Security", Proceedings published by Springer Verlag, Sept. 2003

[SKK02] K. Schloegel, G. Karypis, and V. Kumar. Graph partitioning for high-performance scientific simulations. In Jack Dongara, Ian Foster, Geoffrey Fox, William Gropp, Ken Kennedy, Linda Torczon, and Andy White, editors, Sourcebook on Parallel Computing, chapter 18, pages 491–541. Morgan Kaufmann, San Francisco, CA, 2002.

[WS1998] D. J. Watts. and S. H. Strogatz, 1998, Nature 393, 440

[XC2005] J. Xu and H. Chen. Criminal network analysis and visualization. Communications of ACM 48, 6 (Jun. 2005), 100-107.

[XCM+03] G. Xue , H. Zeng, Z. Chen, W. Ma, H. Zhang, and C. Lu,. “Implicit link analysis for small web search”. In Proceedings of the 26th Annual international ACM SIGIR Conference on Research and Development in information Retrieval (Toronto, Canada, July 28–August 01, 2003).

[ZHH02] J. Zhu, J. Hong, and J. G. Hughes, Using Markov Chains for Link Prediction in Adaptive Websites. In Proc. of ACM SIGWEB Hypertext 2002.